

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-078-2, название «Oracle. Оптимизация производительности» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.

Optimizing Oracle Performance

Cary Millsap and Jeff Holt

O'REILLY®

Oracle

Оптимизация производительности

Кэри Миллсан и Джефф Холт



Санкт-Петербург — Москва
2006

Кэри Миллсап, Джефф Холт

Oracle. Оптимизация производительности

Перевод П. Шера

Главный редактор	<i>А. Галунов</i>
Зав. редакцией	<i>Н. Макарова</i>
Научный редактор	<i>О. Летаев</i>
Редактор	<i>В. Овчинников</i>
Корректор	<i>Н. Аникеева</i>
Верстка	<i>О. Макарова</i>

Миллсап К., Холт Д.

Oracle. Оптимизация производительности. – Пер. с англ. – СПб: Символ-Плюс, 2006. – 464 с., ил.

ISBN 5-93286-078-2

Оптимизация производительности БД Oracle считается очень сложной задачей, подвластной лишь черной магии. Успехи настройки нередко случайны и достигаются скорее за счет интуиции, комбинируемой с методом проб и ошибок. Известные исследователи Oracle, Миллсап и Холт, в практическом руководстве «Oracle. Оптимизация производительности» подробно описывают надежный, воспроизводимый и четкий метод выявления проблем производительности системы, позволяющий с уверенностью сказать, в чем причина любой из них.

Ключом к методу Миллсапа и Холта является тот факт, что программное обеспечение БД Oracle оснащено инструментами, способными предоставить информацию о том, на что тратится время при обработке запросов. Метод включает три этапа: выбор пользовательской операции, оптимизация которой наиболее важна с точки зрения бизнеса; сбор корректно выбранных данных расширенной трассировки SQL, относящихся к данной операции, и выявление по этим данным места и причин перерасхода времени; поиск наиболее эффективного способа повышения производительности (уменьшения времени отклика) данной операции.

Авторы показывают, как применять метод, и объясняют, почему он эффективен. Метод способен помочь не только выявить проблемы производительности, но и оценить рост производительности при увеличении количества и/или мощности процессоров или добавлении оперативной памяти. Издание предназначено администраторам и разработчикам БД Oracle.

ISBN 5-93286-078-2

ISBN 0-596-00527-X (англ)

© Издательство Символ-Плюс, 2006

Authorized translation of the English edition © 2003 O'Reilly Media, Inc. This translation is published and sold by permission of O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

Все права на данное издание защищены Законодательством РФ, включая право на полное или частичное воспроизведение в любой форме. Все товарные знаки или зарегистрированные товарные знаки, упоминаемые в настоящем издании, являются собственностью соответствующих фирм.

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.

Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.

Подписано в печать 15.11.2005. Формат 70х100¹/₁₆. Печать офсетная.

Объем 29 печ. л. Тираж 2000 экз. Заказ N

Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

*Я посвящаю эту книгу, с любовью, Минди,
Александру и Николасу.*

– Кэри Миллсап

*Я посвящаю эту книгу каждому менеджеру,
готовому слушать и сидящему на мягком диване.*

– Джефф Хольт

Оглавление

Вступительное слово редактора	11
Предисловие	14
I. Методика	29
1. Лучший способ оптимизации	31
«Вы делаете это неправильно»	33
Требования к хорошему методу	35
Три важных достижения	36
Средства анализа времени отклика	42
Метод R	49
2. Выбор пользовательских операций	64
Надежность спецификации	64
Создание хорошей спецификации	70
Избыточные ограничения в спецификации	75
3. Выбор диагностических данных	77
О сборе данных	77
Область данных	81
Источники диагностических данных в Oracle	88
Дополнительная информация	90
4. Выбор пути решения задачи	91
Новый стандарт обслуживания клиентов	91
Выбор экономически оптимального пути повышения производительности	93
Анализ диагностических данных	94
Прогнозирование экономической эффективности проекта	96

II. Справочная информация	105
5. Интерпретация данных расширенной трассировки SQL	107
Знакомство с файлом трассировки	107
Справочник по данным расширенной трассировки SQL	110
Учет времени отклика	122
Эволюция модели времени отклика	131
Отсчет времени	135
Опережающее атрибутирование	140
Подробный анализ файла трассировки	142
Упражнения	145
6. Сбор данных расширенной трассировки SQL	148
Знакомство с приложением	148
Включение расширенной трассировки SQL	152
Поиск файлов трассировки	159
Устранение ошибок сбора данных	167
Упражнения	183
7. Измерение времени ядром Oracle	184
Управление процессами операционной системы	184
Измерение времени ядром Oracle	187
Как программное обеспечение измеряет само себя	189
Неучтенное время	193
Влияние измерителя	194
Двойной учет занятости процессора	198
Ошибка квантования	200
Время «невыполнения»	216
Код ядра Oracle без измерительных средств	219
Упражнения	222
8. Данные фиксированных представлений Oracle	224
Изъяны данных фиксированных представлений	225
Справочник по фиксированным представлениям	235
Полезные запросы к фиксированным представлениям	244
«Интерфейс ожидания» Oracle	267
Упражнения	268
9. Теория массового обслуживания для специалиста по Oracle	270
Модели производительности	271
Массовое обслуживание	272
Теория массового обслуживания	276

Модель массового обслуживания М/М/м	295
Резюме	335
Упражнения	337
III. Реализация	341
10. Работа с профилем ресурсов	343
Как работать с профилем ресурсов	344
Как предсказать результат	360
Как узнать, что работа завершена	362
11. Лечение согласно диагнозу	365
За пределами профиля ресурсов	366
Компоненты времени отклика	367
Исключение ненужной работы	377
Признаки масштабируемости приложения	387
12. Учебные примеры	389
Пример 1: обманчивые общесистемные данные	390
Пример 2: большие затраты процессорного времени	396
Пример 3: длительные события SQL*Net.	401
Пример 4: длительные события чтения	409
Заключение	415
IV. Приложения	417
A. Глоссарий	419
B. Греческий алфавит	434
C. Оптимизация коэффициента попаданий в кэш буферов базы данных	436
D. Формулы теории массового обслуживания М/М/м	443
E. Ссылки	445
Алфавитный указатель	453

Вступительное слово редактора

Редактору редко выпадает удача работать с такой книгой. Едва увидев заявку Кэри Миллсапа и Джеффа Хольта на книгу «Optimizing Oracle Response Time» (Оптимизация времени отклика Oracle) – таким было ее рабочее название, я сразу понял, как мне повезло. Эта книга воплощает в себе все мечты редактора: талантливые авторы, серьезное исследование и огромный материал.

Я хорошо помню первое знакомство с настройкой производительности Oracle. Это было еще в дни Oracle7, когда я знакомился с основами Oracle. Мне как раз поручили работу администратора всех баз данных, используемых моей группой разработки, и я подумал, что нет ничего лучше для начала карьеры администратора БД Oracle, чем серьезные работы по настройке производительности. Я купил и прочитал учебник по Oracle. Узнал о кэше буферов, разделяемом пуле и коэффициентах попаданий.

Казалось, это очень просто. Предыдущий администратор вообще не занимался настройкой, поэтому мне достаточно было уточнить значения нескольких параметров, использовать коэффициенты попаданий для определения оптимального размера памяти под кэш буферов и разделяемый пул, после чего можно было купаться в славе хорошо сделанной работы и упиваться похвалами моих приятелей-разработчиков, которые должны были, без сомнения, испытать благоговение перед той скоростью, с которой я собирался заставить работать их программы.

Однако все получилось не так, как я себе представлял. Я удвоил размер кэша буферов, но никакого ускорения не заметил. Я сделал размер кэша буферов равным половине исходного, и ничто не стало работать медленнее. Я увеличил разделяемый пул. Потом уменьшил его. Я метался от одного параметра к другому, но база данных упрямо продолжала работать все с той же скоростью. Было очевидно, что настройка оказалась более сложным делом, чем мне казалось, а я просто недостаточно умен для того, чтобы заниматься этим. Униженный, я оставил свои мечты стать суперпрофессионалом в настройке производительности.

К счастью для моего эго и, вероятно, для моей карьеры, я со временем открыл для себя трассировку SQL и узнал, как генерировать файлы трассировки и использовать *tkprof* для вывода информации об их содержимом. Я решил: ладно, я не специалист по настройке баз данных в целом, но я неплохо умею применять трассировку SQL для выявления и корректировки плохо работающих команд SQL.

В конце концов я понял, что настоящее значение имеет то, на что жалуются пользователи. Пользователей не волнуют ни коэффициенты попаданий, ни скорость обмена с диском; им не важно, возникает ли конкуренция за защелки; не интересует их и загадочная статистика, которая может беспокоить вас или системного администратора. Пользователям важно только одно: *насколько быстро выполняются их задания*. Теперь я практически не беспокоился ни о чем, кроме того, что волновало моих клиентов. Когда клиент жаловался на низкую производительность, я думал не о каких-то коэффициентах или статистиках, а о том, сколько времени я могу для него сэкономить. Однако меня все еще часто мучили мысли о том, что я что-то упускаю в своей настройке, что существует какой-то уровень, которого я не в состоянии достичь. Я был убежден в том, что настоящие администраторы баз данных отслеживают такие статистики, как коэффициенты попаданий, чтобы поддерживать общую эффективность их баз данных. Почему я не мог делать то же самое?

Заявка на книгу Кэри и Джеффа, мои беседы с ними, их курс, который я прошел, освободили меня от остатков чувства вины и стыда по поводу отсутствия у меня успехов по настройке экземпляра базы данных с использованием коэффициентов попаданий и других статистик уровня экземпляра. Я научился концентрироваться на времени отклика; Кэри и Джефф подтверждают правильность этого. Я расстраивался, более того, моя самооценка снизилась, т. к. у меня не получалось управлять производительностью путем отслеживания коэффициентов попаданий и других статистик уровня экземпляра; Кэри и Джефф показывают, что коэффициенты попаданий можно выбросить вон. Моим главным достижением в настройке было применение статистик файлов трассировки (трассировки SQL и *thprof*) конкретного задания для определения команд SQL, потреблявших больше всего времени; Кэри и Джефф поднимают работу со статистиками файлов трассировки на совершенно новый уровень.

Теперь вы понимаете, почему впечатление, произведенное на меня этой книгой, было (и продолжает оставаться) таким сильным. Все, что Кэри и Джефф рассказали о своем методе, резонирует с моим собственным опытом. Во время наших бесед я ловил себя на том, что как только они поднимали новый вопрос, я тут же кивал в ответ: «Да, да, конечно, это именно так!». Рассказывая мне в первый раз, как они представляют себе эту книгу, они «проповедовали перед церковным хором»¹, хотя, скорее всего, и не подозревали об этом тогда. Поразительно было то, что Кэри и Джефф ясно видели там, где я блуждал впотьмах. Несомненно,

¹ Идиома «*preach to the choir*» (англ.) употребляется в значении «делать что-то ненужное, рассказывать о том, что и так всем известно», т. е. ломиться в открытую дверь. Имеется в виду, что в церкви проповедник стоит непосредственно перед хором, участники которого отлично знают его проповеди. — *Примеч. перев.*

мненно, существовал более высокий уровень, чем тот, на котором я находился, но я *мог* его достичь, и вы тоже сможете – с помощью Кэри и Джеффа.

Обсудив с Кэри и Джеффом их планы относительно книги, я сразу понял, что она должна быть опубликована издательством O'Reilly. Мне понравилось редактировать их работу. Перечитывая главу за главой снова и снова, я узнал много нового и уверен, так будет и с вами. Настоятельно рекомендую вам эту книгу. Я рад, что вы купили ее, вы не пожалеете вложенных денег. Если вы читаете эти строки, стоя у книжного прилавка, пожалуйста, не ставьте книгу обратно на полку. Купите ее – вложите средства в собственное развитие. Не сомневайтесь, это окупится сторицей.

– *Джонатан Генник (Jonathan Gennick)*

Предисловие

Можно сказать, что в основном задача оптимизации времени отклика сервера Oracle уже решена. Надеюсь, что я хорошо потрудился и что после прочтения этой книги эта задача будет решена и для вас.

Однако если вы похожи на большинство людей, то, вероятно, пока думаете иначе. Для большинства повышение производительности Oracle — это долгая и тщетная борьба с невидимым врагом, которого невозможно обнаружить вне зависимости от того, сколько времени и дополнительного компьютерного оборудования вы готовы ради этого задействовать. Основная трудность в том, что нет цельного последовательного курса обучения повышению производительности. Цель моей книги — показать, почему так случилось, и рассказать, чем же следует заняться вместо борьбы с невидимым врагом.

Долгое время в сообществе пользователей Oracle процветали неудачные методы настройки. Более десяти лет Oracle-сообщество лихорадило от проблем производительности, при этом практически не было удачных обучающих курсов для аналитиков. Сложившаяся конъюнктура рынка была чрезвычайно выгодна для аналитиков, занимающихся вопросами производительности. В 90-х годах двадцатого века в различных уголках земного шара консультант мог называть любые суммы и брать деньги за каждый час, потраченный на улучшение производительности. Создаваемые в таких условиях методы настройки производительности обеспечивали скорее максимальные доходы консультантов, чем усовершенствование системы заказчика.

Зачем написана эта книга

Я начал работать с СУБД Oracle в 1989 г., поступив на работу в корпорацию Oracle. К 1992 г. я уже чувствовал себя достаточно компетентным специалистом в области производительности. Для оптимизации производительности я применял тот метод, который и сегодня изучает множество пользователей: исправлял десяток известных мне ошибок и молился, чтобы причиной проблем с производительностью была какая-то их комбинация. В конце 1992 г. мне поручили руководить национальной группой. Вскоре после этого назначения на руководящую должность мои полученные на практике технические навыки начали

улетучиваться. Еще через год мне казалось, что я потратил большую часть карьеры не на продукты Oracle, а на Excel и PowerPoint.

В 1995 г. я предложил создать внутри корпорации Oracle группу производительности систем System Performance Group, или SPG. Эта группа стала одной из самых больших и сильных команд профессионалов в области производительности Oracle в мире. К 1996 г. я понял, как сильно заблуждался в 1992, считая себя достаточно компетентным. На это, в частности, повлияли полученные отчеты о работе некоторых из членов моей команды, свидетельствовавшие об абсолютно фантастических прорывах в обеспечении производительности систем.

Эти аналитики практически не теряли времени попусту в своих проектах повышения производительности. Они точно прогнозировали воздействие конкретных рекомендаций по улучшению производительности на время отклика приложений. К концу первого дня работы у заказчика такой аналитик решал больше задач и с большим эффектом, чем я бы это сделал за целую неделю в 1992. Эти люди как будто применяли для хирургических операций над производительностью компьютерные томографы и лазерные скальпели там, где я мог ранее применять лишь кровопускание и костную пилу.

Технологию, которую применяли эти аналитики, неформально называли «интерфейсом ожидания». По моим тогдашним представлениям, этот интерфейс представлял собой совокупность набора V\$-таблиц и некоторых новых данных трассировки, информировавших аналитика о том, чем занимается ядро Oracle во время ожидания пользователем ответа. Сотрудники корпорации Oracle впервые познакомились с возможностями этого нового инструмента благодаря предназначенной для внутреннего использования статье Аньо Колка (Anjo Kolk) «Description of Oracle7 Wait Events and Enqueues» (Описание событий ожидания и блокировок с очередью для Oracle7), которая была выпущена в середине 90-х годов. Однако, как и в случае с другими новыми технологиями, выдающихся успехов удалось достичь лишь немногим специалистам-практикам, обладавшим редким талантом.

Проблема заключалась в воспроизводимости. В работе самых талантливых моих коллег чувствовалось, что метод оптимизации производительности должен поддаваться повторению, но воспроизвести впечатляющие результаты нескольких моих топ-консультантов могли не более 10% из 85 аналитиков группы. Дело в том, что этот метод требовал больше опыта и интуиции, чем можно было ожидать от большинства специалистов.

В октябре 1999 г. я подал в отставку с поста вице-президента группы корпорации Oracle.¹ Немного отдохнув, мы вместе с Гарри Гудманом (Gary Goodman) и Джеффом Хольтом (Jeff Holt) занялись созданием

¹ Имеется в виду System Performance Group. – *Примеч. науч. ред.*

компании, которая теперь известна многим тысячам специалистов по производительности как *hotsos.com*. С 1999 г. моя профессиональная жизнь посвящена достижению одной цели:

Созданию метода оптимизации производительности, который бы *работал* и которому можно было бы *эффективно обучить* любого администратора базы данных Oracle.

В течение последующих трех лет мы посвятили более шести полных человеко-лет исследованиям для получения и проверки тех результатов, которые будут предложены вам в этой книге. Одновременно мы обучали до 250 слушателей ежегодно по программе под названием «Hotsos Clinic». Цель этого курса обучения была такая же, как у книги, – передать владение новым надежным методом, который произвел бы настоящую революцию в работе специалиста, занимающегося оптимизацией производительности. Вернувшись на работу после нашего обучения, слушатели в первый же день, применяя приемы, описанные в этой книге, сводили время отклика важнейших бизнес-операций от часов к секундам.

В настоящее время «интерфейс ожидания Oracle» находится в центре всеобщего внимания в среде администраторов баз данных. Хотя прошло уже около 10 лет после его появления в версии Oracle 7.0.12, сообщения об интерфейсе ожидания и сегодня делаются сотнями практикующих специалистов по производительности, рассказывающих о настройке по времени ожидания на конференциях или на открытых форумах, таких как Oracle-L (http://www.cybcon.com/~jkstill/util/util_master.html).

Однако на момент написания этой книги возможности расширенной трассировки Oracle SQL все еще очень сильно недооценивались и мало применялись в общей практике, что объясняется несколькими причинами:

- Несмотря на то, что отладочное событие псевдоошибки 10046 известно уже долгое время, корпорация Oracle формально не поддерживала возможность применения *расширенной* (т. е. LEVEL > 1) трассировки SQL до момента выпуска процедуры DBMS_SUPPORT.START_TRACE_IN_SESSION.
- В собственной документации корпорации Oracle и в большинстве книг, имеющих в продаже, расширенной трассировке SQL уделялось самое незначительное внимание.
- Расширенную трассировку SQL сопровождают множество заблуждений, несправедливо подрывающих доверие аналитиков к этому методу. Даже ко времени написания этой книги большинство аналитиков еще не осознали, что файлы трассировки действительно содержат информацию о том, сколько времени сеанс Oracle тратит на подкачку страниц, свопинг или ожидание процессора.

- Не были доступны инструменты, способные помочь в сборе и выделении диагностических данных, относящихся к конкретному периоду времени или конкретному программному модулю.¹
- Было очень мало средств, способных помочь в правильной интерпретации должным образом собранных трассировочных данных. Появившаяся в Oracle 6 утилита *tkprof* вполне адекватно работала в контексте поэлементного тестирования. Однако ее уделом модернизации в девятой версии стала унылая работа по учету общего времени отклика для всего сеанса. Она способна помочь лишь в диагностике событий, происходящих *между* вызовами базы данных. А вот определять рекурсивные связи между действиями курсоров *tkprof* совсем не способна.

Расширенная трассировка SQL стала основным диагностическим инструментом ядра Oracle для нашей команды из *hotsos.com*. Это стало возможным благодаря тому, что начиная с 1999 г. мы всесторонне исследовали поведение более чем тысячи подлинных файлов трассировки SQL для реально существующих прикладных систем, работающих на разнообразных платформах по всему миру.

Мы решили восполнить нехватку программных средств и обучающих курсов. В рамках курсов Hotsos Clinic мы скрупулезно проверили наш метод на нескольких сотнях обучавшихся анализу производительности. Предложенный нами бесплатный инструмент под названием Sparky стал первым в мире средством, которое помогает выделять данные трассировки SQL, относящиеся к конкретному периоду времени или конкретному программному модулю. Благодаря нашему программному средству Hotsos Profiler мы помогли нашим клиентам решить сотни сложных и реальных задач производительности, при этом на один анализ было затрачено в среднем не более одного часа (подробную информацию о занятиях Hotsos Clinic и программных средствах Hotsos можно найти по адресу <http://www.hotsos.com>).

Лежащая перед вами книга представляет собой результат наших стараний на всех трех фронтах. Ее цель в том, чтобы ликвидировать те препятствия, которые не давали миру возможности в полной мере использовать чрезвычайно «новый» инструментарий настройки производительности Oracle.

¹ Словосочетание «properly scoped data», неоднократно встречающееся в книге, с трудом поддается столь же лаконичному переводу. Очевидно, что автор подразумевает ограничение области видимости, контекста, т. е. выделение из всего массива собранной информации только данных, относящихся к конкретному периоду времени или конкретной пользовательской операции. — *Примеч. науч. ред.*

Для кого написана эта книга

Решение задачи повышения производительности может оказаться весьма сложным и потребовать участия сотрудников различных подразделений компании (пользователей, системных инженеров, администраторов базы данных, сетевых инженеров, разработчиков приложений и т. д.), а вероятно, и производителей применяемого программного и аппаратного обеспечения. Моя книга предназначена для человека, работающего *аналитиком по производительности*. Такой специалист (или, возможно, небольшая группа специалистов) отвечает за следующие аспекты проектов по повышению производительности:

Определение цели

Аналитик должен корректно определить план работы с тем, чтобы решать именно ту задачу, которую требуется решить.

Анализ

Аналитик должен гарантировать, что поставленная цель по повышению производительности будет достигнута с наименьшими экономическими затратами.

Реализация

Аналитик отвечает за то, что результатом выполненных действий будет ощутимый прогресс для реально действующей системы.

В связи с тем, что книга представляет новый метод повышения производительности, более радикальный, чем то, к чему вы, может быть, привыкли, я попытался мотивировать изменения в фундаментальном подходе к решению задачи для спонсоров и руководителей проектов. Первая часть книги предназначена в основном для тех из них, кто пока еще не понимает необходимости изменения методов улучшения эффективности Oracle.

Структура книги

Книга состоит из двенадцати глав и пяти приложений, разбитых на четыре части.

Часть I «Методика» посвящена *определению цели*. Она написана неформальным разговорным языком, благодаря чему спонсоры и руководители, занимающиеся проектами по повышению производительности, смогут прочесть ее от начала и до конца, не увязнув в технических деталях. В нее входят следующие главы:

- Глава 1 «Лучший способ оптимизации» объясняет, почему так сложно добиться повышения производительности Oracle, применяя традиционные методы. Представлены три важных достижения в других областях, которые десятилетиями игнорировались аналитиками по производительности. Описан новый метод повышения производительности, которому и посвящена оставшаяся часть книги.

- Глава 2 «Выбор пользовательских операций» показывает, что множество проектов повышения производительности с самого начала страдают из-за неудачных спецификаций. Объясняется, как создать надежную спецификацию для проекта.
- Глава 3 «Выбор диагностических данных» рассказывает о том, что ошибки при сборе диагностических данных становятся главной причиной неудач множества проектов повышения производительности. Объясняется, почему многие проекты просто не могут быть успешными в отсутствие корректно собранных данных. Представлены три различных источника таких данных в системах Oracle.
- Глава 4 «Выбор пути решения задачи» объясняет, как можно реализовывать проекты повышения производительности, основываясь на том же принципе *информированного согласия*, который практикуется в других областях. Описано, как можно спрогнозировать затраты на проект повышения производительности и выгоду от него и как найти экономически оптимальное решение среди множества доступных путей повышений производительности.

Часть II «Справочная информация» посвящена *подробностям*. Она написана в сугубо техническом стиле с целью предоставить справочную информацию, необходимую аналитику по производительности для реализации предложенной методики. Включает в себя следующие главы:

- Глава 5 «Интерпретация данных расширенной трассировки SQL» рассматривает содержимое файла расширенной трассировки SQL. Описаны значения полей файла трассировки и отношения временных статистик.
- Глава 6 «Сбор данных расширенной трассировки SQL» рассказывает, как собрать корректные данные расширенной трассировки SQL, необходимые для анализа проблемы производительности.
- Глава 7 «Измерение времени ядром Oracle» показывает, как программное обеспечение (например, ядро Oracle) само производит над собой измерения и как проверить такую самодиагностику в конкретной системе. Указывается ряд источников неучтенного времени в файлах трассировки Oracle и поясняется, почему такие промежутки во временных характеристиках сами в себе содержат диагностические данные производительности.
- Глава 8 «Данные фиксированных представлений Oracle» описывает некоторые из множества недостатков динамических представлений производительности Oracle. Приводятся описания нескольких наиболее популярных фиксированных представлений `V$` и примеры их применения. Вы можете удивиться, обнаружив, что не все из того, что вы думаете о динамических представлениях производительности Oracle, соответствует действительности.
- Глава 9 «Теория массового обслуживания для специалиста по Oracle» – это одна из моих любимых глав. Поясняется физический смысл очередей и рассказывается о том, как применить математику

ческие знания из области *теории массового обслуживания* для оценки и даже предсказания производительности систем, в том числе приложений, работающих с БД Oracle.

Часть III «Реализация», как и часть I, написана простым разговорным языком, что должно вдохновить спонсоров и руководителей на ее прочтение. Составляющие ее главы рассказывают, как сделать, чтобы выполненная работа имела максимальный положительный эффект.

- Глава 10 «Работа с профилем ресурсов» описывает пошаговый метод анализа данных времени отклика Oracle, обеспечивающий максимальное увеличение производительности при минимальных затратах. Описывается огромный экономический эффект от «уборки мусора» и объясняется, как находить нетривиальные решения, обеспечивающие такие результаты, которых ни в каком ином случае достичь бы не удалось. Наконец, объясняется, как понять, завершена ли работа по оптимизации. На этот вопрос чрезвычайно сложно ответить в случае применения традиционных методов повышения производительности.
- Глава 11 «Лечение согласно диагнозу» описывает, как улучшить производительность приложения в разнообразных ситуациях, описываемых диагностическими данными. Особое внимание уделяется способам избавления системы от неэкономных операций. Описаны важные составляющие времени отклика, недостаточно представленные в других работах или же вообще там не упомянутые.
- Глава 12 «Учебные примеры» завершает книгу. Она содержит четыре проекта от начала и до конца – от постановки задачи и определения цели и анализа до реализации, благодаря чему можно посмотреть, как именно работает метод в реальной жизни.

Часть IV «Приложения» включает в себя следующую информацию:

- Приложение А «Глоссарий» содержит определения технических терминов, употребляемых в книге.
- Приложение В «Греческий алфавит» представляет собой таблицу греческих букв и соответствующих английских и русских эквивалентов и призвано помочь в чтении главы 9.
- Приложение С «Оптимизация коэффициента попаданий в кэш буферов базы данных», на которое меня вдохновил Коннор МакДональд своим сайтом <http://www.oracledba.co.uk>, предлагает лучшее из известных мне доказательств того, что высокий коэффициент попаданий в кэш буферов базы данных не гарантирует успешность системы. Приведенная в данном приложении программа на Perl может довести коэффициент попаданий в кэш до любого значения!
- Приложение D «Формулы теории массового обслуживания $M/M/m$ » содержит формулы из главы 9.
- Приложение E «Ссылки» содержит библиографические данные для тех источников, ссылки на которые встречаются в этой книге.

Платформа и версия

Примеры, приведенные в книге, относятся к версиям 8 и 9 ядра Oracle для операционных систем Linux, Sun Solaris, IBM AIX, HP-UX, OSF-1, VMS, MVS и Microsoft Windows. Большая часть представленных возможностей не зависит от выбора операционной системы и будет одинаково хорошо работать в версиях Oracle с 7.0.12 по 9.2.0.

К моменту написания книги корпорация Oracle еще не выпустила 10 версию Oracle. У меня не было никакой предварительной официальной информации о ядре версии 10, но некоторые подозрения о предполагаемых новшествах, конечно же, были. Там, где это было возможно, я указал, в каких областях грядущие изменения версии 10 могут изменить представление об Oracle, которое вы получите, прочитав данную книгу.

Некоторые главы гарантированно защищены от изменений. Вся первая часть книги и большая часть третьей части не изменятся и после выхода на рынок Oracle 10. Это может показаться удивительным, но и большая часть информации из части II не зависит от версии. Основные идеи глав 5, 7, 8 и 9 не изменятся с выходом версии 10. Например, несмотря на то, что в главе 7 приводятся примеры работы для ядер Oracle 7, 8 и 9, там рассказано и о том, как понять, будет ли вести себя иначе версия 10. В главе 8 подробно рассказано о представлениях *V\$* для Oracle9i, и эти представления изменятся в версии 10, но фундаментальные проблемы *опросов (polling)* и *резюмирования (summarization)* останутся не менее важными и в версии 10.

Некоторые из упомянутых в книге возможностей доступны не в каждой из версий Oracle с 7 по 9 (например, `TRACEFILE_IDENTIFIER`). Я не ставил перед собой цель указывать ту версию ядра Oracle, в которой впервые появляется новая возможность. Однако когда речь идет о тех функциях, которые могут быть недоступны в конкретной версии, я обычно предлагаю два или более способов решения задачи. Так что если окружение не позволяет применить элегантный способ решения задачи, скорее всего, в книге можно найти какое-то другое решение.

Чего ждать от этой книги

Она отличается от любых других имеющихся на рынке книг по производительности Oracle. В ней нет перечня советов и приемов. Ее цель состоит в том, чтобы избавить вас от головной боли по поводу производительности так быстро и эффективно, как вы и не надеялись. Мне кажется, что такая книга, меняющая все ваше представление о производительности, была необходима.

Книга описывает определенный метод оптимизации производительности системы Oracle, но не останавливается на этом. Предписанный метод оптимизирует производительность самого *процесса повышения производительности*. Цель книги не в том, чтобы заработала быстрее

какая-то одна система, а в том, чтобы *вы* могли *быстрее* и *эффективнее* оптимизировать работу *любой* системы.

Книга уделяет более пристальное внимание *диагностике* проблемы производительности, а не ее *решению*. Мой опыт говорит о том, что обычно сложнее всего именно поставить диагноз. Множество специалистов может предложить разумное решение на основании плохо собранных диагностических данных для проектов с неудачной спецификацией. Если задача поставлена корректно, то решить ее обычно бывает несложно. Но если решать не ту задачу, то шансы, что в результате будет решена и та, которую следовало решить, ничтожны. В книге приведено множество примеров работы не над теми задачами и объяснено, как никогда не повторить эту ошибку.

Это не книга об управлении системой или планировании загрузки мощностей, хотя практически вся предложенная информация имеет отношение к работе системного инженера и специалиста, занимающегося планированием мощностей. Это и не сборник всех «событий ожидания» Oracle. В книге описаны те события, которые наиболее часто встречались в сотнях проанализированных нами файлов трассировки, но их подробные описания следует искать в других источниках (например, в Интернете). Для меня в качестве основного поставщика сведений о событиях ожидания выступает *google.com*. Авторы не устают добавлять новые сведения о событиях ожидания в копилку Интернета.

Наконец, хочу сказать, что эта книга адресована *практикующим* специалистам по оптимизации. Это не оторванные от жизни теории. Вся включенная в нее информация необходима моим слушателям, коллегам и мне самому для выполнения работы по повышению производительности Oracle.

Об инструментах, примерах и упражнениях

Я выбрал Perl в качестве основного языка программирования для примеров. Может показаться странным, что в книге по Oracle не используется для этой цели SQL или PL/SQL, но вы поймете меня по мере чтения. Дело в том, что значительная часть книги посвящена не Oracle, а *производительности*, а возможности языков SQL и PL/SQL далеки от того, чтобы элегантно обрабатывать все вопросы, связанные с производительностью. Perl позволяет мне иллюстрировать приближенные к жизни эксперименты, используя бесплатный переносимый инструмент, который легко установить. Надеюсь, этот выбор максимально увеличит вероятность того, что вы действительно попытаетесь выполнить некоторые из предложенных мною заданий самостоятельно.

Для иллюстрирования материала о формировании очередей выбран Microsoft Visual Basic, поскольку сегодня Microsoft Excel де-факто является основным инструментом для выполнения упражнений. Опять-таки, я надеюсь, что этот выбор максимально увеличит вероятность того, что вы действительно используете предоставленные мною материалы.

Все фрагменты кода можно скачать из Интернета по адресу:

<http://www.oreilly.com/catalog/optoraclep/>

В книге читателю предложен ряд упражнений. Это необычно как для книги издательства O'Reilly, так и для книги по Oracle. Я решил включить упражнения для того, чтобы:

1. Вдохновить вас попробовать самостоятельно сделать то, о чем вы только что прочитали.
2. Стимулировать применение тех приемов, о которых вы только что прочитали, но с другими входными данными.
3. Стимулировать применение тех приемов, о которых вы только что прочитали, в конкретной системе.
4. Признать, что мне известны ответы не на все интересные вопросы, и хотя бы обозначить эти вопросы. Благодаря этому читатели узнают о существовании проблемы и смогут работать над ее решением только в том случае, если экономический эффект оправдает это.
5. Для того чтобы книгу можно было применять как учебник на официальных курсах обучения. Я очень успешно использовал черновые версии этой книги в качестве материала для курсов Hotsos Clinic 101 (*<http://www.hotsos.com>*), начиная с 2002 г.

Большинство упражнений имеет несколько правильных ответов. Для таких упражнений обновленные решения появляются по адресу *<http://www.oreilly.com/catalog/optoraclep>*.

Цитаты

В тексте книги приводятся ссылки на внешние ресурсы. Например, ссылка [Bach (1986) 148] означает, что речь идет о стр. 148 в документе, занесенном в приложение Е как [Bach (1986)]. В данном случае документ представляет собой книгу, написанную Морисом Бахом (Maurice Bach) и опубликованную в 1986 г.

Принятые обозначения

В этой книге мы будем придерживаться следующих типографских соглашений:

Курсив

Применяется для имен файлов, каталогов и адресов URL. Кроме того, курсивом выделяются технические термины при их первом появлении в тексте.

Моноширинный шрифт

Встречается в примерах, служит для выделения имен событий, а также для отображения содержимого файлов и выводимых данных.

Моноширинный полужирный шрифт

В примерах обозначает вводимую пользователем информацию. Также применяется для выделения участков текста, на которые следует обратить внимание.



Это совет, предложение или примечание.



Это предупреждение или предостережение. Например, о том, что некоторое сочетание параметров может оказать негативное влияние на систему.

Комментарии и вопросы

Мы проверили информацию, содержащуюся в этой книге, настолько хорошо, насколько это было возможно, однако некоторые свойства могли измениться, а мы могли допустить ошибки. Сообщите нам об этом по адресу:

O'Reilly & Associates, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (США или Канада)
(707) 829-0515 (международные/местные)
(707) 829-0104 (факс)

Для этой книги создан сайт, где можно найти примеры программ, список опечаток и другую дополнительную информацию; его адрес:

<http://www.oreilly.com/catalog/optoraclep/>

Чтобы задать вопросы или дать комментарии к книге, пишите по адресу:

bookquestions@oreilly.com

Сведения о книгах и проводимых конференциях можно найти в разделах «Resource Centers» и «O'Reilly Network» на сайте издательства O'Reilly:

<http://www.oreilly.com>

Также можно заглянуть на сайт авторов:

<http://hotsos.com>

Благодарности

Минди, Александр и Николас Миллсан

Моя жена и мои сыновья всегда вдохновляют меня, и я хотел выразить им свою признательность за те жертвы, на которые они пошли,

чтобы позволить мне осуществить этот проект. Я работал над книгой в те часы, которые мог бы провести с ними, и теперь надеюсь, что результаты этого проекта хотя бы в какой-то мере компенсируют им то время, которые они позволили мне на него потратить. Спасибо вам, Минди, Алекс и Ник.

Ван и Ширли Миллсан

В начале каждого учебного года мои родители отводили меня в школу и знакомились с учителями. По дороге они всегда говорили мне одно и то же:

На каждый вопрос, задаваемый учителем, существует два ответа. Один из них правильный, а второй – это тот, которого ждет учитель. Надеемся, что ты будешь знать оба.

Спасибо вам, мама и папа. Вы даже не представляете себе, как помогли мне в жизни.

Джефф Хольт

За эти три года сайт *hotsos.com* получил несколько благословений, но самым значительным событием было согласие на участие в проекте некоего мистера Джеффа Л. Хольта. Джефф был одним из ведущих аналитиков той группы из 85 человек, которую я оставил в корпорации Oracle. Сейчас это ведущий научный сотрудник *hotsos.com*. В последние три года основным занятием Джеффа было обучение *меня* оптимизации системы Oracle. Моя же функция в значительной степени заключалась в том, чтобы вывести из строя интуицию Джеффа.

Джефф относится к тем талантливым людям, которые понимают, как решить задачу, задолго до того, как могут объяснить, как они это сделали. Я же – маниакальный педант, который тратит больше времени на то, чтобы понять, *почему* ответ правильный, чем Джефф тратит собственно на его получение. Я считаю, что если метод основывается только на интуиции и опыте пользователя, то его невозможно воспроизвести и ему невозможно обучить. Я уверен, что высококачественный метод оптимизации производительности Oracle можно создать, только исключив из процесса улучшения производительности интуицию, опыт и удачу.

Теперь вы понимаете, с чем Джеффу приходилось иметь дело. Его пониманию и терпению не было пределов – все то время, пока я копался во внутренностях его мозга, вынимая их и укладывая обратно множество раз. Спасибо тебе, Джефф.

Гарри Гудман

Гарри Гудман – это мой друг и соучредитель *hotsos.com*. Если бы не было долгой прогулки летом 1999 г., то невозможно предсказать, чем бы я занимался сегодня. А если бы не та долгая прогулка летом 1989 г., возможно, я никогда бы не работал в корпорации Oracle. Если бы не та работа, которой Гарри занимается каждый день, не

было бы ни *hotsos.com*, ни книги, которую вы держите в руках. Спасибо тебе, Гарри.

Могенс Норгаард

Могенс Норгаард – это мой друг из Дании, обладатель многих наград, первым познакомивший меня с тогда еще загадочным «интерфейсом ожиданий Oracle». Могенс был первым человеком в мире, который потребовал от всех своих технических специалистов в корпорации Oracle использовать интерфейс ожидания и только интерфейс ожидания для диагностики проблем производительности Oracle. Могенс также является основателем знаменитой сети Oak Table (<http://www.oaktable.net>), объединяющей людей, чьи умы считаются наиболее влиятельными в мире производительности Oracle. Если бы не дружба и поддержка Могенса, не его организаторские способности, эта книга, возможно, никогда бы не появилась на свет. Спасибо тебе, Могенс.

Аньо Колк

Аньо Колк – это отец методов оптимизации времени отклика Oracle. С тех пор, когда я впервые встретил Аньо где-то в начале 1990-х, он при всей своей занятости ни разу не отказался уделить время для того, чтобы научить мои группы и меня самого тому, как на самом деле все работает. Спасибо тебе, Аньо.

Вираг Саксена

Вираг Саксена был первым консультантом моей Группы производительности систем в корпорации Oracle и тем лучом, который на мгновение показал мне, каким мог бы быть мир проектов по улучшению производительности. Можно сказать, что именно дарование Вирага было той искрой, которая зажгла костер этой книги. Спасибо тебе, Вираг.

Джонатан Льюис, Коннор МакДоналд и Фрэнк Хансен

Есть множество причин, по которым я хотел бы поблагодарить вас, в том числе за полученный отзыв, который помог улучшить эту книгу. Спасибо, джентльмены.

Джонатан Генник и сотрудники O'Reilly & Associates

Рик Гринвальд говорил мне, что в мире существует три вида книгоиздателей: те, у которых публикуемая книга оказывается хуже авторского экземпляра; те, кто публикуют книгу, которая столь же хороша, как и рукопись; и те, у которых публикуемый вариант оказывается лучше, чем рукопись. Под руководством Джонатана эта книга стала лучше, чем тот вариант, который я принес в O'Reilly.

Хотелось бы выразить искреннюю признательность клиентам *hotsos.com*, которые обеспечивали средства для существования моей семьи и стимулировали мой разум. Именно благодаря вашей поддержке я смог собрать материал для книги. Наконец, большое спасибо всем тем, кто многому меня научил, а именно:

Стив Адамс (Steve Adams)	Элен Дудар (Ellen Dudar)	Торбен Хольм (Torben Holm)
Мика Адлер (Micah Adler)	Нэнси Душкин (Nancy Dushkin)	Марк Хостман (Mark Horstman)
Филипп Олмес (Philip Almes)	Джулиан Дайк (Julian Dyke)	Мемдух Ибрагим (Mamdouh Ibrahim)
Энди Бэйли (Andy Bailey)	Мортен Иган (Morten Egan)	Джонатан Интнер (Jonathan Intner)
Карла Бэйси (Karla Baisey)	Джин Эмерсон (Jean Emerson)	Линн Изабелла (Lynn Isabella)
Владимир Баррьер (Vladimir Barriere)	Бьерн Энгсис (Bjørn Engsig)	Кен Якобс (Ken Jacobs)
Кен Баумгарднер (Ken Baumgardner)	Дэйв Энсор (Dave Ensor)	Нил Йенсен (Neil Jensen)
Кертис Беннетт (Curtis Bennett)	Барри Эпштейн (Barry Epstein)	Фил Джоэл (Phil Joel)
Дарен Бок (Darren Bock)	Генри Фахей (Henry Fahey)	Гумундур Джисепсон (Gumundur Jysepsson)
Кеннет Брэди (Kenneth Brady)	Марк Фарнхем (Mark Farnham)	Дэрри Кэбснелл (Derry Kabcenell)
Филипп Бригз (Phillip Briggs)	Роберт Фейнер (Robert Feighner)	Джордж Кадифа (George Kadifa)
Майкл Браун (Michael Brown)	Зак Фрис (Zach Frieze)	Майк Кол (Mike Kaul)
Тим Бунс (Tim Bunce)	Питер Грэм (Peter Gram)	Брайан Куш (Brian Kush)
Др. Бёрт Бёрнс (Dr. Burt Burns)	Дональд Гросс (Donald Gross)	Арман Садат Киаэ (Armand Sadat Kyae)
Лассе Кристенсен (Lasse Christensen)	Кайл Хэйли (Kyle Hailey)	Том Кайт (Tom Kyte)
Кэрол Колрейн (Carol Colrain)	Стефан Хэйсли (Stephan Haisley)	Рей Лэйн (Ray Lane)
Руди Корси (Rudy Corsi)	Тереза Хэйсли (Theresa Haisley)	Санг Чул Ли (Sang Chul Lee)
Кэрол Дако (Carol Dacko)	Рей Хэмлит (Ray Hamlett)	Джонатан Льюис (Jonathan Lewis)
Доминик Дельмолино (Dominic Delmolino)	Ахмер Хасан (Ahmer Hasan)	Маргарет Льюис (Margaret Lewis)
Дэвид Дэмпси (David Dempsey)	Джим Хендон (Jim Herndon)	Джим Литтлфилд (Jim Littlefield)
Кёрти Дешпанде (Kirti Deshpande)	Дэйв Херрингтон (Dave Herrington)	Хуан Лоайза (Juan Loaiza)
Йоханнес Джернос (Johannes Djernæs)	Кэрол Хипп (Carol Hipp)	Андреа Лопес (Andrea Lopez)
Грег Догерти (Greg Doherty)	Др. Майрон Глинка (Dr. Myron Hlynka)	Скотт Лавинфосс (Scott Lovingfoss)

Родерик Макалак (Roderick Macalac)	Уиллис Ренни (Willis Ranney)	Торфи Олафур Свериссон (Torfi Ólafur Sverrisson)
Лаура Маццарелла (Laura Mazzarella)	Мэтт Роу (Matt Raue)	Ирфан Сайед (Irfan Syed)
Коннор МакДональд (Connor McDonald)	Энди Ривнес (Andy Rivenes)	Тони Тейлор (Tony Taylor)
Брэд и Вонда МакФарлинг (Brad and Vonda McFarling)	Хасан Ризви (Hasan Rizvi)	Лоуренс То (Lawrence To)
Дэниэл Менасо (Daniel Menascé)	Джесс Рудер (Jesse Ruder)	Дэн Тау (Dan Tow)
Рик Минутелла (Rick Minutella)	Боб Рудзки (Bob Rudzki)	Джоаким Треугут (Joakim Treugut)
Майкл Мюллер (Michael Möller)	Сэнди Сандерсон (Sandy Sanderson)	Хенк Туллис (Hank Tullis)
Джеймс Морл (James Morle)	Мэтт Ситон (Matt Seaton)	Питер Уциг (Peter Utzig)
Крейг Ньюбургер (Craig Newburger)	Крейг Шеллахамер (Craig Shallahamer)	Гаджа Кришна Вайдьянатха (Gaja Krishna Vaidyanatha)
Марк Павкович (Mark Pavkovic)	Пит Шерман (Pete Sharman)	Терри Вергулт (Thierry Vergult)
Чарльз Петерсон (Charles Peterson)	Роберт Шоу (Robert Shaw)	Майкл Вецуипенс (Michel Vetsuypens)
Нареш Пилларицетти (Nagesh Pillarisetti)	Роджер Сименс (Roger Siemens)	Др. Анита Уокер (Dr. Anita Walker)
Джордж Полиснер (George Polisner)	Др. Джон Слокум (Dr. John Slocum)	Др. Билл Уокер (Dr. Bill Walker)
Ник Попович (Nick Popovic)	Джерри Сноу (Jerry Snow)	Майк Вьелонски (Mike Wielonski)
Лин Претт (Lyn Pratt)	Билл Стенгел (Bill Stangel)	Джеральд Вильямсон (Gerald Williamson)
Дерил Пресли (Darryl Presley)	Джаред Стил (Jared Still)	Лиз Вайзман (Liz Wiseman)
Др. Рей Куайт (Dr. Ray Quiett)	Марсела Студница (Marcela Studnicka)	Брайан Вульф (Brian Wolff) Грэхем Вуд (Graham Wood)

Джимми Хаки (Jimmy Harkey), от которого я узнал об аксиоматическом подходе к решению задач.

Рейчел Рутти (Rachel Rutti), которая познакомила меня с книгой «The Goal» Эли Голдрата (Eli Goldratt).^a

Участники сетевого проекта Oak Table.

Многие участники Oracle-L.

И Грэм... Если бы ты знал, как мне тебя не хватает...

^a Голдрат Э.М., Кокс Дж. «Цель: процесс непрерывного совершенствования». – Пер. с англ. П. А. Самсонова. – Минск, ООО «Попурри», 2004 г. – Примеч. перев.

Часть I. Методика

1

Лучший способ оптимизации

Производительность Oracle для многих представляет серьезную проблему. Начиная с 1990 года мне довелось поработать с тысячами профессионалов, занятых в проектах по повышению производительности систем, основанных на Oracle. Развитие всех таких проектов проходит через несколько стандартных стадий. Полагаю, названия этих стадий лежат в сейфовом хранилище¹ где-нибудь в Женеве. Если мне не изменяет память, названия эти таковы:

- Безудержный оптимизм
- Информированный пессимизм
- Паника
- Отрицание
- Безднадежность
- Полная безысходность
- Упадок и отчаянный дефицит

По какой-то причине меня и моих коллег редко приглашают в проект, не достигший стадии «упадка и отчаянного дефицита». Вот так обычно выглядит проект по повышению производительности к моменту нашего появления. Не встречались ли вы с подобными ситуациями?

Технические эксперты расходятся во мнениях по поводу источника проблем

Чем серьезнее трудности с производительностью, тем больше людей приходит на совещания, чтобы поговорить о ней. Если на сове-

¹ В оригинале автор употребляет слово «vault». В данном контексте, скорее всего, имеется в виду хранилище ценностей в стране банков, где все неизменно, стабильно и вечно. Так же вечны и стадии проектов по повышению производительности. – *Примеч. науч. ред.*

паниии собрались «лучшие эксперты» из нескольких фирм – это очень плохой признак. За время своей карьеры я побывал на десятках совещаний, где «лучшие эксперты» консалтинговых компаний, производителей компьютеров и систем хранения данных, поставщиков программного обеспечения, сетевых провайдеров пытались решить проблемы производительности. Ровно в 100% случаев участники совещания проводили время в бесконечных спорах о первопричинах недостаточной производительности. *Неделями*. Как могут умные, образованные, целеустремленные специалисты, глядя на одну и ту же систему, иметь столь разные (и зачастую *противоположные*) мнения о причинах снижения производительностью? Очевидно, производительность Oracle-системы – это очень сложный вопрос.

Эксперты заявляют о прекрасных достижениях, но пользователи не видят улучшений

Многие из моих студентов понимающе усмеваются, когда я рассказываю истории о консультантах, с гордостью рапортовавших об успехах в улучшении какой-либо статистики (будь то увеличение коэффициента попаданий или уменьшение количества экстенгов и т. д.) и получивших обескураживающий ответ, что пользователи и не заметили никаких изменений. Обычно результатом такой работы становится длинный отчет консультанта, в котором в максимально вежливых выражениях объясняется, что, хотя пользователи в силу своей отсталости этого и не заметили, система стала значительно лучше – в полном соответствии с приложенным счетом.

Такие истории вызывают улыбку, если, конечно, вы не владелец компании, оплачивающей потраченное впустую время, или не консультант, не получивший оплаты из-за отсутствия сколько-нибудь значимого результата. Возможно, история кажется смешной, потому что большинство из нас так или иначе *оказывались* в роли подобного консультанта. Как может получиться, что при очевидном улучшении таких важных показателей как коэффициент попаданий, среднее время задержки и время ожидания, пользователи могут даже не заметить результата наших усилий? Очевидно, производительность Oracle-системы – это очень сложный вопрос.

Модернизация оборудования либо не помогает, либо приводит к дальнейшему замедлению

Со времени первого знакомства с публикацией Нейла Гюнтера (Neil Gunther) «The Practical Performance Analyst» («Аналитик-практик производительности») в 1998 году мне неоднократно доводилось демонстрировать возможность такого противоестественного феномена. «Допускаете ли вы, что замена оборудования на более мощное может ухудшить производительность важного приложения?» Во всех группах, где я задавал этот вопрос и приводил соответствующие факты, реакция была практически одинаковой. Большинство слу-

пателей недоверчиво улыбалось, пока я рассказывал, как такое может произойти, в конце же подходили один или двое, радуясь, что наконец поняли, что же произошло несколько месяцев назад, когда после модернизации «все пошло не так».

Улучшение оборудования нечасто приводит к ухудшению производительности, но такое случается. Очень часто эффект от замены аппаратной части практически не заметен, если, конечно, не считать весьма заметного утекания денег в обмен на неощутимые преимущества. То, что модернизация может не дать никаких преимуществ, немного беспокоит. Мысль о том, что улучшение оборудования может в действительности привести к *снижению* производительности, полностью обескураживает. Как может получиться, что замена оборудования на более мощное не только не повышает производительность, но может даже повредить ей? Очевидно, производительность Oracle-системы – это очень сложный вопрос.

Значительная часть ресурсов системы расходуется впустую

Мы с коллегами пришли к выводу, что практически любая система не менее чем на 50% загружена бесполезной работой. Мы очень осторожно подходим к определению «бесполезности», понимая под ней ту часть нагрузки, которая может быть исключена без потерь для функционирования бизнеса. Как может совершенно ненужная работа отнимать большую часть ресурсов во многих грамотно управляемых системах? Очевидно, производительность Oracle-системы – это очень сложный вопрос.

Это умные люди. Почему же их проекты в таком беспорядке? Очевидно, что оптимизация Oracle-системы – это очень сложное дело. Как еще можно объяснить то, что такое количество проектов в стольких не связанных между собой компаниях сталкиваются со столь похожими трудностями?

«Вы делаете это неправильно»

Одно из моих увлечений заключается в изготовлении довольно крупных вещей из дерева. Такое хобби требует применения тяжелых инструментов, которые так и норовят перемолоть мои пальцы вместо толстого куска черного американского ореха (*Juglans Nigra*). Одно из самых интересных занятий, связанных с этим хобби, – читать о новой технике, позволяющей повысить точность, сберечь время и вместе с этим значительно уменьшить риск несчастного случая. Мне нравится чувствовать, что я делаю что-то не так, потому что в такие минуты понимаю, что сейчас узнаю нечто такое, что сделает мою жизнь заметно лучше. Подобные открытия весьма благотворно сказываются на моем эмоциональном здоровье. Конечно, я испытываю легкое разочарование от очередного свидетельства своего несовершенства, но мне очень приятно сознавать, что скоро я стану лучше.

Основываясь на таком подходе, я предлагаю вам рассмотреть следующие гипотезы:

Если вы обнаруживаете, что настройка производительности Oracle чересчур сложна, то, скорее всего, вы делаете это неправильно.

Теперь – самое страшное:

Вы делаете это неправильно, потому что вас так научили.

Я бросаю вызов. Я убежден, что большинство методик настройки, общепринятых и преподаваемых с 1980-х годов, в корне ошибочны. Я взялся за эту книгу, движимый желанием поделиться с вами результатами исследования, убедившего меня, что существует гораздо более удачный путь.

Начнем с краткого обзора «метода», который вы, может быть, сейчас применяете. Вообще метод должен предполагать определенную последовательность действий. Первое, что можно заметить в имеющейся сегодня литературе, – поразительное *отсутствие* настоящего метода. Большинство авторов уделяют больше внимания советам и техническим приемам, нежели методам. В результате получается громоздкий набор «вещей, которые могут понадобиться», но нет почти ни слова о том, *действительно ли* они могут понадобиться и *в каких случаях*. Посмотрев на результаты поиска в *google.com* по строке «Oracle performance method», вы поймете, что я имею в виду.

Большинство рекомендуемых в настоящее время методик повышения производительности Oracle может быть сведено к последовательности шагов, описанных в Методе С – *традиционном (conventional*, отсюда и название) методе проб и ошибок. Если вы испытываете трудности с оптимизацией производительности, то может оказаться, что все дело в Методе С. Что этот метод действительно оптимизирует, так это доходы специалиста по оптимизации, проводящего время в решении проблем производительности.

Метод С: метод проб и ошибок, доминирующий сегодня в культуре настройки производительности Oracle

1. Построить гипотезу о несоответствующем значении некоторой метрики x .
2. Предпринять действия с целью улучшения x . Отменить любые изменения, приведшие к заметному снижению производительности.
3. Если пользователи *не* ощутили достаточного улучшения времени отклика, перейти к шагу 1.
4. Если достаточное увеличение производительности *достигнуто*, все равно перейти к шагу 1, так как продолжение поиска может привести к дальнейшему улучшению.

Конечно, данный метод проб и ошибок – не единственный в природе способ повышения производительности. Метод YAPP, разработанный Аньо Колком (Anjo Kolk) и Шари Ямагучи (Shari Yamaguchi) в 1990-х [Kolk и другие (1999)]¹, был, вероятно, первым, который возвысился над нагромождением советов и приемов и содержал действительно пригодную к использованию определенную последовательность шагов. Этот метод совершил переворот в диагностике проблем производительности и послужил главным источником вдохновения при написании данной книги.

Требования к хорошему методу

Что отличает хороший метод от плохого? Когда мы в 1999 г. запустили проект *hotsos.com*, я потратил кучу времени на выявление неэффективности существующих методик увеличения производительности Oracle. Это оказалось интересным занятием. После долгого изучения мы с коллегами смогли составить перечень объективно измеримых критериев, позволяющих определить, что *хорошо* в методе и что в нем *плохо*. Мы надеялись, что такой список послужит некой системой координат, в которой можно будет измерить эффективность наших усилий по усовершенствованию метода. Вот этот список признаков, которые, я надеюсь, отличают хорошие методы от плохих:

Действенность

Если есть возможность повышения производительности, метод должен обеспечивать это повышение. Нельзя допускать, чтобы мероприятия по повышению производительности требовали значительных вложений, давая конечному пользователю незначительный или отрицательный результат.

Эффективность

Метод всегда должен обеспечивать повышение производительности с наименьшими возможными затратами. Метод не может считаться оптимальным, если существует другой метод, дающий удовлетворительный результат при меньших расходах за такое же или меньшее время.

Измеримость

Эффект повышения производительности должен быть выражен величинами, принятыми в *бизнесе*. Результаты измерений производительности не представляют интереса, если они выражены в технических единицах, не отражающих прироста движения денежных потоков, чистой прибыли или возврата инвестиций.

¹ Имеется в виду статья «Yet Another Performance Profiling Method» (Еще один метод профилирования производительности). – *Примеч. ред.*

Прогнозируемость

Метод должен позволять аналитику прогнозировать результат предполагаемого воздействия. Прогноз должен оперировать теми же единицами измерения, в которых будут оцениваться результаты повышения производительности с точки зрения бизнеса.

Достоверность

Метод должен верно идентифицировать первопричину проблемы независимо от источника ее возникновения.

Детерминизм

Метод должен предоставлять аналитику однозначно определенную последовательность шагов, основанную на документированных постулатах, а не на опыте или интуиции. Недопустимо, чтобы один и тот же метод привел двух специалистов к разным выводам относительно причин недостаточной производительности.

Конечность

Метод должен содержать четко определенное условие завершения, например, доказательство оптимальности.

Практичность

Метод должен быть применим в любых разумных рабочих условиях. В частности, нельзя считать удовлетворительным метод, основанный на применении инструментов, существующих не на всех вычислительных платформах.

Метод С ни в малейшей степени не соответствует ни одному из восьми перечисленных показателей качества. Я больше не стану распространяться на эту тему, просто проверьте прямо сейчас, насколько применяемый вами метод повышения производительности соответствует перечисленным критериям. Результат анализа может побудить вас к действию. Надеюсь, после прочтения первой части книги вы вернетесь к этому списку, чтобы проверить, сколько дополнительных очков вы набрали благодаря прочитанному.

Три важных достижения

Я начал предисловие так:

Задача оптимизации времени отклика в Oracle, по большей части, решена.

Это утверждение полностью противоречит мрачной картине, нарисованной мною в начале этой главы: «Производительность Oracle для многих представляет серьезную проблему». Такое противоречие требует логичного объяснения. Вот оно:

Ряд технических усовершенствований позволил повысить действенность, эффективность, измеримость, прогнозируемость, достоверность, детерминизм, конечность и практичность в деле оптимизации производительности Oracle.

В частности, я уверен, что нынешний прогресс обязан трем важным достижениям. Любопытно, что, хотя эти достижения лишь недавно вошли в практику работающих с Oracle профессионалов, ни одно из них не является действительно новым. Все они применяются специалистами по оптимизации в несвязанных с Oracle областях, и некоторым из них уже больше века.

Концентрация на пользовательских операциях

Первое важное правило оптимизации Oracle следует из элементарного правила математики:

По агрегированным данным нельзя восстановить составляющие.

Поясню это на таком примере. Предположим, я говорю вам, что среди 1000 камней 999 серых и один особенный, покрашенный в красный цвет. Все камни весят 1000 фунтов. Теперь скажите, сколько весит красный камень? Сказав, что *знаете*, что красный камень весит один фунт», вы, осознавая это или нет, не дадите правильный ответ. Неизвестно, весит ли красный камень один фунт. Основываясь на предоставленной информации, нельзя это знать. Ответив: «Я *предполагаю*, что красный камень весит один фунт», вы сделаете слишком самонадеянное предположение, рискуя придти к ошибочным, возможно, катастрофически ошибочным выводам.

Правильный ответ гласит, что красный камень может иметь практически любой вес – от 0 до 1000 фунтов. Единственное ограничение снизу определяется минимальным количеством атомов, которое еще может называться *камнем*. Определив, насколько малым может быть камень, мы получим и верхнюю границу искомого ответа. Она составит 1000 фунтов за вычетом веса 999 камней минимально возможного размера. Таким образом, вес красного камня может находиться в пределах практически от нуля до тысячи фунтов. Любой ответ с большей точностью будет *ошибочным*, если только вам не повезет. Но искусству выигрывать в подобных играх нельзя ни научиться, ни научить других, и его нельзя воспроизвести с достаточной достоверностью.

Это одна из причин, по которым специалисты по Oracle так не любят диагностировать проблемы производительности, основываясь на данных общесистемной статистики, вроде той, что предоставляет *Statspack* (или его ближайшие родственники, произошедшие от старых SQL-сценариев *bstat* и *estat*). Два аналитика, глядя на одни и те же результаты работы *Statspack*, могут сделать совершенно разные выводы, причем ни один из них полностью не подтверждается и не опровергается представленными данными. Это не ошибка *Statspack*. Это принципиальный недостаток метода, отправной точкой в котором служат данные по *системе* в целом (`V$SYSSTAT`, `V$SYSTEM_EVENT` и т. д.). На самом деле с помощью *Statspack* можно собрать данные с достаточной степенью детализации, но нигде в документации я не встретил ни малейшей попытки объяснить, зачем это может быть необходимо.

Отличной иллюстрацией к сказанному будет случай с Oracle-системой, в которой роль красного камня исполняла проблема формирования платежной ведомости. Руководство компании обрисовало проблему медленного создания платежной ведомости, создающую сложности в ведении бизнеса. Администраторы базы данных обнаружили проблему с защелками, конкретно – с защелками *цепочек буферов кэша*. Обе точки зрения были хорошо аргументированы. Бизнес действительно страдал из-за медленной обработки ведомостей. Это было очевидно: система выводила чеки с большой задержкой. Сервер тоже страдал вследствие конкуренции за защелки. И это было очевидно: запросы к `V$SYSTEM_EVENT` ясно показывали, что система тратит массу времени на события ожидания *освобождения защелки (latch free)*.

Системные администраторы и администраторы БД этой компании потратили три месяца на отчаянную борьбу с конкуренцией за защелки, но это нисколько не улучшило ситуацию с платежной ведомостью. Причина оказалась проста: ведомость не ожидала освобождения защелок. Как мы это узнали? Мы получили данные о распределении времени выполнения программы формирования ведомости. Результаты нас удивили. Да, множество других программ действительно проводили время в ожидании защелок цепочек буферов кэша. Но медленная программа формирования ведомости из 1985,40 секунд общего времени выполнения потратила на ожидание защелок лишь 23,69. Это составляет 1,2% от общего времени. Если бы компания полностью *исключила* ожидания освобождения защелок в своей системе, скорость формирования платежной ведомости возросла бы всего на 1,2%.

Почему же общесистемная статистика может так вводить в заблуждение? Да, множество других программ простаивало из-за проблем с освобождением защелок. Но было большой ошибкой предполагать, что проблема данной программы и проблема системы в целом – это одно и то же. Ложная предпосылка о наличии причинно-следственной связи между ожиданием освобождения защелок и медленным формированием ведомости стоила компании трех месяцев потраченного времени, разочарования и нескольких тысяч долларов на оплату работ и модернизацию оборудования. В противоположность этому, определение истинной причины низкой производительности заняло около десяти минут с того момента, как компания получила корректные диагностические данные.

Я и мои коллеги регулярно сталкиваемся с подобными проблемами. Для аналитика по производительности решение заключается в том, чтобы сосредоточиться на *пользовательских операциях*, требующих оптимизации. Бизнес может определить, какие операции самые важные, система – нет. Если определены процедуры, требующие оптимизации, то главной задачей становится сбор данных о выполнении *именно* этих процедур – не больше и не меньше.

Концентрация на времени отклика системы

Последние пару десятилетий специалисты по производительности Oracle исходили из предположения, что объективное измерение времени отклика Oracle невозможно [Ault and Brinson (2000), 27]. В отсутствие путей достоверного измерения времени отклика они взяли следующий по важности показатель: *счетчики событий (event counts)*. За счетчиками, естественно, последовали коэффициенты. А за коэффициентами – разного рода доводы о том, какие действия по «настройке» важны, а какие – нет.

Однако пользователей не заботят счетчики, коэффициенты и аргументы за и против; их интересует *время отклика* – продолжительность ожидания от того момента, когда они сделали запрос, до получения ответа. Неважно, насколько изощренно обрабатываются данные, основанные на счетчиках событий и не содержащие сведений о времени. Тщетность усилий обусловлена неизбежным фактом, лежащим в основе второго важного правила:

Нельзя судить о длительности события по количеству его возникновений.

Пользователей волнует только время отклика. Занимаясь счетчиками событий, вы измеряете не то, что их действительно интересует. Если вам понравилась задача про красный камень, вот вам еще одна: что мешает выполняться программе, порождающей приведенные ниже данные?

Пример 1.1. Составляющие времени отклика в порядке убывания количества вызовов

Response Time Component	# Calls
-----	-----
CPU service	18,750
SQL*Net message to client	6,094
SQL*Net message from client	6,094
db file sequential read	1,740
log file sync	681
SQL*Net more data to client	108
SQL*Net more data from client	71
db file scattered read	34
direct path read	5
free buffer waits	4
log buffer space	2
direct path write	2
log file switch completion	1
latch free	1

В примере 1.2 показаны результаты того же прогона программы, но здесь они дополнены данными о распределении времени (в секундах) и отсортированы по убыванию своего вклада в общее время ответа. Ваш ответ изменился?

Пример 1.2. Составляющие времени отклика в порядке убывания доли в общем времени ответа

Response Time Component	Duration		#Calls	Dur/Call
SQL*Net message from client	166.6s	91.7%	6,094	0.027338s
CPU service	9.7s	5.3%	18,750	0.000515s
unaccounted-for	2.2s	1.2%		
db file sequential read	1.6s	0.9%	1,740	0.000914s
log file sync	1.1s	0.6%	681	0.001645s
SQL*Net more data from client	0.3s	0.1%	71	0.003521s
SQL*Net more data to client	0.1s	0.1%	108	0.001019s
free buffer waits	0.1s	0.0%	4	0.022500s
SQL*Net message to client	0.0s	0.0%	6,094	0.000007s
db file scattered read	0.0s	0.0%	34	0.001176s
log file switch completion	0.0s	0.0%	1	0.030000s
log buffer space	0.0s	0.0%	2	0.005000s
latch free	0.0s	0.0%	1	0.010000s
direct path read	0.0s	0.0%	5	0.000000s
direct path write	0.0s	0.0%	2	0.000000s
Total	181.8s	100.0%		

Разумеется, ответ изменился – время отклика имеет решающее значение, по сравнению с ним счетчики событий малоинтересны. Для рассматриваемой программы критичной является составляющая SQL*Net message from client, а вовсе не CPU service.



Те, у кого есть опыт анализа производительности Oracle, наверняка слышали, что событие SQL*Net message from client – это событие простая и может быть проигнорировано. Такие события нельзя игнорировать, если диагностика выполняется способом, описанным в главе 3.

Если бы дело происходило в 1991 г., у нас возникли бы серьезные затруднения, т. к. в то время ядро Oracle не могло предоставить данные, показанные во второй таблице. Но если обновить Oracle хотя бы до версии 7, то счетчики событий перестанут быть «следующими по важности» после времени отклика показателями. Начиная с Oracle 7.0.12 основополагающий тезис, утверждающий, что невозможно судить о времени выполнения операций ядром Oracle, больше не действует.

Закон Амдала

Последнее из упоминавшихся мною «великих достижений» в оптимизации производительности Oracle основано на опубликованном в 1967 г. Джинном Амдалом (Gene Amdahl) наблюдении, названном впоследствии законом Амдала [Amdahl (1967)]:

Увеличение производительности, достигаемое некоторым усовершенствованием, ограничено потребляемой усовершенствованным компонентом долей общего времени выполнения.

Другими словами, повышение производительности пропорционально доле, занимаемой улучшаемым компонентом в процессе выполнения программы. Закон Амдала объясняет, почему составляющие времени отклика следует рассматривать в порядке убывания. Именно поэтому в примере 1.2 не следует заниматься «проблемой» CPU service, не разобравшись предварительно с проблемой SQL*Net message from client. Если потребление процессорного времени уменьшится на 50%, то время отклика улучшится всего на 2%. Но если удастся уменьшить на те же 50% составляющую SQL*Net message from client, то общее время отклика уменьшится на 46%. В примере 1.2 каждый процент уменьшения SQL*Net message from client дает эффект приблизительно в двадцать раз больший, чем один процент уменьшения CPU service.

Закон Амдала формализует здравый смысл применительно к оптимизации. Он поясняет, как достичь наибольшей отдачи от вложенных в повышение производительности усилий.

Собираем воедино

Объединив три рассмотренных принципа технологии оптимизации Oracle в одном предложении, получим следующее простое правило:

В первую очередь следует стремиться к уменьшению самой значимой составляющей времени отклика в наиболее критичных для бизнеса операциях.

Казалось бы, просто, не так ли? Тем не менее я почти полностью уверен, что вы *не* придерживаетесь этого правила при оптимизации своих Oracle-систем. Этому правилу не следуют ваши консультанты и применяемые вами инструменты. Такой способ настройки не имеет ничего общего с рекомендациями, данными в книгах и в большинстве материалов семинаров и конференций по Oracle, начиная с 1980 г. В чем же дело?

А в том, что этот простой метод оптимизации Oracle-систем недоступен тому, кто не умеет измерять время отклика и интерпретировать полученные результаты. Как раз этой теме и посвящена основная часть этой книги.



Я надеюсь, что когда вы прочитаете эту книгу, мое утверждение «сейчас вы делаете это *не* так» уже потеряет актуальность. Сейчас, когда я пишу эту главу, имеются свидетельства того, что описываемый мной способ оптимизации набирает популярность среди пользователей Oracle. Хорошо, если книга, которую вы держите в руках, способствовала этой эволюции.

Средства анализа времени отклика

Определение *времени отклика*, данное Международной организацией по стандартизации ISO, несмотря на простоту, достаточно полезно:

Время отклика – это полное время, прошедшее с момента завершения запроса или команды компьютерной системе до начала ответа; например, период времени между событием окончания запроса и появлением первого символа ответа на терминале пользователя (взято на http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci212896,00.html).

Время отклика представляет собой *объективную* меру взаимодействия между поставщиком и потребителем. Потребители компьютерных услуг хотят получать правильный ответ за наименьшее время и с наименьшими затратами. Цель аналитика по производительности Oracle заключается в минимизации времени отклика в рамках экономических ограничений, наложенных владельцем системы. Если разложить время отклика на компоненты, то станет понятнее, каким путем этого достичь.

Диаграмма последовательности

Диаграмма *последовательности* – это удобный способ изображения составляющих времени отклика для действий пользователя. Диаграмма показывает передачу управления между уровнями технологического стека в процессе выполнения пользовательской операции. *Технологический стек* служит для представления многоуровневой архитектуры, образованной всеми элементами системы: пользователями, сетью, прикладными программами, ядром базы данных и оборудованием. Все элементы стека получают услуги от элементов нижележащего уровня и предоставляют услуги элементам, расположенным уровнем выше. На рис. 1.1 показана диаграмма последовательности для многоуровневой Oracle-системы.

Рис. 1.1 иллюстрирует приведенную ниже последовательность действий, наглядно показывая вклад каждого из уровней технологического стека в общее время отклика:

1. Пользователь, решив, что он хочет получить от системы, инициирует запрос на получение данных, нажимая в броузере кнопку «ОК». Практически мгновенно браузер принимает этот запрос. Ожидание пользователем ответа начинается с нажатия кнопки «ОК».
2. Броузер, затратив несколько мгновений на формирование изображения кнопки в прежнем, не нажатом состоянии, посылает HTTP-пакет в глобальную сеть (WAN). Некоторое время спустя запрос, пройдя по глобальной сети, достигает сервера приложений.
3. Сервер приложений, выполнив некоторые прикладные операции промежуточного уровня, формирует запрос к базе данных и отправляет его по протоколу SQL*Net в локальную сеть (LAN). Запрос за некоторое время (быстрее, чем в глобальной сети) проходит по локальной сети и поступает на сервер базы данных.

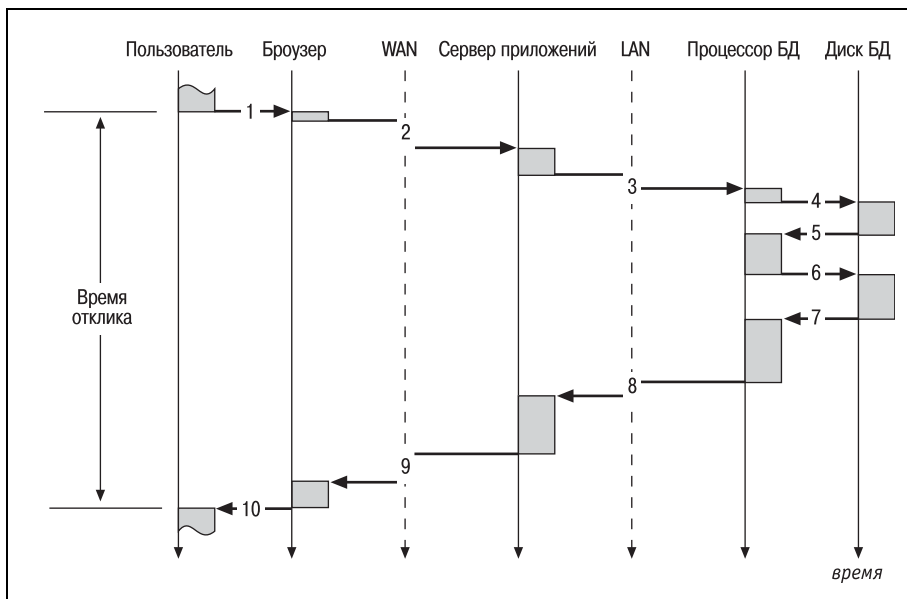


Рис. 1.1. Диаграмма последовательности многоуровневой Oracle-системы

4. Процессор сервера БД выполняет ряд вычислений, и ядро Oracle формирует системный вызов с запросом на чтение с диска.
5. Через некоторое время системный вызов заканчивает чтение и возвращает управление процессу сервера БД.
6. Выполнив еще ряд вычислений на сервере, ядро Oracle формирует еще один вызов на чтение.
7. Затратив еще некоторое время на выполнение дисковых операций, системный вызов снова возвращает управление процессу сервера БД.
8. Ядро Oracle, затратив немного процессорного времени на завершающую обработку, формирует ответ на запрос сервера приложений и отправляет его по протоколу SQL*Net в локальную сеть.
9. Сервер приложений преобразует полученные из базы данных результаты в формат HTML и отправляет их по глобальной сети броузеру по протоколу HTTP.
10. После отображения результатов на дисплее пользователя броузер возвращает управление пользователю. Получив запрошенную информацию, пользователь заканчивает ожидание, длившееся в течение *времени отклика*.



Хорошая диаграмма последовательности имеет детализацию, соответствующую решаемой задаче. Например, на рис. 1.1 в целях упрощения не показаны задержки, возникающие в броузере, сервере приложений и сервере БД, когда их операционные

системы выполняют переключение процессов из состояния *выполняется* в состояние *готов к выполнению* и обратно. В некоторых задачах анализа производительности такой уровень детализации был бы необходим. Влияние таких переключений на производительность рассматривается в главе 7.

На мой взгляд, идеальное средство анализа производительности Oracle еще не создано. Такой идеальный инструмент анализа должен иметь графический интерфейс для представления диаграммы последовательности, с точностью до микросекунды показывающей распределение составляющих времени отклика для любого действия пользователя. Большой объем данных, обрабатываемых таким приложением, потребует наличия развитых средств для обобщения и детализации, чтобы представить информацию именно так, как это требуется.

Вероятно, такое приложение скоро появится. Как вы увидите из этой книги, ядро Oracle уже может предоставить значительную часть данных, необходимых для создания такой программы. Основными проблемами на нынешний момент являются:

- Отсутствие в большинстве узлов многоуровневой системы средств представления данных о времени отклика, аналогичных тем, которые предоставляет ядро Oracle. О каких данных идет речь, уточняется в главе 7.
- В некоторых архитектурах трудность сбора диагностических данных, относящихся к конкретным пользовательским операциям. В главе 3 рассматриваются требования к представлению диагностических данных, а в главе 6 объясняется, как обойти трудности сбора данных, возникающие в различных прикладных архитектурах.

Однако многое из того, что нам необходимо, уже существует. Начиная с версии Oracle 7.0.12, в ядро включена и развивается поддержка средств измерения времени отклика. Эта книга поможет вам разобраться в том, как преимущества данного инструментария могут быть использованы в деле повышения производительности систем Oracle.

Профиль ресурсов

Полная диаграмма последовательности для любого мало-мальски сложного пользовательского действия содержит так много данных, что это заметно осложняет работу с ними. Следовательно, соображения удобства диктуют необходимость каким-то образом обобщить эту информацию. В примере 1.2 в качестве такого обобщения показан *профиль ресурсов*. Профиль ресурсов – это просто таблица, содержащая удобное представление структуры времени отклика. Обычно профиль ресурсов содержит, как минимум, следующие атрибуты:

- Категория операций
- Суммарное время выполнения операций данной категории
- Количество обращений к операциям данной категории

Профиль ресурсов особенно полезен, когда он содержит список категорий в порядке убывания потраченного на них времени. Такое представление профиля очень удобно для анализа производительности, т. к. фокусирует внимание на задачах, требующих решения в первую очередь. Профиль ресурсов – главный инструмент в моем наборе средств диагностики производительности.

Идея применения профилей ресурсов далеко не нова. Наша компания вдохновилась ей благодаря статье о программах-профилировщиках, опубликованной в 1980-х годах [Bentley (1988) 3–13] и основанной, в свою очередь, на работе Дональда Кнута, вышедшей в начале 1970-х [Knuth (1971)]. Идея разложения времени отклика на составляющие настолько практична, что вы наверняка эксплуатировали ее, даже не сознавая того. Вспомните, как вы оптимизируете маршрут в свое любимое место. Представьте себе тот «райский уголок», в котором вы всегда чувствуете прилив сил. Для меня это ближайший магазин «Woodcraft Supply» (<http://www.woodcraft.com>), где продаются не только всевозможные инструменты, способные отрезать пальцы и ломать ребра, но и книги и журналы, объясняющие, как этого избежать.

Вот как может выглядеть профиль ресурсов для путешествия по городу с оживленным движением в час пик (время выражено в минутах):

Составляющая времени отклика	Продолжительность		Кол-во вызовов	Время/вызов
Езда в час пик по автомагистрали	90 мин	90%	2	45 мин
Езда по окрестности	10 мин	10%	2	5 мин
Всего	100 мин	100%		

Если, предположим, до магазина всего пятнадцать миль, вам, вероятно, не очень понравится перспектива полторачасового стояния в пробках. Независимо от того, мыслите вы категориями профилей ресурсов или нет, вы скорее всего придете к тому же способу оптимизации, что и я: пожалуй, лучше отправиться в магазин, когда движение станет менее интенсивным.

Составляющая времени отклика	Продолжительность		Кол-во вызовов	Время/вызов
Езда в спокойное время по автомагистрали	30 мин	75%	2	15 мин
Езда по окрестности	10 мин	25%	2	5 мин
Всего	40 мин	100%		

Пример с поездкой достаточно прост, и ставки здесь невысоки, так что в данном случае можно обойтись и без формального анализа. Однако в более сложных задачах анализа производительности удобный формат профиля ресурсов помогает в решении вопроса, особенно когда речь идет о значительных финансовых и временных затратах.

Профили ресурсов помогают однозначно поставить задачу для проекта повышения производительности Oracle. В примере 1.3 представлен профиль ресурсов программы формирования платежной ведомости, рассмотренной ранее в разделе «Концентрация на пользовательских операциях». Не имея такого профиля, администраторы БД три месяца бились над известной им проблемой конкуренции за защелки. Придя в отчаяние, они потратили несколько тысяч долларов на новый процессорный блок, что привело к замедлению выполнения операции, которую они пытались ускорить. Получив в свое распоряжение профиль ресурсов, администратор в течение десяти минут нашел решение, сокращающее время отклика примерно на 50%. Более подробно эта проблема и ее решение рассмотрены в третьей части книги.

Пример 1.3. Профиль ресурсов для случая неправильной конфигурации сети. Первоначально предполагалось, что дело в конкуренции за защелки и недостаточной производительности ЦПУ

Response Time Component	Duration		# Calls	Dur/Call
SQL*Net message from client	984.0s	49.6%	95,161	0.010340s
SQL*Net more data from client	418.8s	21.1%	3,345	0.125208s
db file sequential read	279.3s	14.1%	45,084	0.006196s
CPU service	248.7s	12.5%	222,760	0.001116s
unaccounted-for	27.9s	1.4%		
latch free	23.7s	1.2%	34,695	0.000683s
log file sync	1.1s	0.1%	506	0.002154s
SQL*Net more data to client	0.8s	0.0%	15,982	0.000052s
log file switch completion	0.3s	0.0%	3	0.093333s
enqueue	0.3s	0.0%	106	0.002358s
SQL*Net message to client	0.2s	0.0%	95,161	0.000003s
buffer busy waits	0.2s	0.0%	67	0.003284s
db file scattered read	0.0s	0.0%	2	0.005000s
SQL*Net break/reset to client	0.0s	0.0%	2	0.000000s
Total	1,985.4s	100.0%		

Пример 1.4 демонстрирует еще один профиль ресурсов, спасший проект от обескураживающего и дорогостоящего провала. До появления этого профиля предполагаемое решение проблемы формирования отчета заключалось в увеличении объема памяти или модернизации подсистемы ввода/вывода. Профиль ресурсов однозначно показал, что данные мероприятия если и улучшат время отклика, то не более чем на 2%. Практически все время уходило на выполнение единственной команды SQL, вызывавшей около миллиарда обращений к блокам буферного кэша БД.



Глядя на профиль ресурсов из примера 1.4, невозможно сделать вывод о том, что процессорное время было затрачено на миллиард операций чтения. Каждый из 192072 «вызовов» ресурса CPU service представляет собой обращение к БД Oracle (например,

разбор, выполнение, выборку). С помощью детальной трассировки SQL для всех этих вызовов мне удалось определить, что они породили около миллиарда операций чтения. О том, как это сделать, подробно рассказано в главе 5.

Подобные проблемы обычно бывают вызваны ошибками в обслуживании, например, случайным удалением статистики, используемой *оптимизатором по стоимости* (*Cost-Based Optimizer – CBO*).

Пример 1.4. Профиль ресурсов для случая неэффективной команды SQL. Первоначально предполагалось, что проблема заключается в подсистеме ввода/вывода

Время отклика Component	Duration		# Calls	Dur/Call
CPU service	48,946.7s	98.0%	192,072	0.254835s
db file sequential read	940.1s	2.0%	507,385	0.001853s
SQL*Net message from client	60.9s	0.0%	191,609	0.000318s
latch free	2.2s	0.0%	171	0.012690s
other	1.4s	0.0%		

Total	49,951.3s	100.0%		

Пример 1.4 ясно показывает, как профиль ресурсов помогает не стать жертвой мифов. Здесь в качестве расхожего заблуждения, дезориентировавшего аналитика относительно причин задержек в сеансе, выступает утверждение о том, что высокий *коэффициент попаданий в кэш буферов базы данных* свидетельствует об эффективности выполнения команд SQL. Команда, вызывавшая замедление работы, показывала исключительно высокий коэффициент попаданий в кэш буферов. Причину легко понять, взглянув на формулу вычисления коэффициента попадания CHR (*cache hit ratio*) для данного случая:

$$\begin{aligned} CHR &= \frac{LIO - PIO}{LIO} \\ &\approx \frac{10^9 - 507385}{10^9} \\ &\approx 0.9995 \end{aligned}$$

В этой формуле *логический ввод/вывод* LIO (*logical I/O*) – это количество блоков, считанных из памяти Oracle (кэша буферов БД); *физический ввод/вывод* PIO (*physical I/O*) – количество блоков, считанных вызовами операционной системы.¹ Таким образом, значение выраже-

¹ Эта формула порождает и другие проблемы, помимо показанной в данном примере. Многие авторы, включая Адамса (Adams), Льюиса (Lewis), Кайта (Kyte) и меня самого, указывают на многочисленные и серьезные изъяны такого способа представления коэффициента попаданий в кэш буферов. Наиболее подробно это вопрос рассмотрен в работе [Lewis (2003)].

ния (LIO – PIO) равно количеству чтений блоков из кэша, не потребовавших обращений к операционной системе.¹

Несмотря на то, что многие специалисты сочли бы значение коэффициента 0,9995 «хорошим», оно все же не может считаться «идеальным». В отсутствие данных, представленных в примере 1.4, большинство известных мне аналитиков предположили бы, что низкая производительность вызвана неидеальностью коэффициента попаданий в кэш. Но профиль ресурсов ясно показывает, что даже если бы все 507385 операций физического чтения были обслужены кэшем буферов, экономия времени составила бы только 940,1 секунды. Максимально возможное улучшение от решения этой «проблемы» составило бы лишь 16 минут при общей продолжительности выполнения 14 часов.

Применение профилей ресурсов для анализа пользовательских операций радикально повысило эффективность работы специалистов по производительности. Прежде всего это превосходный инструмент для определения наиболее важной точки приложения усилий – в соответствии с провозглашенной нами целью:

Если вы где-то слышали, что добавление памяти решает все проблемы производительности

Пример 1.4 заставляет меня вспомнить первый пройденный мной курс по «настройке». Это было в 1989 г., когда я только пришел на работу в корпорацию Oracle. Наш преподаватель предложил простой способ оптимизации запросов к базе данных Oracle: просто исключить операции физического ввода/вывода. Я задал вопрос: «А как насчет доступа к памяти?», имея в виду большое значение в столбце *query* выходных данных *thprof*, которые мы в этот момент изучали. Инструктор ответил, что выборка из памяти выполняется настолько быстро, что ее влиянием на производительность можно пренебречь. Такой ответ меня разочаровал – до того, как начать карьеру в Oracle, я много занимался оптимизацией кода на С. Одним из наиболее важных моментов в этой работе было исключение избыточных обращений к памяти [Dowd (1993)].

Из примера 1.4 видно, почему и для вас устранение излишних обращений к памяти должно стать приоритетным. Лишние обращения к памяти увеличивают время отклика. Если таких обращений много, то расходуется *много* времени. При частоте процессора 2 ГГц выполнение кода, реализующего операцию логического ввода/вывода Oracle (LIO), занимает, как правило, несколько десятков микросекунд процессорного времени в пользовательском режиме. Следовательно, миллион операций LIO увеличат время отклика на несколько десятков секунд. Чрезмерная интенсивность операций LIO, в силу ряда причин, затрудняет масштабирование системы; причины рассмотрены во второй и третьей частях книги. Дополнительная информация имеется в работе [Millsap (2001c)].

¹ Имеется в виду – к диску. – *Примеч. науч. ред.*

В первую очередь стремиться к уменьшению самой значимой составляющей времени отклика в наиболее критичных для бизнеса операциях.

Другое важное преимущество использования профилей ресурсов заключается в том, что от них не может укрыться практически ни одна проблема производительности. Неформальное доказательство этой гипотезы состоит из двух утверждений:

Доказательство: Если нечто представляет собой проблему со временем отклика, это проявляется в профиле ресурсов. Если это не проблема со временем отклика, то это и не проблема производительности. ЧТД

О создании таких профилей ресурсов, от которых не укроется ни одна проблема производительности, рассказывается во второй части книги.

Метод R

Эта книга написана *не* для того, чтобы просто помочь вам ускорить работу вашей Oracle-системы. Настоящая ее цель – в оптимизации *проекта*, имеющего целью ускорение Oracle-системы. В мои планы не входит помощь в настройке какой-то одной системы. Я хочу помочь вам научиться ускорять работу любых систем, я также хочу, чтобы вы могли решать такие задачи с наибольшей экономической эффективностью, достижимой в вашем бизнесе. Для достижения этой цели я предлагаю Метод R. Именно на этом методе основана оставшаяся часть книги.

Концептуально Метод R очень прост. Как нетрудно заметить, это лишь слегка формализованное представление простого правила «В первую очередь стремиться к уменьшению самой значимой составляющей времени отклика в наиболее критичных для бизнеса операциях», которое встретилось нам уже неоднократно.

Метод R: метод повышения производительности, основанный на времени отклика и дающий максимальный экономический эффект для бизнеса

1. Выявите пользовательские операции, требующие ускорения с точки зрения *бизнеса*.
2. Получите диагностические данные, относящиеся к конкретному периоду времени или конкретной операции, которые позволят вам идентифицировать причины задержек в каждой из выбранных пользовательских операций, при их выполнении в наилучших условиях.
3. Выполните действия по оптимизации тех операций, которые дают наилучший совокупный эффект для бизнеса. Если даже наиболее действенные мероприятия не дают достаточного улучшения, отложите оптимизацию производительности до лучших времен.

Перейдите к шагу 1.

Кто применяет этот метод

Первая отличительная черта Метода R – это требования к специалистам, применяющим его. Метод *не* годится для технического специалиста, не заинтересованного в вашем бизнесе. Как я уже говорил, конечной целью Метода R является повышение ценности системы для *бизнеса*. Эта цель не может быть достигнута без понимания проблем бизнеса. Но где в организационной структуре отдела информационных технологий должен находиться человек, применяющий данный метод?

Эти отвратительные дымовые трубы

В большинстве крупных компаний службы поддержки технической инфраструктуры организованы в стиле «копящих труб», как показано на рис. 1.2. Организационная структура такого рода серьезно осложняет работу по оптимизации системы в силу одной фундаментальной причины:

Организационно раздробленные подразделения предпочитают заниматься оптимизацией изолированно от других подразделений, что приводит к появлению локально оптимизированных компонентов. Даже если это им удастся, результат получается далеко не оптимальный. Система, состоящая из локально оптимизированных компонентов, не обязательно оптимальна в целом.

Одно из наиболее значительных достижений Голдрата (Goldratt) в области оптимизации систем заключается в наглядной иллюстрации того, что локальная оптимизация совсем не обязательно ведет к оптимизации глобальной [Goldratt (1992)].

Модель «дымовых труб» заразна. Даже в кратких заявках на участие в конференциях на тему Oracle от нас требуют выбрать такую

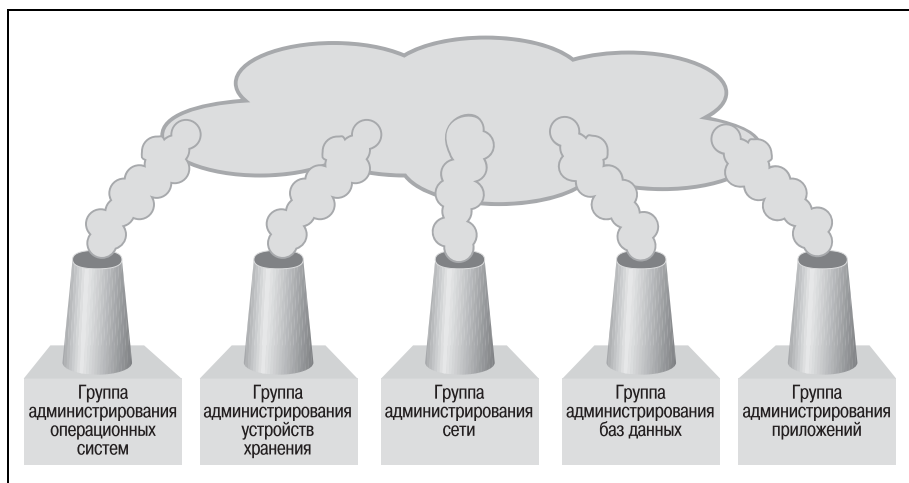


Рис. 1.2. Типичная организационная структура технического отдела

Та самая Цель

Одним из толчков к созданию Метода R послужила история, рассказанная Эли Голдратом в книге «The Goal» (Та самая цель) [Goldratt (1992)]. В книге описана победа нового революционного метода оптимизации производительности над старым, глубоко укоренившимся методом, дававшим худшие результаты. Метод Голдрата применяется в оптимизации производства, но рассказанная история до боли напоминает сегодняшнюю ситуацию в сообществе специалистов по Oracle: ниспровержение метода оптимизации, основанного на неверной системе измерений.

Эта книга опровергает массу ложных идей, на которых были основаны «знания» об оптимизации многих аналитиков. Вот два наиболее ярких урока, которые я усвоил из этой книги:

- Практика *калькуляции стоимости* часто приводит к неверным решениям при оптимизации. Пользователи Oracle придерживаются аналогичной практики, когда в качестве параметра оптимизации выбирают коэффициент попаданий..
- Система, состоящая из оптимизированных элементов, *сама не обязательно является оптимальной*. Это объясняет, почему производительность систем, полностью построенных из «лучших в своем роде» элементов, может хромать на обе ноги. Объясняет это и причину существования такого количества медленных Oracle-систем, компонентами которых управляют разные администраторы, уверенные в том, что уж их-то подсистемы точно не могут быть причиной низкой производительности.

Если вы еще не читали эту книгу – уверен, она доставит вам немало приятных минут. Если читали, перечитайте еще раз, сопоставляя прочитанное с тем, что происходит в области производительности Oracle. Как сказано в аннотации, «Читатели книги занимаются сейчас лучшим делом в своей жизни». Данное утверждение в точности отражает мое собственное отношение к этой книге.

«трубу» для каждой из наших презентаций (организаторы предпочитают называть эти трубы «секциями»). Например, имеется секция, посвященная настройке баз данных, и совершенно отдельная секция по настройке операционных систем. Что, если бы метод оптимизации производительности требовал поочередно уделять внимание разным элементам технологического стека? Думаю, малейший намек на такое разделение отвратит специалистов от применения такого метода. По крайней мере, аналитики, пользующиеся методом, охватывающим все уровни стека, испытывают трудности с выбором подходящей секции для своих докладов.



С кем бы из владельцев систем Oracle я ни говорил, почти всем им доставляет неудобство классический подход к разделению специалистов на прикладных программистов и администраторов баз данных. Какая из этих групп отвечает за производительность системы? Ответ: *обе*. Есть проблемы производительности,

которые прикладные программисты не в состоянии обнаружить без помощи администраторов баз данных. Точно так же, есть проблемы производительности, не устранимые силами администраторов без помощи разработчиков.

Лучший специалист по производительности

Наилучшим способом защиты компании от понижения производительности следует признать наличие у нее хорошего аналитика по производительности, свободно ориентирующегося во всех уровнях технологического стека и способного их диагностировать. В терминах рис. 1.2 такой специалист должен уметь не потеряться в «дымном облаке». Аналитик по производительности парит над трубами, пока не выберет, в какую из них нырнуть. А самый лучший аналитик обладает знаниями, интеллектом, обаянием и мотивацией, позволяющими ему вмешаться во взаимодействие этих труб, когда он уверен, что найден нужный вентиль.

Имея честь общаться с десятками аналитиков по производительности, я заметил, что большинство из них обладает общим набором качеств, которые, как мне представляется, и лежат в основе их успеха. Лучшая из известных мне характеристик этих талантливых специалистов принадлежит Джиму Кеннеди (Jim Kennedy) и Анне Эверест (Anna Everest) [Kennedy and Everest (1994)], выделившим четыре группы личных качеств:

Образование/опыт/компетентность

В категории образование/опыт/компетентность от хорошего аналитика требуются понимание *целей и процессов бизнеса*, а также *пользовательских операций*, обеспечивающих его функционирование. Хороший аналитик знает *финансы* в достаточной степени, чтобы понимать, какие сведения потребуются оперирующему финансовыми понятиями спонсору проекта для принятия информированного решения о вложении средств в проект по повышению производительности. И, разумеется, хороший аналитик разбирается в технических средствах прикладной системы, включающих *оборудование, операционную систему, сервер БД, прикладные программы* и все остальные компоненты, связывающие сервер и клиентов в единое целое. Многие важные технические составляющие описаны во второй части книги.

Интеллект

Хорошему аналитику присущ и ряд интеллектуальных качеств. Прежде всего, на мой взгляд, он должен чувствовать *значимость* — под этим понимается способность разобраться, что важно, а что нет. Это широкая категория. Сюда входят понятия *восприимчивости, здравого смысла и рассудительности*. Обязателен навык решения проблем, равно как и способность к быстрому восприятию и усвоению информации.

Межличностные отношения

В области межличностных отношений хороший аналитик должен обладать определенным набором качеств. *Доброжелательность* служит ключом к получению достоверной информации от пользователей, владельцев бизнеса и персонала, обслуживающего компоненты системы. *Самообладание* требуется для поддержания порядка в кризисные моменты, особенно с наступлением неизбежной «панической» стадии проекта. *Уверенность в себе* необходима для поддержания среди жертв и виновников проблемы правильного морального духа, вселяющего уверенность в выполнимости проекта. Хороший аналитик обладает *тактом* и умением *объединять* усилия для достижения поставленной цели.

Мотивация

Наконец, хороший аналитик по производительности демонстрирует высокую мотивацию. Он *ориентирован на клиентов и заинтересован в бизнесе*. Ему *интересны сложные задачи*, он *изобретателен*. Я заметил, что лучшие аналитики по производительности всегда уверены в том, что технические, интеллектуальные, межличностные и мотивационные проблемы разрешимы, но разные виды задач требуют зачастую совершенно разных подходов к их решению. Лучшие из аналитиков не только понимают это, но и извлекают немалую *пользу* из такого многообразия.

Ваша роль

Я хочу, чтобы уверенность в своих способностях к диагностике проблем производительности, приобретенная вами после прочтения этой книги, позволила вам ни на секунду не испугаться сценария, подобно-го приведенному ниже:

Место действия: Большое совещание. В числе участников несколько менеджеров подразделений, отвечающих за инфраструктуру, вы, а также генеральный директор, который, озаботившись медленной работой формы онлайн-овых заказов, специально пришел на ваше совещание – посмотреть, как вы собираетесь это исправить...

Старший менеджер отдела системного администрирования («Системный менеджер»): «В течение двух недель мы намерены увеличить производительность процессоров; стоимость нового оборудования и дополнительных лицензий составит 65 000 долларов. Благодаря удвоению мощности ЦПУ наши пользователи получают заметное повышение производительности».

Генеральный директор: (одобрительно кивает.) «Мы должны повысить производительность при оформлении онлайн-овых заказов, иначе потеряем одного из крупнейших розничных заказчиков».

Вы: «Но наша онлайн-овая форма, тратя на свое выполнение 45 секунд, расходует примерно 1,2 секунды процессорного времени. Даже если бы нам удалось *полностью исключить* эту составляющую из времени отклика, оно уменьшится лишь примерно на одну секунду».

Системный менеджер: «Я не согласен. Я считаю, что в представленных вами данных о времени отклика слишком много необъяснимых противоречий, чтобы делать такие выводы».

Вы: «Давайте обсудим это отдельно. Я объясню вам, как я пришел к этим выводам».

(Позже, на повторно созванном совещании.)

Системный менеджер: «Хорошо, я согласен. Он прав. Замена процессора не даст нужного прироста производительности, как мы надеялись».

Вы: «Зато, перераспределив нагрузку способом, который я могу изложить, мы сократим время отклика при формировании заказа не менее чем на 95%, *не тратя денег на новые процессоры*. Как видно из приведенного профиля времени отклика формы, замена ЦПУ никак нам не помогла бы».

Я был свидетелем множества обсуждений, в которых персонаж «Вы» получает первое слово, после чего дискуссия так и не возвращается в верное русло. Результат зачастую бывает ужасен. Компания постигает азы в поисках решения, способного поднять производительность. Иногда она останавливается, лишь полностью исчерпав отведенное время или деньги, или и то и другое.

Пожалуй, еще больнее видеть, как персонаж «Вы» выступает со своим предложением, а затем его заглушает хор скептиков, не верящих представленным данным. Если вы не сможете доказать достоверность всех своих данных, включая их источник и то, как они соотносятся с данными других участников, вы имеете все шансы проиграть в этом споре, несмотря на свою правоту.

Преодоление общего неодобрения

Надеюсь, своей книгой я смогу убедить вас опробовать Метод R на своей системе. На пути того, кто работает в одиночестве, большинство препятствий будут чисто техническими, и ему, вероятно, придется потрудиться, чтобы их устранить. Я приложил все усилия к тому, чтобы эта книга помогла с ними справиться.

Однако более вероятно, что повышение производительности системы потребует коллективных усилий. Скорее всего, реализация рекомендаций аналитика потребует участия коллег. Действия, рекомендуемые им в результате применения Метода R, зависят от следующих обстоятельств:

- Коллеги знакомы с этими идеями и не разделяют их.
- Они никогда раньше не слышали об этом.

В противном случае ваша система уже была бы оптимизирована. В обоих случаях аналитик, вероятно, окажется в окружении, готовом поспорить с его идеями. Чтобы достичь успеха, ему придется доказывать справедливость своих рекомендаций на языке, понятном его оппонентам.



Такой подход к обоснованию своего мнения будет продуктивным в любом случае – даже в самом дружественном окружении, где пожелания аналитика выполняются практически мгновенно.

Вот самый эффективный из найденных мною способов доказательства:

Доказательство проверкой

Нет лучшего способа доказательства результата, чем его демонстрация. Дэйв Энсор (Dave Ensor) описал это как «Метод Ювелира». Любой хороший ювелир, продавая товар, *как можно раньше* даст потенциальному покупателю поддержать его. Когда покупатель держит изделие *в руках*, он может гораздо полнее оценить его красоту и достоинство. Пока он мечтает, как изменится к лучшему его жизнь (стоит только ему стать обладателем этой вещицы), воображение покупателя работает на продавца. Этот метод прекрасно работает для дорогого товара: драгоценностей, автомобилей, домов, яхт и производительности систем. Пожалуй, нет лучшего способа заручиться поддержкой своих предложений, чем дать пользователям на деле *почувствовать*, насколько улучшится их жизнь в результате предлагаемых мероприятий.

Доходчивая статистика, понятная конечным пользователям

Если доказательство проверкой слишком сложно в реализации, следующим по действенности аргументом будет демонстрация статистики, имеющей смысл для конечных пользователей. Есть только *три* подходящих для такой статистики единицы измерений:

- Местная валюта
- Величина, на которую будет уменьшено время отклика
- Количество бизнес-операций в единицу времени, на которое возрастет производительность для какого-либо пользователя

Любая другая единица измерения породит одну из двух проблем. Либо аргументы не возымеют действия на аудиторию, либо, что еще хуже, убедить ее удастся, но из-за неправильно выбранной системы мер полученный результат не будет соответствовать реальному положению дел. Фактический результат *всегда* измеряется в единицах времени или денег. Преуспев в убеждении, но ошибившись в конечном результате, вы обречете свои будущие рекомендации на неминуемое недоверие.

Репутация, основанная на сбывшихся предсказаниях

Если вы обладаете незыблемой репутацией, помноженной на дар убеждения, малейшего вашего пожелания может оказаться достаточно для начала действий. Если это так, *будьте осторожны*. Любой прогноз связан с риском потери кредита доверия. Даже если полномочия позволяют вам ставить задачи сотрудникам, основываясь на своих предположениях, настоятельно советую провести сначала неофициальную проверку рекомендаций путем «доказатель-

ной проверки» или ориентированной на пользователя статистикой. Не стоит полагаться на кредит доверия, пока вы не уверены полностью в своих рекомендациях.

«Но вся система работает медленно»

На сайте *hotsos.com* Метод R является стилем жизни. Многократно испробовав этот метод, я могу с уверенностью сказать, что самым сложным этапом этого метода оказывается тот, который вообще отсутствует в списке: этап убеждения людей применять его. Первое возражение, с которым сталкиваемся я и мои коллеги, применяя подход, основанный на пользовательских операциях, предсказуемо, как восход солнца:

«Но вся система работает медленно»

«Надо настраивать *всю систему*, а не только одного пользователя»

«Когда вы перейдете к методу, который позволит настраивать *систему в целом?*»

Мы слышали это везде, где бы мы ни были.

Что делать, если вся система работает медленно? Специалисты обычно нервно реагируют на метод повышения производительности, если он основан на анализе выполнения только одной операции в данный момент. А если пользователи замечают, что «вся система тормозит», возникает сильное искушение начать анализ со сбора общесистемной статистики. Причина заключается в опасении упустить что-то важное, ограничившись анализом части системы. Да, действительно, сосредоточившись на приоритетных пользовательских операциях, вы *действительно* пренебрежете некоторыми вещами:

Концентрация внимания на высокоприоритетных операциях заставляет игнорировать не относящиеся к делу данные о производительности. Под «не относящимися к делу» я понимаю любые данные, которые могут помешать обнаружению и устранению *наиболее значимой* проблемы производительности.

Вот почему Метод R эффективен независимо от происхождения проблемы – порождена ли она отдельной пользовательской операцией или же набором различных операций. На рис. 1.3 показана информация, в первую очередь получаемая аналитиками, когда они приступают к изуче-

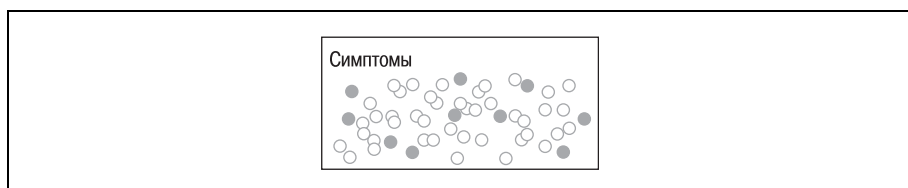


Рис. 1.3. Картина, наблюдаемая аналитиком при ухудшении производительности. Заштрихованные круги обозначают пользовательские операции, характеризующиеся плохой производительностью

нию проблемы. Достоверная информация о проблемах производительности в первую очередь обычно поступает в виде жалоб пользователей.



Иногда поставщики информации первыми узнают о проблемах с производительностью. В главе 9 описана ситуация, в которой такая ситуация может быть предсказана *заранее*. Хотя это редкий случай, чтобы поставщики информации узнали о проблемах производительности прежде, чем им о них расскажут ее потребители.

Получив такую информацию, большинство аналитиков первым делом выясняют причинно-следственные связи между наблюдаемыми *симптомами* и их возможными *основными причинами*. Я полностью согласен с тем, что это верный шаг. Однако многие проекты потерпели неудачу из-за того, что аналитики не сумели установить *правильную* связь между причиной и следствием. Сила Метода R заключается в том, что он позволяет установить причинно-следственные связи быстрее и точнее, чем любой другой метод.

Рис. 1.4 объясняет, почему это так. Показаны три возможных набора причинно-следственных связей между источниками плохой производительности и ее проявлениями. Понимание эффективности метода R в каждом из этих сценариев, по сравнению с обычными методами настройки, поможет вам решить, эффективен ли метод R при общесистемной оптимизации. Вот эти три:

- Первый крайний случай (а) предполагает, что все симптомы, замеченные пользователем системы, вызваны единственной «универсальной» причиной.

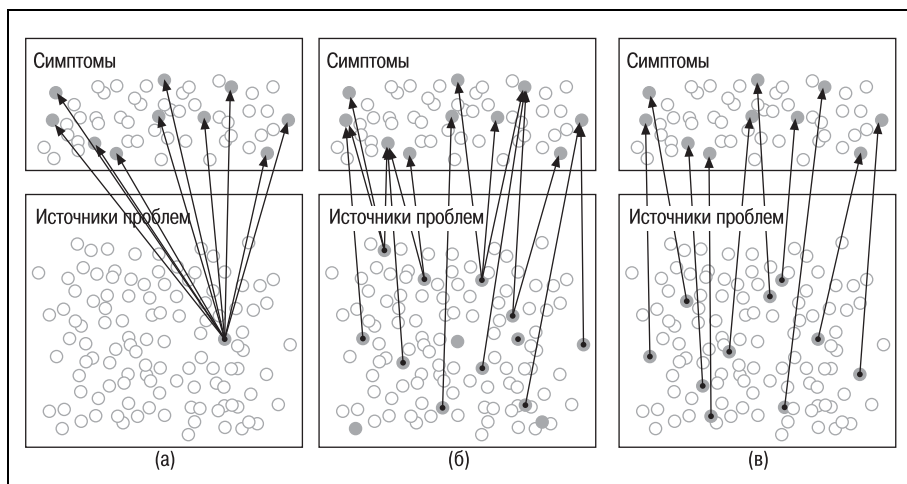


Рис. 1.4. Три возможных набора причинно-следственных связей (показаны стрелками) между причинами и симптомами сниженной производительности

- Во втором случае (б) между причинами и их проявлениями наблюдаются отношения «многие ко многим». Некоторые симптомы вызваны двумя или больше причинами, а некоторые причины вносят свой вклад в несколько симптомов.
- Другой крайний случай (в) соответствует ситуации, в которой каждый симптом вызван своей отдельной причиной. Нет такой причины, которая оказала бы негативное влияние на более чем одну пользовательскую операцию.

Конечно, легко рисовать картинки с причинно-следственными связями источников проблем и их симптомов. Совсем другое дело – выявить такие связи в реальности. В способности к этому, как мне представляется, и заключается основная мощь, выделяющая метод R. Сейчас объясню почему.

В случаях, подобных (а), метод R работает достаточно хорошо. Даже если вы напортачили в определении приоритетов бизнеса на первом этапе, главная причина все равно проявит себя в первых же диагностических данных. Причина проста. Если все симптомы вызваны одной и той же причиной, то неважно, какой из них вы рассматриваете, – все равно эта единственная, универсальная причина отыщется в профиле времени отклика этого симптома.

Так же хорошо метод R работает в случаях (б) и (в). Здесь единственным способом улучшить работу системы в целом будет устранение каждой из причин, вносящих свой вклад в наблюдаемые симптомы. Ограниченные (временем) возможности аналитика, возможно, не позволят ему заняться всеми проблемами одновременно, в таком случае важно установить приоритеты работ. Именно этим объясняется предусмотренное методом R распределение приоритетов. Помня о том, что фактическая *цель* проекта по повышению производительности определяется *экономикой*, наивысший приоритет следует отдавать устранению наиболее важных *симптомов*. Для метода R характерно, что он заставляет выстраивать приоритеты процесса в соответствии с интересами бизнеса.

Рассмотрим для сравнения эффективность метода C для каждого из трех описанных случаев. Как вы помните, на первом этапе метода C требуется

Построить гипотезу о несоответствующем значении некоторой метрики х.

В контексте рис. 1.4 этот этап аналогичен выявлению заштрихованных кругов в той части, которая называется *основные причины*. После обнаружения вероятных источников проблем метод C предлагает аналитику установить причинно-следственные связи между причинами и их проявлениями. Изъян метода C в том, что, поскольку нет плана для поиска причинно-следственных связей, приходится действовать наугад. Обычно такой подход состоит в том, чтобы что-нибудь «поправить», а потом посмотреть, к чему это привело. Это метод проб и ошибок.

Успешность метода С определяется тем, как быстро будет найден параметр системы с «неподходящим» значением. И чем длительнее его поиски, тем сильнее затягивается проект. Безусловно, шанс правильно идентифицировать проблему, требующую решения, гораздо выше, когда она – единственная в системе. Однако далеко не факт, что источник неприятностей будет быстро обнаружен даже в таком «простом» случае, как (а). Из того, что весь букет проблем вызван одной единственной причиной, вовсе не следует, что только одна системная статистика будет иметь «несоответствующее» значение.

По-настоящему метод С начинает пробуксовывать при попытке применить его в ситуациях (б) и (в). В обоих случаях движение «снизу вверх» приводит к необходимости выбора из нескольких потенциальных источников проблем. Как узнать, какой из них требует первоочередного внимания? Наилучшим способом расстановки приоритетов будет «проход по стрелкам» назад – от наиболее критичных для бизнеса симптомов к порождающим их причинам. Претендентами на первоочередное внимание будут те причины, которые способны вызвать наиболее неприятные симптомы.

Однако здесь притаилась очередная трудность метода С:

Общесистемные показатели производительности не содержат информации о причинно-следственных связях.

Невозможно достоверно установить причинно-следственные связи, показанные на рис. 1.4, пока не определена структура времени отклика для каждой пользовательской операции – «сверху вниз» в контексте этого рисунка. Понимая, какая информация необходима для установления причинно-следственных связей, мы ясно видим как недостатки метода С, так и силу метода R. Нельзя достоверно определить связи в направлении от причин к симптомам (снизу вверх). В то же время не составляет труда провести стрелки от симптомов к причинам (сверху вниз), т. к. профиль ресурсов для выбранной пользовательской операции точно указывает их местоположение.

Проект, в котором не установлены причинно-следственные связи, неуправляем. Выбор приоритетов при повышении производительности всегда *должен* основываться на экономических приоритетах бизнеса. Не нарисовав такие стрелки, нельзя правильно организовать свои действия, основанные на внутренних показателях производительности, полученных из отчетов *Statspack*. Без этих стрелок едва ли не единственной возможностью для вас останется «калькуляция стоимостей», в частности коэффициентов попаданий. К сожалению, такие величины плохо коррелируют с экономическими показателями бизнеса. В рассмотренном выше примере с платежной ведомостью ситуация три месяца оставалась неопределенной. Проект завершился в тот день, когда команда получила данные, приведенные в примере 1.3.



Ирония в том, что свойство метода R, часто вызывающее возражения, в действительности является его большим преимуществом. Фактически мы специально создали его для эффективной работы с системами, плохая производительность которых вызвана несколькими причинами одновременно.

Причина успешной работы метода R в случае общесистемного кризиса производительности в том, что «вся система» не является единой сущностью, она состоит из пользовательских операций разной степени важности. Все операции могут выполняться медленно не по одной и той же причине. Если же причин несколько, то как узнать, какой из них следует заняться в первую очередь? Правильный путь – в составлении списка приоритетов пользовательских операций в порядке убывания их важности с точки зрения бизнеса. А что если все задержки *на самом деле* вызваны единственной причиной? Тогда вам повезло: первый же набор диагностических данных, собранных для одного процесса, укажет на общий источник проблем всей системы. Устранив проблемы одного процесса, вы сделаете это и для всех остальных. В табл. 1.1 приведены сравнительные характеристики нового и существующего методов.

Таблица 1.1. Достоинства методов C и R. Преимущества метода R особенно заметны на «медленных в целом» системах

Вариант на рис. 1.4	Эффективность метода C	Эффективность метода R
(а)	Эффективен в ряде случаев. Наличие единственного источника проблем увеличивает вероятность его выявления в ходе анализа общесистемной статистики.	Эффективен. Даже при неправильном определении приоритетов бизнеса единственный источник проблем идентифицируется с первой попытки.
(б)	Неприменим. Не позволяет проследить связь причины с вызванным ею эффектом, что заставляет устранять проблемы «снизу вверх» без учета приоритетов бизнеса.	Эффективен. Учет приоритетов бизнеса гарантирует, что самые важные проблемы будут выявлены и устранены первыми.
(в)	Неприменим. Причины те же, что и в случае (б).	Эффективен. Причины те же, что и в случае (б).

«Метод работает только в случае проблем с базой данных»

Другое часто возникающее возражение против метода R связано с мнением о его непригодности для поиска и устранения проблем производительности, если они вызваны внешними по отношению к уровню хранения данных причинами. Сейчас, когда практически все сложные системы имеют многозвенную архитектуру, из такого предположения можно сделать вывод о серьезных ограничениях области применения метода R.

Сам по себе метод R никоим образом не содержит подобных ограничений. Обратите внимание: ни в одном из четырех этапов метода нет указаний на то, что информация о времени отклика собирается *только для базы данных*. Впечатление об ориентированности на базу данных возникает в связи с *реализацией* шага 2, на котором происходит сбор подробных диагностических данных о времени отклика. Эта книга, как вы увидите, посвящена исключительно средствам измерения времени отклика, встроенным в ядро Oracle. Такой подход основывается на нескольких причинах:

- Когда производительность падает, люди стремятся первым делом обвинить в этом наименее понятную им часть системы. Поэтому база данных Oracle часто оказывается первым компонентом, на который сыплются обвинения в низкой производительности. На самом деле ядро Oracle предоставляет достаточно диагностических данных для однозначного решения о том, в нем ли кроется источник проблем.
- В момент написания этих строк из всех уровней технологического стека ядро Oracle, несомненно, имеет самый полный набор средств измерения. К сожалению, аналитики зачастую неспособны правильно воспользоваться полученными от этих средств данными. Структура средств диагностики Oracle весьма развита, несмотря на свою простоту и эффективность (см. главу 7). Производители остальных компонентов технологического стека уже обратили на это внимание. Я полагаю, что встроенные в ядро Oracle средства диагностики времени отклика станут стандартными и для остальных звеньев.

Если источник проблем заключен в базе данных, метод R позволяет решить их быстро и эффективно даже при ограниченной оснащенности остальных уровней средствами измерений. В том случае, если источник проблем находится *вне* базы данных, метод R позволяет так же быстро и эффективно *доказать* этот факт. Независимо от места возникновения проблемы метод R страхует аналитика от попыток борьбы с несуществующими проблемами.

Практика – критерий истины. Ниже перечислены ситуации, в которых метод R последовательно подводит нас к местам возникновения проблем, требующих решения, независимо от того, находятся они внутри базы данных или вне ее:

- Неэффективность запросов, вызванная неудачно написанными в приложении командами SQL, плохой организацией данных, ошибочной стратегией индексирования, проблемами с плотностью данных и т. д.
- Неэффективность прикладных программ, связанная с избыточным разбором, плохо спроектированными механизмами сериализации (блокировками), неправильным применением (или неприменением) способов работы с массивами и т. д.
- Эксплуатационные ошибки, вызванные неправильным сбором статистики для оптимизатора по стоимости, случайными изменения-

ми схемы (например, удалением индексов), переполнением файловой системы и т. д.

- Сетевые ошибки, связанные с неправильной конфигурацией программного обеспечения, сбоями оборудования, неудачно спроектированной топологией и т. д.
- Ошибки дискового ввода/вывода, связанные с исчерпанием кэшей, несбалансированностью нагрузки на разные устройства и т. д.
- Ошибки в планировании вычислительных ресурсов, проявляющиеся в недостаточной производительности ЦПУ, памяти, дисков, сети и т. д.

«Этот метод необычен»

Даже если метод R окажется важнейшим достижением с момента изобретения строк и столбцов, я вполне допускаю, что еще пару лет после выхода этой книги он будет встречать определенное сопротивление. Метод нов и не похож на то, к чему все привыкли. По мере того, как описанные здесь приемы будут проникать в практику, книги и инструменты, сопротивление, надеюсь, будет ослабевать. Тем временем коллеги будут требовать от вас подробно объяснять, почему вы рекомендуете им такой необычный метод оптимизации производительности, который не основывается ни на пакете *Statspack*, ни на каком другом популярном средстве мониторинга производительности, возмозжно, стоившем вашей компании немалых денег. То, что вы пользуетесь таким непривычным методом, может послужить для них причиной отказа от ваших предложений.

В этой книге одна из моих целей состоит в том, чтобы вооружить вас таким знанием о технологии Oracle, которое позволит полностью реализовать потенциал, заключенный в диагностических данных. Полагаю, прочитав эту книгу, вы сможете аргументированно защитить свою точку зрения. Надеюсь, это укрепит ваши позиции, и предложенные вами меры по повышению производительности будут оцениваться по критериям экономической эффективности, а не по названию метода, из которого они позаимствованы.

Оценка эффективности

Выше в этой главе приведены восемь критериев, по которым, на мой взгляд, следует оценивать метод повышения производительности. Закончу главу сравнением метода R с обычными методами, с точки зрения этих характеристик:

Действенность

Метод R всегда максимально эффективен, поскольку заставляет сосредоточиться на цели, имеющей смысл для бизнеса, – времени отклика выбранной пользовательской операции.

Эффективность

Метод R обеспечивает наибольшую отдачу от проекта в силу того, что нацеливает вас на наивысшие приоритеты бизнеса и позволяет принимать полностью осознанные решения на каждой стадии проекта. Фактически эффективность *проекта* была главным условием при проектировании метода.

Измеримость

В методе R в качестве критерия выступает время отклика, а не внутренние технические характеристики, которые не всегда могут быть непосредственно преобразованы в показатели преимущества для конечного пользователя.

Прогнозируемость

Метод R дает беспрецедентные возможности для прогнозирования результата предлагаемых действий для выбранной пользовательской операции, не требуя для этого дорогостоящих экспериментов.

Достоверность

Метод R работает достоверно практически для всех мыслимых проблем производительности; метод отличается тем, что позволяет указать на источник проблем любого типа, не прибегая к помощи опыта, интуиции или везения.

Детерминизм

Метод R исключает диагностику «угадыванием» благодаря, во-первых, нацеленности на приоритеты бизнеса; во-вторых, обладанию надежной методикой для определения истинных связей между видимыми симптомами и источниками проблем.

Конечность

Метод R имеет четко выраженное условие завершения. Его отличает способность установить момент, когда дальнейшие усилия по оптимизации перестают быть экономически оправданными.

Практичность

Метод R поддается изучению и уже был использован сотнями аналитиков самой различной квалификации для быстрого и эффективного улучшения производительности Oracle.

Последующие главы посвящены описанию применения метода R.

2

Выбор пользовательских операций

В любом проекте на одном из первых этапов необходимо определить, в чем должен заключаться его результат. Формальное описание поставленной цели называется *спецификацией* проекта. Проект по повышению производительности Oracle, как и все прочие проекты, нуждается в спецификации. Иначе невозможно будет оценить степень его успеха или провала.

Многие проекты повышения производительности Oracle с самого начала хромают из-за плохих спецификаций. Вы, наверное, видели карикатуру, на которой изображен менеджер, говорящий программисту: «Начинай писать. А я пойду узнаю, чего они хотят». Очень многие пытаются «настраивать свою систему», не имея представления о том, чего хотят достичь. С другой стороны, никому не нужна система, месяцами прозябающая в ожидании «совершенной» спецификации от аналитиков, каждый дюйм продвижения которых вызывает новые расходы. На создание хорошей спецификации проекта по повышению производительности Oracle обычно требуется не больше двух часов.

Эта глава имеет целью помочь вам правильно начать свой проект по повышению производительности – так, чтобы он повысил экономическую ценность системы. Мы рассмотрим несколько плохих спецификаций и выясним, почему они вместо помощи принесли проектам вред. Рассмотрим и ряд удачных спецификаций, способствовавших быстрому завершению проектов с большим экономическим эффектом. Также мы введем ряд характеристик, отличающих хорошие спецификации от плохих.

Надежность спецификации

Спецификация проекта может считаться «надежной», только если любой проект, который в точности последовал «букве» спецификации,

достиг и той цели, которая ставилась перед началом проекта. К несчастью, большинство спецификаций проектов по повышению производительности ненадежны. Спецификация может содержать такие требования:

- Максимально равномерно распределить дисковый ввод/вывод по нескольким дисковым устройствам.
- Обеспечить не менее $x\%$ незанятых ресурсов ЦПУ в часы пиковой нагрузки.
- Увеличить коэффициент попаданий в кэш буферов базы данных как минимум до $x\%$.
- Исключить все полные просмотры таблиц в системе.

Каждое из этих требований ненадежно в том смысле, что их выполнение может не привести к желаемому изменению в поведении системы. Определить, надежна ли спецификация, вам поможет простая игра:

Для того чтобы выяснить, надежна ли спецификация проекта по повышению производительности, задайте себе вопрос: «Можно ли выполнить требования спецификации, не повысив при этом производительность системы?».

Вот хороший способ сыграть в такую игру: представить себе злого джинна. Может ли злой джинн, следуя букве вашего желания (спецификации проекта), привести проект к результату, противоречащему вашей очевидной цели? Если злому джинну удастся, выполнив условия спецификации проекта, создать систему с неудовлетворительной производительностью, то это и будет доказательством ненадежности такой спецификации.

Прием с игрой в злого джинна применял в своих мысленных экспериментах Рене Декарт (Rene Descartes) в 1600-х годах, и, сравнительно недавно, героиня Элизабет Херли (Elizabeth Hurley) в фильме «Bedazzled» (в российском прокате – «Ослепленный желаниями»). Вот как злой джинн мог бы сыграть по приведенным выше правилам:

Максимально равномерно распределить дисковый ввод/вывод по нескольким дисковым устройствам

Такое требование вполне оправданно, если вы пытаетесь предотвратить ухудшение производительности при конфигурировании новой системы, но с точки зрения проекта по *повышению* производительности оно представляется ненадежным. Во многих системах ощутимое повышение производительности дискового ввода/вывода сказывается на общей производительности незначительно или даже отрицательно.

Представьте, например, систему, в которой каждый из наиболее важных бизнес-процессов, требующих ускорения, расходует на операции дискового ввода/вывода менее 5% от общего времени отклика. (На сайте *hotsos.com* есть сотни трассировочных файлов с такими данными.) В этой системе никакая «настройка» ввода/вывода

не приведет к ускорению работы более чем на 5%. Поскольку равномерное распределение ввода/вывода по нескольким устройствам может и не привести к сколько-нибудь значимому приросту производительности, это требование ненадежно.

Обеспечить не менее $x\%$ свободных ресурсов ЦПУ в часы пиковой нагрузки

У злого джинна есть несколько способов выполнить такое требование, не затронув производительность системы. Один из них – создать узкое место в дисковом вводе/выводе, например, поместив всю базу данных на один гигантский накопитель с низкой пропускной способностью. По мере роста очереди пользовательских процессов на ввод/вывод будет увеличиваться и неиспользуемая часть мощности процессора. Выполнение этого требования может привести к снижению производительности, следовательно, это требование ненадежно.

Увеличить коэффициент попаданий в кэш буферов БД минимум до $x\%$

И это просто: воспользуйтесь новой демонстрационной программой Коннора МакДональда (Connor McDonald), приведенной в приложении С. Вы узнаете, как увеличить коэффициент попаданий в кэш буферов до любого количества девяток, загружая процессор ненужной работой. Такая дополнительная нагрузка, разумеется, снизит производительность системы, зато «улучшит» коэффициент попаданий. Конечно, программа Коннора – это трюк, демонстрирующий ошибочность представления о коэффициенте попадания, как о критерии качества системы. (Мне достоверно известно, что Коннор – не злой, хотя и был однажды замечен в поведении, отчасти присутствующем джинну.)

Есть и более изощренные способы снижения производительности системы при одновременном «улучшении» коэффициента попаданий в кэш. Например, так часто поступают «настройщики» SQL, привлекаемые в проект для исключения операций полного просмотра TABLE SCAN FULL (обсуждаемых в следующей спецификации). Еще один способ, которым злой джинн может увеличить коэффициент попаданий в кэш, снизив при этом производительность, заключается в уменьшении размера буфера для выборки массивом¹ до одной строки [Millsap (2001b)]. В силу того, что можно так просто увеличить коэффициент попаданий в кэш буферов, снизив при этом производительность системы, данное требование следует признать исключительно ненадежным.

Исключить все полные просмотры таблиц в системе

К сожалению, многие из тех, кто изучает оптимизацию производительности SQL, очень рано усваивают ошибочное эмпирическое правило, утверждающее, что «полные просмотры таблиц всегда вред-

¹ Array fetch size. – Примеч. науч. ред.

ны». Злой джинн легко состряпает сотни команд SQL, производительность которых будет падать при исключении операции TABLE SCAN FULL [Millsap (2001b); (2002)]. Из-за того, что исключение полного просмотра таблиц способно реально снизить производительность, такое требование не может служить надежной основой для спецификации проекта по повышению производительности.

Избежать ненадежности спецификации проекта очень легко. Просто говорите, что думаете. Хотя, конечно, по такой логике играть в гольф несложно: достаточно забивать мяч в лунку каждым ударом. Трудность борьбы с ненадежностью спецификаций в том, чтобы сформулировать действительные требования таким способом, который не приведет впоследствии к ошибкам. Например, спецификация, более точно отражающая фактическое намерение, может выглядеть так:

Ускорить работу системы.

Однако даже такая спецификация ненадежна. Я встречал десятки проектов, выполненных по такой спецификации, формально успешных, но неудачных по существу. Например, консультант, исследуя V\$SQL, находит пакетное задание, выполняющееся четыре часа. Он «настраивает» его так, что оно выполняется за 30 минут. Проект реализован успешно – выданное консультантами заключение подтверждает это. Однако достижение лишено смысла. Пакетное задание и так выполнялось с достаточной скоростью, поскольку запускалось во время свободного от других работ восьмичасового перерыва. Средства, вложенные в повышение производительности (оплата консультантов), не принесли выгоды бизнесу.

Хуже того, я знаю аналитиков, ускорявших некую программу А ценой замедления другой, гораздо более важной, программы В. Во многих системах существуют взаимные зависимости между процессами, способные привести к такому результату. В таких системах «настройка» второстепенной программы не только отнимает время и деньги на выполнение этих работ, она фактически *снижает* ценность системы для бизнеса (см. раздел «Пример 1: Заблуждения, вызванные общесистемными данными» в главе 12).

Спецификация «ускорить работу системы» слишком расплывчата, чтобы быть полезной. Работая в отделе обслуживания корпорации Oracle, я неоднократно участвовал в дискуссиях, посвященных спецификациям проектов, – внедрение сервисных пакетов невозможно без спецификации целей проекта на уровне контракта. Большинство участников этих совещаний очень быстро понимали, что требование «ускорить работу системы» звучит слишком неопределенно. Что меня поражает сейчас – это то, что многие из них видели неопределенность совершенно не там, где надо.

Большинство людей видят суть проблемы в той части спецификации, которая предписывает *ускорить работу системы*. Они считают это тре-

бование неполным, т. к. в нем отсутствует указание, *насколько* ускорить. На упоминавшихся мною совещаниях в Oracle попытки конкретизировать такую спецификацию обычно выливались в обсуждение различных методик измерения фактической и субъективной скорости, способов введения «эквивалентных» метрик, основанных на счетчиках событий (аналогично коэффициенту попаданий), и т. д. Конечно, поиски «эквивалентных метрик» заканчивались ничем, потому что (если вы правильно выполните тест на злого джинна) такие метрики, основанные на предположениях, обычно оказываются ненадежными.

Выясняя, насколько «должна ускориться» система, зачастую удается лишь без пользы растратить ресурсы проекта. (Исключение составляет случай, когда аналитик находит максимально допустимое время выполнения операции путем моделирования с применением теории массового обслуживания, как показано в главе 9.) Сегодня, когда наши слушатели обсуждают требование по ускорению системы, их почти не приходится подталкивать к правильному решению о том, что суть проблемы скрывается в слове *система*. Рассмотрим, например, такие часто предлагаемые «улучшения» исходной спецификации «ускорить работу системы»:

- Ускорить работу системы на 10%.
- Заставить систему выполнять все бизнес-функции менее чем за 1 секунду.

Прежде всего, подобные требования так же беззащитны перед трюками злого джинна, как и исходная спецификация. Но в действительности, добавив детали, мы ослабили первоначальное утверждение. Например:

Ускорить работу системы на 10%

Вы действительно уверены в том, что каждая бизнес-транзакция в системе может выполняться на 10% быстрее? Даже та, которая требует лишь пары обращений к логическому вводу/выводу (LIO) Oracle? С другой стороны, действительно ли достаточно ускорения на 10% для онлайн-запроса, если его время отклика составляет 17 минут?

Заставить систему выполнять все бизнес-функции менее чем за 1 секунду

Что хорошего в том, что время выполнения выборки единственной строки по первичному ключу составит 0,99 секунды? С другой стороны, разумно ли ожидать, что приложение Oracle сможет построить 72-страничный отчет менее чем за одну секунду?

Привели ли эти два дополнения к улучшению исходного требования «ускорить работу системы»? Нет. Гораздо более серьезная проблема связана с недостаточно четко определенным понятием «система».

Система

Что такое *система*? Большинство администраторов систем и баз данных интерпретируют этот термин иначе, чем все остальные участники бизнеса. Для большинства администраторов *система* – это сложная совокупность процессов, областей разделяемой памяти, файлов, блокировок и защелок, а также разных технических штучек, доступных через «V\$-таблицы», утилиты операционной системы и, возможно, графические средства системного мониторинга. Однако *никто*, кроме администраторов, не видит систему с этой стороны. Для пользователя *система* – это набор экранных форм и пакетных заданий, соответствующих его области деятельности. Для менеджера *система* – это инструмент повышения эффективности бизнеса. И пользователям, и менеджерам красные, желтые и зеленые индикаторы на вашей панели мониторинга абсолютно безразличны.

Вот простой тест, который поможет вам убедиться, что я говорю правду. Попробуйте представить себя на месте пользователя, ожидающего завершения отчета, который должен был быть сдан два часа назад сегодня утром, – и все из-за того, что на создание «пятнадцатиминутного отчета» требуется полных три часа. Подумайте, что вы ответите администратору базы данных, который на совещании в присутствии ваших коллег заявит примерно следующее: «Во время выполнения вашего отчета система работала совершенно нормально, все наши индикаторы находились в зеленой зоне в течение этих трех часов».

Пожалуйста, помните об этом, когда выступаете в роли аналитика по производительности: система – это набор пользовательских программ. Конечные пользователи относятся к ним очень внимательно. (Если на какую-то программу никто не обращает внимания, это означает, что ее надо запускать в нерабочее время или, возможно, не запускать вообще.) Период времени, который требуется программе для выработки очередной порции данных для бизнеса, – это ее время отклика. Время отклика отдельной пользовательской операции – это единственный показатель производительности, который представляет интерес для бизнеса. Следствие:

В первую очередь вас должно интересовать время отклика операций конечного пользователя.

Экономические ограничения

Избавившись от неоднозначного толкования слова «система», мы значительно приблизились к правильному формулированию цели:

В интересах бизнеса необходимо увеличить производительность программы А в рабочие дни с 14:00 до 15:00. В этот период производительность программы А должна быть увеличена настолько, насколько это возможно.

Но является ли эта спецификация «джинноустойчивой»? Еще нет. Допустим, среднее время выполнения программы А составляет две минуты. Предположим, злой джинн мог бы сократить время отклика с двух минут до 0,25 секунды. Великолепно... Но это обойдется в \$1 000 000 000. Увы. Возможно, достаточно будет уменьшить время отклика до 0,5 секунды, затратив на это всего \$2000. В приведенной спецификации не упоминаются никакие экономические ограничения.

И все же существует спецификация проекта, которая, я уверен, может устоять перед происками джинна. Это цель оптимизации, сформулированная Эли Голдратом (Eli Goldratt) в [Goldratt (1992), 49]:

Делать деньги, одновременно увеличивая чистую прибыль, возврат инвестиций и усиливая движение денежных потоков.

Эта спецификация задает обязательные условия, которым должны соответствовать все прочие спецификации проектов. Однако она тоже страдает излишней обобщенностью, как и уже упоминавшееся правило «забивать мяч в лунку каждым ударом».

Создание хорошей спецификации

Оставим в покое ошибочные спецификации и займемся составлением правильных. На создание хорошей спецификации для большинства проектов повышения производительности у вас должно уйти не больше двух часов. Вот рецепт:

1. Выявить пользовательские операции, оптимизация которых необходима для *бизнеса*, и определить, в каком контексте проявляется их важность.
2. Разбить найденные пользовательские операции на группы по пять элементов и назначить им приоритеты.
3. Для каждой операции из первой группы определить, *за кем и когда* вы будете вести наблюдение при выполнении операции в наиболее благоприятном контексте.

Пользовательская операция

В этой книге я стараюсь провести четкое различие между *пользовательскими операциями, программами и сеансами Oracle*. Смысл *пользовательской операции* полностью соответствует названию – это операция, выполняемая пользователем. Такая операция может заключаться в заполнении поля формы или в выполнении одной или нескольких программ. Пользовательская операция определяется как некоторая единица работы, результат и скорость выполнения которой имеют значение для бизнеса. Понятие *пользовательской операции* особенно важно при составлении спецификации проекта, поскольку именно эта единица работы важна с точки зрения бизнеса.

Понятно, что *программа* – это последовательность компьютерных команд, выполняющих некоторую бизнес-функцию. Пользовательская операция может состоять из программы, из ее части или из нескольких программ. *Сеанс Oracle* – это конкретная последовательность обращений к БД, осуществляемых в течение соединения между пользовательским процессом и экземпляром Oracle. Программа может создать несколько сеансов Oracle или не создавать ни одного, а в некоторых конфигурациях несколько программ могут совместно пользоваться одним сеансом. Важность понятия *сеанса Oracle* при сборе диагностической информации обусловлена тем, что ядро Oracle ведет статистику производительности на уровне сеансов.



В Oracle разделены понятия *соединения* (маршрута взаимодействия) и *сеанса*. Можно установить соединение с Oracle и не иметь сеанса. С другой стороны, можно создать несколько сеансов одновременно, используя единственное соединение.

Выбор пользовательских операций и контекстов

Первым шагом в создании спецификации должно быть выявление тех пользовательских операций, оптимизации которых требуют интересы *бизнеса*. Если этого не сделать, проект повышения производительности, скорее всего, провалится. Жизненно важно иметь список *конкретных пользовательских операций*. Необходимо выбрать из них те, которые имеют наибольшее влияние на прирост чистой прибыли, возврата инвестиций и движения денежных потоков.

Я особенно подчеркиваю, что «оптимизации требуют интересы *бизнеса*», т. к. в этот момент вас не интересует мнение администратора базы данных о ее производительности. Одна из наиболее распространенных ошибок, допускаемых аналитиками по производительности Oracle, заключается в том, что они изучают данные в представлениях $\forall\$, чтобы понять, где требуется «настройка». А представления $\forall\$ не могут подсказать вам этого. В главе 3 я приведу ряд технических причин, которые не позволяют полагаться на данные представлений $\forall\$ в этом вопросе.$$$

Выяснить потребности бизнеса обычно не составляет труда. Крайне редко для этого приходится затевать длительный проект по определению целей. Практически всегда эти сведения можно получить, задав наделенному здравым смыслом руководителю предприятия вопрос: «Если бы мы могли сегодня к вечеру ускорить работу одной из программ, какую из них вы бы выбрали?». Следующий пример иллюстрирует возможные типы ответов:

- Мы производим дисковые накопители. Склад заполнен готовыми к отправке изделиями. Каждое утро мы получаем сотни звонков от рассерженных клиентов, сделавших заказы больше двух недель назад, которые требуют сообщить о состоянии этих заказов. В среднем на погрузочной площадке в каждый момент времени находится

около двадцати свободных машин службы доставки. Если вы сейчас пройдете на погрузочную площадку, то увидите, что наши упаковщики вместе с водителями сидят на коробках и распивают кофе. Они не могут погрузить эти коробки, потому что программа, печатающая транспортные накладные, работает слишком медленно. Главная причина плохой производительности в нашем бизнесе заключается в программе печати транспортных накладных.

- Мы слишком много тратим на серверные лицензии и техническую поддержку. У нас в компании 57 серверов уровня предприятия, и нам надо сократить их количество до десяти или меньше. Мы уже перенесли 80% данных компании в сетевое хранилище данных (Storage Area Network – SAN). Однако десять серверов вряд ли справятся с объемом вычислений, выполняемых 57 машинами. Основная проблема производительности в нашем бизнесе заключается в исключении лишней нагрузки, что позволит нам консолидировать вычислительные мощности и избавиться приблизительно от 50 серверов.

Обычно самым трудным делом оказывается поиск того сотрудника, который обладает нужной вам информацией. Возможно, придется покопаться в длинном списке. Вот некоторые полезные советы:

Спросите у своего босса, где сосредоточены риски, связанные с производительностью

Уведите разговор от обсуждения технических вопросов функционирования базы данных. Ведите разговор на языке пользователей. Выясните, кто из пользователей чаще всего жалуется на производительность системы, и встретьтесь с ним за обедом. Проблема самого недовольного пользователя может и не быть главной для бизнеса, но знакомство с проблемами этого пользователя может стать хорошим началом.

Пригласите пользователя на обед

Угостите его и задайте простой вопрос: «Если бы я мог что-нибудь ускорить в твоей работе, что бы ты выбрал?»

Найдите прогноз сбыта для вашего бизнеса

Подумайте, какие прикладные процессы приобретают наибольшее значение в связи с планируемым ростом продаж компании. Выполняются ли эти процессы настолько эффективно, насколько это возможно?

Если вы вплотную занялись выяснением у сотрудников степени важности пользовательских операций для бизнеса, узнайте у них, какие операции относятся к следующим категориям:

- Операции, критичные для бизнеса.
- Долго выполняющиеся операции.
- Наиболее часто выполняющиеся операции.

- Операции, потребляющие большую часть того ресурса, который вы пытаетесь сэкономить.

В дополнение к определению операций, требующих оптимизации, требуется определить *контекст*, в котором проявляется их важность. Например:

- Всегда ли операция выполняется медленно?
- Наблюдается ли замедление в определенное время дня (недели, месяца, года)?
- Связано ли замедление операции с одновременным запуском других программ?
- Не вызвано ли замедление достижением некоторого порогового количества пользовательских сеансов?
- Не возникает ли замедление после выполнения некоторых других программ (загрузки, удаления и т. д.)?

Не определив контекст, вы рискуете, собрав диагностическую информацию для проблемной операции, после всех приложенных усилий обнаружить очевидные свидетельства отсутствия проблем с данной операцией. Необходимо выяснить, как заставить интересующую вас операцию в момент ее наихудшей производительности. Иначе нельзя обнаружить проблему. Это правило настолько важно, что я повторю его еще раз:

Необходимо выяснить, как заставить исследуемую операцию в момент ее *наихудшей* производительности.

На этом этапе бывает важно выбрать несколько пользовательских операций, особенно в тех ситуациях, когда несколько различных проблем оказывают влияние на многих пользователей. Это верно даже в тех случаях, когда имеется одна главная проблема с производительностью системы, намного более серьезная, чем все остальные. Этот совет продиктован опытом многократного применения данного метода:

- Чистая прибыль зависит от издержек, поэтому прибыль бизнеса от улучшения, например, операции №3 может в действительности превышать прибыль от улучшения операции №1.
- Быстрое улучшение ситуации по *любой* из пяти основных проблем может дать большое политическое преимущество, включающее такие факторы, как повышение морального духа в коллективе и доверие спонсора проекта.
- Возможно, неизвестно, как повысить производительность пользовательской операции №1. Но оптимизация, к примеру, операции №3, может настолько снизить избыточную нагрузку, что №1 потеряет свое значение.
- Нельзя сказать, какое из мероприятий по повышению производительности даст наибольший прирост прибыли, пока не будет проведен высокоуровневый анализ затрат и результатов по всем пяти операциям первой группы.

Назначение приоритетов пользовательским операциям

Итак, список пользовательских операций составлен и необходимо оценить степень важности их улучшения с точки зрения бизнеса. Все последующие действия подразумевают, что выбор важнейших кандидатов на оптимизацию уже сделан. Расстановка бизнес-приоритетов жизненно важна по нескольким причинам, среди которых:

Наиболее важные операции будут оптимизированы раньше

Это самая важная причина. Тут все просто, если вы не занимаетесь оптимизацией важнейших бизнес-процессов, значит, то, чем вы занимаетесь, – это не оптимизация.

В компромиссных решениях приоритет будет отдан более важной операции

Вы можете вдруг обнаружить, что оптимизация одной пользовательской операции отрицательно сказывается на производительности другой. Такое часто случается, когда выбранная стратегия оптимизации заключается в увеличении производительности некоторого компонента. Надеюсь, однако, что я убедил вас прибегать к замене компонентов только в случае необходимости (то есть *редко*), и такие компромиссы будут скорее исключением, чем правилом.

На менее важных операциях сказывается сопутствующая выгода

Термин *сопутствующие потери* пришел в наш язык из сводок о несчастных случаях во время военных действий. Противоположный по смыслу термин *сопутствующая выгода* означает выгоду, неожиданно полученную в связи с другим событием. Сопутствующая выгода в производительности часто имеет место в тех вычислительных системах, где бывает исключена значительная бесполезная нагрузка.

На этом этапе можно чересчур увлечься анализом, но на самом деле на него не надо тратить много времени. Достаточно грубой классификации. Я рекомендую объединить пользовательские операции в группы по пять или больше и назначить им приоритеты. В этом случае у вас не возникнет искушения заняться точным ранжированием близких по важности операций. Например, если у вас десять проблемных операций, разбейте их не больше чем на две группы по пять. Если проблемных операций больше десяти (я бывал в компаниях, где их количество переваливало за пятьдесят), предлагаю разделить список на три части:

1. Пять наиболее важных операций (первая группа).
2. Пять следующих по важности операций (вторая группа).
3. Все оставшиеся важные пользовательские операции из списка (объединение третьей и последующих групп).

Будьте особенно осторожны, когда в расстановке приоритетов участвует большая группа людей. Конечно, каждый пользователь попытается убедить вас в исключительнейшей важности выполняемых им действий для всей системы. Разумеется, все операции в системе не могут

иметь высший приоритет. Значительная часть времени, которое вы потратите на дискуссии о том, в какую из групп следует поместить ту или иную операцию, может быть потрачена с гораздо большей пользой на следующих этапах. Если вы видите, что задача расстановки приоритетов занимает больше нескольких минут, вернитесь на шаг назад и примите благоразумное решение. Заверьте пользователей, чьи операции не получили высшего приоритета, в том, что они ничего не потеряют; их проблемы тоже будут рассмотрены.

Кто и когда выполняет каждую из операций

На последнем этапе составления спецификации проекта по повышению производительности надо определить, как следует идентифицировать каждую из выбранных операций, когда она в следующий раз будет выполняться в установленном контексте. Эта информация поможет найти программы, реализующие данную операцию, с тем, чтобы измерить их производительность.

Часто успешность сбора статистических данных зависит от вашей способности установить хорошие отношения с человеком, отвечающим за выполнение медленной операции и способным ответить на следующие простые вопросы:

- Когда, по его мнению, произойдет очередное замедление в выполнении операции?
- Как это можно наблюдать?

Ответы на эти вопросы однозначно определяют параметры процесса сбора диагностических данных, описанного в главе 3.

Если в вашем распоряжении есть средство непрерывного мониторинга соответствующей статистики производительности всех пользовательских операций системы, то необязательно выяснять, кто и когда будет запускать проблемную программу. Роскошь обладания такими данными обо всех операциях системы позволит вам реагировать на уже произошедшие случаи вместо того, чтобы заниматься предсказаниями их появления в будущем. Такие инструменты дороги, но они существуют.

Если такого инструмента нет, то придется проявить большую избирательность в отношении собираемой диагностики, и описанный здесь этап необходим. Надеюсь, главы 6 и 8 будут хорошим подспорьем в этом деле.

Избыточные ограничения в спецификации

Мы обсудили проблемы надежности, обусловленные неоднозначностью спецификаций. Спецификации, страдающие излишней подробностью, ничуть не лучше. Чрезмерная детализация требований часто приводит к тому, что спецификация противоречит цели оптимизации. Спецификация, содержащая определенные требования к повышению произво-

длительности программы *одновременно* с повышением на 10 пунктов коэффициента попаданий в кэш буферов, может оказаться просто невыполнимой. Вполне может получиться так, что повышение производительности определенной программы приведет к резкому снижению коэффициента попаданий в кэш для системы в целом. Соответствующий пример приведен в [Millsap (2001b)].

Еще один забавный случай имел место, когда я работал в Oracle. Спецификация производительности требовала, чтобы в некотором клиент-серверном приложении переход с одного поля формы на другое занимал не более 0,5 секунды. Далее шло требование о расположении клиента системы в Сингапуре, а сервера – в Чикаго. Более того, спецификация запрещала модификацию имевшегося приложения, которое выполняло в среднем шесть синхронных обращений к базе данных по глобальной сети (WAN) для каждого поля.

Цель в том виде, как она была поставлена в спецификации, была недостижима в силу *избыточных ограничений*. Фактически она противоречила физическим законам нашей вселенной. Нет никакой возможности за полсекунды шесть раз передать сообщение по сети из Сингапура в Чикаго и обратно. Даже при исключении всех составляющих времени отклика, за исключением минимального теоретически возможного времени распространения сигнала с максимальной теоретически возможной скоростью (т. е. без учета всех задержек в кабелях, концентраторах, маршрутизаторах, базах данных и т. д.), шестикратное прохождение сигнала туда и обратно заняло бы *не менее* 0,6 секунды.

Доказательство: Допустим, что на время перехода от поля к полю не оказывают влияния никакие факторы, кроме скорости света. Скорость света в вакууме составляет приблизительно 299 792 458 метров в секунду. Расстояние между Сингапуром и Чикаго по поверхности Земли приблизительно 15 000 000 метров. Следовательно, расстояние пройденное при шести запросах составит $2 \times 6 \times 15\,000\,000$, т. е. приблизительно 180 000 000 метров для каждого поля. Из соотношения $d = rt$ получаем $t = d/r \approx 0,6$ секунды на каждое поле. Если же учесть все отброшенные ранее факторы задержки, время отклика будет еще больше. Следовательно, требования спецификации не могут быть выполнены. ЧТД.

Не существует способа выполнить требования спецификации, не ослабив хотя бы одно из условий. Главный кандидат на сокращение – условие, согласно которому переход к каждому полю должен сопровождаться в среднем шестью обращениями клиента к серверу. Основной задачей этого проекта повышения производительности стало доказательство того, что проект с такой спецификацией обречен на неудачу. До тех пор, пока доказательство не было представлено, люди продолжали тратить время и деньги в погоне за недостижимой целью.

Хорошие проекты не рождаются из плохих спецификаций. Будь то нечеткая формулировка проблемы или невыполнимые условия в спецификации – в любом случае проект улучшения производительности не может быть основан на спецификации, содержащей ошибки.

3

Выбор диагностических данных

Теперь, когда определены пользовательские операции, в ускорении которых бизнес заинтересован в первую очередь, наступает время сбора данных. Сбор диагностических данных – это та фаза проекта, на которой типичный аналитик по производительности действительно начинает ощущать, что дело движется. В имеющейся на сегодняшний день литературе по Oracle очень скупо отражены вопросы сбора диагностических данных о производительности. Это большое упущение. Метод получения диагностических данных имеет огромное влияние на успех проекта. Только исключительное везение может предотвратить крах проекта, основанного на неверно собранных данных.

Полагаю, эта глава вас удивит. Здесь описаны два серьезных изъяна стандартных процедур сбора данных, глубоко укоренившихся в мире Oracle. Практически в каждом из тех сотен неудачных проектов повышения производительности Oracle, которые мне и моим коллегам приходилось спасать, среди причин неудач были ошибки в сборе данных. К сожалению, почти все документы о производительности Oracle, выпущенные до 2000 г., подталкивают своих читателей к совершению этих ошибок. Надеюсь, основанные на здравом смысле примеры из этой главы навсегда изменят ваше отношение к сбору диагностических данных.

О сборе данных

В проекте повышения производительности, основанном на методе R, суть сбора данных заключается в получении информации о времени отклика *правильно выбранной пользовательской операции*. Не больше и не меньше. К сожалению, многие разработчики приложений, не обеспечивая в своих программах соответствующего инструментария, сильно усложняют процесс получения данных.



Многие компании, в том числе Oracle, в последних версиях начали улучшать инструментарий для определения времени отклика.

Уроки по сбору данных, полученные в этой главе, заставят вас думать, что задача сбора данных труднее, чем вы ожидали. Умение правильно решать эту задачу поможет уменьшить общую стоимость и продолжительность проекта за счет исключения дорогостоящего и мучительно-го поиска ответов методом проб и ошибок.

Проект повышения производительности, реализуемый в соответствии с методом R, протекает существенно иначе, чем проект, основанный на методе С – обычном методе проб и ошибок, описанном в главе 1. Различия проиллюстрированы на рис. 3.1. Обычно исполнитель проекта ощущает, что лед тронулся, в тот момент, когда закончены выбор целей и сбор данных и можно приступать к этапу анализа и проверок. Как правило, исполнитель, действующий по методу С, достигает этого момента (момент t_1 на рис. 3.1) раньше, чем тот, кто работает над той же проблемой по методу R (момент t_2). Если вы не ожидали такого поворота событий, то метод R может стать политически уязвимым.

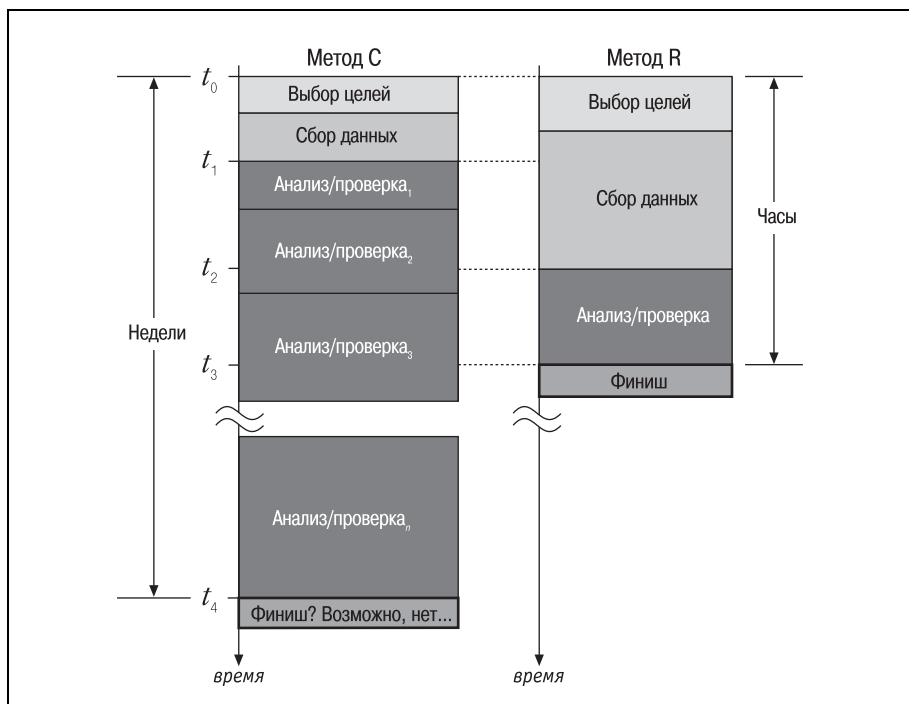


Рис. 3.1. Фазы выбора целей и сбора данных для метода R требуют больше времени, чем для обычных методов, но общее время выполнения проекта, как правило, значительно меньше

В промежутке между моментами t_1 и t_2 риск потери сторонников нового метода максимален.

Быстрое завершение фазы сбора данных не является целью проекта повышения производительности. Действительная цель состоит в оптимизации системы с наименьшими возможными затратами. Метод R оптимизирован по данному критерию. Фактически метод R разрабатывался нами именно с целью помочь нашим клиентам справиться с проектами повышения производительности, длившимися неделями и даже месяцами без заметного продвижения. В подавляющем большинстве проектов, выполненных по методу R, мы смогли продемонстрировать, что цель оптимизации может быть достигнута в течение часа с момента получения корректно ограниченных по областям видимости диагностических данных. Как только получены *правильные* данные для диагностики производительности, методу R достаточно одной итерации анализа/проверки для приближения к поставленной цели в отношении выбранной пользовательской операции.

Специалисты, применяющие метод С, большую часть времени проводят в попытках методом проб и ошибок установить причинно-следственные связи между сотнями возможных источников проблем и их проявлений, требующих первоочередного внимания. Исключительно низкая эффективность метода С вызвана необходимостью выполнения, как правило, нескольких итераций анализа и проверки, прежде чем будет выявлена причина возникновения симптома. Каждая следующая итерация требует все больше времени, т. к. обычно аналитики в первую очередь проверяют самые простые и очевидные предположения, оставляя более сложные и дорогостоящие процедуры настройки до того момента, когда все простые предположения будут отвергнуты.

Последний изъян метода С связан с тем, что он практически не позволяет количественно оценить момент «окончания настройки». Во многих проектах пользователи метода С не в состоянии обоснованно судить о действительном вкладе каждого обнаруженного источника в возникновение проблемы. Даже в «успешных» проектах исполнители в течение недель и даже месяцев не имели представления о том, достигнуто ли полное решение проблемы производительности (*оптимизация*) или лишь частичное (*настройка*). Проблема неопределенности в вопросе о том, возможно ли дальнейшее улучшение выполнения пользовательской операции, приводит к состоянию, которое Гаджа Вайдьянатха (Gaja Vaidyanatha) и Кирти Дешпанде (Kirti Deshpande) остроумно назвали «маниакально-настроечным расстройством» (Compulsive Tuning Disorder – CTD) [Vaidyanatha and Deshpande (2001) 8]. В продолжение их шутки могу добавить, что CTD – это болезненное состояние, вызванное *надеждой*. Если быть более точным, к состоянию CTD приводит отсутствие полной информации, позволяющей обоснованно судить о возможности дальнейшего повышения производительности некоторой пользовательской операции. Метод R заполняет этот

Разные методы для разных проблем производительности?

Возможно ли, что в случае «простых» проблем настройки производительности обычные методы более эффективны, чем метод R, подходящий для «сложных» ситуаций? Вопрос поставлен не совсем корректно: как оценить сложность проблемы, не занимаясь сбором каких-либо диагностических данных?

Создавая метод R, мы пришли к необходимости предварительного получения легкодоступных диагностических данных, позволяющих принять решение о необходимости сбора дополнительных, более труднодоступных данных. Мы сочли такой подход близким к оптимальному. Его недостаток в том, что практически нельзя быть *уверенным* в наличии причинно-следственной связи, не имея *корректных* диагностических данных (разумеется, получить *корректные* данные не всегда просто). Элементы сомнения и неопределенности, привносимые в проект анализом легкодоступных данных, приводят к быстрой его деградации. Я неоднократно наблюдал, как некачественные диагностические данные заводят аналитика в тупик. В связи с этим уместно вспомнить замечательное высказывание, приписываемое кардиналу Томасу Вулси (Thomas Wolsey) (1471–1530): «Будьте весьма и весьма осторожны, вкладывая что-то себе в голову, потому что уже никогда не сможете вынуть это обратно».

Главным требованием при создании метода R было сделать его *детерминистским*. Детерминизм – ключевое свойство, определяющее пригодность метода к обучению и автоматизации. Мы стремились к тому, чтобы любые два человека, применяющие метод R к решению некоторой проблемы производительности, выполняли бы одинаковую последовательность действий, не прибегая к помощи опыта, интуиции или везения для определения следующего шага. Наш метод гарантирует это благодаря наличию единой точки входа и хорошо определенной последовательности логических условий для каждой последующей точки принятия решений.

информационный пробел, устраняя причины возникновения расстройства CTD.

Для того, кто впервые применяет метод R, получение диагностических данных может стать самой сложной частью проекта. Есть приложения, в которых сбор диагностических данных чрезвычайно прост. Другие превращают получение корректных диагностических данных в тяжелое испытание. В главе 6 рассмотрены простые и сложные типы приложений и показаны приемы, помогающие мне и моим коллегам преодолевать различные трудности. Хорошая новость: как только вы поймете, каким способом следует получать правильные диагностические данные для выбранной пользовательской операции, работа в следующих проектах пойдет значительно легче и быстрее. Ну а метод C так никогда и не избавится от многочисленных итераций анализа/проверки, на количество которых не влияет уровень квалификации аналитика.

Надеюсь, в будущем большинство производителей прикладного ПО облегчат пользователям и аналитикам сбор точных диагностических данных, необходимых методу R. В последних версиях Oracle E-Business Suite процесс диагностики упрощен, и, судя по дошедшим до меня сведениям, в 10-й версии Oracle ядро и сервер приложений двигаются в том же направлении. Методы, аналогичные методу R, доминируют в других областях промышленности, и прогресс в области сбора диагностических данных приведет к повсеместному его признанию как основы проектов повышения производительности.

Область данных

Хорошая методика сбора данных о производительности Oracle требует соответствующих методик принятия решений в двух измерениях. Необходимо получить данные в правильной *временной области* и в правильной *области операций*. Начнем с построения диаграммы, показывающей структуру времени отклика в виде последовательности временных промежутков, соответствующих потреблению различных ресурсов. Результат показан на рис. 3.2. Для простоты будем считать, что наша воображаемая система состоит из ресурсов трех видов, обозначенных C, D и S. Пусть это будут процессор, диск и некий ресурс с последовательным доступом (как, например, ядро Oracle, предоставляющее последовательный доступ блокировкам, защелками и определенным операциям с буферами памяти). На рис. 3.2 ось времени расположена горизонтально.

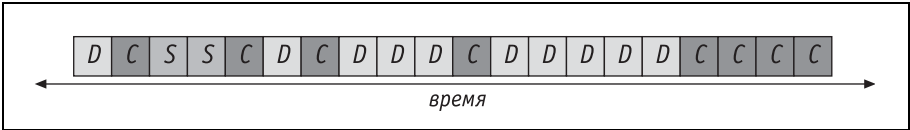


Рис. 3.2. Распределение времени между ресурсами в ходе выполнения пользовательской операции

Аналогично можно изобразить систему, выполняющую несколько пользовательских операций одновременно, поместив такие диаграммы одну под другой, как это показано на рис. 3.3. На этой диаграмме ось пользовательских операций расположена вертикально.

В следующих разделах данный способ представления помогает объяснить, почему методы сбора данных, практикуемые многими экспертами по Oracle с 1980-х годов, лишь губят проекты повышения производительности Oracle по всему миру.

Ошибки в определении области

Представьте, что в системе, изображенной на рис. 3.3, процесс выбора пользовательских операций, описанный в главе 2, показал следующее: наибольшая проблема производительности для бизнеса заключа-

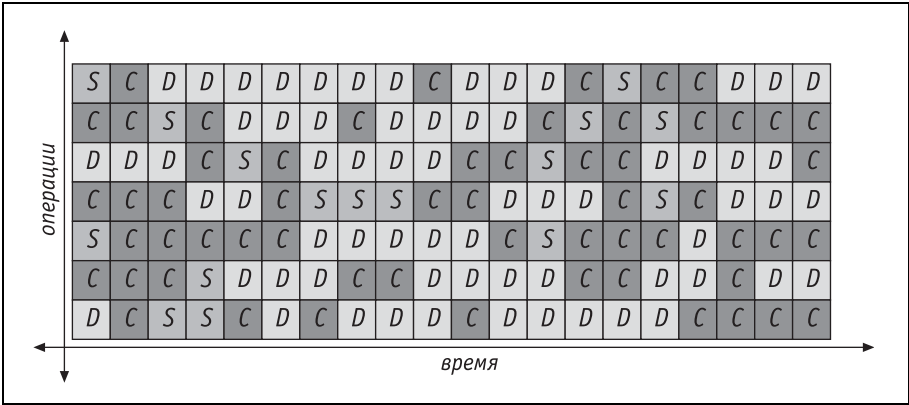


Рис. 3.3. Двумерная диаграмма показывает систему, выполняющую семь одновременных операций, каждая из которых в процессе выполнения потребляет ресурсы трех видов

ется в том, что для пользователя Уоллеса время отклика (между моментами t_1 и t_2) недопустимо велико (рис. 3.4).



В последующем обсуждении будем использовать математическое обозначение для закрытого интервала. Обозначение $[a, b]$ соответствует множеству значений, заключенному между a и b , включая их:

$$[a, b] = \{ \text{все значения } x, \text{ для которых } a \leq x \leq b \}$$

Из рис. 3.4 легко видеть, что на проблемном интервале большая часть времени потребляется ресурсом S, а оставшаяся – ресурсом C (табл. 3.1). Понятно, что решение проблемы производительности для Уоллеса требует уменьшения времени, потребляемого ресурсом S, или C, или ими обоими. Закон Амдала говорит, что эффект от сокращения по-

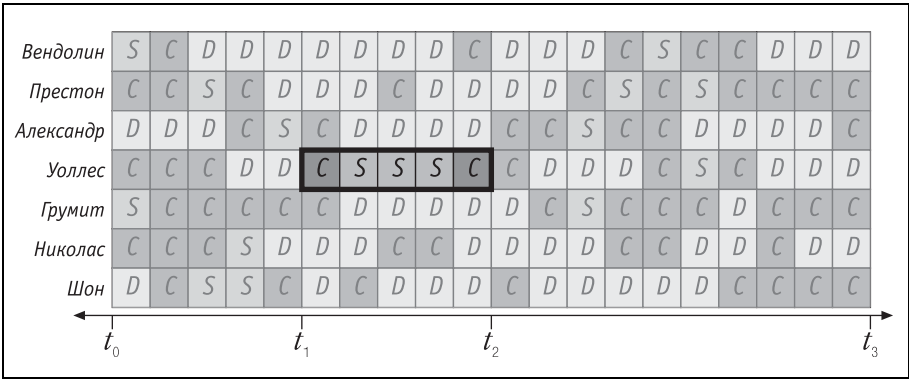


Рис. 3.4. Наиболее важный пользователь системы Уоллес страдает от недопустимо низкой производительности в интервале времени $[t_1, t_2]$

требления ресурса S будет в 1,5 раза выше, чем от ресурса C вследствие того, что доля S во времени отклика в 1,5 раза больше доли C.

Таблица 3.1. Профиль ресурсов для операции пользователя Уоллеса в интервале $[t_1, t_2]$

Ресурс	Время использования	Доля в общем времени
S	3	60,0%
C	2	40,0%
Всего	5	100,0%

Пожалуй, самая распространенная ошибка в получении данных – это сбор данных, агрегированных по обоим измерениям. Рис. 3.5 иллюстрирует эту ошибку. Толстая черная линия, охватывающая все блоки диаграммы, показывает, что были получены общие данные по всем процессам (не только для Уоллеса) и во всем временном интервале $[t_0, t_3]$ (не только для $[t_1, t_2]$). Подсчет составляющих времени для всей системы в интервале $[t_0, t_3]$ дает профиль ресурсов, приведенный в табл. 3.2. Как видите, проблема пользователя Уоллеса, которая, как нам известно, заключается в слишком больших временных затратах на выполнение операции S, оказалась погребенной под кучей не относящихся к делу данных. Результатом небрежности, допущенной при сборе данных, будет задержка в выполнении проекта повышения производительности и, возможно, его меньшая результативность.

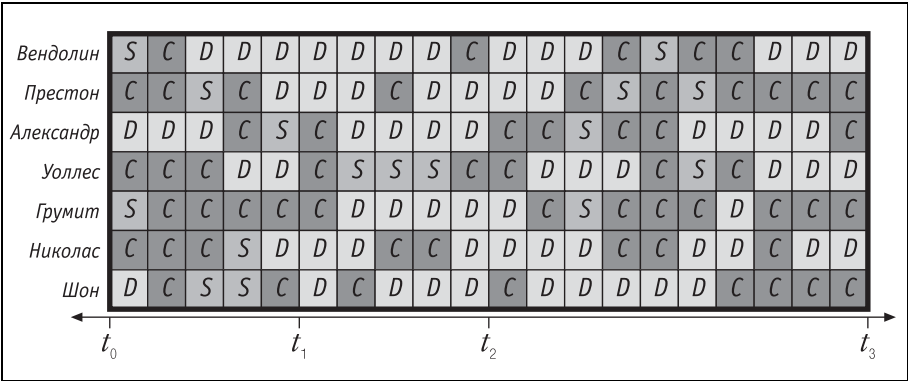


Рис. 3.5. Сбор данных в неверно определенной по обоим измерениям области полностью скрывает источник проблемы пользователя Уоллеса на интервале $[t_1, t_2]$

На основании данных табл. 3.2 невозможно сделать вывод о том, что проблема пользователя Уоллеса связана с расходом ресурса S. Более того, предположение о том, что проблема Уоллеса заключается в ресурсе S, было бы безответственным.

Таблица 3.2. Профиль ресурсов всей системы на интервале $[t_0, t_3]$

Ресурс	Время использования	Доля в общем времени
D	66	47,1%
C	58	41,4%
S	16	11,4%
Всего	140	100,0%

К сожалению, разобранный здесь порочный метод сбора данных применяется по умолчанию пакетом *Statspack*, парой сценариев *utlbstat.sql* и *utlestat.sql*, а также практически всеми инструментами исследования производительности Oracle, созданными между 1980 и 2000 годами. Во всех наиболее провальных проектах повышения производительности, в которых мне довелось участвовать, основной причиной неудачи чаще всего был именно такой способ сбора данных.

Решение описанной проблемы требует корректировки сбора данных по обоим измерениям. Недостаточно исправить ошибку в одном измерении. Посмотрите, например, на результаты сбора данных, показанные на рис. 3.6. Здесь временная область выбрана правильно, но область операций все еще слишком велика. Соответствующий профиль ресурсов приведен в табл. 3.3. Не забывайте, вам известна истинная причина проблем пользователя Уоллеса, и она заключается в ресурсах S и C. Но данные, полученные для системы в целом, недвусмысленно «свидетельствуют» об обратном, несмотря на то, что были собраны в правильном временном интервале.

Наконец, рассмотрим результаты сбора данных в правильной области операций, но в неправильной временной области, как показано на рис. 3.7. В табл. 3.4 приведен соответствующий профиль ресурсов. И здесь даже опытный специалист по производительности, основываясь

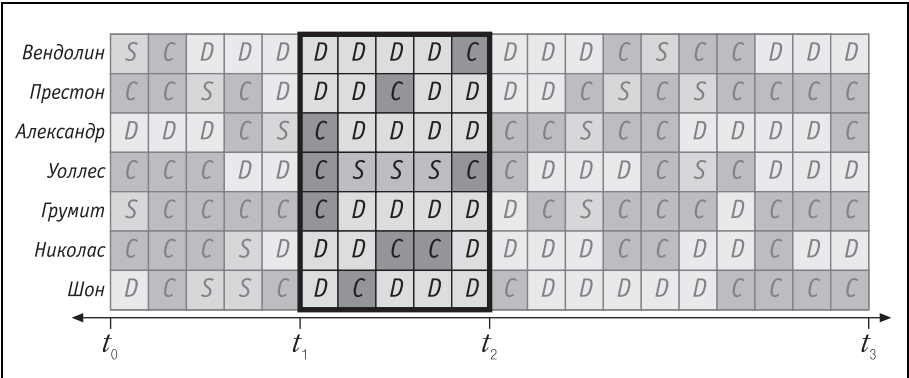


Рис. 3.6. Данные, собранные в неправильной области операций, также не позволяют судить о природе проблемы пользователя Уоллеса, несмотря на то, что были собраны в соответствующем временном интервале

на представленных данных, не справится с задачей диагностики. Проблема пользователя Уоллеса связана с ресурсами S и C, но данные табл. 3.4 не позволяют сделать такой вывод.

Таблица 3.3. Профиль ресурсов системы в целом на интервале $[t_1, t_2]$

Ресурс	Время использования	Доля в общем времени
D	23	65,7%
C	9	25,7%
S	3	8,6%
Всего	35	100,0%

Таблица 3.4. Профиль ресурсов операции пользователя Уоллеса на интервале $[t_0, t_3]$

Ресурс	Время использования	Доля в общем времени
D	8	40,0%
C	8	40,0%
S	4	20,0%
Всего	20	100,0%

На основании этих простых примеров хорошо видно, почему правильная организация сбора диагностических данных жизненно важна для проекта повышения производительности. Примеры также раскрывают равнозначность двух измерений, на основании которых можно судить о применимости имеющихся диагностических данных:

Надежная диагностика проблемы невозможна до тех пор, пока не будут получены данные о времени отклика для строго определенных временной области и области операций.

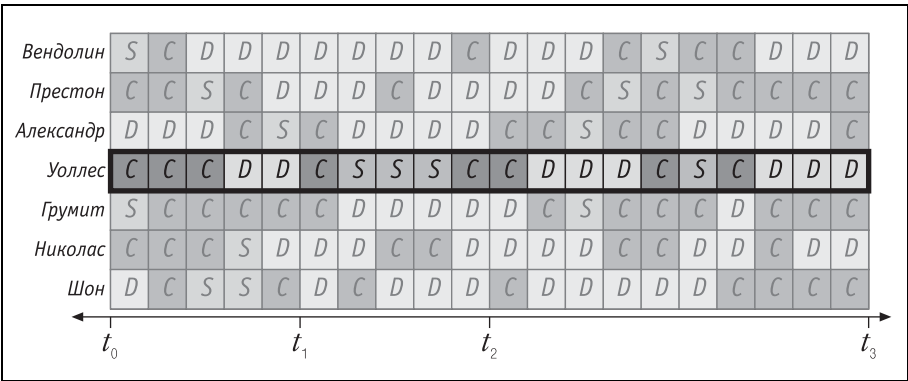


Рис. 3.7. Данные, собранные в неправильной временной области, также не позволяют судить о природе проблемы пользователя Уоллеса, несмотря на то, что были собраны в правильной области операций

Протяженные во времени пользовательские операции

Целесообразно ли накапливать диагностические данные производительности на всем протяжении пользовательской операции, которая выполняется действительно долго? К примеру, операция, выполнявшаяся на прошлой неделе за десять минут, сегодня висит уже четвертый час, и вы подумываете, не пора ли ее отменить. Надо ли перезапускать задание для получения диагностики? Мне приходилось слышать о пакетных заданиях, выполнявшихся по несколько *дней*, прежде чем потерявшие терпение пользователи снимали их.¹ Действительно ли необходимо собирать диагностические данные для всего задания целиком?

Ответ – нет. Конечно, сбор диагностики лишь на части проблемного интервала представляется нарушением правила выбора временной области, но в некоторых обстоятельствах такие данные могут быть действительно полезными. Например:

- Если пользовательская операция должна выполняться за n минут, то данные, собранные в течение $n+m$ минут, будут содержать не относящуюся к делу информацию как минимум за m минут. Например, если предполагается, что задание выполнится за 10 минут, данные, собранные в течение 25 минут будут содержать не менее 15 минут не относящейся к рассматриваемой операции нагрузки.
- Если пользовательская операция состоит из последовательности повторяющихся действий, диагностические данные, полученные только для нескольких из них, дадут представление о расходах ресурсов для всей операции благодаря ее однородности.²

В главе 6 обсуждаются некоторые ошибки, которые могут возникнуть в случае начала процесса сбора данных в середине операции над базой данных. Но во многих случаях сбор данных на частичных интервалах может оказаться весьма полезным.

«Слишком много данных» – на самом деле их недостаточно

Может показаться, что проблемы с данными, приведенными в табл. 3.2, 3.3 и 3.4, возникли из-за того, что было собрано «слишком много дан-

¹ В некоторых из этих случаев я мог бы представить доказательства того, что всей нашей жизни не хватило бы на то, чтобы дожидаться окончания выполнения заданий.

² Хотя в этом разделе речь идет о допустимости или недопустимости *сужения* временной области, первый пример возвращает нас к вопросу о расширении временной области. Он напоминает, что расширение временной области сбора диагностических данных – это явная ошибка, нарушение правила выбора временной области, последствия которого описаны выше. Второй же пример показывает, что сужение временной области вполне допустимо в некоторых случаях. По-видимому, эти примеры должны противопоставляться друг другу, хотя явно автор этого не делает. – *Примеч. науч. ред.*

ных». Однако дело не в том, какие данные были *собраны*, а в том, как они были *агрегированы*. Посмотрите еще раз на рис. 3.5. Здесь вполне достаточно информации для построения профиля ресурсов в правильной области. Проблема с данными в табл. 3.2 возникла вследствие неправильного обобщения данных из рис. 3.5. То же можно сказать и о рис. 3.6, 3.7 и соответствующих им профилях ресурсов. Дело не в избытке данных, а в том, что они некорректно агрегированы в профилях ресурсов.

Неправильное агрегирование данных особенно сильно сказывается на проектах, в которых в качестве источника диагностических данных выступают запросы к фиксированным представлениям Oracle V\$. Эти представления по природе своей способны предоставить только совокупные данные либо для экземпляра с момента его запуска, либо для сеанса с момента установки соединения. Например, используя представления V\$SYSSTAT или V\$SYSTEM_EVENT, вы *гарантированно* получите ошибки в представлении данных, аналогичные показанным в табл. 3.2 и 3.3. Как бы аккуратно вы ни применяли V\$SESSTAT и V\$SESSION_EVENT, вы не избежите ошибок в определении временного интервала, проиллюстрированных данными в табл. 3.4 (в этом можно убедиться, поэкспериментировав с программой *vproof* из главы 8).

При должном внимании к временной области данных представления V\$SESSTAT и V\$SESSION_EVENT могут дать общее представление о причинах задержек при выполнении пользовательских операций. Однако в них отсутствуют данные, необходимые для более детального анализа. Как быть, если, к примеру, предварительный анализ показал, что рассматриваемая операция большую часть времени проводит в ожидании события освобождения блокировки latch free? Тут потребуются данные из V\$LATCH (и возможно, V\$LATCH_CHILDREN) за тот же период времени. Но, получив их, вы заметите, что ни в одном из представлений нет идентификатора сеанса, в силу чего сбор информации о защелках в корректной области операций оказывается невозможным.

Невозможность получения подробных данных из представлений V\$ представляет собой серьезную проблему. И корень зла отнюдь не в представлении V\$LATCH. Что, если бы большую часть времени занимало использование ЦПУ? Тогда вам понадобились бы данные, как минимум, из представления V\$SQL для заданной операции в определенном интервале времени. А если бы время уходило на ожидание db file scattered read? Тогда потребовались бы аналогичные данные из V\$FILESTAT. А если проблема в ожидании buffer busy waits? Тогда надо обратиться к V\$WAITSTAT. В Oracle9i имеется около 300 событий, требующих для анализа подробных данных из нескольких десятков представлений V\$. Даже если бы вам удалось получить из них данные для строго определенных моментов времени (в правильной временной области), вы все равно получите агрегированные значения, страдающие отсутствием детализации, которую можно получить из других источников.

К счастью, есть как минимум три пути получения нужных нам подробных данных. Первый не очень хорош. Второй обойдется дорого, но у вас, возможно, уже есть все необходимое. Третий доступен по цене книги, которую вы держите в руках. Подробности – в следующем разделе.

Источники диагностических данных в Oracle

Имеются по меньшей мере три различных способа получения данных о времени выполнения операций в Oracle:

- SQL-запросы к фиксированным представлениям Oracle (фиксированные представления содержат в именах префиксы V\$, GV\$ или X\$).
- Прямой опрос сегментов разделяемой памяти Oracle, дающий ту же информацию, что и представления V\$ (т. е. доступ к данным представлений V\$ без привлечения механизма запросов SQL).
- Активирование в Oracle функции расширенной трассировки SQL, сохраняющей в трассировочном файле все хронометрические сведения о ходе выполнения команд для сессии Oracle.

На первый взгляд, данные представлений V\$ имеют мало общего с данными файла трассировки SQL, но на самом деле это разные формы представления одной и той же информации. Об общем источнике исходных данных рассказывается в главе 7.

Посвятив полных три года изучению метода R и его требований к данным, я пришел к выводу о наличии у перечисленных трех способов следующих достоинств:

Запросы SQL к представлениям V\$

SQL-запросы к представлениям V\$ – превосходный источник данных о *потреблении ресурсов* (т. е. о количестве обращений к различным ресурсам). Дополнительную информацию можно почерпнуть из примеров Тома Кайта (Tom Kyte), доступных по адресу <http://asktom.oracle.com/~tkyte/runstats.html>. Представления V\$ особенно полезны на этапе разработки приложений. Доступ к *хронометрическим* данным из представлений V\$ при помощи SQL – это относительно простой способ начать работу с этими данными. Однако по ряду причин, изложенных в главе 8, хронометрические данные из этого источника оказываются ненадежными при решении проблем некоторых типов. Возможности, предоставляемые хронометрическими данными из фиксированных представлений, намного скромнее, чем в случае применения двух других подходов.



Одно из фиксированных представлений, X\$TRACE, предоставляет средство доступа при помощи SQL к данным расширенной трассировки SQL. Однако эта возможность в настоящее время не документирована, не поддерживается и работает нестабильно. Если в будущем корпорация Oracle усовершенствует реализацию X\$TRACE, мой пессимистический взгляд на возможность получе-

ния детальных данных из фиксированных представлений может потерять актуальность. Но в Oracle 9.2 данная возможность не готова к промышленному использованию.

Получение данных V\$ непосредственно из разделяемой памяти Oracle

Если у вас уже есть инструмент, позволяющий получать диагностические данные в заданных временной области и области операций, то выборка данных с высокой частотой непосредственно из разделяемой памяти, возможно, будет для вас наилучшим выбором. Высокочастотная выборка позволяет получить данные, весьма полезные в решении разнообразных проблем производительности. Однако доступ к разделяемой памяти и последующее хранение гигантских массивов данных требуют или больших трудозатрат, или расходов на специальный инструментарий. Такие инструменты не дешевы.

Использование возможностей расширенной трассировки SQL

Расширенная трассировка SQL также способна предоставить данные для надежной диагностики, не требуя при этом тех трудовых или финансовых ресурсов, которые характерны для способа с опросом памяти. Принципиальный недостаток расширенной трассировки в том, что она позволяет получить диагностические данные только для тех операций, о которых заранее известно, что они выполняются в неоптимальных условиях. Возможно, вам придется набраться терпения, если проблема производительности проявляется лишь время от времени. Проведя опрос, можно собирать диагностические данные в любой области для любой выполнявшейся пользовательской операции, но *только* при том условии, что в хранение первичных данных были вложены достаточные средства. Расширенная трассировка – эффективная и недорогая альтернатива высокочастотного опроса.

В табл. 3.5 для удобства восприятия сделана попытка дать количественное представление вышеизложенных соображений.



Этот способ создания иллюзии, что арифметика позволяет манипулировать мнением читателей, позаимствован мной из журнала *Car & Driver*.

Я уверен, что расширенная трассировка SQL – лучший из описанных в этой главе трех источников данных для целей оптимизации производительности. За последние три года мы с коллегами из *hotsos.com* более тысячи раз помогали диагностировать и решать проблемы производительности, пользуясь *только* данными расширенной трассировки SQL. Наш практический опыт показывает, что при надлежащем применении расширенная трассировка представляет собой *исключительно* надежный инструмент диагностики.

Таблица 3.5. Мнение автора о сравнительных достоинствах трех источников хронометрических данных в Oracle. Оценки выставлены по десятибалльной шкале, более высокая оценка соответствует лучшему значению указанной характеристики

Характеристика	Источник диагностических данных		
	Фиксированные представления V\$	Разделяемая память Oracle	Данные расширенной трассировки SQL
Простота получения текущих результатов	9	1	8
Простота хранения полученных данных	7	1	10
Простота анализа полученных данных	8	1	7
Минимальное влияние на ядро Oracle	2	10	7
Минимальное влияние на остальные ресурсы	8	4	7
Пригодность для детального анализа исторических данных	1	8	7
Стоимость вспомогательного инструментария	9	1	6
Надежность диагностики	3	9	9
Всего	45	35	61

Дополнительная информация

Главы 5 и 6 содержат сведения, которые понадобятся вам для начала работы с расширенной трассировкой SQL, когда вы будете к этому готовы. В главе 8 дан ряд инструкций по работе в Oracle с представлениями V\$. Способы получения данных непосредственно из Oracle SGA в этой книге не рассматриваются. Мало кто из тех, кто умеет обращаться с SGA, хочет обсуждать эту тему публично. Кайл Хейли (Kyle Hailey) – один из специалистов, пожелавших описать данный процесс [Hailey (2002)]. С технической точки зрения, прямой доступ к Oracle SGA представляется мне безупречным. Однако, с точки зрения практики и экономики оптимизации, я отдаю пальму первенства методу расширенной трассировки SQL.

4

Выбор пути решения задачи

Итак, корректные диагностические данные для выбранной операции собраны, и можно приступить к выбору путей решения проблемы:

Выполните действия по оптимизации, которые дадут наибольший эффект для бизнеса. Если даже самые результативные мероприятия не дадут нужного эффекта, отложите оптимизацию производительности до лучших времен.

Выполнение этой задачи требует, чтобы вы мыслили категориями, которые связаны с двумя совершенно различными сферами деятельности. Во-первых, есть чисто технический аспект, о котором все знают: необходимо выяснить, какие изменения приведут к повышению производительности. А во-вторых, есть работа, которой вы, возможно, *не ожидали*, – это анализ финансовых последствий предполагаемых действий. Это та часть работы, которую многие аналитики по производительности выполняют не очень хорошо (в большинстве своем потому, что даже не пытаются этого делать). В этом деле практически неприменимы обычные методы «настройки Oracle». Но тот, кто пренебрегает прогнозированием финансовых последствий предполагаемых изменений, теряет возможность принимать обоснованные решения по повышению производительности в соответствии с интересами бизнеса.

Новый стандарт обслуживания клиентов

Анализ производительности Oracle во многом пока еще находится в стадии становления. Эра времени отклика (точнее, основанных на нем методов), начало которой было возведено Хуаном Лоайза (Juan Loaiza), Родериком Макалаком (Roderick Macalac), Аньо Колком (Anjo Kolk) и Шари Ямагучи (Shari Yamaguchi), – это, бесспорно, большой шаг вперед. Но технические достижения составляют лишь часть необходимого прогресса в этой области. Тот стандарт качества, который мы предлагаем нашим клиентам (пользователям, руководителям – тем,

кого мы консультируем...), дает еще одну возможность для неограниченного роста.

На мой взгляд, наши отношения с клиентами должны быть такими, как описано в изданном Стэнфордским университетом руководстве по работе с людьми «Human Subjects Manual». Приведенный ниже фрагмент взят из главы под названием «Informed Consent» (Информированное согласие) [Stanford (2001)]:

Добровольное согласие человека совершенно необходимо. Это означает, что соглашающийся *должен*:

- Иметь законные полномочия на дачу согласия;
- Находиться в положении, допускающем свободу выбора, в отсутствии элементов силы, обмана, принуждения, помех или других форм ограничения или насилия, а также
- В достаточной степени знать и понимать обсуждаемую тему и сопутствующие вопросы, чтобы быть в состоянии принять обоснованное и информированное решение.

Последний пункт подразумевает, что человеку известно все из перечисленного ниже:

- a) суть эксперимента;
- b) продолжительность эксперимента;
- c) цель эксперимента;
- d) средства и методы, задействованные в эксперименте;
- e) все вероятные неудобства и риски;
- f) влияние эксперимента на здоровье или личность испытуемого, возможное в результате его участия в эксперименте.

Полагаю, идея *информированного согласия* как нельзя лучше подходит к нашей профессии. К счастью, очень немногие специалисты по Oracle испытывают тяжесть принятия решений, связанных с вопросами жизни и смерти, с которыми медики сталкиваются в своей практике ежедневно. Но многие из нас регулярно привлекаются к решению чисто технических, понятных только специалистам вопросов, исключительно важных для нуждающихся в нашей помощи клиентов. Принцип информированного согласия – это своего рода «билль о правах», позволяющий клиенту противостоять давлению стремительно развивающихся технологий.

Но высокая планка информированного согласия практически недостижима для специалистов, применяющих к настройке Oracle подход, основанный на общепринятом методе C. Технология последнего просто не предоставляет информации, необходимой для того, чтобы предсказать поведение проекта (или даже его небольшой части). Нельзя сообщить клиентам то, чего не знаешь сам. Одно из важнейших преимуществ метода R в том, что он предоставляет технические средства, позволяющие применять стандарт информированного согласия в нашем профессиональном общении с клиентами.

Выбор экономически оптимального пути повышения производительности

Вы, наверное, уже заметили, что в основе метода R лежит внимательный и аккуратный *выбор*. Это относится и к тому, какие мероприятия предпримет аналитик, основываясь на полученных данных. Его действия состоят из трех этапов выбора:

1. Во-первых, он *выбирает* пользовательские операции, повышение производительности которых даст бизнесу наибольшие экономические выгоды (глава 2).
2. Затем он *выбирает* соответствующую временную область и область операций для сбора диагностических данных (глава 3).
3. И наконец, он *выбирает* способ повышения производительности, наиболее эффективный с точки зрения чистой прибыли (глава 4).

Как уже говорилось в главе 1, наилучшим способом представления данных для третьего этапа выбора является *профиль ресурсов*. Однако профиль ресурсов – это лишь часть необходимой информации. После того, как вы с его помощью определите, на что уходит время при выполнении пользовательской операции, нужно будет на основе диагностических данных определить, *почему* данный компонент времени отклика так велик. К счастью, если область сбора данных в расширенной трассировке SQL определена правильно, то у вас уже есть вся необходимая информация. Если же таких данных еще нет, то, вероятно, следует вернуться к главе 3.

По завершении сбора диагностических данных в соответствующей временной области и области операций для всех выбранных (от одной до пяти) пользовательских операций, на основании технических и финансовых соображений осуществляется выбор действий, оптимальных для бизнеса, по следующему алгоритму:

1. Для каждой выбранной пользовательской операции:
 - Представьте диагностические данные в формате, облегчающем сопоставление источников проблем составляющим времени отклика.
 - Оцените экономический эффект нескольких предпочтительных вариантов повышения производительности. Составьте список вариантов действий и соответствующих им финансовых показателей.
2. Из списка возможных действий выберите те, которые дадут наибольший экономический эффект для бизнеса.

Эти шаги позволят определить, какие способы повышения производительности будут наиболее выгодны бизнесу. Следующее, что вы должны сделать, – отложить эту книгу и выполнить свой план повышения производительности.



Метод R совершенно не похож на обычный способ «настройки». Этот метод не предполагает поиска подозрительных значений в списках коэффициентов или событий ожидания. Метод выстраивает приоритеты повышения производительности в соответствии с потребностями бизнеса. Выбором приоритетов проекта по повышению производительности управляет бизнес, а не технология.

Анализ диагностических данных

Во второй части книги содержится вся необходимая информация для выбора действий, основанных на корректно полученных диагностических данных. Как вы увидите, порядок действий в методе R полностью основан на диагностических данных. Следовательно, задача представления данных трассировки в удобном для анализа виде поддается автоматизации. Если вам предстоит проанализировать несколько мегабайт необработанных данных, без автоматизации не обойтись. На момент написания этих строк мне известны три инструмента корпорации Oracle, способные помочь в этом:

tkprof

Утилита *tkprof* предназначена для форматирования трассировочных данных Oracle; она генерирует текстовый файл, содержащий данные о статистике производительности, сгруппированные по командам SQL. Параметры командной строки позволяют указать порядок, в котором команды будут показаны. Утилита была разработана для тестирования производительности отдельных приложений SQL и прекрасно справляется с этой задачей. Версия Oracle9i стала первой, в которой *tkprof* обрабатывает необходимые методу R данные о «событиях ожидания» Oracle. В предыдущих версиях *tkprof* эти данные игнорировались. (В главе 5 рассказывается о важности «событий ожидания» Oracle.) Сведения об утилите *tkprof* содержатся в документе «Performance Tuning Guide and Reference» (Справочное руководство по настройке производительности), который можно найти по адресу <http://technet.oracle.com> и в документах MetaLink 41634.1, 29012.1 и 1012416.6.

trcsmary

Утилита *trcsmary* позиционируется корпорацией Oracle как «не предназначенная для широкого применения». Для разбора трассировочного файла Oracle в ней применяются средства *awk* и *nawk*, формируя выходные данные аналогично утилите *tkprof*. По-видимому, эта утилита была разработана для преодоления некоторых недостатков, свойственных ранним версиям *tkprof*. Информация о *trcsmary* имеется в документе MetaLink 62160.1.

Trace Analyzer

Средство Trace Analyzer представляет собой набор сценариев SQL*Plus и программ на PL/SQL, предназначенный для чтения исходного файла трассировки SQL и загрузки его содержимого в базу данных с последующим формированием подробного отчета. Trace Analyzer позволяет обрабатывать данные о «событиях ожидания» Oracle. Более подробное описание Trace Analyzer имеется в документе MetaLink 224270.1.

Из трех описанных средств самое новое и развитое – Trace Analyzer, но оно же и самое неудобное. Оно работает очень медленно, а на анализ выводимых им данных может уйти не один день, т. к. среди них непорочно много второстепенных.

Мой любимый анализатор трассировочных файлов *Hotsos Profiler*TM является коммерческим продуктом; его основной разработчик – Джефф Холт (Jeff Holt). Причиной создания Hotsos Profiler стало отсутствие на рынке инструмента, позволяющего с достаточной скоростью перейти из состояния «данные собраны» в состояние «проблема решена». Hotsos Profiler всего за несколько секунд преобразует многомегабайтный файл расширенной трассировки SQL в документ HTML, после чего выяснение причины практически любой проблемы производительности требует всего пары щелчков мышью. Выходные данные Hotsos Profiler позволяют определить экономический эффект от наиболее значимых мероприятий по повышению производительности не больше чем за час с момента получения корректного трассировочного файла. Подробнее о Hotsos Profiler можно прочитать на сайте <http://www.hotsos.com>.

После приведения собранных диагностических данных к пригодному для анализа виду надо определить пути повышения производительности выбранной пользовательской операции. На этом этапе действия аналитика состоят из ряда итераций примерно такого вида:

1. По профилю ресурсов определить компоненты времени отклика, обещающие наибольший экономический эффект. Затем найти диагностические данные, указывающие на причину столь заметного вклада этих компонентов в общее время отклика.
2. Сделать предположения о путях наибольшего сокращения доли времени отклика, приходящейся на компоненты, выбранные в п.1. Эти предположения по повышению производительности обычно проверяются на тестовой системе. По результатам тестов оценить ожидаемую выгоду от реализации предложенных идей. Следует рассмотреть достаточное количество идей, чтобы убедиться, что ни одна из возможностей, сулящих большую выгоду от повышения производительности, не упущена.

Технические подробности выполнения этих шагов отложим до третьей части. Остаток данной главы посвящен предсказанию экономической эффективности проекта.

Прогнозирование экономической эффективности проекта

Цель проекта повышения производительности состоит в *экономической оптимизации*. Приступая к проекту повышения производительности, вы реализуете готовность затратить время и материальные ресурсы в экономическую эффективность системы. Успех оптимизации зависит от того, будут ли ваши действия на каждом этапе *информированными* действиями. Другими словами, необходимо *заранее* знать, каких затрат потребует каждый шаг и какую он принесет выгоду.

Следует руководствоваться простым правилом: прежде чем делать инвестиции, необходимо понять, какую они принесут *чистую прибыль*. *Чистая прибыль* от проекта равна разности приведенной стоимости (ПС от англ. *present value* – *PV*) приносимой им прибыли и приведенной стоимости *затрат* на него. Концепция приведенной стоимости достаточно проста:

Сегодняшний доллар дороже доллара завтрашнего в силу того, что, будучи инвестирован сегодня, он немедленно начнет приносить прибыль. [Brealey and Myers (1988), 12]

Формула расчета ПС пригодится вам для приведения будущих денежных потоков к текущим ценам для корректного сравнения инвестиций и ожидаемой от них отдачи в будущем. Прогнозирование экономического эффекта от предлагаемого проекта требует выполнения следующих шагов:

1. Сделать временной прогноз роста прибыльности бизнеса в результате предлагаемых мероприятий.
2. Сделать прогноз затрат на предлагаемые мероприятия с указанием времени.
3. Рассчитать приведенную стоимость каждой составляющей движения денежных потоков по формуле

$$PV = \frac{C}{1+r},$$

где C – размер средств в будущем, r – норма прибыли, принимаемая при приеме отложенных платежей. В программе Microsoft Excel имеется встроенная функция ПС, выполняющая указанные вычисления.

Например, если ожидаемая норма прибыли составляет $r = 0,07$ (приблизительно такова средняя норма прибыли на фондовом рын-

ке США за последние 90 лет), то приведенная стоимость доллара, полученного год спустя, составит всего \$0,934579. Другими словами, при норме прибыли 7% получение одного доллара через год эквивалентно получению 93,4579 центов сегодня. Причина в том, что если вы сегодня вложите сумму \$0,934579 и получите на нее 7% прибыли в течение года, она увеличится до \$1.

4. Рассчитать экономический эффект предложенных мероприятий, сложив ПС всех составляющих прибыли и вычтя ПС всех составляющих затрат.

Тот, кому известны приведенные стоимости всех предполагаемых мероприятий (аналитик или сотрудник проекта, ответственный за принятие решений), без труда сравнит их по ожидаемым финансовым показателям.

Прогнозирование прибыльности проекта

Задача предсказания прибыли, приносимой бизнесу в результате предлагаемого повышения производительности, значительно упрощается при наличии профиля ресурсов. Я с болью вспоминаю те времена, когда в отсутствие метода R единственным способом прогнозирования прибыли была экстраполяция, выходявшая далеко за пределы математически обоснованной. Например, на моей памяти профессиональные аналитики занимались такими предсказаниями:

«Я уменьшил количество экстенгов вашей таблицы заказов с 3482 до 8. Можно ожидать, что такое улучшение приведет к повышению производительности при вводе заказов в целом на 30%».

«Увеличение коэффициента попаданий с 95% до 99% может привести к росту производительности более чем на 400%».

У меня есть *книги*, в которых написана подобная чепуха. Вы когда-нибудь интересовались, откуда в этих утверждениях появляются такие числа, как «30» или «400»? Если эти значения взяты не из правильно составленного профиля ресурсов, можете быть уверены, что их источник наверняка находится там же, откуда берут свои необоснованные оценки те консультанты, чьи гонорары определяются их способностью вселять надежду в заказчика.



Я всегда испытываю определенное злорадство, встречая в высказываниях о прибыли фразы типа «Ожидается, что ...». В данной ситуации страдательный залог служит грамматическим инструментом, который авторы используют для того, чтобы впоследствии иметь возможность отречься от собственных оценок.

Надеюсь, эта книга поможет в принятии более эффективных решений, направленных на повышение производительности, как вам, так и работающим с вами специалистам.

Подсчет прибыли

Прогнозирование прибыли от предложенных действий по повышению производительности состоит из двух шагов:

1. Оценить уменьшение времени отклика для выбранной пользовательской операции. Как это сделать, рассказывается в третьей части книги.
2. Оценить в денежном выражении положительный эффект для бизнеса, достигаемый этим уменьшением времени отклика. С точки зрения арифметики тут все просто. Результат получается из трех величин:

Количество секунд, на которое можно сократить время выполнения операции, умноженное на

Количество выполнений данной операции за период, для которого рассчитывается прибыль, умноженное на

Стоимость одной секунды ожидания для бизнеса

Первый шаг достаточно прост. Как вы увидите в части III, профиль ресурсов, полученный на производственной системе, содержит всю необходимую *предыдущую* статистику, а профиль, полученный на тестовой системе, поможет предсказать *будущую*. Выполнение второго шага тоже обычно не вызывает затруднений, пока не приходит время оп-ределить, во что обходится бизнесу одна секунда времени ожидания ответа. Часто никто в компании просто не знает этой стоимости. В некоторых знают. Например, могут быть известны показатели, аналогичные приведенным ниже:

Повышение скорости процесса авторизации возвращаемых материалов с 5 дней до 2 часов необходимо для предотвращения потери нашего крупнейшего розничного клиента. На чаше весов лежит годовая сумма продаж более чем на 100 000 000 долларов США.

Уменьшение времени оформления счетов клиентам с 4 дней до 1 дня сократит требования к оборотным средствам на 325 000 евро.

Повышение производительности работы формы, через которую наши пользователи оплачивают счета, поднимет боевой дух бухгалтерии, снизит текучесть кадров и выплаты за сверхурочные работы, что в целом приведет к снижению годовых затрат более чем на 60 000 фунтов.

Увеличение количества заказов, обрабатываемых системой в час пик, с 40 штук в час (как в настоящее время) до 100 увеличит годовые доходы от продаж более чем на 100 000 000 иен.

Если компания не имеет таких конкретных соотношений времени и стоимости, обычно бывает достаточно выразить выгоду от проекта в терминах экономии времени отклика за некоторый разумный период. Тогда прогноз экономического эффекта может выглядеть так:

Я полагаю, что Проект А будет выполнен в течение двух недель и обойдется в 10 000 долларов США. В результате время ожидания для конечных пользователей сократится на 128 часов в неделю.

Если же величина «доллары в секунду» известна, рассчитанный экономический эффект позволяет более точно обосновать решение:

Я полагаю, что Проект А будет выполнен в течение двух недель и обойдется в 10 000 долларов. В результате время ожидания для конечных пользователей сократится на 128 часов в неделю, стоимость сэкономленного рабочего времени для компании составит 70 000 долларов США.

Денежное выражение ожидаемой от проекта выгоды часто представляет лишь академический интерес. В наибольшей степени это характерно для проектов, в которых стоимость работ незначительна по сравнению с очевидной привлекательностью их результатов. Например, среди приведенных ниже требований есть как весьма расплывчатые, так и совершенно конкретные. Однако применительно к каждой из ситуаций они имеют вполне достаточный уровень детализации:

Я не знаю, сколько времени должен выполняться этот отчет. Я знаю только, что сейчас он выполняется недопустимо долго. Мы будем рады, если вы сделаете хоть что-нибудь.

Мы не сможем работать с этим приложением, если вы не сумеете уменьшить время отклика с нескольких минут до двух секунд.

Данная транзакция должна завершаться не позднее чем через 1 секунду в 95% случаев.

Данная транзакция должна занимать менее 13 миллисекунд на незагруженной системе.

О том, как и почему на основании неполных тестовых данных приложения формулируются столь детальные требования, подобные последним двум из приведенных здесь, будет рассказано в главе 9. При наличии столь подробных спецификаций спонсор проекта, как правило, должен определить ценность подобных требований для бизнеса. Несколько раз мне доводилось видеть, как проект выживал только благодаря тому, что в процессе работы заказчик, обнаружив, сколько будет стоить выполнение первоначальных требований, допускал ослабление некоторых из них.

Если прибыль оценить невозможно

Как правило, невозможность оценить стоимость секунды сэкономленного времени отклика не вызывает проблем. Если вы предлагаете *недорогое* решение, заказчик, скорее всего, не станет требовать финансового обоснования. В этом случае единственный, кто пострадает от недостатка данных, – это вы, т. к. лишитесь возможности количественно оценить финансовый результат хорошо сделанной работы. Если предлагаемое решение требует серьезных затрат, то неспособность дать разумную оценку экономии от планируемого сокращения времени отклика приведет к одному из трех итогов:

- Спонсор проекта оценит финансовую выгоду и примет экономически обоснованное решение о его дальнейшем развитии.

Будьте внимательны, подсчитывая проценты

Формат профиля ресурсов позволяет легко оперировать процентными отношениями. Если мероприятия по повышению производительности на x процентов сократят время выполнения операции, отнимающей у процентов общего времени отклика, то в целом время отклика сократится на x процентов. Например, если на 50% уменьшить время операции, составляющей 80% полного времени отклика, результирующий выигрыш составит 40% ($0,5 \times 0,8 = 0,4$)

Но будьте осторожны, принимая решения, основанные на процентных соотношениях: они могут стать причиной ошибки. К примеру, что лучше: сократить время выполнения операции А на 20% или операции В – на 90%? Правильный ответ на этот вопрос заключается в том, что на него нельзя ответить, не располагая сведениями о продолжительности А и В. Если вы ответили А или В, значит, вы попали в процентную ловушку.

- Проекту будет позволено двигаться дальше, несмотря на отсутствие экономического обоснования, что может быть ошибкой, а может и не быть – этого нельзя узнать до его окончания.
- Проект может быть просто свернут на том основании, что «он стоит слишком дорого». Лучший из известных мне способов борьбы с угрозой закрытия проекта – обоснованный расчет экономического эффекта в денежном выражении. Если проект не может устоять перед строгим финансовым анализом, его закрытие почти наверняка будет правильным решением.

Задача определения финансовой выгоды сводится к получению данных, основываясь на которых, вы или спонсор проекта можете принять экономически обоснованное решение о судьбе проекта. Большого здесь не требуется. Не рискуйте своим авторитетом, делая необоснованные прогнозы финансового успеха, – это может погубить проект.

Прогнозирование затрат на проект

Предсказание экономического эффекта проекта требует определения как приносимой им прибыли, так и затрат на его выполнение. Прогнозирование *затрат* – более простая из этих двух задач, т. к. в вашем распоряжении ориентированная на это обширная инфраструктура. Множество людей способны достаточно корректно оценить объем необходимых инвестиций. Консультанты проходят курс по планированию развития, в ходе которого обучаются оценивать стоимость проектов. Консультанты, не способные с достаточной точностью оценить стоимость работ, уходят из бизнеса (и обычно вынуждены работать на тех консультантов, которые умеют это).

Для разумной оценки стоимости проекта, безусловно, потребуется понимание задач, которые требуется решить, чтобы выполнить предложенные мероприятия по улучшению производительности. В части III

представлены обширные сведения, призванные помочь вам лучше понять требования к этим работам и материалам.

Прогнозирование рисков проекта

Даже для лучших проектов прогнозы обычно не оправдываются. Фактически невозможно предсказать точную прибыль или стоимость для любой сложной операции. Поэтому при оценке прибыльности проекта учитываются возможные риски. Пока проект не завершен, его стоимость и прибыль представляют собой случайные переменные. Случайная переменная – это понятие, используемое в математике для описания результата процесса, который невозможно предсказать точно, но для которого существуют какие-то понятные ограничения. У каждой переменной есть предполагаемое значение, но при этом необходимо понимать, что каждая из них характеризуется изменчивостью (вариацией). Финансовые аналитики оперируют термином «риски», обозначая с его помощью то, что статистики называют «вариацией» [Kachigan (1986); Bodie, et al. (1989)].

Для того чтобы вычислить риск реализованного в прошлом проекта, необходимо иметь данные для достаточного количества проектов с тем, чтобы прийти к статистически обоснованным выводам. Для того чтобы понять, как можно оценить риск проекта, давайте представим себе, что тридцать различных проектных групп работают над идентичными проектами повышения производительности. Практически исключено, что для всех тридцати групп затраты на проект и полученные прибыли совпадут. Если бы можно было провести этот эксперимент в реальной жизни, то при таком количестве групп-участников нам вполне бы хватило данных для определения статистической модели затрат и прибылей. Если составить диаграмму, например, затрат на проект для всех

Что такое «Ограничение проекта по стоимости»?

Многие на вопрос «Каково ограничение проекта по стоимости?» поддаются искушению дать ответ, выразив его в цифрах, например «десять тысяч долларов». Но хороший финансист, скорее всего, скажет, что реальное ограничение зависит от ценности усовершенствования для бизнеса и нормы прибыли от инвестиций, которой требует бизнес.

Представим, например, что предполагаемой лимит бюджета составляет \$10 000. Пусть норма прибыли от инвестиций составляет 35%, а предполагаемое повышение производительности принесет в этом году компании дополнительную сумму в 1 000 000 долларов США. Тогда если компания действительно поверит в оценку прибыли в 1 000 000 долларов, она должна быть готова инвестировать в проект до 740 741 доллара. Анализ сводится к тому, чтобы понять, принесет ли сумма в 740 741 доллар, которую компании придется достать из внебюджетных источников, наибольшую отдачу при вложении в этот проект или же в какой-то другой.

тридцати его реализаций, то получится нечто, напоминающее диаграмму на рис. 4.1.

Стоимости, представленные на рис. 4.1, отражают следующую тенденцию: наибольшее скопление наблюдается вокруг величины в 15 000 (будем считать, что речь идет о единицах местной валюты). Если бы суммы затрат тяготели к правой части диаграммы и были бы распределены по более широкому диапазону, как это показано на рис. 4.2, то это означало бы, что риск проекта выше и велика вероятность того, что предполагаемая стоимость будет превышена.

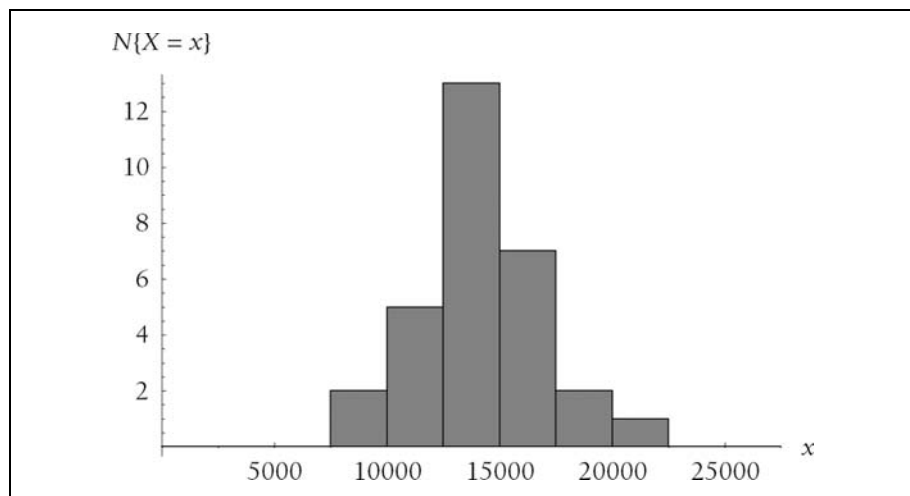


Рис. 4.1. Тридцать реализаций одного проекта с различными стоимостями

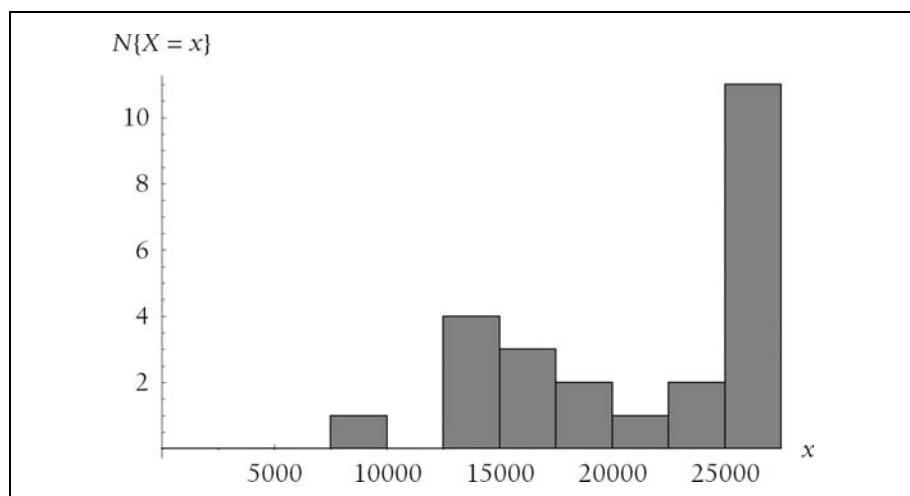


Рис. 4.2. Большое колебание стоимости данного проекта указывает на повышенный риск

Если вы можете позволить себе такую роскошь, как данные о стоимости и прибыли большого количества проектов, в точности повторяющих реализуемый вами, то, конечно же, гораздо лучше сможете предсказать реальную стоимость и прибыль проекта. Но такие данные доступны только в компаниях, которые постоянно осуществляют однотипные проекты (и надо сказать, что такие компании, как правило, очень хорошо предсказывают их стоимость и ожидаемую прибыль).

Возможность перерасхода средств на проект, как и недополучения прибыли, может определяться несколькими факторами. Основным таким фактором является *опыт*. Понятно, что выполнение какого-то действия, которое раньше никто не осуществлял, связано с повышенным риском. Но даже реализация простого проекта группой, которая раньше не занималась подобными вещами, может привести к неожиданным результатам. Однако опыт – это еще не все. Один из моих любимых уроков, извлеченных из спортивной жизни [Pelz (2000)]:

Тренировки приводят к *стабильности*. Только *безупречные тренировки* приводят к *совершенству*.

Это научно сформулированная мысль о том, что если кто-то сделал что-то десять тысяч раз, это еще не означает, что он делает это хорошо.¹ Постоянство успеха при осуществлении похожих проектов – вот фактор значительного снижения риска.

К прогнозированию можно подходить со всей имеющейся изобретательностью и фантазией. Но не теряйте из виду цель вашей деятельности по прогнозированию рисков. На самом деле нужна возможность предсказывать две вещи, которые необходимо знать клиенту и которые взяты непосредственно из текста об «информированном согласии», приведенного в начале главы:

- а) все вероятные неудобства и риски;
- б) влияние эксперимента на здоровье или личность испытуемого, возможное в результате его участия в эксперименте.

Невозможно точно спрогнозировать стоимость и прибыль рекомендуемых аналитиком действий. Но предложенные в книге методики помогут очень быстро усовершенствовать его пророческие способности.

¹ Приведем еще одно родственное замечание: соответствие некоторому стандарту качества гарантирует только *однородное* качество, но совсем не обязательно – *высокое*.

Часть II. Справочная информация

5

Интерпретация данных расширенной трассировки SQL

Для того чтобы стать хорошим аналитиком по производительности, необходимо понимать тот язык, на котором система сообщает о своей производительности. К сожалению, в течение более десяти лет временная статистика Oracle относилась к наиболее недооцененным и недопонятым областям ядра Oracle. Для того чтобы научиться обращаться с инструментами анализа времени отклика, предлагаемыми ядром Oracle, надо понять, каким образом ядро Oracle взаимодействует с операционной системой. Речь идет об операционной системе, которая выделяет ресурсы для процесса ядра Oracle, именно она и предоставляет ту временную статистику, которую Oracle использует для описания собственной производительности.

Знакомство с файлом трассировки

Мне кажется, что лучше всего начать изучение данных о функционировании Oracle с содержимого файлов *расширенной трассировки SQL*. Данные о трассировке SQL предоставляют непревзойденный учебный и диагностический материал, т. к. представляют собой последовательно записанную историю действий ядра Oracle, предпринятых в ответ на запросы приложения к базе данных.

Функция трассировки SQL входит в состав ядра Oracle начиная с версии 6, которая наверняка старше любой из используемых вами версий. В 1992 году, с выпуском версии 7.0.12, корпорация Oracle значительно усовершенствовала трассировку SQL, дополнив ее информацией о продолжительности не потребляющих времени процессора инструкций, выполняемых ядром Oracle.

Начнем наше изучение данных о времени отклика с запроса «Hello, world». В примере 5.1 приведен один из простейших сеансов SQL*Plus, который можно себе представить. Сеанс активирует для себя механизм расширенной трассировки SQL. Затем он запрашивает из базы данных строку «Hello, world; today is sysdate» и завершается.

*Пример 5.1. Сценарий сеанса SQL*Plus, генерирующего данные расширенной трассировки SQL для простого запроса*

```
alter session set max_dump_file_size=unlimited;
alter session set timed_statistics=true;
alter session set events '10046 trace name context forever, level 12';
select 'Hello, world; today is '||sysdate from dual;
exit;
```

Файл трассировки из примера 5.1 приводит последовательность действий, выполненных ядром Oracle от имени данного сеанса. Если вы привыкли смотреть на данные трассировки SQL только через объектив утилиты tkprof, то вас ждет приятный сюрприз. Научившись обращаться с необработанными данными расширенной трассировки, вы сможете диагностировать большее количество типов ошибок производительности, чем это позволяет tkprof. Многие аналитики, научившись свободно обращаться с необработанными данными трассировки, с удивлением обнаруживают множество недостатков в tkprof.

*Пример 5.2. Необработанные данные расширенной трассировки SQL, сформированные сеансом SQL*Plus, из примера 5.1*

```
/u01/oradata/admin/V901/udump/ora_9178.trc
Oracle9i Enterprise Edition Release 9.0.1.0.0 - Production
With the Partitioning option
JServer Release 9.0.1.0.0 - Production
ORACLE_HOME = /u01/oradata/app/9.0.1
System name: Linux
Node name: research
Release: 2.4.4-4GB
Version: #1 Fri May 18 14:11:12 GMT 2001
Machine: i686
Instance name: V901
Redo thread mounted by this instance: 1
Oracle process number: 9
Unix process pid: 9178, image: oracle@research (TNS V1-V3)

*** SESSION ID:(7.6692) 2002-12-03 10:07:40.051
APPNAME mod='SQL*Plus' mh=3669949024 act='' ah=4029777240
=====
PARSING IN CURSOR #1 len=69 dep=0 uid=5 oct=42 lid=5 tim=1038931660052098
hv=1509700594 ad='50d6d560'
alter session set events '10046 trace name context forever, level 12'
END OF STMT
EXEC #1:c=0,e=1,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=1038931660051673
WAIT #1: nam='SQL*Net message to client' ela= 5 p1=1650815232 p2=1 p3=0
WAIT #1: nam='SQL*Net message from client' ela= 1262 p1=1650815232 p2=1 p3=0
```

```

=====
PARSING IN CURSOR #1 len=51 dep=0 uid=5 oct=3 lid=5 tim=1038931660054075
hv=1716247018 ad='50c551f8'
  select 'Hello, world; today is '||sysdate from dual
  END OF STMT
  PARSE #1:c=0,e=214,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,
                                     tim=1038931660054053

  BINDS #1:
  EXEC #1:c=0,e=124,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,
                                     tim=1038931660054311
  WAIT #1: nam='SQL*Net message to client' ela= 5 p1=1650815232 p2=1 p3=0
  FETCH #1:c=0,e=177,p=0,cr=1,cu=2,mis=0,r=1,dep=0,og=4,
                                     tim=1038931660054596
  WAIT #1: nam='SQL*Net message from client' ela= 499 p1=1650815232 p2=1 p3=0
  FETCH #1:c=0,e=2,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=0,tim=1038931660055374
  WAIT #1: nam='SQL*Net message to client' ela= 4 p1=1650815232 p2=1 p3=0
  WAIT #1: nam='SQL*Net message from client' ela= 1261 p1=1650815232 p2=1 p3=0
  STAT #1 id=1 cnt=1 pid=0 pos=0 obj=221 op='TABLE ACCESS FULL DUAL '
  XCTEND rlbk=0, rd_only=1

```

Такой маленький файл трассировки несложно пройти вручную. В конце главы я опишу каждую операцию, чтобы вы осознали, какого рода данные можно найти в файле трассировки. Пока же обратимся только к наиболее значимым пунктам.

В начале файла трассировки содержится преамбула – заголовок, включающий в себя сведения о файле: его имя, версию сформировавшего его ядра Oracle и т. д. Затем идет строка, в которой определен сеанс, для которого выполняется трассировка (в нашем случае – сеанс 7, порядковый номер 6692), и время появления данной строки. Обратите внимание, что ядро идентифицирует каждую команду SQL, используемую сеансом, в секции PARSING IN CURSOR. В этой секции приводятся атрибуты используемого текста SQL, в том числе и сам текст.

Рабочие строки файла трассировки начинаются с лексем PARSE, EXEC и FETCH (а также некоторых других), также к ним относятся строки WAIT. Каждая из строк PARSE, EXEC и FETCH отражает выполнение одного вызова базы данных. Статистики с и e сообщают, сколько было потрачено на данный вызов времени процессора и какова была общая продолжительность вызова соответственно. Другие статистические показатели, присутствующие в строке, представляют собой количество блоков Oracle, полученных системными вызовами чтения (р) или обращениями к кэшу буферов базы данных в двух режимах (сг для чтения в согласованном режиме и су для чтения в текущем режиме), количество непопаданий в библиотечный кэш (mis) и количество возвращенных вызовом строк (r). Значение tim в конце строки позволяет приблизительно оценить время, когда этот вызов был завершен.

Строки WAIT – это замечательное «новое» дополнение к файлам трассировки Oracle (они доступны только начиная с 1992 года). Строки WAIT как раз и относятся к тому, что отличает данные *расширенной* трасси-

ровки SQL от обычных старых трассировочных данных. Каждая строка WAIT сообщает о продолжительности определенной последовательности инструкций, выполняемой внутри процесса ядра Oracle. Статистика ela содержит время отклика такой последовательности инструкций. Атрибут nam определяет вызов; значения p1, p2 и p3 предоставляют полезную информацию о вызове, формат которой зависит от значения nam.

Строки STAT не передавали непосредственной информации о времени отклика до версии 9.2. Однако и до выхода версии 9.2 они широко применялись при анализе производительности, т. к. содержали сведения о плане выполнения, выбранном оптимизатором запросов Oracle для выполнения команд SQL. Наконец, строка XTEND появляется в том случае, когда исследуемое приложение выполняет команду фиксации или отката.

Вот так. Все, что надо знать о времени отклика сеанса, есть в файле трассировки. Одно из главных достоинств этих данных состоит в том, что они показывают, что *именно* сеанс делал в ходе своего выполнения. Нет необходимости восстанавливать составляющие из агрегированных данных (чем приходится заниматься, например, при оценке данных V\$). Все составляющие как на ладони, они расположены в хронологическом порядке¹ и хранятся в удобном для разбора формате ASCII.

Справочник по данным расширенной трассировки SQL

Одна из причин невероятного успеха корпорации Oracle на рынке высокопроизводительных баз данных заключается в простоте доступа к подробным данным о времени отклика. Ядро Oracle обеспечивает аналитика всеми сведениями (в файлах расширенной трассировки SQL и различных фиксированных представлениях), необходимыми для того, чтобы понять, *почему* время отклика приложения было именно таким. Остается «только» понять, что делать со всей этой информацией. Этот пробел мы надеемся заполнить нашей книгой.

Определения элементов файла трассировки

Форматы строк файла трассировки описаны во многих достойных источниках [Oracle MetaLink note 39817.1; Kyte (2001) 464–475; Morle (2000) 133–142]. Однако ни один из них не предоставляет достаточной информации, чтобы сделать возможным полный учет времени отклика. А именно полный учет времени отклика – та цель, которой вы должны достичь при помощи этой книги. В последующих разделах описано значение каждой из связанных с производительностью статистик, встречающихся в данных расширенной трассировки SQL.

¹ С несколькими незначительными исключениями из строгого хронологического порядка вы вскоре познакомитесь.

Номера курсоров

Каждая строка файла трассировки соответствует одному «действию», выполненному процессом ядра Oracle. Имеющиеся в каждой строке символы #ID идентифицируют курсор, согласно которому ядро выполняло действие. Например, приведенная ниже строка соответствует выборке по курсору #1:

```
FETCH #1:c=0,e=177,p=0,cr=1,cu=2,mis=0,r=1,dep=0,og=4,tim=1038931660054596
```

Номера курсоров имеют значение только в рамках файла трассировки. Более того, ядро Oracle делает номер курсора доступным для повторного использования в файле трассировки после закрытия курсора.¹ Поэтому строки файла трассировки, содержащие ссылки на один и тот же номер курсора, совсем необязательно ссылаются на один и тот же курсор. К счастью, каждый конкретный файл трассировки содержит упорядоченную по времени историю создания всех курсоров; каждая лексема PARSING IN CURSOR указывает на рождение (или возрождение) курсора. Например, рассмотрим две строки PARSING IN CURSOR из файла трассировки в примере 5.2:

```
=====
PARSING IN CURSOR #1 len=69 dep=0 uid=5 oct=42 lid=5 tim=1038931660052098
                                hv=1509700594 ad='50d6d560'
alter session set events '10046 trace name context forever, level 12'
END OF STMT
...
=====
PARSING IN CURSOR #1 len=51 dep=0 uid=5 oct=3 lid=5 tim=1038931660054075
                                hv=1716247018 ad='50c551f8'
select 'Hello, world; today is '||sysdate from dual
END OF STMT
```

Первая секция PARSING IN CURSOR сообщает, что курсор #1 был сопоставлен команде ALTER SESSION. Далее в том же самом файле трассировки ядро Oracle повторно использует идентификатор #1 для курсора, сопоставленного команде SELECT.

Идентификация сеанса и временные метки

Строка, начинающаяся с лексемы ***, указывает системное время, полученное непосредственно перед передачей строки *** в файл трассировки. Например:

```
*** 2002-12-02 22:25:53.716
*** SESSION ID:(8.6550) 2002-12-02 22:25:53.714
```

Эта информация помогает аналитику по производительности установить соответствие между показаниями времени tim в статистике Oracle

¹ Автор несколько упрощает: номера курсоров становятся доступными для повторного использования не в файле трассировки, а в сессии, независимо от того, трассируется данная сессия или нет. — *Примеч. науч. ред.*

и показаниями системных часов. Ядро Oracle любезно вставляет в файл трассировки строку `***` каждый раз, когда после вывода предыдущей строки трассировки проходит значительное количество времени (десятки секунд). Это удобно, потому что позволяет восстановить синхронизацию событий с фактическим временем после больших интервалов строк `WAIT`, содержащих приблизительное истекшее время (`ela`), а не значения внутренних часов (`tim`). Такую строку можно вставить в свой файл данных самостоятельно, вызвав `DBMS_SYSTEM.KSDDDT`.

Строка, содержащая лексему `SESSION ID: (m.n)`, идентифицирует строки файла трассировки, следующие за ней, как относящиеся к сеансу Oracle с идентификатором `V$SESSION.SID=m` и порядковым номером `V$SESSION.SERIAL#=n`. Строки идентификации сеанса обеспечивают уверенность в том, что анализируется именно тот файл, который нужен. Эти строки особенно важны в многопоточной конфигурации Oracle, т. к. каждый процесс ядра может обслуживать запросы от многих сеансов. Строки с идентификатором сеанса показывают, работа какого из них представлена в последующих строках трассировки.

Обратили ли вы внимание на то, что приведенные строки идентификации сеанса и временной метки выводятся не в хронологической последовательности? В первой строке отмечено время `22:25:53.716`, а во второй строке – время на `0,002` секунды раньше. Это явление аналогично описанному ниже в разделе «Идентификация курсоров».

Идентификация приложения

Если название модуля или операции приложения заданы при помощи процедур пакета `DBMS_APPLICATION_INFO`, то при активации трассировки SQL уровня 1 ядро Oracle передает строку `APPNAME`. Например:

```
APPNAME mod='SQL*Plus' mh=3669949024 act='' ah=4029777240
```

Рассмотрим входящие в эту строку значения:

`mod`

Имя модуля, заданное процедурой `SET_MODULE`.

`mh`

Хеш-значение, идентифицирующее модуль.

`act`

Имя операции, заданное процедурой `SET_MODULE` или `SET_ACTION`.

`ah`

Хеш-значение, идентифицирующее операцию.

Идентификация курсора

Секция `PARSING IN CURSOR` содержит информацию о курсоре, например:

```
=====
```

```
PARSING IN CURSOR #135 len=358 dep=0 uid=173 oct=3 lid=173 tim=3675359494
hv=72759792 ad='bb13f788'
```



```

select vendor_number, vendor_id, vendor_name, vendor_type_lookup_code,
type_1099, employee_id, num_1099, vat_registration_num, awt_group_id,
allow_aws_flag, hold_all_payments_flag, num_active_pay_sites,
total_prepays, available_prepays from po_vendors_ap_v where (VENDOR_NUMBER
LIKE : 1) AND ( active_flag = 'Y' and enabled_flag = 'Y' ) order by
vendor_number
END OF STMT

```

Сама строка `PARSING IN CURSOR` содержит информацию об идентификаторе курсора `#ID`. Текст между строкой `PARSING IN CURSOR` и соответствующей строкой `END OF STMT` — это собственно SQL-текст курсора. Ядро Oracle обычно отображает эту секцию по завершении вызова разбора, непосредственно перед строкой курсора `PARSE`. Однако если трассировка была отключена в момент завершения вызова разбора, ядро обычно передает секцию где-то в начале данных трассировки (непосредственно перед завершением первого трассируемого вызова базы данных, возможно, после одной или нескольких строк `WAIT`), как если бы ядро выполняло следующий псевдокод:

```

# При завершении операций ядра Oracle, связанных с вызовом базы данных...
if SQL tracing level >= 1 {
    if db call is PARSE or pic[cursor_id] is unset {
        emit "PARSING IN CURSOR" section
        pic[cursor_id] = 1
    }
    emit statistics for the db call
}

```

Таким образом, Oracle показывает в файле трассировки данные о курсоре, даже если трассировка была отключена в момент завершения вызова разбора курсора.

Каждая строка `PARSING IN CURSOR` содержит следующие сведения о курсоре:

len

Длина текста SQL.

dep

Рекурсивная глубина курсора. Курсор глубиной $dep = n + 1$ является потомком некоторого курсора глубиной $dep = n$ ($n = 0, 1, 2, \dots$). Некоторые действия приводят к использованию рекурсивного SQL, например вызовы базы данных, которым необходима информация из словаря БД; команды, запускающие триггеры; PL/SQL-блоки, содержащие команды SQL. В разделе «Двойной учет рекурсивного SQL» далее в этой главе мы продолжим разговор о рекурсивных отношениях в SQL.

uid

Идентификатор пользователя, выполняющего разбор команды.

oct

Идентификатор типа команды Oracle [Oracle OCI (1999)].

lid

Идентификатор пользователя, обладающего привилегиями. Так, если пользователь FRED вызывает пакет, принадлежащий пользователю JOE, то для команды SQL, выполненной внутри пакета, uid будет ссылаться на FRED, а lid – на JOE.

Проверка в Oracle версии 9 показала, что независимо от того, выполнялась ли программа под привилегиями владельца или под привилегиями вызвавшего ее пользователя, uid и lid в файле трассировки имеют одно и то же значение: идентификатор пользователя, зарегистрировавшегося в системе и вызвавшего эту программу.

tim

Если значение tim равно 0, значит, параметр инициализации TIMED_STATISTICS имел значение FALSE в момент, когда должно было подсчитываться время вызова. Таким образом, на основе значений tim можно сделать вывод о том, какое значение имел параметр инициализации TIMED_STATISTICS. В процессе работы мы с коллегами пришли к выводу о том, что конкретные ненулевые значения tim, относящиеся к секциям PARSING IN CURSOR, не представляют особого интереса.

В Oracle9i значение tim измеряется в микросекундах (1 μ s = 0,000001 секунды). В некоторых системах (например, на наших Linux-серверах) значения поля tim – это чистые значения gettimeofday. В других системах (например, на наших Windows-компьютерах) происхождение значений поля tim может быть весьма загадочным. В версиях, предшествующих Oracle9i, tim было значением V\$TIMER.HSECS, выраженным в сотых долях секунды.

hv

Идентификатор команды SQL. Может показаться, что значение hv уникально, но это не так. Может случиться (хотя это бывает редко), что разные тексты SQL имеют одинаковые значения hv.

ad

Адрес курсора в библиотечном кэше, как это показано в V\$SQL.

Вызовы базы данных

Вызов *базы данных (database call)* – это обращение к подпрограмме ядра Oracle. Если в момент завершения вызова базы данных активна трассировка SQL уровня 1, то ядро Oracle выводит в файл трассировки строку, информирующую о состоявшемся вызове. Чаще всего встречаются вызовы базы данных типа PARSE, EXEC и FETCH. Например:

```

PARSE #54:c=20000,e=11526,p=0,cr=2,cu=0,mis=1,r=0,dep=1,og=0,
                                     tim=1017039304725071
EXEC #1:c=10000,e=12137,p=0,cr=22,cu=0,mis=0,r=1,dep=0,og=4,
```

```
tim=1017039275981174
FETCH #3:c=10000,e=306,p=0,cr=3,cu=0,mis=0,r=1,dep=2,og=4,
tim=1017039275973158
```

О других типах вызовов (например, ERROR, UNMAP и SORT UNMAP) можно прочитать в документе Oracle *MetaLink* 39817.1. Каждая строка вызова базы данных содержит следующие статистики:

c

Общее процессорное время, потраченное процессом Oracle в ходе вызова. Oracle*9i* измеряет c в микросекундах (1 μ s = 0,000001 секунды). В более ранних версиях ядра значение c выражалось в сотых долях секунды.

e

Фактическая продолжительность вызова. Oracle*9i* измеряет e в микросекундах (1 μ s = 0,000001 секунды). В более ранних версиях ядра значение e выражалось в сотых долях секунды.

p

Количество блоков базы данных Oracle, полученных вызовом за счет обращения к дисковой подсистеме ОС. Предполагается, что p означает первую букву слова «physical» – физический, но имейте в виду, что не каждое так называемое «физическое» чтение Oracle обращается к физическому дисковому устройству. Многие такие чтения обслуживаются различными кэшами, расположенными между ядром Oracle и физическим диском.

sg

Количество блоков базы данных Oracle, полученных вызовом в *режиме согласованного чтения* из кэша буферов базы данных Oracle. Чтение в согласованном режиме может потребовать дополнительных согласованных чтений из блоков отката, хранящихся в сегментах отката.

cu

Количество блоков базы данных Oracle, полученных вызовом в *режиме текущего чтения* из кэша буферов базы данных Oracle. Чтение в текущем режиме – это просто чтение текущего содержимого блока.

mis

Количество непопаданий в библиотечный кэш в ходе вызова. Результатом каждого непопадания в библиотечный кэш является операция *полного разбора (hard parse)*.

r

Количество строк, возвращаемых вызовом.

dep

Рекурсивная глубина курсора. Курсор глубиной $dep=n+1$ является потомком некоторого курсора глубины $dep=n$ ($n = 0, 1, 2, \dots$). Под-

робнее об этом рассказано ниже в разделе «Двойной учет рекурсивного SQL» данной главы.

og

Фактическая цель оптимизатора в ходе вызова. Oracle использует значения, приведенные в табл. 5.1.

tim

См. раздел «Идентификация курсора» ранее в этой же главе.

Таблица 5.1. Цели оптимизатора запросов Oracle, соответствующие значениям og (источник: документ Oracle MetaLink 39817.1)

Значение og	Цель оптимизатора запросов Oracle
1	ALL_ROWS
2	FIRST_ROWS
3	RULE
4	CHOOSE

Имейте в виду, что ядро Oracle не передает в файл трассировки строку вызова базы данных до тех пор, пока операция не *завершена*. То есть чрезвычайно долгая операция над базой данных может привести к тому, что ядро Oracle будет работать несколько часов, *ничего* не передавая в файл трассировки. Плохо оптимизированный SQL может порождать вызовы EXEC (для обновлений или удалений) или FETCH (для выборок), способные загрузить процессор на несколько *дней*.

События ожидания

Событие *ожидания* Oracle – это последовательность инструкций ядра Oracle, для которой время учитывается специальным образом. Если в момент завершения события ожидания включена трассировка SQL уровня 8 или 12, то ядро Oracle по завершении данного события вставляет в файл трассировки строку WAIT. Например:

```
WAIT #1: nam='SQL*Net message to client' ela= 40 p1=1650815232 p2=1 p3=0
WAIT #1: nam='SQL*Net message from client' ela= 1709 p1=1650815232 p2=1 p3=0
WAIT #34: nam='db file sequential read' ela= 14118 p1=52 p2=2755 p3=1
WAIT #44: nam='latch free' ela= 1327989 p1=-1721538020 p2=87 p3=13
```

Каждая строка WAIT содержит следующую информацию о работе, выполненной в ходе события:

nam

Имя, присвоенное кем-то из разработчиков ядра Oracle для обозначения того, какая часть ядра Oracle отвечает за данную часть времени отклика.

ela

Фактическое время, потраченное на исполнение названного события. Oracle9i измеряет ela в микросекундах (1 μ s = 0,000001 секунды).

В более ранних версиях ядра значение `ela` выражалось в сотых долях секунды.

p1, p2, p3

Значения данных параметров зависят от значения `nam`. Полный перечень описаний параметров для всех типов событий можно получить, выполнив такой SQL-код:

```
select name, parameter1, parameter2, parameter3
from v$event_name order by name
```

Обратите внимание, что строки `WAIT` появляются в файле трассировки *до* строки того вызова базы данных, результатом которого является данное событие ожидания. Дело в том, что ядро Oracle передает строки в файл по мере завершения событий. Другими словами, если вызов выборки требует выполнения трех системных вызовов чтения, то три события ожидания для вызовов чтения появятся в файле трассировки прежде, чем Oracle выведет информацию о завершении вызова выборки.

Строки `WAIT` в файле трассировки SQL представляют собой интерфейс к чрезвычайно важной новой функциональности, введенной Oracle в 1992 г., которая коренным образом упрощает диагностику и решение проблем производительности.

Переменные связывания

Если трассировка SQL уровня 4 или 12 активна в тот момент, когда ядро Oracle связывает значения с заполнителями в SQL-тексте приложения, то ядро выводит в файл трассировки секцию `BINDS`. Например:

```
=====
PARSING IN CURSOR #1 len=105 dep=0 uid=56 oct=47 lid=56 tim=1017039275982462
                                hv=2108922784 ad='98becef8'
declare dummy boolean;begin fnd_profile.get_specific(:name, :userid,
:respid, :applid, :val, dummy);end;
END OF STMT
...
Несколько строк пропущено для удобства восприятия
...
BINDS #1:
  bind 0: dty=1 mxl=2000(1998) mal=00 scl=00 pre=00 oacflg=01 oacfl2=0
                                size=2000 offset=0
        bfp=025a74a0 bln=2000 avl=19 flg=05
        value="MFG_ORGANIZATION_ID"
  bind 1: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=01 oacfl2=0
                                size=72 offset=0
        bfp=025a744c bln=22 avl=04 flg=05
        value=118194
  bind 2: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=01 oacfl2=0
                                size=0 offset=24
        bfp=025a7464 bln=22 avl=05 flg=01
        value=1003677
  bind 3: dty=2 mxl=22(22) mal=00 scl=00 pre=00 oacflg=01 oacfl2=0
```

```

size=0 offset=48
bfp=025a747c bln=22 avl=03 flg=01
value=140
bind 4: dty=1 mxl=2000(1998) mal=00 scl=00 pre=00 oacflg=01 oacfl2=0
size=2000 offset=0
bfp=025ba490 bln=2000 avl=00 flg=05

```

Секция BINDS может включать в себя одну или несколько подсекций связывания, по одной для каждой связываемой переменной. Номер, следующий за словом «bind», указывает позицию (порядковый номер, начиная с 0) переменной связывания в тексте SQL. Каждая секция связывания содержит различные характеристики связывания, наиболее важными из которых для анализа производительности являются следующие:

dty

Внешний тип данных значения, переданного приложением [Oracle OCI (1999)]. В Oracle определены два набора типов данных: *внутренний* и *внешний*. Определения внутреннего типа данных показывают, каким образом ядро Oracle хранит свои данные в операционной системе, а определения внешнего типа – каким образом ядро Oracle взаимодействует с SQL-кодом приложения.

Внешний тип данных переменной связывания весьма важен. Время от времени встречаются команды SQL, для которых оптимизатор запросов Oracle отказывается использовать индекс, безусловно, помогающий делу. Иногда это вызвано несоответствием типа столбца типу значения, что может привести к выполнению для столбца функции неявного приведения типа, что не позволит оптимизатору выбрать данный индекс.

avl

Длина связываемого значения (в байтах).

value

Значение, которое передается в выполняемую команду. Ядро Oracle может усекаать значения, передаваемые в файл трассировки. Понять, что это произошло, несложно – усечение будет иметь место всегда, когда значение avl превышает длину поля value.

Если значение value не попало в файл трассировки, значит, переменная-заполнитель была связана со значением NULL. Так, в приведенном выше примере `fn$profile.get_specific` отсутствие поля value в секции bind 4 говорит о том, что приложение передало NULL в качестве значения заполнителя :val. Это подтверждается нулевой длиной (avl=00) значения переменной связывания.

Операции над источником строк

Если в момент закрытия курсора включена трассировка SQL уровня 1, то ядро Oracle передает в файл трассировки по одной строке STAT для

каждой операции над источником строк из плана выполнения запроса. Например:

```
STAT #1 id=1 cnt=55 pid=0 pos=1 obj=0 op='SORT UNIQUE (cr=39741 r=133 w=0
time=1643800 us)'  
STAT #1 id=2 cnt=23395 pid=1 pos=1 obj=0 op='VIEW (cr=39741 r=133 w=0  
time=1614067 us)'  
STAT #1 id=3 cnt=23395 pid=2 pos=1 obj=0 op='SORT UNIQUE (cr=39741 r=133 w=0  
time=1600554 us)'  
STAT #1 id=4 cnt=23395 pid=3 pos=1 obj=0 op='UNION-ALL (cr=39741 r=133 w=0  
time=1385984 us)'
```

Если в файле трассировки не оказывается ожидаемых строк STAT, значит, трассировка была отключена до закрытия курсора. Естественно, строки STAT будут отсутствовать при трассировке хорошо спроектированного постоянно работающего сервиса, который закрывает свои курсоры не чаще одного раза в несколько недель.

Каждая строка STAT содержит следующие данные о плане выполнения курсора:

id

Уникальный идентификатор операции над источником строк внутри множества строк STAT.

cnt

Количество строк, возвращенных данной операцией над источником строк.

pid

Идентификатор операции, родительской по отношению к данной.

pos

Произвольное число – ничего лучше не придумать. Можно предположить, что это значение могло бы определять «позицию» операции над источником данных во множестве операций, принадлежащих одному родителю, но похоже, что сестринские операции над источником данных упорядочены по возрастанию идентификаторов.

obj

Идентификатор объекта для операции над источником строк, если операция выполняется над «базовым объектом».¹ Такие операции над источником строк, как NESTED LOOPS, которые сами не обращаются к базовому объекту, имеют obj=0. (Дочерние операции для NESTED LOOPS обращаются к базовым объектам, но сама операция NESTED LOOPS над источником данных этого не делает.)

¹ Имеется в виду, что некоторые операции обращаются к объектам базы данных – базовым объектам, а некоторые – только к результатам предшествующих операций. – *Примеч. науч. ред.*

op

Имя операции над источником данных. Начиная с версии Oracle 9.2.0.2.0, ядро передает дополнительную информацию в строки STAT [Rivenes (2003)]. Новая информация включает в себя ряд полезных характеристик для каждой операции над источником строк, а именно:

cr

Количество чтений в согласованном режиме.

r

Количество блоков Oracle, прочитанных вызовами чтения ОС.

w

Количество блоков Oracle, записанных вызовами записи ОС.

time

Фактическая длительность (в микросекундах).

Статистика для родительской операции над источником строк включает в себя статистики для ее потомков.



Утилита Oracle *tkprof* выдает ошибочные результаты гораздо чаще, чем можно было бы предположить, особенно при обработке строк STAT. Утилита имеет репутацию исключительно надежной, но лично я убежден в том, что единственная причина такой репутации состоит в том, что никто никогда не утруждал себя тем, чтобы перепроверить ее выходные данные. Невозможно определить, корректен ли вывод *tkprof*, не изучив необработанные данные трассировки. А большинство пользователей не хочет этим заниматься. Надеюсь, что эта книга убедит вас сделать над собой усилие.

Маркеры конца транзакции

Если трассировка SQL уровня 1 активна в момент выполнения фиксации или отката транзакции, то ядро Oracle при завершении вызова передает в файл трассировки строку XCTEND. Например:

```
XCTEND rlbk=0, rd_only=0
```

Каждая строка XCTEND содержит следующие сведения о работе, выполняемой в ходе фиксации или отката транзакции:

rlbk

Истина (1) – в том и только в том случае, если транзакция была откатчена.

rd_only

Истина (1) – в том и только в том случае, если транзакция не изменила никаких данных в базе данных.

Обратите внимание, что маркер `XCTEND` не ссылается на идентификатор курсора. Это объясняется тем, что транзакция и участвующие в ней курсоры находятся в отношении «один-ко-многим».

Краткий справочник

Доступные в файле трассировки статистики, представляющие наибольший интерес при анализе производительности, собраны в табл. 5.2.

Таблица 5.2. Описания некоторых элементов данных расширенной трассировки SQL

Поле	Встречается в ...			Описание
	Идентифика- тор курсора	Вызов БД	Событие ожидания	
c		✓		Общее время процессора, потрачен- ное в ходе вызова БД. Измеряется в микросекундах в Oracle9i, в ранних версиях – в сотых долях секунды.
cr		✓		Количество блоков Oracle, получен- ных вызовом из кэша буферов в ре- жиме согласованного чтения.
cu		✓		Количество блоков Oracle, получен- ных вызовом из кэша буферов в ре- жиме текущего чтения.
dep	✓	✓		Рекурсивная глубина курсора.
e		✓		Фактическая продолжительность вы- зова. Измеряется в микросекундах в Oracle9i, в ранних версиях – в со- тых долях секунды.
ela			✓	Фактическая продолжительность со- бытия ожидания. Измеряется в мик- росекундах в Oracle9i, в ранних вер- сиях – в сотых долях секунды.
hv	✓			Идентификатор команды.
mis		✓		Количество непопаданий в библио- течный кэш.
nam			✓	Имя события ожидания.
p		✓		Количество блоков Oracle, получен- ное посредством системных вызовов чтения.
p1, p2, p3			✓	Сведения о событии ожидания, зави- сит от значения nam.
tim	✓	✓		Внутреннее время Oracle, указываю- щее момент завершения события.

Единицы времени Oracle

Ядро Oracle*9i* отображает временные характеристики трассировки SQL в микросекундах (1 μ с = 0,000001 сек). В версиях Oracle 6, 7 и 8 временные значения выводятся в сантисекундах (1 сантисек = 0,01 сек). В табл. 5.3 приведен перечень единиц измерения, которые ядро Oracle использует для каждого вида временной статистики в данных расширенной трассировки SQL.

Таблица 5.3. Единицы измерения временных характеристик для файла трассировки в зависимости от версии Oracle

Версия Oracle	c	e	ela	tim
9	μ с	μ с	μ с	μ с
8	сс	сс	сс	сс
7	сс	сс	сантисек	сс
6	сс	сс	N/A	сс

В табл. 5.4 поясняется значение единиц измерения времени, с которыми вам придется иметь дело при анализе производительности Oracle.

Таблица 5.4. Единицы времени, обычно используемые аналитиками по компьютерной производительности

Название единицы	Сокращение	Продолжительность в секундах		
Секунда	1 с	1	1E-0	1.
Сотая доля секунды (сантисекунда)	1 сс	1/100	1E-2	0.01
Миллисекунда	1 мс	1/1000	1E-3	0.001
Микросекунда	1 μ с	1/1000000	1E-6	0.000001

Учет времени отклика

Ядро Oracle передает в файл трассировки значения времени двух категорий:

1. Время, потраченное *внутри* вызова базы данных.
2. Время, прошедшее *между* вызовами базы данных.

Общее время отклика сеанса представляет собой сумму всех длительностей вызовов базы данных и длительностей промежутков между вызовами. Для того чтобы не забыть учесть какой-нибудь промежуток времени или, наоборот, не учесть что-то лишнее, необходимо понимать, к чему относится каждая строка файла трассировки.

Время внутри вызова базы данных

Фрагмент файла трассировки, приведенный в примере 5.3, отображает действия, потребляющие время в рамках трех различных вызовов базы данных. Первый вызов представляет собой разбор, который продолжался 306 микросекунд. Ядро любезно выдало секцию PARSING IN CURSOR перед строкой PARSE, так что мы можем сказать, что же было подвергнуто разбору. Далее ядро выводит строку EXEC, означающую, что вызов исполнения для курсора завершен, что потребовало дополнительных 146 микросекунд времени. Затем речь идет о двух системных вызовах чтения, которые указаны в двух строках WAIT. «Родительской» операцией, породившей эти вызовы чтения, является вызов выборки, статистика которого отражена в строке FETCH.

Пример 5.3. Фрагмент файла трассировки, иллюстрирующий потребление времени тремя вызовами базы данных

```
=====
PARSING IN CURSOR #4 len=132 dep=1 uid=0 oct=3 lid=0 tim=1033064137929238
                                hv=3111103299 ad='517ba4d8'
select /*+ index(idl_ub1$ i_idl_ub11) */ piece#,length,piece from idl_ub1$
where obj#=1 and part=:2 and version=:3 order by piece#
      END OF STMT
      PARSE
#4:c=0,e=306,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=1033064137929139
      EXEC
#4:c=0,e=146,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=1033064137931262
❶ WAIT #4: nam='db file sequential read' ela= 13060 p1=1 p2=53903 p3=1
❷ WAIT #4: nam='db file sequential read' ela= 6978 p1=1 p2=4726 p3=1
❸ FETCH #4:c=0,e=21340,p=2,cr=3,cu=0,mis=0,r=0,dep=1,og=4,tim=1033064137953092
STAT #4 id=1 cnt=0 pid=0 pos=0 obj=72 op='TABLE ACCESS BY INDEX ROWID IDL_UB1$'
STAT #4 id=2 cnt=0 pid=1 pos=1 obj=120 op='INDEX RANGE SCAN'
```

Строки вызовов чтения находятся в файле трассировки *перед* строкой родительской выборки, т. к. ядро Oracle выдает статистику для операции только по ее завершению. Инструкции ядра Oracle, породившие приведенные строки трассировки, должны были выглядеть как-то так:

```
fetch IDL_UBL$ query
    выполнить какие-то инструкции, необходимые для выборки IDL_UBL$
    выполнить вызов ввода/вывода для одного блока в файле 1, блок 53903
    вывести ❶ "WAIT #4: nam='db file sequential read' ela=13060 ..."
    выполнить еще какие-то инструкции по выборке
    выполнить вызов ввода/вывода для одного блока в файле 1, блок 4726
    вывести ❷ "WAIT #4: nam='db file sequential read' ela=6978 ..."
    выполнить оставшиеся инструкции по выборке
    вывести ❸ "FETCH #4:c=0,e=21340,..."
закрыть курсор
и т. д.
```

Общая продолжительность вызова выборки составила 21340 мс. Составляющие времени отклика для этого вызова приведены в табл. 5.5.

Таблица 5.5. Составляющие времени отклика вызова выборки

Время отклика	Составляющая
13060 мксек	Последовательное чтение файла базы данных
6978 мксек	Последовательное чтение файла базы данных
0 мксек	Общее процессорное время
1302 мксек	Потерянное время
21340 мксек	Общая продолжительность выборки

Статистика *e* для вызова базы данных – это фактическая продолжительность всего вызова базы данных. Другими словами, значение *e* включает в себя все время процессора, потребленное в течение вызова (которое передано в значении *c*), плюс все длительности событий ожидания в контексте данного вызова базы данных (переданы в значениях *ela*). Это соотношение отображено на рис. 5.1, формально же его можно записать следующим образом:

$$e \approx c + \sum_{db\ call} ela$$

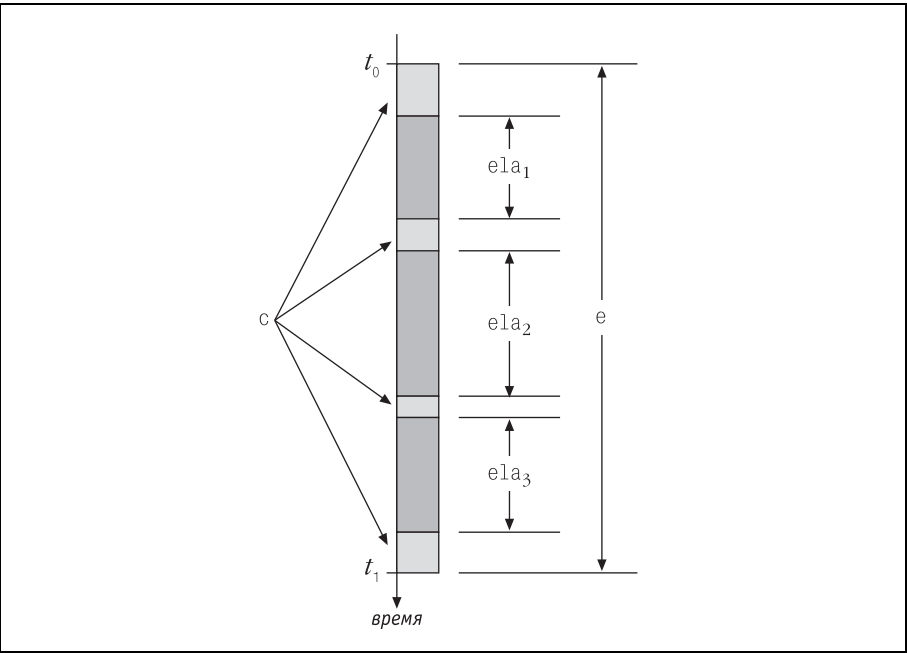


Рис. 5.1. Базовое соотношение временных статистик Oracle в пределах одного вызова базы данных: общая фактическая продолжительность (*e*) приблизительно равна общему времени процессора, потраченного на вызов (*c*), к которому прибавлена сумма длительностей соответствующих событий ожидания ($\sum ela$)

Речь идет о базовом соотношении временных статистик Oracle в рамках одного вызова базы данных. Оно является приблизительным за счет наличия таких факторов, как влияние измерительного инструмента, ошибка квантования, время без выполнения каких-либо действий и наличие в ядре Oracle сегментов кода, лишенных измерительных средств, о которых мы поговорим в главе 7.

Время между вызовами базы данных

Ядро Oracle также выводит сведения о фактической продолжительности событий ожидания, которые имеют место *между* вызовами базы данных. К примерам событий ожиданий, происходящих между вызовами базы данных, относятся:

```
SQL*Net message from client
SQL*Net message to client
single-task message
pipe get
rdbms ipc message
pmon timer
smon timer
```

Фрагмент файла трассировки, приведенный в примере 5.4, иллюстрирует события ожидания, которые имеют место между вызовами базы данных. Рассматриваемое приложение содержит ошибку – слишком часто выполняется разбор, что ограничивает возможность масштабирования. Как видите, фрагмент содержит два последовательных вызова разбора (выделенных жирным шрифтом) абсолютно одинакового текста SQL. Строки WAIT (помеченные жирным курсивом) находятся *между* вызовами разбора – как в смысле их физического расположения в файле трассировки, так и потому, что продолжительности этих событий не учитываются во времени, истекшем с начала второго вызова. Это легко проверить – фактическая длительность, показанная во второй строке PARSE (e=0), слишком мала для того, чтобы вмещать в себя продолжительность события SQL*Net message from client (ela=3).

Пример 5.4. Фрагмент файла трассировки, иллюстрирующий затраты времени между двумя одинаковыми вызовами разбора в системе Oracle8i

```
=====
PARSING IN CURSOR #9 len=360 dep=0 uid=26 oct=2 lid=26 tim=1716466757
hv=2475520707 ad='d4c55480'
INSERT INTO STAGING_AREA (TMSP_LAST_UPDT, OBJECT_RESULT, USER_LAST_UPDT,
DOC_ OBJ_ID, TRADE_NAME_ID, LANGUAGE_CODE) values(TO_DATE('11/05/2001
16:39:06', 'MM/DD/YYYY HH24:MI:SS'), 'if ( exists ( stdphrase ( "PCP_MAV_1"
) ) , langconv ( "Incompatibility With Other Materials" ) + " : " ,
log_omission ( "Materials to Avoid: " ) )', 'sa', 222, 54213, 'NO_LANG')
END OF STMT
PARSE #9: c=0, e=0, p=0, cr=0, cu=0, mis=1, r=0, dep=0, og=4, tim=1716466757
WAIT #9: nam='SQL*Net message to client' ela= 0 p1=1413697536 p2=1 p3=0
WAIT #9: nam='SQL*Net message from client' ela= 3 p1=1413697536 p2=1 p3=0
```

```

=====
PARSING IN CURSOR #9 len=360 dep=0 uid=26 oct=2 lid=26 tim=1716466760
hv=2475520707 ad='d4c55480'
INSERT INTO STAGING_AREA (TMSP_LAST_UPDT, OBJECT_RESULT, USER_LAST_UPDT,
DOC_OBJ_ID, TRADE_NAME_ID, LANGUAGE_CODE) values(TO_DATE('11/05/2001
16:39:06', 'MM/DD/ YYYY HH24:MI:SS'), 'if ( exists ( stdphrase ( "PCP_MAV_1"
) ) , langconv ( "Incompatibility With Other Materials" ) + " : " ,
log_omission ( "Materials to Avoid: " ) )', 'sa', 222, 54213, 'NO_LANG')
END OF STMT
PARSE #9: c=0, e=0, p=0, cr=0, cu=0, mis=0, r=0, dep=0, og=4, tim=1716466760

```

Теперь, когда вы все это знаете, вам легче будет представить себе соотношение между величинами c , e и ela в рамках всего файла трассировки. Учитывая все вышесказанное, общее время отклика сеанса равно общей продолжительности вызовов базы данных плюс общая продолжительность интервалов времени между такими вызовами. Формально можно представить это отношение так:

$$\begin{aligned}
 R &= \sum e + \sum_{\substack{\text{between} \\ \text{calls}}} ela \\
 &\approx \left[\sum c + \sum_{\substack{\text{within} \\ \text{calls}}} ela \right] + \sum_{\substack{\text{between} \\ \text{calls}}} ela \\
 &= \sum c + \sum ela
 \end{aligned}$$

Есть, однако, еще одна сложность: наличие рекурсивного SQL приводит к двойному учету времени.

Двойной учет рекурсивного SQL

Рекурсивный SQL – это SQL, соответствующий вызову базы данных, значение dep которого больше нуля. Вызов базы данных глубиной $dep = n + 1$ ($n = 0, 1, 2, \dots$) можно рассматривать как потомка какого-то вызова базы данных со значением $dep = n$. Сеансы приложений регулярно порождают достаточно сложные данные трассировки для представления всего спектра отношений между командами SQL, выступающими друг по отношению к другу в роли предков, потомков и т. д. Любой файл трассировки SQL содержит достаточно информации для того, чтобы можно было точно определить отношения типа «родитель-потомок» между вызовами базы данных. Для того чтобы не учесть какую-нибудь характеристику дважды при вычислении времени отклика, надо понимать, как выявить рекурсивные отношения между вызовами базы данных.

Отношения типа «родитель-потомок»

Термин *рекурсивный* характеризует исполнение ядром Oracle вызовов базы данных в контексте других вызовов базы данных. Рекурсивность

может быть вызвана: выполнением команд DDL; выполнением блоков PL/SQL, включающих в себя команды DML; вызовами базы данных, к которым приводят к запуску триггеров, а также разнообразными командами DML, требующими доступа к словарию данных. Любой вызов базы данных, способный выполнять другой вызов базы данных, может служить причиной возникновения рекурсивного SQL.

Фрагмент файла трассировки, приведенный в примере 5.5, иллюстрирует использование рекурсивного SQL. В этом фрагменте есть информация о новом курсоре, помеченном как #2, который сопоставлен следующему тексту SQL:

```
select text from view$ where rowid=:1
```

Такого текста не было в исходном тексте исследуемого приложения — он появился в результате разбора запроса из представления DBA_OBJECTS.

Пример 5.5. Фрагмент файла трассировки, демонстрирующий рекурсивный SQL. Три операции курсора #2 глубины dep=1 являются рекурсивными потомками операции разбора курсора #1 глубины dep=0

```
=====
❶ PARSING IN CURSOR #2 len=37 dep=1 uid=0 oct=3 lid=0 tim=1033174180230513
hv=1966425544 ad='514bb478'
select text from view$ where rowid=:1
END OF STMT
❷ PARSE #2:c=0,e=107,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=1033174180230481
❸ BINDS #2:
  bind 0: dty=11 mxl=16(16) mal=00 scl=00 pre=00 oacflg=18 oacfl2=1
                                     size=16 offset=0
      bfp=0a22c34c bln=16 avl=16 flg=05
      value=00000A88.0000.0001
❹ EXEC #2:c=0,e=176,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=1033174180230878
❺ FETCH #2:c=0,e=89,p=0,cr=2,cu=0,mis=0,r=1,dep=1,og=4,tim=1033174180231021
❻ STAT #2 id=1 cnt=1 pid=0 pos=0 obj=62 op='TABLE ACCESS BY USER ROWID VIEW$'
=====
❼ PARSING IN CURSOR #1 len=85 dep=0 uid=5 oct=3 lid=5 tim=1033174180244680
hv=1205236555 ad='50cafbec'
select object_id, object_type, owner, object_name from dba_objects
                                     where object_id=:v
END OF STMT
❽ PARSE #1:c=10000,e=15073,p=0,cr=2,cu=0,mis=1,r=0,dep=0,og=0,
tim=1033174180244662
```

Правило определения рекурсивного отношения между вызовами базы данных формулируется очень просто:

Вызов базы данных, имеющий глубину $dep=n+1$, является рекурсивным потомком первого следующего за ним в потоке данных трассировки SQL вызова базы данных, имеющего глубину $dep=n$.

Пример 5.6 поясняет это правило. Ядро Oracle может выдать данные трассировки для вызова базы данных только по завершении действий (например, ядро не может вычислить фактическую продолжитель-

ность вызова, пока он не завершен). Поэтому мы можем воспроизвести последовательность инструкций, приведших к формированию данных трассировки SQL, приведенных в примере 5.5. В частности, в этом примере все вызовы базы данных для запроса к VIEW\$ являются рекурсивными потомками вызова разбора запроса к DBA_OBJECTS. Для того чтобы подчеркнуть рекурсивное отношение «родитель-потомок» между вызовами базы данных, в тексте примера 5.6 для процедур стека вызовов использованы различные уровни отступа от левого края.

Пример 5.6. Последовательность инструкций ядра Oracle, которая выдает данные трассировки SQL в порядке, приведенном в примере 5.5. Величина отступа выбрана пропорционально глубине вызова

```
Разбор запроса к DBA_OBJECTS
# запрос к VIEW$ для получения определения DBA_OBJECTS
Разбор запроса к VIEW$
    # выполнить инструкции, необходимые для разбора VIEW$
    вывод ❶ "PARSING IN CURSOR #2 ..."
    вывод ❷ "PARSE #2: ..."
связывание переменных курсора VIEW$
    # выполнить инструкции, необходимые для связывания VIEW$
    вывод ❸ "BINDS #2: ..."
выполнение курсора VIEW$
    # выполнить инструкции, необходимые для выполнения VIEW$
    вывод ❹ "EXEC #2: ..."
выборка из курсора VIEW$
    # выполнить инструкции, необходимые для выборки из VIEW$
    вывод ❺ "FETCH #2: ..."
закрытие курсора VIEW$
    # выполнить инструкции, необходимые для закрытия VIEW$
    вывод ❻ "STAT #2: ..."
# выполнить оставшиеся инструкции для разбора DBA_OBJECTS
вывод ❼ "PARSING IN CURSOR #1 ..."
вывод ❸ "PARSE #1: ..."
```

Графическое представление отношения «родитель-потомок» между вызовами базы данных дано на рис. 5.2.

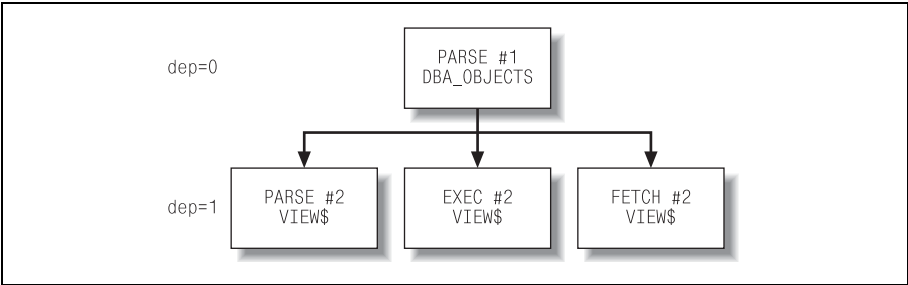


Рис. 5.2. Графическое представление рекурсивных вызовов из примера 5.5

Рекурсивные статистики

В версиях Oracle (по крайней мере, вплоть до Oracle9i Release 2) такие статистики вызовов базы данных, как *c*, *e*, *p*, *cr* и *cu*, включают в себя ресурсы, потребленные собственно вызовом базы данных и всем его рекурсивным потомством.



Рекурсивное потомство вызова базы данных состоит из всех рекурсивных потомков вызова базы данных, включая детей, внуков, правнуков и т. д.

На рис. 5.3 представлено такое отношение для вымышленного набора вызовов базы данных. Каждый узел графа (прямоугольник) – это вызов базы данных (например, *PARSE*, *EXEC* или *FETCH*). Направленная линия от некоторого узла *A* к узлу *B* означает, что вызов базы данных *A* является рекурсивным родителем (т. е. *вызывающим*) для вызова базы данных *B*. Строка *cr*=*n* внутри узла – это то значение, которое ядро Oracle передаст в файл трассировки для вызова базы данных. Значение *crself* – это количество чтений в согласованном режиме, выполненных самим вызовом базы данных, без учета значений для дочерних вызовов.

Ядро выдает в файл трассировки только те значения статистик, которые включают в себя соответствующие значения для потомства, но из них можно извлечь составляющие, исключающие учет работы потомства (величины, указанные внутри дочерних вершин). Например, если бы числа внутри узлов на рис. 5.3 отсутствовали, было бы несложно их проставить. Значение каждого узла – это просто значение статистики для данного узла за вычетом суммы значений статистик для прямых потомков данного узла. Значение узла глубины *dep*=*k* – это значение *cr*, указанное для данного вызова базы данных за вычетом суммы значений *cr* его потомков глубины *dep*=*k* + 1. Обобщая, можно сказать,

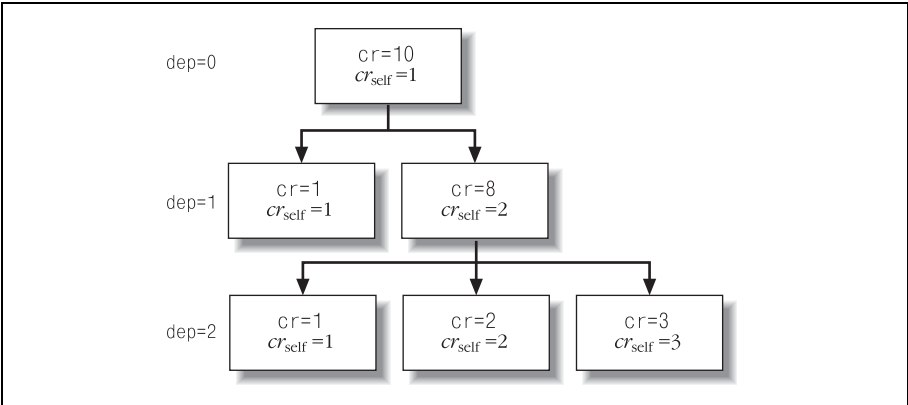


Рис. 5.3. Каждая из величин *c*, *e*, *p*, *cr* и *cu* для вызова базы данных включает в себя соответствующие значения для всего рекурсивного генеалогического дерева данного вызова

что количество ресурсов s , потребленных вызовом базы данных глуби-
ны $dep=k$, вычисляется следующим образом:

$$s = s_k - \sum_{children} s_{k+1},$$

где s_i – это значение статистики из набора $\{c, e, p, cr, cu\}$, выведенное
ядром Oracle в файл трассировки для рекурсивной глубины i .

Описанный метод нетрудно применить к реальным данным трасси-
ровки. Обратимся еще раз к вызовам базы данных из примера 5.5.
На рис. 5.4 приведено значение фактической продолжительности,
включающее статистики для потомства, для каждого вызова базы дан-
ных (обозначенное e) и вклад каждого вызова в фактическую продол-
жительность без учета статистик потомков (обозначенный e_{self}).

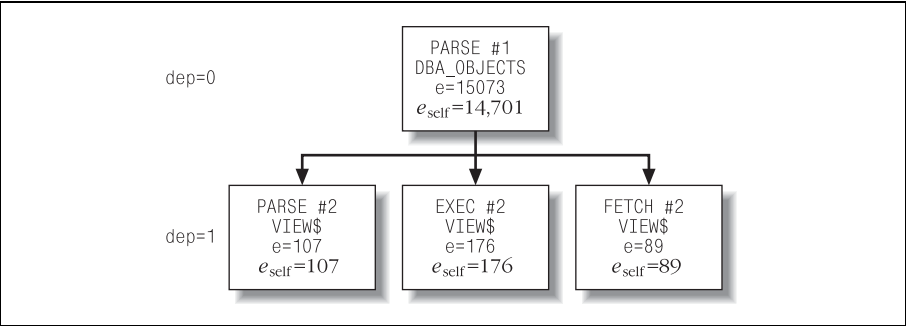


Рис. 5.4. Графическое представление стека рекурсивных вызовов
из примера 5.5

В табл. 5.6 собраны все статистики для всех вызовов базы данных из
примера 5.5, не учитывающие влияние потомства данных вызовов. На-
пример, вклад (без учета потомков) в фактическую продолжительность
вызова базы данных PARSE #1 можно вычислить следующим образом:

$$\begin{aligned} e &= e_0 - \sum_{children} e_i \\ &= 15,073 - (107 + 176 + 89) \\ &= 14,701 \end{aligned}$$

Таблица 5.6. Статистики c, e, p, cr и cu для курсора включают в себя
активность самого курсора, а также операции всех его рекурсивных потом-
ков. Собственная активность курсора оценивается с помощью вычитания

Ресурсы, потребляемые...	c	e	p	cr	cu
PARSE #1, включая его рекурсивное потомство	10000	15073	0	2	0
PARSE #2, потомок	0	107	0	0	0

Ресурсы, потребляемые...	c	e	p	cr	cu
EXEC #2, потомок	0	176	0	0	0
FETCH #2, потомок	0	89	0	2	0
PARSE #1 без учета его рекурсивного потомства	10000	14701	0	0	0

Теперь у нас достаточно информации для завершения формулы учета времени отклика. Избавившись от двойного учета рекурсивного SQL, получим:

$$\begin{aligned} R &= \sum_{dep=0} e + \sum_{\text{between calls}} ela \\ &\approx \left[\sum_{dep=0} c + \sum_{\text{within calls}} ela \right] + \sum_{\text{between calls}} ela \\ &= \sum_{dep=0} c + \sum ela \end{aligned}$$

То есть общее время отклика по файлу трассировки приблизительно равно сумме значений *e* для вызовов базы данных рекурсивной глубины 0 плюс сумма значений *ela* для событий ожидания, которые имеют место между вызовами базы данных. С другой стороны, общее время отклика для файла приблизительно равно сумме значений *c* вызовов базы данных глубины 0 плюс сумма всех значений *ela* для файла.

Эволюция модели времени отклика

В 1980-х годах, когда была изобретена большая часть «методов настройки», функция трассировки SQL в Oracle еще не поддерживала выдачу информации о продолжительности событий ожидания – строк WAIT – в файл трассировки. Доступ имелся только к таким данным, как *c*, *e* и *tim*. Конечно, если основное время отклика было потрачено на потребление времени процессора, то данные *c* и *e* могут предоставить практически всю необходимую информацию о производительности наших вызовов базы данных. Однако если какая-то часть времени отклика вызова базы данных не связана с потреблением времени процессора, то анализ сильно усложнялся.

Рассмотрим статистику для вызова выборки, полученную от приложения, работающего с версией Oracle 8.1.7.2:

```
FETCH #1:c=80741,e=151841,p=9628,cr=34304348,cu=10,mis=0,r=0,dep=0,
og=4,tim=87762034
```

Фактическая продолжительность данного вызова составила 1518,41 с, и только 807,41 из них было потрачено на процессоре. На что же пошли

остальные 711,00 секунд времени отклика? Возникала ли конкуренция блокировок? Событие ожидания блокировки? Длинные дисковые очереди? Избыточная подкачка страниц? Просто посмотрев на строку FETCH, на эти вопросы ответить невозможно. В статистике слишком мало информации для определения того, на что были потрачены неучтенные 711 секунд продолжительности вызова. Конечно, большое значение *p* указывает на то, что некоторая часть неучтенного времени могла быть потрачена на системные вызовы чтения, но дело в том, что существует приблизительно 200 различных событий ожидания, которые Oracle мог бы выполнять в течение этих 711 секунд. Имея перед собой только приведенные сведения для выборки, мы *не можем узнать*, каким образом были потрачены 711 секунд.

В 1992 г., выпустив версию ядра 7.0.12, корпорация Oracle элегантно решила эту проблему. Новый механизм, предложенный Oracle, заключался в снабжении измерительными средствами различных выполняемых ядром событий из числа тех, которые вносят вклад в фактическую продолжительность, но не потребляют процессорного времени. Так называемые «данные ожидания» имели исключительную ценность. Они помогли заполнить временной провал между значениями *e* и *s*. Аньо Колк (Anjo Kolk) и Шари Ямагучи (Shari Yamaguchi) были первыми, кто описал использование «данных ожидания» в документе, ставшем поворотной вехой в истории, – «YAPP Method» (Метод YAPP) [Kolk and Yamaguchi (1999)].

Вернемся к предыдущему примеру, в котором было 711 секунд неучтенного времени. После того как ядро Oracle стало формировать статистику WAIT, в файл трассировки добавилось еще 9748 строк данных перед информацией о вызове выборки. Выполнив программу Perl, приведенную в примере 5.7, для 9749 строк данных трассировки, мы получим следующий профиль ресурсов:

\$ prof-cid waits.1.trc		
Duration	Pct	Oracle kernel event

807.41s	53.2%	total CPU
426.26s	28.1%	direct path write
197.29s	13.0%	db file sequential read
76.23s	5.0%	unaccounted-for
8.28s	0.5%	latch free
2.87s	0.2%	db file scattered read
0.05s	0.0%	file open
0.02s	0.0%	buffer busy waits
0.00s	0.0%	SQL*Net message to client

1518.41s	100.0%	Total response time

Теперь все понятно. Более 53% времени отклика для выборки было потрачено на работу процессора в пользовательском режиме. Более 28% было потрачено на *запись* на диск (что удивительно!). Еще 13%

ушло на чтение с диска и приблизительно 6% времени отклика было потрачено на другие события ожидания.

Пример 5.7. Программа на Perl, формирующая профиль ресурсов по данным трассировки SQL для отдельного простого вызова базы данных Oracle (при отсутствии рекурсивных вызовов базы данных)

```
#!/usr/bin/perl

# $Header: /home/cvs/cvm-book1/sqltrace/prof-cid.pl,v 1.4 2003/03/20
23:32:32 cvm Exp $
# Cary Millsap (cary.millsap@hotsos.com)
# Copyright (c) 1999-2003 by Hotsos Enterprises, Ltd. All rights reserved.

# Создание профиля ресурсов для отдельного вызова базы данных.
# Usage: $0 file.trc

# Требуется ввод данных расширенной трассировки SQL (уровень 8 или 12),
# которые были заранее отфильтрованы так, чтобы содержать только один вызов
# базы данных (т.е. отдельные вызовы PARSE, EXEC, FETCH, UNMAP или SORT UNMAP
# без рекурсивных потомков) и строки WAIT, соответствующие данному вызову.
# Пример содержимого входного файла:
#
# WAIT #2: nam='db file sequential read' ela= 0 p1=2 p2=3240 p3=1
# WAIT #2: nam='db file sequential read' ela= 0 p1=2 p2=3239 p3=1
# FETCH
#2:c=213,e=998,p=2039,cr=100550,cu=5,mis=0,r=0,dep=0,og=4,tim=85264276

use strict;
use warnings;
my $cid;          # идентификатор курсора
my %ela;          # $ela{событие} содержит сумму значений ela для события
my $sum_ela = 0;   # сумма всех значений ela для событий
my $r = 0;        # время отклика для вызова базы данных
my $action = "(?:PARSE|EXEC|FETCH|UNMAP|SORT UNMAP)";
while (<>) {
    if (/^WAIT #(\d+): nam='([~']*)' ela=\s*(\d+)/i) {
        $ela{$2} += $3;
        $sum_ela += $3;
    }
    elsif (/^$action #(\d+):c=(\d+),e=(\d+)/i) {
        $ela{"total CPU"} += $2;
        $r = $3;
    }
    if (!defined $cid) {
        $cid = $1;
    } else {
        die "can't mix data across cursor ids $cid and $1" if $1 != $cid;
    }
}
$ela{"unaccounted-for"} = $r - ($ela{"total CPU"} + $sum_ela);
printf "%9s %6s %-40s\n", "Duration", "Pct", "Oracle kernel event";
printf "%8s- %5s- %-40s\n", "-x8", "-x5", "-x40;
```

```
printf "%.2fs %.1f%% %-40s\n", $ela{$_}/100, $ela{$_}/$r*100, $_ for sort
{ $ela{$b} <=> $ela{$a} } keys %ela;
printf "%8s- %5s- %-40s\n", "-x8, "-x5, "-x40;
printf "%.2fs %.1f%% %-40s\n", $r/100, 100, "Total response time";
```

Обратите внимание на строку «unaccounted-for» (неучтенное) в профиле ресурсов. Посмотрим, как она была получена. Общая продолжительность (по существу, *время отклика*) вызова выборки – это просто значение *e* для выборки. В исходных трассировочных данных это время учитывается двумя способами:

- Составляющая полного времени занятости процессора во времени отклика вызова выборки записывается в статистику *c* в самой строке FETCH.
- Составляющие времени системных вызовов во времени отклика вызова выборки записываются в статистики *ela* во все строки WAIT, относящиеся к выборке.

Так что «неучтенное» время оказывается равным значению Δ (дельта), вычисляемому по формуле:

$$e = c + \sum_{\text{db call}} ela + \Delta$$

Интересна эволюция учета времени отклика со времен версии Oracle 6. В версии 6 в файле трассировки отображалось время отклика для вызовов базы данных (*e*) и время занятости процессора (*c*), но это были *единственные* данные о времени отклика, которые выводило ядро Oracle. Первая модель времени отклика Oracle была чрезвычайно простой и формулировалась так: «время отклика равно времени использования процессора плюс еще что-то несущественное» или же так:

$$e = c + \Delta$$

Эта модель эффективна, когда величина Δ мала, но не заслуживает доверия при диагностике многих проблем со временем отклика, возникающих при больших значениях Δ . Во времена версии 6 большинство аналитиков были обучены считать, что большие значения Δ соответствуют времени, потраченному на вызовы чтения операционной системы. Это предположение часто оказывается неверным (как, например, в случае с только что рассмотренным профилем ресурсов), но все же оно помогло аналитикам решить многие проблемы производительности приложений. Причина успеха модели (несмотря на ее упрощенность) в том, что огромное количество проблем приложений Oracle связано с вызовами выборки, которые избыточно обращаются к кэшу буферов базы данных. Этим случаям соответствуют маленькие значения Δ , и модель $e = c + \Delta$ работает отлично.

Разработчики ядра Oracle одними из первых столкнулись с неадекватностью модели. Диапазон потенциально возможных причин для боль-

ших значений Δ так велик, что некоторые важные и сложные проблемы времени отклика просто невозможно решить без дополнительных оперативных данных. Представленные корпорацией Oracle в 1992 г. с выходом версии 7.0.12 *данные расширенной трассировки SQL* стали превосходным решением задачи. Расширенные трассировочные данные включают в себя строки `WAIT`, рассказывающие о том, сколько времени ядро Oracle провело за «ожиданием» исполнения ключевых событий. Новая, значительно усовершенствованная модель времени отклика, ставшая возможной благодаря новой функциональности расширенной трассировки SQL из версии Oracle 7.0.12, – это та модель, с которой мы работаем и по сей день:

$$e = c + \sum_{\text{db call}} ela + \Delta$$

С этого момента расширенная трассировка SQL смогла предоставить такие мощные диагностические возможности, о которых большинство аналитиков и не мечтало. Из тех немногих аналитиков, которые хотя бы осознают наличие интервала Δ , некоторые считают, что его существование объясняется недостатками расширенной трассировки, делающими данные недостоверными. На самом же деле, как вы увидите далее, в значении Δ скрывается весьма полезная информация. Существует несколько причин, вносящих свой вклад в ненулевое значение Δ , – об этом мы поговорим в главе 7. Понимание этих причин поможет вам в полной мере использовать всю диагностическую мощь предоставляемых Oracle данных расширенной трассировки SQL.

Отсчет времени

Для того чтобы извлечь из исходных трассировочных данных сведения о времени отклика, потребуется корректно интерпретировать хронологическую последовательность событий, обратившись к процессу, который мы называем «отсчет времени». Для отсчета времени требуются определенные знания о том, каким образом ядро Oracle обрабатывает информацию о времени:

- Значение поля `tim` в строке – это приблизительное время, в которое *завершилось* действие, представленное данной строкой.
- Значение поля `e` для вызова базы данных содержит общую фактическую продолжительность данного действия. Это значение включает в себя время, потраченное на использование процессора (значение поля `c`), и время, потраченное на события, произошедшие в ходе выполнения действия (сумма соответствующих значений полей `ela`).
- Рекурсивный SQL приводит к двойному учету. То есть значение поля `e` для вызова базы данных глубины `dep=n+1` уже включено в последующее значение `e` для вызова глубины `dep=n`.

- Не ждите совершенства от измерений времени. В файлах трассировки Oracle8i часто встречаются ошибки занижения или завышения числа на единицу. На первый взгляд кажется, что в файлах трассировки Oracle9i нередко содержатся существенно более значимые ошибки; однако, учитывая то, что измерение времени в Oracle9i ведется в микросекундах, эти ошибки не так страшны, как кажутся.

Версии Oracle 8 и ниже

Рассмотрим некоторые данные трассировки, на которых покажем, как выполняется отсчет времени в файлах трассировки, сформированных ядром Oracle8i и более ранних версий:

```
EXEC #13:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=2,og=3,tim=198360834
FETCH #13:c=0,e=0,p=0,cr=3,cu=0,mis=0,r=1,dep=2,og=3,tim=198360834
EXEC #12:c=2,e=4,p=0,cr=27,cu=0,mis=0,r=0,dep=1,og=4,tim=198360837
FETCH #12:c=2,e=10,p=10,cr=19,cu=4,mis=0,r=1,dep=1,og=4,tim=198360847
```

В табл. 5.7 показан соответствующий отсчет времени.

Таблица 5.7. Отсчет времени *tim* для вызовов базы данных Oracle8i

Строка (<i>k</i>)	<i>e</i>	Предполагаемое время $tim_k = tim_{k-1} + e_k$	Реальное время tim_k	Погрешность
1	0		198360834	
2	0	$198360834 + 0 = 198360834$	198360834	0
3	4	$198360834 + 4 = 198360838$	198360837	1
4	10	$198360837 + 10 = 198360847$	198360847	0

Здесь как раз встречается ошибка занижения/завышения на единицу – в строке 3, где наблюдается отличие прогнозируемого значения *tim* от фактического. Пусть подобные ошибки (±1 сотая секунды) вас не беспокоят. Ядра Oracle8i округляют значения времени до ближайшей сотой секунды, поэтому то, что кажется результатом сложения ...834 + 4, могло бы быть результатом сложения ...833,7048 + 3,5827, что после округления привело бы к имеющемуся значению ...837.

Следующий фрагмент файла трассировки Oracle8i содержит вызовы базы данных и события ожидания:

```
PARSE #494:c=4,e=5,p=11,cr=88,cu=0,mis=1,r=0,dep=2,og=0,tim=3864619462
WAIT #494: nam='latch free' ela= 2 p1=-2147434220 p2=95 p3=0
WAIT #494: nam='latch free' ela= 2 p1=-2147434220 p2=95 p3=1
EXEC #494:c=0,e=4,p=0,cr=0,cu=0,mis=0,r=0,dep=2,og=4,tim=3864619466
FETCH #494:c=0,e=0,p=0,cr=2,cu=0,mis=0,r=1,dep=2,og=4,tim=3864619466
```

В табл. 5.8 приведен отсчет времени для этих строк. Исследуя этот фрагмент, обратите внимание, что метки *k* присвоены только строкам вызовов базы данных (но не строкам WAIT). Можно отслеживать ожидаемый ход времени *tim* для событий ожидания, но не забывайте о том,

что значение *e* вызова базы данных уже включает в себя время, содержащееся в значениях *ela* событий ожидания, порожденных данным вызовом базы данных. Поэтому прогнозирование значения *tim_k* для вызова базы данных всегда основывается на значении *tim_{k-1}* из предыдущей строки вызова базы данных.

Таблица 5.8. Отсчет времени *tim* для вызовов базы данных и событий ожидания Oracle8i

Строка (k)	<i>e</i>	Предполагаемое время <i>tim_k</i> = <i>tim_{k-1}</i> + <i>e_k</i>	Реальное время <i>tim_k</i>	Погрешность
1	5		38646194 62	
	2	3864619462 + 2 = 3864619464		
	2	3864619464 + 2 = 3864619466		
2	4	3864619462 + 4 = 3864619466	38646194 66	0
3	0	3864619466 + 0 = 3864619466	38646194 66	0

Теперь рассмотрим более сложный фрагмент – проверим, насколько вы внимательны. Попробуйте объяснить, почему фактическое значение *tim* 198360796 в строке EXEC #8 настолько отличается от ожидаемого значения, 198360795 + 19 = 198360814?

```
EXEC #9:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=2,og=3,tim=198360795
FETCH #9:c=0,e=0,p=0,cr=3,cu=0,mis=0,r=1,dep=2,og=3,tim=198360795
EXEC #9:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=2,og=3,tim=198360795
FETCH #9:c=0,e=0,p=0,cr=3,cu=0,mis=0,r=1,dep=2,og=3,tim=198360795
EXEC #8:c=4,e=19,p=16,cr=162,cu=0,mis=0,r=0,dep=1,og=4,tim=198360796
FETCH #8:c=0,e=5,p=4,cr=4,cu=0,mis=0,r=1,dep=1,og=4,tim=198360801
FETCH #8:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=0,tim=198360801
FETCH #7:c=0,e=0,p=0,cr=2,cu=0,mis=0,r=1,dep=1,og=4,tim=198360801
EXEC #8:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=198360801
```

Ответ заключается в том, что вызов базы данных EXEC #8 – это рекурсивный родитель глубины *dep*=1 всех операций глубины *dep*=2, приведенных для курсора #9. Поэтому поле *e*=19 содержит все указанные значения *e* курсора #9, а также некоторые другие не упомянутые операции, потребляющие время, и не указанные. Вероятно, операция EXEC #8 началась вблизи значения *tim* 198360796 – 19 = 198369777. Между значениями *tim* ...777 и ...796 произошел ряд операций с глубиной *dep*=2, каждая из которых потребляла время, отсчитываемое таймером *tim*. Но помните, что все эти действия с глубиной *dep*=2 состоялись во время выполнения одной операции EXEC #8.

Версия Oracle 9

Увеличение разрешения статистик времени до микросекунд, появившееся в Oracle9i, – очень полезное усовершенствование. При переходе на версию Oracle9i в данных трассировки в первую очередь обращает

на себя внимание появление – благодаря использованию микросекунд – реальных данных, в тех случаях, когда Oracle8i передавал бы только множество нулей.



Однако нет сомнений в целесообразности использования данных расширенной трассировки SQL в системах версий 8 или даже 7. Описанный в этой книге метод оптимизации *надежно работает* и для диагностических данных, измеряемых в сотых долях секунды. В большей части реальных проектов улучшения производительности вывод в микросекундах – это просто роскошь.

Новое разрешение позволяет чуть точнее представить себе поведение ядра Oracle. В этом разделе будет рассмотрено несколько случаев, в которых улучшенная точность вывода ядра Oracle позволяет извлечь больше информации.

Отсчет времени для файла трассировки версии Oracle9i требует чуть больше терпения. Первое отличие, которое сразу бросается в глаза, – числа стали гораздо больше, и осуществлять этот подсчет в уме становится несколько сложнее. Например:

```
EXEC #1:c=0,e=1863,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=1017039275956134
FETCH #1:c=0,e=2566,p=0,cr=23,cu=0,mis=0,r=1,dep=1,og=4,tim=1017039275958821
FETCH #1:c=0,e=50,p=0,cr=0,cu=0,mis=0,r=1,dep=1,og=4,tim=1017039275959013
FETCH #1:c=0,e=34,p=0,cr=0,cu=0,mis=0,r=1,dep=1,og=4,tim=1017039275959155
FETCH #1:c=0,e=34,p=0,cr=0,cu=0,mis=0,r=1,dep=1,og=4,tim=1017039275959293
FETCH #1:c=0,e=35,p=0,cr=0,cu=0,mis=0,r=1,dep=1,og=4,tim=1017039275959433
```

Следующее, на что вы могли обратить внимание, – числа не сходятся. Посмотрите на большие числа в столбце «Погрешность» табл. 5.9.

Таблица 5.9. Отсчет времени *tim* для вызовов базы данных Oracle9i. Значения кажутся большими, но не забывайте о том, что на самом деле они очень даже маленькие, т. к. выражены в микросекундах

Строка (<i>k</i>)	<i>e</i>	Предполагаемое время $tim_k = tim_{k-1} + e_k$	Реальное время tim_k	Погрешность
1	1863		...956134	
2	2566	...956134 + 2566 = ...958700	...958821	-121
3	50	...958821 + 50 = ...958871	...959013	-142
4	34	...959013 + 34 = ...959047	...959155	-108
5	34	...959155 + 34 = ...959189	...959293	-104
6	35	...959293 + 35 = ...959328	...959433	-105

Большие значения в столбце ошибок кажутся ужасающими, если не вспоминать, что они измерены в микросекундах. Небольшие временные погрешности, подобные этой, всегда присутствовали в диагностических данных Oracle. Они были незаметны, когда измерение велось в сотых долях секунды. Теперь же, когда мы видим данные в микро-

секундах, становится очевидным влияние другого типа ошибок измерения времени отклика: вызовы к `gettimeofday` и `getrusage` расходуют фактическое время, которое сами при этом не измеряют (см. обсуждение *эффекта влияния измерителя* в главе 7).

Можно заметить, файлы трассировки Oracle9i имеют одну раздражающую особенность – не все строки трассировки расположены в порядке возрастания времени. В частности, значение `tim` секции `PARSING IN CURSOR` всегда находится «в будущем», относительно значения `tim` вызова базы данных, следующего сразу же за секцией `PARSING IN CURSOR`. Например:

```
PARSING IN CURSOR #1 len=32 dep=0 uid=5 oct=42 lid=5 tim=1033050389206593
hv=1197935484 ad='50f93654'
alter session set sql_trace=true
END OF STMT
EXEC #1:c=0,e=33,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=1033050389204497
```

Для того чтобы объяснить, почему так происходит, выполним трассировку событий ожидания ядра Oracle при помощи `strace` или аналогичного средства. Ядро Oracle завершает обработку вызова `EXEC` прежде, чем начинает вычислять данные для секции `PARSING IN CURSOR`. Но ядро *выводит* секцию `PARSING IN CURSOR` прежде, чем печатает строку `EXEC`. В этом и заключается причина неупорядоченности по времени.

Надо сказать, что ядро Oracle8i ведет себя точно так же. Вы просто не обращали на это внимания, т. к. передаваемая Oracle8i статистика в сотых долях секунды в большинстве случаев скрывала реальную информацию о хронологической последовательности. Статистика Oracle9i, выдаваемая в микросекундах, делает порядок событий очевидным.

Формулы отсчета времени

Познакомившись с несколькими примерами отсчета времени, можно уловить общую закономерность. Если вы еще помните, что не следует дважды учитывать наличие различных уровней рекурсивных вызовов базы данных, то согласитесь с тем, что значения полей `tim` и `e` связаны следующим отношением:

$$tim_{k+1} \approx tim_k + e_{k+1}$$

То есть значение поля `tim` следующей строки приблизительно равно значению поля `tim` текущей строки плюс значение поля `e` следующей строки. Формулу можно переписать так:

$$tim_k \approx tim_{k+1} - e_{k+1}$$

То есть значение поля `tim` текущей строки приблизительно равно значению поля `tim` следующей строки минус значение поля `e` следующей строки.

Конечно, в строке `WAIT` нет поля `tim`, поэтому, если надо оценить, каким будет для нее значение `tim`, необходимо сделать шаг вперед по от-

ношению к самому последнему доступному значению `tim`, используя такое соотношение:

$$tim_{k+1} \approx tim_k + ela_{k+1}$$

Эти формулы окажутся полезными, когда вы в главе 7 научитесь вводить поправку на погрешность сбора данных.

Опережающее атрибутирование

Итак, вы обнаружили в файле расширенной трассировки SQL событие ожидания, занимающее много времени. Следующая задача состоит в том, чтобы определить, какую операцию приложения можно изменить с тем, чтобы уменьшить расход времени. Эта задача легко решается с помощью данных расширенной трассировки SQL. Необходимо сопоставить длительность каждого события `WAIT #n` первому вызову базы данных для курсора `#n`, который *следует* за данной строкой `WAIT` в файле трассировки. Я называю этот метод *опережающим атрибутированием* (*forward attribution*). Опережающее атрибутирование помогает безошибочно определить, какая из SQL-команд приложения отвечает за возникновение каждого периода ожидания. Как это ни удивительно, но опережающее атрибутирование работает для событий, исполняемых как *внутри* вызовов базы данных, так и *между* ними.

Опережающее атрибутирование для событий внутри вызовов

Для событий, исполняемых *внутри* вызовов базы данных, обоснование опережающего атрибутирования вполне понятно. Строки записываются в файл трассировки в момент завершения соответствующего действия, поэтому события ожидания, исполняемые определенным вызовом базы данных, появляются в потоке трассировки *перед* строкой вызова. Следующий отрывок кода (фрагмент примера 5.3), показывает, каким образом ядро Oracle выдает в файл трассировки строки для событий ожидания, исполняемых внутри вызова:

```
=====
PARSING IN CURSOR #4 len=132 dep=1 uid=0 oct=3 lid=0 tim=1033064137929238
                                hv=3111103299 ad='517ba4d8'
select /*+ index(idl_ub1$ i_idl_ub11) */ piece#,length,piece from idl_ub1$
where obj#=:1 and part=:2 and version=:3 order by piece#
END OF STMT
PARSE #4:c=0,e=306,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=1033064137929139
EXEC #4:c=0,e=146,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=1033064137931262
❶ WAIT #4: nam='db file sequential read' ela= 13060 p1=1 p2=53903 p3=1
❷ WAIT #4: nam='db file sequential read' ela= 6978 p1=1 p2=4726 p3=1
❸ FETCH
#4:c=0,e=21340,p=2,cr=3,cu=0,mis=0,r=0,dep=1,og=4,tim=1033064137953092
```

В этом примере события `db file sequential read`, которым соответствуют строки ❶ и ❷, были исполнены в контексте вызова `FETCH`, представленного в строке ❸.

Опережающее атрибутирование для событий между вызовами

Для событий, исполняемых *между* вызовами базы данных, основание для применения опережающего атрибутирования не так очевидно. Следующий отрывок кода (фрагмент примера 5.4) помогает понять, как это работает. Из-за недостатков драйвера базы данных данное приложение *дважды*¹ передает каждый вызов разбора в базу данных. Обратите внимание на две идентичных секции `PARSING IN CURSOR`, разделенные парой строк для событий `to/from SQL*Net message`:

```
=====
PARSING IN CURSOR #9 len=360 dep=0 uid=26 oct=2 lid=26 tim=1716466757
hv=2475520707 ad='d4c55480'
INSERT INTO STAGING_AREA (TMSP_LAST_UPDT, OBJECT_RESULT, USER_LAST_UPDT,
DOC_OBJ_ID, TRADE_NAME_ID, LANGUAGE_CODE) values(TO_DATE('11/05/2001
16:39:06', 'MM/DD/YYYY HH24:MI:SS'), 'if ( exists ( stdphrase ( "PCP_MAV_1" )
), langconv ( "Incompatibility With Other Materials" ) + ": " ,
log_omission ( "Materials to Avoid: " ) )', 'sa', 222, 54213, 'NO_LANG')
END OF STMT
PARSE #9:c=0,e=0,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=1716466757
❶WAIT #9: nam='SQL*Net message to client' ela= 0 p1=1413697536 p2=1 p3=0
❷WAIT #9: nam='SQL*Net message from client' ela= 3 p1=1413697536 p2=1 p3=0
=====
PARSING IN CURSOR #9 len=360 dep=0 uid=26 oct=2 lid=26 tim=1716466760
hv=2475520707 ad='d4c55480'
INSERT INTO STAGING_AREA (TMSP_LAST_UPDT, OBJECT_RESULT, USER_LAST_UPDT,
DOC_OBJ_ID, TRADE_NAME_ID, LANGUAGE_CODE) values(TO_DATE('11/05/2001
16:39:06', 'MM/DD/YYYY HH24:MI:SS'), 'if ( exists ( stdphrase ( "PCP_MAV_1" )
), langconv ( "Incompatibility With Other Materials" ) + ": " ,
log_omission ( "Materials to Avoid: " ) )', 'sa', 222, 54213, 'NO_LANG')
END OF STMT
❸PARSE #9:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=1716466760
```

Несмотря на то, что вызовы разбора были незатратными (два значения `e=0` специально выделены в тексте), время отклика для всей пользовательской операции сильно увеличилось от огромного количества ненужных выполнений события `SQL*Net message from client`, на которые было потрачено в среднем 0,027 секунды на вызов. Общее влияние на вре-

¹ Множество драйверов предоставляют возможность выбрать такой способ работы. Дополнительный разбор служит для формирования «описания» разбираемого SQL, с тем чтобы драйвер мог генерировать более информативные сообщения об ошибках для разработчика. Даже Perl DBI по умолчанию ведет себя подобным образом. В Perl такое поведение можно отключить, задав атрибут `ora_check_sql=>0` в вызовах `prepare`.

мя отклика составило несколько минут для пользовательской операции, которая в целом должна была занять менее 10 секунд (см. раздел «Ситуация 3: большая длительность события SQL*Net» в главе 12). Для того чтобы избавиться от исполнения событий SQL*Net, приведенных в строках ❶ и ❷, можно удалить вызов разбора, представленный в строке ❸, которая следует за строками событий ожидания. В общем можно сказать, что вызов базы данных, «породивший» событие между вызовами, — это вызов базы данных, строка трассировки для которого следует за соответствующей строкой WAIT.

Подробный анализ файла трассировки

В начале главы я обещал привести подробное исследование файла трассировки из примера 5.2. Этот момент настал.

Любой файл трассировки SQL начинается с преамбулы, которая содержит информацию о файле: имя файла, версию ядра Oracle и различные характеристики системного окружения и анализируемого сеанса. Вот преамбула из примера 5.2:

```
/u01/oradata/admin/V901/udump/ora_9178.trc
Oracle9i Enterprise Edition Release 9.0.1.0.0 - Production
With the Partitioning option
JServer Release 9.0.1.0.0 - Production
ORACLE_HOME = /u01/oradata/app/9.0.1
System name:   Linux
Node name:    research
Release:      2.4.4-4GB
Version:      #1 Fri May 18 14:11:12 GMT 2001
Machine:      i686
Instance name: V901
Redo thread mounted by this instance: 1
Oracle process number: 9
Unix process pid: 9178, image: oracle@research (TNS V1-V3)
```

После преамбулы ядро Oracle выводит информацию, идентифицирующую сеанс, для которого выполняется трассировка, и время передачи первой строки трассировки:

```
*** SESSION ID:(7.6692) 2002-12-03 10:07:40.051
```

Следующая строка отображает сведения об имени модуля и операции, заданные клиентской программой (в нашем случае — SQL*Plus) при помощи процедур пакета DBMS_APPLICATION_INFO:

```
APPNAME mod='SQL*Plus' mh=3669949024 act='' ah=4029777240
```

Первое реальное действие, которое ядро отобразило в файле трассировки, — это исполнение команды ALTER SESSION. Ядро не передало данных о разборе команды ALTER SESSION, т. к. трассировка была включена только после завершения разбора. Для удобства ядро Oracle вывело сек-

цию, описывающую курсор для вызова исполнения, перед информацией о самом вызове EXEC. Вызов исполнения проделал очень небольшую работу. Значение e=1 показывает, что фактическая продолжительность вызова составила всего 1 микросекунду (0,000001 секунды).

```
=====
PARSING IN CURSOR #1 len=69 dep=0 uid=5 oct=42 lid=5 tim=1038931660052098
                                     hv=1509700594 ad='50d6d560'
alter session set events '10046 trace name context forever, level 12'
END OF STMT
EXEC #1:c=0,e=1,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=1038931660051673
```

После завершения выполнения команды ALTER SESSION ядро Oracle отправило результат обратно в клиентскую программу посредством записи в сокет, управляемый драйвером SQL*Net. На этот вызов ушло 5 микросекунд.

```
WAIT #1: nam='SQL*Net message to client' ela= 5 p1=1650815232 p2=1 p3=0
```

По завершении вызова записи ядро Oracle запускает чтение для этого же сокета (обратите внимание, что значения p1 для чтения и записи совпадают) и ожидает следующего запроса от клиентской программы. Приблизительно через 1262 микросекунды после выдачи вызова чтения этот вызов возвращается с новым запросом к ядру:

```
WAIT #1: nam='SQL*Net message from client' ela= 1262 p1=1650815232 p2=1 p3=0
```

Запрос, полученный в результате чтения сокета, — это фактически инструкция по разбору запроса «Hello, world». Как видите, прежде чем печатать статистики для PARSE, ядро любезно выводит секцию, начинающуюся с последовательности символов «=» и заканчивающуюся строкой END OF STMT, которая описывает разбираемый курсор. Собственно вызов разбора продолжается 214 микросекунд.

```
=====
PARSING IN CURSOR #1 len=51 dep=0 uid=5 oct=3 lid=5 tim=1038931660054075
                                     hv=1716247018 ad='50c551f8'
select 'Hello, world; today is '||sysdate from dual
END OF STMT
PARSE #1:c=0,e=214,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=1038931660054053
```

Следующий вызов базы данных — это EXEC, который указывает на исполнение разобранного ядром курсора. Непосредственно перед строкой EXEC расположена пустая секция BINDS, означающая, что программа SQL*Plus запросила операцию связывания, но связывать оказалось нечего. Общее время, потраченное на исполнение, составило 124 микросекунды.

```
BINDS #1:
EXEC #1:c=0,e=124,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=1038931660054311
```

После завершения вызова EXEC ядро отправляет результат обратно в клиентскую программу (т. е. в SQL*Plus). Запись в сокет занимает 5 микросекунд.

```
WAIT #1: nam='SQL*Net message to client' ela= 5 p1=1650815232 p2=1 p3=0
```

Сразу же после записи в сокет ядро переходит к операции выборки. Статистика для FETCH показывает, что общая продолжительность составила 177 микросекунд, возвращена была одна строка ($r=1$), что потребовало трех чтений кэша буферов базы данных, одно из которых – в согласованном режиме ($cr=1$), а два – в текущем ($cu=2$).

```
FETCH #1:c=0,e=177,p=0,cr=1,cu=2,mis=0,r=1,dep=0,og=4,tim=1038931660054596
```

Следующий вызов базы данных, отображенный в файле трассировки, – это еще одна выборка, имевшая место после чтения сокета SQL*Net, потребовавшего 499 микросекунд. Эта выборка не возвращает строку, ее продолжительность составляет всего 2 микросекунды.

```
WAIT #1: nam='SQL*Net message from client' ela= 499 p1=1650815232 p2=1 p3=0
```

```
FETCH #1:c=0,e=2,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=0,tim=1038931660055374
```

Затем ядро передает результат обратно в клиентское приложение посредством операции записи в сокет, выполненной за 4 микросекунды.

```
WAIT #1: nam='SQL*Net message to client' ela= 4 p1=1650815232 p2=1 p3=0
```

Отправив клиенту результат выборки, ядро ожидает следующего обращения к нему. И ожидание не будет долгим. Уже через 1261 микросекунду после инициации чтения сокета SQL*Net вызов чтения завершается.

```
WAIT #1: nam='SQL*Net message from client' ela= 1261 p1=1650815232 p2=1 p3=0
```

Инструкция, полученная ядром в результате вызова чтения сокета, приводит к закрытию курсора «Hello, world» и, в конечном счете, к завершению транзакции чтения. При закрытии курсора ядро выводит строку STAT, содержащую выбранный оптимизатором для данного запроса план выполнения. В данном случае запрос потребовал полного просмотра таблицы DUAL.

```
STAT #1 id=1 cnt=1 pid=0 pos=0 obj=221 op='TABLE ACCESS FULL DUAL'
XCTEND rlbk=0, rd_only=1
```

Как видите, даже для выполнения такого тривиального сеанса SQL*Plus ядру Oracle пришлось проделать достаточно большую работу. Что касается проблем производительности в реальных системах, можно представить себе, насколько более сложными будут их файлы трассировки. Но даже на простом примере видны некоторые действия, выполняемые *внутри* вызовов базы данных, и некоторые другие действия, выполняемые *между* вызовами базы данных. Эти действия представляют собой стандартные блоки, составляющие гораздо более объемные и сложные файлы трассировки, которые вы встретите в реальной жизни.

Упражнения

1. Какие строки WAIT примера 5.8 относятся к событиям ожидания, происходящим *внутри* вызовов базы данных, а какие – к событиям, происходящим *между* вызовами базы данных? Опишите, каким образом каждая из приведенных статистик c , e и ela входит в соотношение $e \approx c + \sum ela$.

Пример 5.8. Фрагмент файла данных расширенной трассировки SQL

```
...
Множество строк WAIT #1 пропущено для удобства
...
=====
PARSING IN CURSOR #1 len=253 dep=0 uid=18 oct=3 lid=18 tim=1024427939516845
hv=1223272015 ad='80cbc5b8'
...
Текст SQL пропущен для удобства
...
END OF STMT
PARSE #1:c=60000,e=55973,p=3,cr=44,cu=6,mis=1,r=0,dep=0,og=4,
tim=1024427939516823
EXEC #1:c=0,e=140,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=1024427939517471
WAIT #1: nam='SQL*Net message to client' ela= 15 p1=1650815232 p2=1 p3=0
WAIT #1: nam='db file sequential read' ela= 678 p1=7 p2=11146 p3=1
WAIT #1: nam='db file sequential read' ela= 815 p1=7 p2=11274 p3=1
FETCH #1:c=200000,e=259460,p=2,cr=12,cu=24,mis=0,r=1,dep=0,og=4,
tim=1024427939777318
WAIT #1: nam='SQL*Net message from client' ela= 1450 p1=1650815232 p2=1 p3=0
WAIT #1: nam='SQL*Net message to client' ela= 5 p1=1650815232 p2=1 p3=0
FETCH #1:c=0,e=339,p=0,cr=0,cu=0,mis=0,r=12,dep=0,og=4,tim=1024427939779621
WAIT #1: nam='SQL*Net message from client' ela= 7828 p1=1650815232 p2=1 p3=0
...
Строки STAT пропущены для удобства
...
=====
PARSING IN CURSOR #1 len=55 dep=0 uid=18 oct=42 lid=18 tim=1024427939789693
hv=3381932903 ad='80c9e33c'
alter session set events '10046 trace name context off'
END OF STMT
PARSE
#1:c=0,e=810,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=1024427939789677
```

2. Постройте схему, аналогичную представленной на рис. 5.3, которая отражает рекурсивные отношения между вызовами базы данных из примера 5.9. Вычислите вклад каждого вызова базы данных в величину e . Какого рода приложение могло бы выполнять приведенные операции?


```

=====
PARSING IN CURSOR #2 len=22 dep=1 uid=5 oct=3 lid=5 tim=1053274499535321
                                hv=4140187373 ad='521444c8'
SELECT count(*) from t
END OF STMT
PARSE #2:c=0,e=1003,p=0,cr=0,cu=0,mis=1,r=0,dep=1,og=3,
                                tim=1053274499535287
EXEC #2:c=0,e=115,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=3,
                                tim=1053274499535550

*** 2003-05-18 11:15:13.212
FETCH #2: c=3730000,e=13676722,p=127292,cr=127894,cu=260,mis=0,r=1,
                                dep=1,og=3,tim=1053274513212315
EXEC #1: c=3730000,e=13695999,p=127293,cr=127897,cu=264,mis=0,r=1,
                                dep=0,og=3,tim=1053274513212610
=====
PARSING IN CURSOR #4 len=52 dep=0 uid=5 oct=47 lid=5 tim=1053274513254792
                                hv=1697159799 ad='51f59e44'
BEGIN DBMS_OUTPUT.GET_LINES(:LINES, :NUMLINES); END;
END OF STMT
PARSE #4:c=0,e=149,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=3,
                                tim=1053274513254759
EXEC #4:c=0,e=38900,p=0,cr=0,cu=0,mis=0,r=1,dep=0,og=3,
                                tim=1053274513293822
STAT #2 id=1 cnt=1 pid=0 pos=0 obj=0 op='SORT AGGREGATE '
STAT #2 id=2 cnt=1 pid=1 pos=1 obj=31159 op='TABLE ACCESS FULL T '
XCTEND rlbk=0, rd_only=0

```

- 3. Выполните трассировку DDL-команды DROP TABLE. Сколько операций со словарем неявно выполняет ядро Oracle при удалении таблицы? Как изменится количество операций, если удаляемая таблица имеет индексы? А если для столбцов существуют гistogramмы? Как обстоит дело с ограничениями целостности? Что если таблица участвует в материализованном представлении или к ней применяется политика безопасности?**

6

Сбор данных расширенной трассировки SQL

Процесс сбора данных расширенной трассировки SQL сродни стрельбе по движущейся мишени. Долгое время корпорация Oracle оставалась верной мучительно медленному механизму Oracle Trace. Однако в документации к версии Oracle9i Release 2 открытым текстом сказано, что Oracle Trace подлежит замене средством SQL Trace (реализующим, по-видимому, *расширенную* трассировку SQL) [Oracle (2002)]. В стандартной документации Oracle имеется описание применения SQL Trace, но тем, кого интересуют *расширенные* возможности, придется поискать дополнительную информацию. Данная глава поможет решить эту проблему для версий Oracle с 7 по 9. Архитекторы ядра Oracle понимают важность данных о времени отклика, особенно если эти данные могут быть точно сопоставлены бизнес-операциям конечных пользователей. Посмотрим, какие возможности Oracle версии 10 позволяют облегчить задачу сбора нужных нам данных.



Имейте в виду, что при включенной трассировке SQL данные приложения записываются в файл в формате ASCII. Каждый файл трассировки SQL содержит SQL-текст приложения. Во многих трассировочных файлах присутствуют данные приложений. Доступ к этой информации может строго регламентироваться. Необходимо убедиться, что применение файлов трассировки SQL не угрожает конфиденциальности данных и не создает возможности их утечки.

Знакомство с приложением

Как известно из главы 3, *необходима* возможность трассировки операций, выполняемых строго определенным пользователем или пакет-

ным заданием на строго определенном временном интервале. Ниже в этой главе показано, что ядро Oracle версий 7, 8 и 9 позволяет управлять расширенной трассировкой SQL только на уровне *сеансов* Oracle. В зависимости от архитектуры приложения возможность управлять трассировкой лишь на уровне сеанса может создать серьезные трудности для процесса сбора диагностических данных. Поэтому, к сожалению, прежде чем приступить к трассировке приложения, надо понять его архитектуру.



Получение диагностических данных в корректно определенной области представляет собой наиболее сложную часть задачи диагностики производительности приложений, работающих с Oracle версий 7, 8 и 9. Как только такие данные получены, дальнейшие шаги не вызывают затруднений.

Для начала определим некоторые понятия. *Пользовательская операция* – это функционально законченная часть работы, выполняемой некоторым человеком. Именно *пользовательская операция* представляет интерес с точки зрения производительности для пользователя (а значит, и для вас тоже). Такая операция может требовать выполнения кода на любых (в том числе и на всех) узлах многозвенной архитектуры – таких как броузер клиента, сервер приложений, сервер БД и различные сетевые устройства.

В центре внимания этой книги находится узел сервера базы данных, т. к. именно он обладает инструментальными средствами, позволяющими наиболее эффективно диагностировать большинство проблем производительности. Пользовательская операция может затрагивать несколько *процессов* (или даже потоков) сервера базы данных, а может не затрагивать ни одного. *Процессом* называется объект операционной системы, представляющий собой экземпляр некой исполняемой программы. Процесс идентифицируется присвоенным ему операционной системой уникальным идентификатором (PID), управление процессами осуществляется средствами ОС. Например, приведенная команда Linux `ps` (вывести состояние процессов) показывает четыре процесса (8233, 8325, 8326 и 8327), созданные тремя различными программами (`ksh`, `ps` и двумя копиями `t`):

```
$ ps
  PID TTY          TIME CMD
 8233 pts/4    00:00:00 ksh
 8325 pts/4    00:00:00 t
 8326 pts/4    00:00:00 t
 8327 pts/4    00:00:00 ps
```

Нас в первую очередь будут интересовать два типа процессов ОС на сервере базы данных. Первый и самый главный из них – это *серверный* процесс Oracle, отвечающий за совместное использование памяти, доступ к файлам базы данных и выполняющий основную работу в боль-

шинстве систем, основанных на Oracle. Имена таких процессов обычно содержат подстроку «oracle». Показанная ниже команда Linux выводит список всех процессов, содержащих в таблице процессов подстроку «oracle» и не содержащих «grep»:

```
$ ps -ef | grep oracle | grep -v grep
oracle   756      1  0 Feb04 ?        00:00:19 ora_pmon_V816
oracle   758      1  0 Feb04 ?        00:00:04 ora_dbw0_V816
oracle   760      1  0 Feb04 ?        00:00:03 ora_lgwr_V816
oracle   762      1  0 Feb04 ?        00:00:43 ora_ckpt_V816
oracle   764      1  0 Feb04 ?        00:00:01 ora_smon_V816
oracle   766      1  0 Feb04 ?        00:00:00 ora_reco_V816
oracle  8834  8833  0 16:12 ?        00:00:00 oracleV816 (DESCRIPTION=(LO
oracle  8859  8858  0 16:13 ?        00:00:00 oracleV816 (DESCRIPTION=(LO
```

Обратите внимание, что эта команда вывела также данные обо всех *фоновых* процессах Oracle в моей системе (т. к. их владельцем является пользователь oracle).

Серверные процессы могут называться по-разному, в частности:

Серверные процессы

Теневые процессы

Процессы ядра

Процессы переднего плана

Второй интересный тип процессов, выполняемых на сервере, – это *клиентские* процессы, устанавливающие соединения с базой данных. Например, такие программы, как отчеты или пакетная загрузка, интенсивно обращающиеся к базе данных, часто запускают непосредственно на сервере БД. Такой способ хорошо подходит для любой клиентской программы, большая часть времени выполнения которой проходит в ожидании ответов на запросы к базе данных. В таком случае расходы на выполнение клиентской программы на сервере многократно окупаются уменьшением нагрузки на сеть, избавленную от массовой пересылки по SQL*Net сообщений между клиентом и серверными процессами oracle.

В качестве примеров клиентских программ Oracle можно привести:

sqlplus (SQL*Plus)

f60run (Oracle*Forms)

FNDLIBR (программа Oracle Financials Concurrent Manager)

PYUGEN (программа Oracle Human Resources)

Сеанс Oracle (или, далее в этой книге, просто *сеанс*) представляет собой конкретную последовательность обращений к базе данных, передаваемых через соединение между клиентским процессом и экземпляром Oracle. Сеанс имеет уникальный идентификатор, образованный сочетанием значений V\$SESSION.SID и V\$SESSION.SERIAL#. Например, следующая команда SQL*Plus выводит данные о девяти сеансах Oracle:

```
SQL> select sid, serial#, username, type from v$session;
```

SID	SERIAL#	USERNAME	TYPE
1	1		BACKGROUND
2	1		BACKGROUND
3	1		BACKGROUND
4	1		BACKGROUND
5	1		BACKGROUND
6	1		BACKGROUND
7	13	SYSTEM	USER
8	11	SYSTEM	USER
9	337	CVM	USER

9 rows selected.

Сбор данных не вызывает затруднений, если в пользовательской операции участвует только один клиентский процесс, один серверный процесс Oracle и один сеанс Oracle. К счастью, такая ситуация довольно часто встречается в случае таких проблем с производительностью, как долго выполняющиеся отчеты и пакетные задания. Сложность сбора данных возрастает, если пользовательская операция задействует несколько процессов или сеансов Oracle. Например:

Многопоточный сервер Oracle (MTS)

В многопоточной конфигурации несколько клиентских процессов совместно пользуются меньшим числом серверных процессов Oracle. Такая конфигурация сокращает количество процессов, необходимых для обслуживания приложения, для которого характерно большое количество подключенных пользователей, в основном находящихся в состоянии ожидания.

Пул соединений

В конфигурации с пулом соединений единственный процесс операционной системы (называемый *службой*) на промежуточном узле создает единственное соединение с Oracle и поддерживает единственный сеанс с единственным серверным процессом Oracle. Эта служба затем от имени множества пользователей выполняет вызовы к базе данных в рамках собственного единственного сеанса. Масштабируемость такой конфигурации для большого количества пользователей лучше, чем у многопоточного сервера.

Мы с коллегами в своей практике встречались с умопомрачительными комбинациями этих и других технологий, особенно в таких системах, где отдельная пользовательская операция обращается к службам, распределенным по нескольким базам данных. Как уже говорилось, сейчас самой сложной частью методов диагностики обычно становится сбор диагностических данных в корректно определенной области. Хорошо то, что как только найден способ сделать это для данной архитектуры, дальнейший сбор данных для нее заметно упрощается. Кроме

того, я полагаю, что архитектурные изменения, планируемые в Oracle версии 10, упростят процесс получения корректно выбранных данных по отдельной пользовательской операции.

Ключ к успешному сбору данных расширенной трассировки SQL в том, чтобы понять, как идентифицировать нужные сеансы Oracle. А для архитектур с пулом соединений – в правильной идентификации вызовов БД и событий ожидания, относящихся к диагностируемой пользовательской операции.

Включение расширенной трассировки SQL

Первый секрет синтаксиса расширенной трассировки SQL в Oracle находится в файле `$ORACLE_HOME/rdbms/mesg/oraus.msg`. Это файл сообщений об ошибках для ядра Oracle. Поиск первого вхождения подстроки «10000», расположенной в начале строки (например, командой `/^1000` в редакторе `vi`), приведет вас в следующий фрагмент этого файла:

```
/ Pseudo-error debugging events:
/   Error codes 10000 .. 10999 are reserved for debug event codes that are
/   not really errors.
/
// NLS_DO_NOT_TRANSLATE [10000,10999] - Tag to indicate messages should
// not be translated.
10000, 00000, "controlfile debug event, name 'control_file'"
// *Cause:
// *Action:
```

Разработчики ядра Oracle отвели диапазон кодов от 10000 до 10999 для отладочных событий, используемых ими для тестирования и отладки ядра.



Корпорация Oracle не предоставляет файл `oraus.msg` в дистрибутивах для Microsoft Windows. Найти его можно только в не-Windows дистрибутивах.

Одноточные описания кодов этих событий достаточно информативны. Из них можно узнать, что существуют отладочные события, позволяющие разработчикам ядра имитировать такие события, как сбой памяти или разнообразные виды повреждений файлов, изменять поведение компонентов, в частности, оптимизатора запросов, трассировать внутренние операции ядра (например, установку защепок). Отладочные события, интересные аналитику по производительности, включают в себя:

```
10032, 00000, "sort statistics (SOR*)"
10033, 00000, "sort run information (SRD*/SRS*)"
10053, 00000, "CBO Enable optimizer trace"
10079, 00000, "trace data sent/received via SQL*Net"
```



```
10104, 00000, "dump hash join statistics to trace file"  
10241, 00000, "remote SQL execution tracing/validation"
```

Среди более чем 400 отладочных событий имеется и то, которое включает расширенную трассировку SQL:

```
10046, 00000, "enable SQL statement timing"
```

Эта скромная неприметная возможность, погребенная в толще 16 000 строк недокументированного файла, и является одним из главных персонажей нашего повествования. Именно она дает возможность получить полный отчет о том, на что расходует время прикладная программа Oracle, когда пользователи ожидают от нее ответа.



До 10 версии Oracle все псевдоошибочные отладочные события официально не поддерживались, если только вы не действовали по прямым указаниям службы технической поддержки Oracle. Ниже в этой главе описана процедура DBMS_SUPPORT.START_TRACE_IN_SESSION, предоставляющая полностью поддерживаемый способ использования события 10046.

Трассировка собственного кода

Трассировка сеанса не вызывает затруднений, если имеется доступ на чтение и запись исходного кода, выполняющегося в трассируемом сеансе. Включение и выключение расширенной трассировки SQL требует лишь, чтобы ядро Oracle выполнило приведенные в примере 6.1 команды SQL. Первая строка гарантирует, что параметр временной статистики TIMED_STATISTICS для сеанса активирован, независимо от его значения, установленного для всего экземпляра. Если временная статистика Oracle не активирована, все значения e, c, ela и tim будут нулевыми, следовательно, бесполезными с точки зрения анализа времени отклика.

Пример 6.1. Код, включающий и выключающий расширенную трассировку SQL для сеанса

```
alter session set timed_statistics=true  
alter session set max_dump_file_size=unlimited  
alter session set tracefile_identifier='POX20031031a'  
alter session set events '10046 trace name context forever, level 8'  
/* здесь находится трассируемый код */  
alter session set events '10046 trace name context off'
```

Вторая строка запрещает ядру Oracle усекать файл трассировки без явного указания. Параметр MAX_DUMP_FILE_SIZE позволяет администратору базы данных Oracle ограничить размер создаваемых сеансами трассировочных файлов. Такая возможность предусмотрена для того, чтобы аналитики по производительности случайно не переполнили файловую систему, указанную параметрами USER_DUMP_DEST и BACKGROUND_DUMP_DEST. Однако, сделав это ограничение слишком строгим, можно допустить досадную ошибку, способную привести к дорогостоящим

последствиям в проекте, посвященном повышению производительности.¹ Последнее, что хотел бы увидеть аналитик, три недели тщательно готовившийся к трассировке большого ежемесячного пакетного задания, — это обрубок трассировочного файла, заканчивающийся строкой:

```
*** DUMP FILE SIZE IS LIMITED TO 1048576 BYTES ***
```

Возможность отмены ограничения на размер дампа влечет за собой ответственность за поддержание порядка в файловой системе, в которую будут записаны трассировочные файлы. Если в файловой системе, в которую пишет ядро Oracle, возникает ошибка переполнения, трассировочный файл будет усечен. В конце файла расположится нечто примерно следующее:

```
WAIT #42: nam='db file sequential read' ela= 17101 p1=10 p2=2213 p3=1  
WAIT #42: nam='db file se
```

Имейте в виду, что некоторые версии Oracle (в особенности Oracle8i для Microsoft Windows) не поддерживают ключевое слово UNLIMITED. В таких случаях достаточно указать в качестве значения параметра MAX_DUMP_FILE_SIZE большое целое число. В 32-разрядных реализациях Oracle, имеющихся в нашей лаборатории, максимально допустимое значение составляет $2^{31} - 1 = 2\,147\,483\,647$. Заметьте также, что параметры TIMED_STATISTICS и MAX_DUMP_FILE_SIZE могут устанавливаться для сеанса, начиная с Oracle 7.3. В более ранних версиях единственным способом задать значение для любого из них для определенной сессии была установка данного значения для всего экземпляра.

Возможно, когда-нибудь и параметр USER_DUMP_DEST можно будет устанавливать на уровне сеанса. Это будет полезно, т. к. позволит при выборе места для трассировочных файлов руководствоваться соображениями экономии дискового пространства, производительности или просто удобства доступа. Документация для Oracle 9.2 утверждает, что параметр USER_DUMP_DEST может быть установлен для сеанса [Oracle (2002)]. Однако это не соответствует действительности, по крайней мере, в Oracle 9.2.0.1.0 для MS Windows.

Команда в третьей строке примера 6.1 помещает строку «POX20031031a» в имя результирующего файла трассировки (такая возможность введена в версии 8.1.7). Использование такого уникального идентификатора в имени трассировочного файла впоследствии облегчит поиск файла с собранной нами информацией. Подойдет любой уникальный идентификатор. В этом примере выбрано мнемоническое имя, означающее выполнение «а» отчета «POX», запущенного 31 октября 2003 года.

Четвертая строка примера 6.1 запускает сам механизм расширенной трассировки SQL, вынуждая ядро Oracle выводить статистику в файл трассировки процесса ядра. Обратите внимание, в этом примере меха-

¹ В Oracle9 по умолчанию установлено значение UNLIMITED.

низм расширенной трассировки SQL активирован с уровнем трассировки 8. Для выключения трассировки добавлено ключевое слово OFF, устанавливающее уровень трассировки в 0. Уровни трассировки описаны в табл. 6.1.

Таблица 6.1. Уровни трассировки псевдоошибочного отладочного события Oracle с номером 10046

Уровень	Маска	Функция
0	0000	Не выводить статистику.
1	0001	Выводить строки ***, APPNAME, PARSING IN CURSOR, PARSE ERROR, EXEC, FETCH, UNMAP, SORT UNMAP, ERROR, STAT и XCTEND.
2	0010	Предположительно идентично уровню 1.
4	0100	Выводить секции BINDS в дополнение к данным уровня 1.
8	1000	Выводить строки WAIT в дополнение к данным уровня 1.
12	1100	Выводить данные уровней 1, 4 и 8.

Несмотря на то, что трассировка может быть отключена явно, зачастую этого лучше не делать. Параметр трассировки сеанса оканчивает жизнь вместе с сеансом, поэтому, когда пользователь отсоединяется от Oracle, трассировочный файл аккуратно закрывается. Окончание трассировки по завершении сеанса – лучшая гарантия того, что все строки STAT для сеанса выведены в трассировочный файл. Причина этого объясняется в главе 5. Разумеется, если трассируется постоянное соединение с Oracle, подобное тому, которое используется процессами, сконфигурированными как «linked internal» в приложении Oracle Applications Concurrent Manager, то придется отключать трассировку явно. К счастью, отсутствующие строки STAT достаточно легко воспроизводятся с помощью команды EXPLAIN PLAN или нового фиксированного представления V\$SQL_PLAN (доступного в версии 9).

Трассировка чужого кода

Можно трассировать *любые* сеансы Oracle по своему выбору, включая фоновые. Вам кажется, что запись в файлы БД выполняется слишком медленно? Выполните трассировку DBWR и разберитесь. Считаете, что запись в оперативные журнальные файлы занимает слишком много времени? Выполните трассировку LGWR и выясните причину. А для того чтобы узнать, чего стоит ядру Oracle автоматическое слияние табличных пространств, выполните трассировку SMON – и узнаете.



Не пытайтесь выполнить *расширенную* трассировку PMON – это может привести к сбою экземпляра (ошибка Oracle 2329767, предположительно, исправленная в Oracle версии 10). К счастью, имеется очень мало разумных причин, по которым можно было бы *захотеть* трассировать PMON.

Включение трассировки триггером сеанса

Если возникает необходимость трассировки программы, к исходному коду которой нет доступа на чтение и запись, план несколько усложняется. Как правило, это не намного сложнее, достаточно иметь некоторое воображение. Можно, например, использовать триггер AFTER LOGON (появившийся в версии 8.1) для включения трассировки с уровнем 8 для любого сеанса с заданным атрибутом. В примере 6.2 показано создание триггера, включающего трассировку всех сеансов, в которых имя пользователя Oracle содержит суффикс `_test`.

Пример 6.2. Создание триггера, включающего трассировку всех сеансов, создаваемых пользователями, чьи имена содержат суффикс `_test`

```
create or replace trigger trace_test_user after logon on database
begin
  if user like '%_test' escape '\\' then
    execute immediate 'alter session set timed_statistics=true';
    execute immediate 'alter session set max_dump_file_size=unlimited';
    execute immediate
      'alter session set events ''10046 trace name context forever,
        level 8''';
  end if;
end;
/
```

Особенности реализации подобных триггеров могут существенно меняться от приложения к приложению. Здесь важно хорошо понимать работу приложения, чтобы творчески подойти к выбору способа активации расширенной трассировки SQL для выбранного сеанса.

Включение трассировки из другого сеанса

В состав Oracle входит пакет процедур, позволяющих управлять атрибутами сеансов, не устанавливая с ними соединения. Первая задача заключается в том, чтобы идентифицировать сеанс, подлежащий трассировке. Большинству администраторов баз данных хорошо знакомы способы поиска значений SID и SERIAL# (из фиксированного представления V\$SESSION) для процессов, принадлежащих управляемым ими приложениям. В примере 6.3 показана соответствующая команда SQL.

Пример 6.3. Команда SQL, показывающая атрибуты пользовательского сеанса. Имя пользователя передается через переменную связывания :uname

```
select
  s.sid db_sid,
  s.serial# db_serial,
  p.spid os_pid,
  to_char(s.logon_time, 'yyyy/mm/dd hh24:mi:ss') db_login_time,
  nvl(s.username, 'SYS') db_user,
  s.osuser os_user,
  s.machine os_machine,
```

```

      nvl(decode(
        instr(s.terminal, chr(0)), 0,
        s.terminal, substr(s.terminal, 1, instr(s.terminal, chr(0))-1)
      ), 'none') os_terminal,
      s.program os_program
from
  v$session s,
  v$process p
where
  s.paddr = p.addr
  and s.username like upper(:uname)

```

Приложение может заметно упростить задачу идентификации сеанса, предоставив пользователю какую-либо характерную информацию о сеансе. Представьте себе приложение, способное вывести значения `V$SESSION.SID` и `V$SESSION.SERIAL#` непосредственно на экранную форму. Это очень поможет пользователю, когда он будет объяснять аналитику, как тому определить сеанс, требующий углубленного анализа производительности.

Пакет `DBMS_APPLICATION_INFO` содержит три полезные процедуры, `SET_MODULE`, `SET_ACTION` и `SET_CLIENT_INFO`, способные помочь в идентификации сеансов Oracle. Каждая из процедур записывает значение в фиксированное представление `V$SESSION` для сеанса, выполняющего процедуру. Атрибуты `MODULE`, `ACTION` и `CLIENT_INFO` образуют иерархию, хорошо подходящую для идентификации пользовательских операций. Например, приложение пользователя `Nikolas Alexander` может установить следующие значения:

```

dbms_application_info.set_module('Accounts Payable')
dbms_application_info.set_action('Pay Invoices')
dbms_application_info.set_client_info('Nikolas Alexander')

```

После того как приложение таким образом «отметилось» при помощи процедур пакета `DBMS_APPLICATION_INFO`, не составляет труда найти сеанс определенного клиента, все сеансы Oracle, выполняющие определенную пользовательскую операцию, и даже все сеансы, участвующие в выполнении операций определенного модуля. Например, приведенный ниже запрос возвращает идентификационные данные всех сеансов, выполняющих операцию «Pay Invoices» модуля «Accounts Payable»:

```

select session, serial#
from v$session
where v$session.module = 'Accounts Payable'
  and v$session.action = 'Pay Invoices'

```



Употребление атрибутов `MODULE`, `ACTION` и `CLIENT_INFO` в Oracle9 связано с одной трудностью: запись этих атрибутов создает дополнительную нагрузку на базу данных (это не только операции ядра Oracle, но и передача данных по сети). Для небольших пользовательских операций эта дополнительная нагрузка составляет значительную долю от общего объема вычислений.

Если для сеанса, требующего трассировки, получены значения SID и SERIAL#, включение трассировки не вызывает затруднений. В примере 6.4 показано, как с помощью пакета DBMS_SYSTEM активировать параметр TIMED_STATISTICS для заданного сеанса и как установить для него значение параметра MAX_DUMP_FILE_SIZE. Даже если пакетом DBMS_SYSTEM почему-либо нельзя воспользоваться, начиная с версии 7.3, можно, не останавливая всю систему, управлять этими параметрами на уровне экземпляра посредством команд ALTER SYSTEM.

Пример 6.4. Управление параметрами сеанса, определяемого значениями :sid и :serial

```
sys.dbms_system.set_bool_param_in_session(
    :sid, :serial,
    'timed_statistics', true
)
sys.dbms_system.set_int_param_in_session(
    :sid, :serial,
    'max_dump_file_size', 2147483647
)
```

Есть несколько способов включения расширенной трассировки заданного сеанса. Два из них показаны в примерах 6.5 и 6.6. Корпорация Oracle рекомендует применять пакет DBMS_SUPPORT вместо DBMS_SYSTEM, когда есть возможность выбора (комментарий 62294.1 от Oracle *Meta-Link*). Однако файлы dbmssupp.sql и prvt supp.plb поставляются не во всех дистрибутивах. Если в системе отсутствует DBMS_SUPPORT, не отчаивайтесь. Мы с коллегами в своих проектах повышения производительности сотни раз использовали DBMS_SYSTEM.SET_EV без каких-либо неприятных последствий. Мои знакомые из службы технической поддержки Oracle сообщили мне, что все равно процедура DBMS_SUPPORT.START_TRACE_IN_SESSION реализована как вызов SET_EV.



Применение START_TRACE_IN_SESSION более безопасно в силу того, что она исключает риск опечатки при указании события 10046. К несчастью, случайная ошибка при вводе номера события может иметь катастрофические последствия.

Пример 6.5. Включение расширенной трассировки SQL уровня 8 процедурой START_TRACE_IN_SESSION для сеанса, определяемого значениями :sid и :serial

```
sys.dbms_support.start_trace_in_session(
    :sid, :serial,
    waits=>true, binds=>false
)
/* трассируемый код выполняется в этот период времени */
sys.dbms_support.stop_trace_in_session(
    :sid, :serial
)
```



Не используйте для включения расширенной трассировки процедуру `DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION`, т. к. она способна активизировать только трассировку уровня 1. Для активизирования расширенной трассировки она не подходит.

Пример 6.6. Включение расширенной трассировки SQL уровня 8 процедурой `SET_EV` для сеанса, определяемого значениями `:sid` и `:serial`

```
sys.dbms_system.set_ev(:sid, :serial, 10046, 8, '')  
/* трассируемый код выполняется в этот период времени */  
sys.dbms_system.set_ev(:sid, :serial, 10046, 0, '')
```

Поиск файлов трассировки

Завершив трассировку сеанса, надо решить следующую задачу: выяснить, в какой файл (или файлы) были записаны данные. Каждый процесс ядра Oracle создает собственный трассировочный файл, следовательно, в зависимости от конфигурации приложения данные исследованного сеанса могут находиться как в одном, так и в нескольких файлах. Например, многопоточный сервер MTS может вывести данные трассировки одного сеанса в два или больше трассировочных файла. Первым делом надо определить, в каком каталоге находятся эти файлы. Это не трудно, поскольку есть всего два варианта. Ответом будет значение параметра `USER_DUMP_DEST` или `BACKGROUND_DUMP_DEST`.¹

Теперь надо правильно выбрать файл (или файлы) в этом каталоге. Если имя файла было помечено уникальным идентификатором посредством установки атрибута сеанса `TRACEFILE_IDENTIFIER`, то поиск файла не вызовет затруднений. Достаточно найти в каталоге файлы, имена которых содержат указанный идентификатор. Если же пометить имена файлов не удалось, например из-за того, что включалась трассировка стороннего кода из другого сеанса, то задача немного усложняется.

Имена трассировочных файлов

Первая сложность связана с тем, что разные группы портирования в корпорации Oracle приняли разные соглашения об именовании трассировочных файлов. В табл. 6.2 приведены некоторые из встречавшихся нам имен. Из-за того, что нет межплатформенного стандарта наименований, может показаться затруднительным создание универсального инструмента, способного предсказать имя трассировочного файла для заданного сеанса. На самом деле все не так плохо, если в вашей сети используется лишь несколько конфигураций. Достаточно выяснить, по какому шаблону ядро Oracle именует свои трассировочные файлы,

¹ Мне доводилось слышать о случающейся время от времени ошибке, вследствие которой ядро Oracle игнорирует параметры, указывающие каталог для дампов, и записывает трассировочные файлы в `$ORACLE_HOME/rdbms/log`.

и можно предсказывать, какие имена оно создаст. Так, на нашем исследовательском Linux-сервере трассировочные файлы имеют имена вида ora_SPID.trc, где SPID – это значение поля V\$PROCESS.SPID для сеанса.

Таблица 6.2. Соглашения об именовании трассировочных файлов меняются в зависимости от группы, выполняющей портирование ядра, и версии Oracle

Имя трассировочного файла	Версия Oracle	Операционная система
ora_1107.trc	8.1.6.0.0	Linux 2.2.15
ora_31641.trc	9.0.1.0.0	Linux 2.4.4
ora_31729.trc	8.1.5.0.0	OSF1 V4.0
proa021_ora_9452.trc	8.0.5.2.1	SunOS 5.6
cdap_ora_17696.trc	9.2.0.1.0	SunOS 5.8
ora_176344_crswp.trc	8.1.6.3.0	AIX 3
MERKUR_S7_FG_ORACLE_013.trc	8.1.7.0.0	OpenVMS 7.2-1
ora_3209_orapatch.trc	8.1.6.3.0	HP-UX B.11.00
ORA01532.TRC	8.1.7.0.0	Windows 2000 V5.0
v920_ora_1072.trc	9.2.0.1.0	Windows 2000 V5.1

Простые клиент–серверные приложения

Даже в нынешний век сложных многозвенных архитектур многие программы выполняются в простом клиент-серверном режиме, в особенности пакетные задания. Выделяя компонент приложения для тестирования, вы наверняка не откажете себе в такой роскоши. В подобной конфигурации любой сеанс Oracle создает единственный трассировочный файл, содержащий данные только этого сеанса (см. рис. 6.1).

Отсутствие межплатформенного стандарта именования трассировочных файлов вызвало у нашей компании затруднения при создании переносимого инструмента для поиска таких файлов. Мы рассматривали вариант с пополняемой таблицей шаблонов имен (т. е. регулярных выражений), которую можно было бы исправлять по мере переноса Oracle на новые платформы и выхода новых версий. Но мы пришли к выводу, что при поддержании актуальности такой таблицы неизбежно возникло бы множество ошибок. В результате мы остановились на таком алгоритме:

1. По заданным идентификатору и порядковому номеру выбранного сеанса (значения V\$SESSION.SID и V\$SESSION.SERIAL#) определить системный идентификатор (SPID) соответствующего серверного процесса. Этот идентификатор содержится в поле V\$PROCESS.SPID, соответствующем выбранному сеансу, и может быть получен при помощи соединения, аналогичного приведенному в примере 6.3.

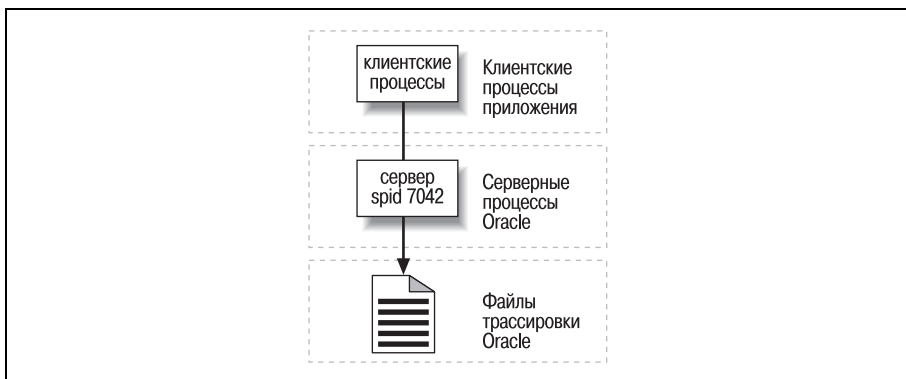


Рис. 6.1. В простых клиент-серверных конфигурациях сеансу соответствует один серверный процесс и, следовательно, один трассировочный файл

2. Найти, в каком каталоге располагаются файлы трассировки. Этот каталог определяется значением параметра `USER_DUMP_DEST`, если `V$SESSION.TYPE='USER'`, или параметра `BACKGROUND_DUMP_DEST`, если `V$SESSION.TYPE='BACKGROUND'`.
3. Отсортировать содержимое каталога по убыванию даты изменения файла (например, командой `ls -lt` в UNIX). Имейте в виду, что время изменения файла (атрибут `mtime`) обычно указывается с точностью до секунды. Поэтому, если несколько файлов были созданы в течение одной секунды, сравнение значений `mtime` не даст ответа на вопрос, какой из них был создан последним.
4. Для каждого файла из полученного списка, значение `mtime` которого превышает время начала сбора данных (можно задать и более точное условие, но сравнение времени изменения со временем начала сбора данных – это более консервативный подход):
 - а) Найти в файле *последнюю* преамбулу. В особенности это касается платформ Microsoft Windows, где ядро Oracle часто стремится повторно использовать трассировочные файлы, *дописывая* новые данные к уже существующим. (Поэтому в одном файле может быть несколько преамбул.)
 - б) Найти в преамбуле строку, содержащую подстроку «pid» (для Unix и OpenVMS) или «thread id» (для Windows). Преамбулу составляют все строки вплоть до той, которая начинается с подстроки «***».
 - в) Если число, следующее за подстрокой «pid» или «thread id», совпадает с идентификатором SPID выбранного сеанса, нужный файл найден, и поиск заканчивается.

Если в файлах из выбранного списка отсутствует подходящий идентификатор процесса, поиск также заканчивается: искомые файлы отсутствуют.

Для проекта Sparky, о котором можно прочитать на сайте <http://www.hotsos.com>, мною был написан переносимый код на Perl, выполняющий действия, аналогичные описанным.

Такой метод просмотра содержимого файлов может показаться не очень элегантным, особенно если в данной организации используется только одна или две операционные системы. Однако достоинство приведенного алгоритма в его надежности для различных платформ и различных версий программного обеспечения Oracle. Алгоритм хорошо масштабируется в зависимости от количества файлов трассировки, находящихся в каталоге. Несколько хуже он масштабируется в случае, когда файлы трассировки имеют очень большие размеры и содержат по несколько преамбул.

Параллельное выполнение (Oracle PX)

При параллельном выполнении Oracle (Oracle Parallel eXecution – PX) серверный процесс Oracle порождает два или более дочерних процесса (называемых *подчиненными процессами PX*) для выполнения параллельного чтения и параллельной сортировки. Подчиненные процессы PX наследуют атрибуты трассировки от координатора запроса. Следовательно, включение расширенной трассировки SQL для сеанса, который использует функциональность PX, приведет к формированию нескольких файлов трассировки. Задача заключается в том, чтобы опознать и проанализировать *все* нужные файлы. Обычно она решается достаточно просто – необходимо оценить время изменения файлов трассировки, сформированных последними. Для запросов, использующих степень параллелизма p , количество n соответствующих файлов трассировки будет находиться в диапазоне $1 \leq n \leq 2p + 1$ для каждого вовлеченного экземпляра.

Многопоточный сервер Oracle

Применение многопоточного сервера Oracle (MTS) несколько усложняет поиск данных трассировки. MTS делает возможным использование коммутируемых соединений, что приводит к созданию отношения «один-ко-многим» между сеансом и серверными процессами Oracle, которые обслуживают вызовы базы данных, выполненные сеансом (рис. 6.2). Соответственно, данные трассировки для одного сеанса могут оказаться разбросанным по двум или более трассировочным файлам. Ядро Oracle предоставляет полные сведения по идентификации сеанса и временные метки каждый раз, когда сеанс мигрирует в новый серверный процесс (а следовательно, и в новый файл трассировки). Создать логический эквивалент единого файла трассировки для конкретного сеанса несложно. Рассмотренный ранее способ поиска файлов трассировки претерпевает следующие изменения:

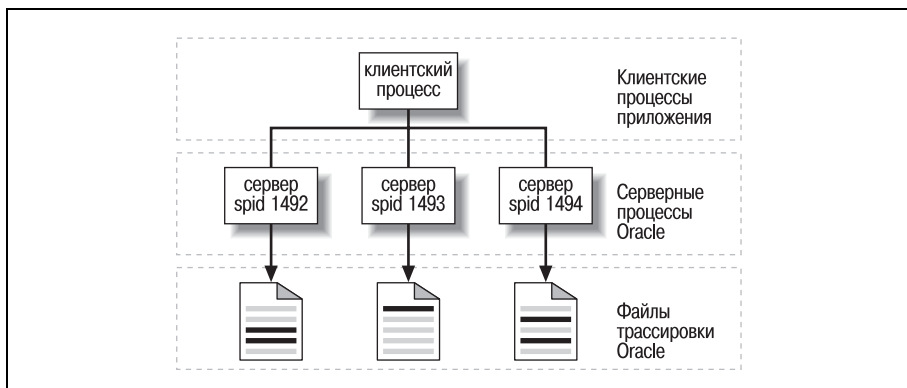


Рис. 6.2. Многопоточный сервер Oracle использует отношение один-ко-многим для клиентского и серверных процессов, поэтому данные о сеансе могут направляться в несколько файлов трассировки

- В зависимости от версии Oracle, совместно используемые серверные файлы трассировки могут храниться в параметре `BACKGROUND_DUMP_DEST` (мы с коллегами видели это на некоторых платформах для версий 7 и 8) или же в `USER_DUMP_DEST` (наблюдается в версии 9).
- Вместо того чтобы завершать поиск, обнаружив один файл трассировки с нужной информацией, идентифицирующей сеанс, необходимо продолжить просмотр *всех* файлов трассировки с подходящим временем изменения.
- Определив все файлы, которые содержат нужные данные трассировки, необходимо отбросить данные, относящиеся не к вашему сеансу, а затем объединить оставшиеся. Сначала избавляемся от сегментов данных трассировки, относящихся к сеансам, отличным от исследуемого. Чтобы определить, какие секции следует сохранить, достаточно просто просмотреть строки идентификации сеансов, которые начинаются с символов `***`. Затем объединяем оставшиеся участки данных трассировки по возрастанию времени. Это тоже несложно, т. к. строки `***` содержат и значения времени. В результате получаем «виртуальный файл трассировки», содержащий сведения только для исследуемого сеанса. Эту операцию можно выполнить вручную в многооконном текстовом редакторе, а можно приобрести специальное средство, которое все сделает за вас. Наша команда на *hotsos.com* создала подобный продукт и предлагает его на коммерческой основе.

Пул соединений

Как я уже говорил, организация пула соединений – это замечательная возможность, призванная уменьшить количество вызовов *подключения* к базе данных и *отключения* от нее. Степень сложности диагно-

стики приложений, работающих с пулами соединений, полностью определяется реализованными в них возможностями. Если приложение позволяет идентифицировать вызовы базы данных, сделанные от имени пользовательской операции, то работа по сбору данных будет простой. К сожалению, многие приложения, работающие с пулами соединений, не обеспечивают такой возможности. Надеюсь, Oracle версии 10 упростит оснащение приложений такими средствами на несколько следующих лет.

Проблемы диагностики производительности для пулов соединений возникают тогда, когда сервер приложений «утаивает» личность конечного пользователя от базы данных. Один сеанс разделяют между собой несколько пользователей, из-за чего невозможно по одному лишь файлу трассировки определить, чьи действия вызвали появление некоторой строки в трассировочных данных (рис. 6.3).



Лучшим общим решением проблемы диагностики приложений, работающих с пулом соединений, является такая архитектура приложения, которая делает возможным включение расширенной трассировки SQL для действий каждого отдельного пользователя.

Если приложение ничем не может помочь в трассировке команд SQL отдельного пользователя, не отчаивайтесь. Конечно же, существуют и другие пути, ведущие к успеху. Рассмотрим такой пример: пользо-

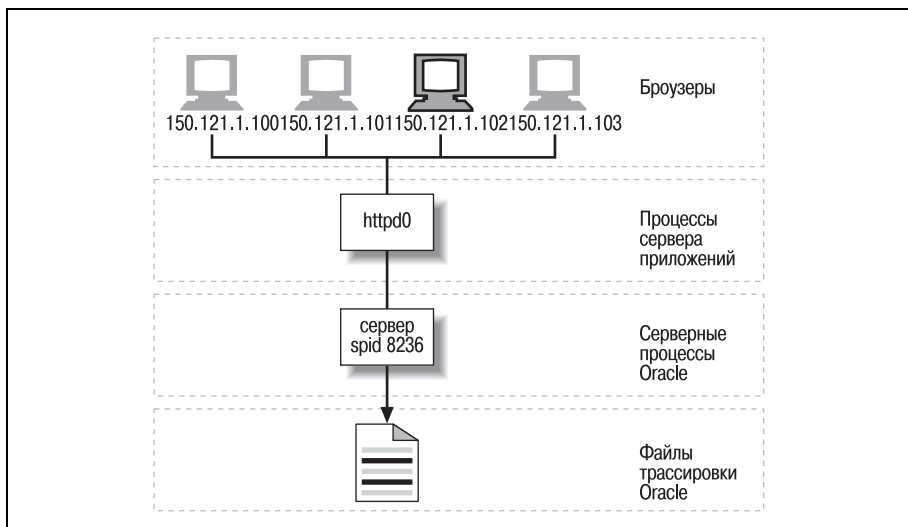


Рис. 6.3. Архитектура, основанная на пуле соединений. Если среднее звено не отслеживает соответствие конечных пользователей вызовам базы данных, то нельзя определить, какой из пользователей вызвал появление той или иной строки данных трассировки

ватель Нэнси, имеющая IP-адрес 150.121.1.102, сообщила об ухудшении производительности приложения ввода заказов, основанного на пуле соединений (архитектура приложения приведена на рис. 6.3). Приложение никак не способствует выделению данных расширенной трассировки SQL для Нэнси.

Есть простой способ: временно запретить работу с системой всем пользователям, кроме Нэнси. Включить расширенную трассировку процесса, обслуживающего Нэнси, и позволить ей выполнить ее медленные операции. Когда Нэнси сделает все, что нужно, отключить трассировку и разрешить всем пользователям вернуться в систему. Этот способ весьма эффективен в отдельных случаях, но надо сказать, что в дополнение к очевидному вмешательству в ход процессов бизнеса, он имеет и серьезный диагностический недостаток. Если ухудшение производительности, о котором сообщила Нэнси, было вызвано конкуренцией с другими сеансами, то данные, собранные предложенным способом, никак не будут указывать на основной источник неприятностей.

Более эффективный обходной маневр – временное изменение архитектуры, которое бы изолировало сеанс Нэнси. Один из способов реализации такого изменения изображен на рис. 6.4, где изоляция сеанса Нэнси обеспечена за счет предоставления ей собственного процесса на сервере приложений и отдельного выделенного серверного процесса Oracle. Для того чтобы реализовать такой переход, можно, например, назначить Нэнси специальный «идентификатор службы» (аналог специального псевдонима TNS для уровня служб приложения), который

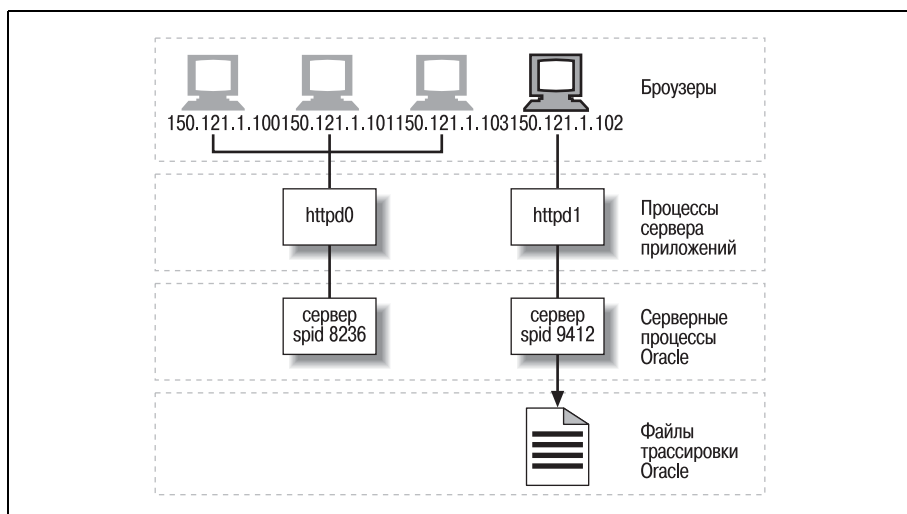


Рис. 6.4. Если изолировать действия пользователя так, чтобы в назначенный ему файл трассировки не попадали данные об операциях других пользователей, то сбор диагностических данных становится таким же простым, как в случае простого клиент-серверного приложения

обеспечивает соединение со специальным диагностическим процессом сервера приложений.

Противники этого метода обычно отмечают, что изменение архитектуры может оказать влияние на производительность исследуемого сеанса Нэнси. Однако эти изменения имеют более локальный характер, чем в первом случае, т. к. мы не изменяем нагрузку, конкурирующую с действиями Нэнси. Конечно же, необходимо будет изучить сделанные изменения, особенно если окажется, что перемены в архитектуре повлияли на производительность. Например, если измененная архитектура, представленная на рис. 6.4, демонстрирует значительно более высокую производительность, чем архитектура на рис. 6.3, то следует задуматься, не является ли виновником низкой производительности процесс сервера приложений `httpd0`.

Последний предлагаемый способ возможен лишь в том случае, если *все*, кто совместно с Нэнси использует один или несколько серверных процессов, выполняют приблизительно те же операции, что и Нэнси. Если все соединения, которые задействуют серверные процессы, выполняют однотипные операции, то каждая из строк результирующего файла трассировки достаточно показательна для изучения действий Нэнси (рис. 6.5).

Конечно же, утверждение о том, что невозможно восстановить составляющие из усредненного значения, остается истинным. И поэтому рискованно рассматривать общий файл трассировки, стремясь полу-

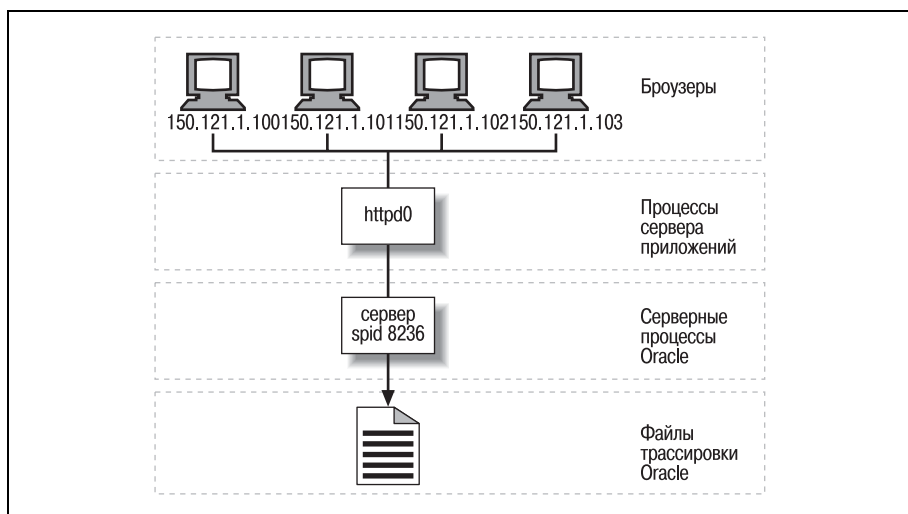


Рис. 6.5. Если все, кто использует показанные серверные процессы Oracle, выполняют однотипную работу, то любая из характеристик рабочей нагрузки в файле трассировки приблизительно отображает рабочую нагрузку любой отдельной пользовательской операции

чить характеристику рабочей нагрузки Нэнси (см. главу 3). Но в данном случае все решает наличие информации об однородности сеансов. Допустим, я сообщаю, что среднее значение для набора чисел равно 11 и что числа *приблизительно равны между собой*. Эта дополнительная информация (о том, что числа приблизительно равны между собой) позволяет делать оправданные предположения об исходных значениях. Если все, кто совместно использует серверный процесс Oracle, выполняют однотипные действия, то можно считать, что каждая строка данных трассировки в результирующем файле приблизительно отображает характеристики нагрузки каждого пользователя.

Немного хороших новостей

Сбор данных труднее реализовать в приложениях, конфигурация которых разрешает распределение вызовов одной пользовательской операции между двумя или более процессами ядра Oracle. Именно такая технология применяется практически во всех современных приложениях. На мой взгляд, проблема идентификации трассировочных данных стала одной из причин значительной переработки диагностических механизмов в Oracle версии 10. Надеюсь, что в будущем решение этой задачи упростится. Однако и на нынешний момент имеются две хорошие новости:

- Сбор данных расширенной трассировки SQL для многих пакетных заданий весьма прост, и так должно быть и дальше, несмотря на все увеличивающееся количество звеньев архитектуры, поскольку лучшей конфигурацией для множества пакетных заданий является использование выделенного серверного процесса Oracle.
- Любая проблема сбора данных, с которой когда-либо сталкивались мои коллеги, имеет свое решение. Надеюсь, что сайт *hotsos.com* — это одно из тех мест, куда вы в первую очередь будете заглядывать в поиске новых решений.

Устранение ошибок сбора данных

Как говорилось в главе 3, первостепенную важность имеет сбор данных трассировки SQL *точно* для выбранной временной области. Включение и отключение трассировки SQL точно в нужный момент особенно важно в тех случаях, когда выполняется диагностика операций с небольшим временем отклика. Если трассировкой сеанса управляет сторонний сеанс, то непопадание в заданную временную область возможно как в начале, так и в конце файла трассировки. Оставшаяся часть главы рассказывает о том, как и почему возникают ошибки сбора данных и как получить максимум информации из трассировочных данных.

Погрешность времени включения трассировки

Если сеанс сам включает собственную трассировку SQL, то в его файле трассировки вы в первую очередь обнаружите информацию о команде `ALTER SESSION SET EVENTS`. Если же атрибут трассировки сеанса устанавливается другим сеансом (например, при помощи, `DBMS_SYSTEM.SET_EV` или `DBMS_SUPPORT.START_TRACE_IN_SESSION`), то сказать, каким будет первое событие, попавшее в файл трассировки, становится сложнее.

Пропуск событий ожидания при включении трассировки

Если трассировка включается во время события ожидания, имеющего место между вызовами базы данных, то в начале файла трассировки могут отсутствовать некоторые данные. Рассмотрим последовательность действий, показанную на рис. 6.6. Я создал два сеанса SQL*Plus: один идентифицируется как 7.10583 (значения `SID` и `SERIAL#` для сеанса), а другой назван «вторым». Во втором сеансе я выполнил приведенный ниже блок PL/SQL, введя в ответ на приглашения значения 7 и 10583:

```
set serveroutput on
undef 1
undef 2
```

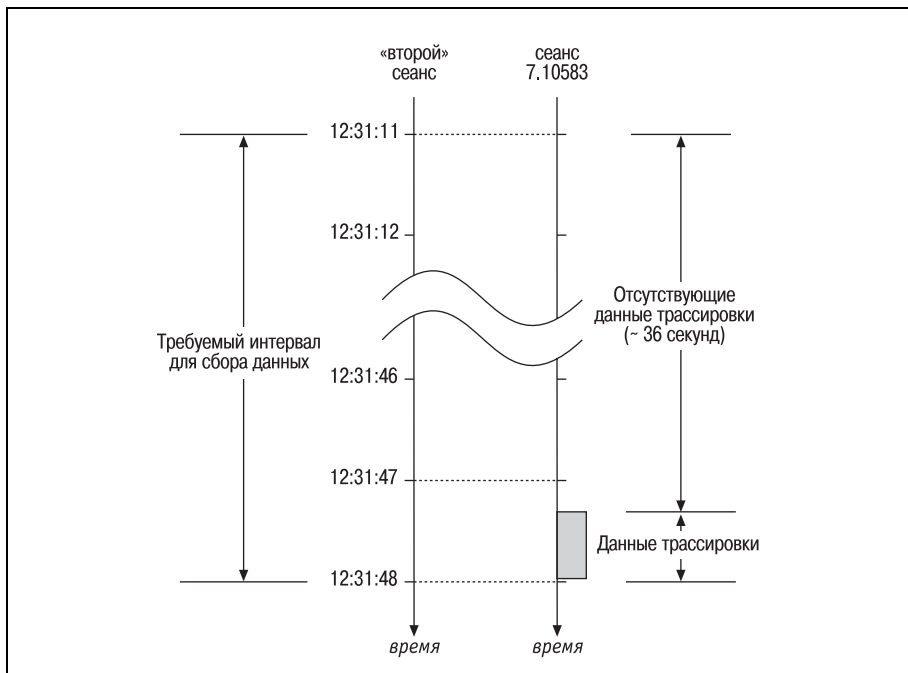


Рис. 6.6. Вызов `SET_EV` был выполнен во втором сеансе в 12:31:11, но первая запись файла трассировки для сеанса 7.10583 появилась только в 12:31:47.330, при этом приблизительно 36 секунд остались неучтенными


```

declare
  t varchar(20);
begin
  dbms_system.set_ev(&1,&2,10046,8,'');
  select to_char(sysdate, 'hh24:mi:ss') into t from v$timer;
  dbms_output.put_line('time='||t);
end;
/

```

Выполнение этого блока включает трассировку для сеанса 7.10583 и выводит время включения – 12:31:11. Следовательно, я знаю, что этот момент времени и есть точка начала нужного мне интервала сбора данных.

В сеансе 7.10583 я подождал несколько секунд, затем выполнил простой запрос текущего времени системы. Результатом запроса было 12:31:47. Затем я завершил сеанс 7.10583. Файл трассировки этого сеанса приведен в примере 6.7. Обратите внимание, что в файле трассировки отражены операции, выполненные между 12:31:47.330 и приблизительно 12:31:47.983 (я получил второе значение, выполнив отсчет времени), но он не содержит данных для периода приблизительно в 36 секунд между 12:31:11 и 12:31:47.330.

Пример 6.7. Файл трассировки, созданный запросом текущего системного времени. Результатом запроса явилось значение 12:31:47, что совпадает с первой временной меткой (специально выделенной в тексте) файла трассировки

```

/u01/oradata/admin/V901/udump/ora_31262.trc
Oracle9i Enterprise Edition release 9.0.1.0.0 - Production
With the Partitioning option
JServer release 9.0.1.0.0 - Production
ORACLE_HOME = /u01/oradata/app/9.0.1
System name:    Linux
Node name:     research
Release:       2.4.4-4GB
Version:       #1 Fri May 18 14:11:12 GMT 2001
Machine:       i686
Instance name: V901
Redo thread mounted by this instance: 1
Oracle process number: 8
Unix process pid: 31262, image: oracle@research (TNS V1-V3)

*** 2003-01-28 12:31:47.330
*** SESSION ID:(7.10583) 2003-01-28 12:31:47.330
APPNAME mod='SQL*Plus' mh=3669949024 act='' ah=4029777240
=====
PARSING IN CURSOR #1 len=47 dep=0 uid=5 oct=3 lid=5 tim=1043778707330593
hv=2972477985 ad='51302734'
select to_char(sysdate, 'hh24:mi:ss') from dual
END OF STMT
PARSE
#1:c=10000,e=1510,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=1043778707330128

```

```
EXEC #1:c=0,e=97,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=1043778707330810
WAIT #1: nam='SQL*Net message to client' ela= 5 p1=1650815232 p2=1 p3=0
FETCH #1:c=0,e=156,p=0,cr=1,cu=2,mis=0,r=1,dep=0,og=4,tim=1043778707331088
WAIT #1: nam='SQL*Net message from client' ela= 452 p1=1650815232 p2=1 p3=0
FETCH #1:c=0,e=2,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=0,tim=1043778707331819
WAIT #1: nam='SQL*Net message to client' ela= 4 p1=1650815232 p2=1 p3=0
WAIT #1: nam='SQL*Net message from client' ela= 650421 p1=1650815232 p2=1 p3=0
STAT #1 id=1 cnt=1 pid=0 pos=0 obj=221 op='TABLE ACCESS FULL DUAL'
XCTEND rlbk=0, rd_only=1
```

Я мог полностью контролировать сеанс 7.10583 на протяжении его существования, поэтому знаю, что 36 секунд, отсутствующие в файле трассировки, относятся к событию ядра SQL*Net message from client. Однако если бы я этого не знал, у меня не было бы способа корректно подсчитать неучтенное время. Поэтому программа сбора данных *Sparky* (<http://www.hotsos.com>) выполняет запрос к V\$SESSION_WAIT при включении (и отключении) трассировки. Выполнив при включении трассировки следующий запрос, я буду знать, какое событие не было завершено на момент включения (12:31:11):

```
select event from v$session_wait where sid=7 and state='WAITING'
```

Пропуск вызовов базы данных при включении трассировки

Более сложная проблема возникает в случае, если трассировка сеанса включается в ходе вызова базы данных. Например, я включил трассировку для сеанса 8.1665 в процессе выполнения долгой выборки, в результате чего были сформированы данные трассировки, представленные в примере 6.8. При внимательном изучении этот файл может вызывать беспокойство. Из более чем 87 700 строк данных трассировки, которые здесь не показаны, тысячи сантисекунд относятся к событиям ожидания курсора #1 (сумма значений поля ela для строк WAIT #1). Однако первым в файле трассировки отображен вызов базы данных UNMAP (он специально выделен в тексте примера). Обратите внимание, что его общая продолжительность составляет всего 3 сотых доли секунды (e=3). Имеются тысячи сантисекунд, потраченных на события ожидания, порожденные некоторым вызовом базы данных, но вызов, отвечающий за это время, так и не появляется в файле трассировки!

Пример 6.8. Файл трассировки, полученный в результате включения трассировки в ходе выполнения длительной выборки. Вызов выборки, который выполнялся в момент включения трассировки, отсутствует в данных трассировки

```
Dump file C:\oracle\admin\ora817\udump\ORA02124.TRC
Tue Jan 28 02:13:21 2003
ORACLE V8.1.7.0.0 - Production vsnsta=0
vsnsql=e vsnxtr=3
Windows 2000 Version 5.0 Service Pack 3, CPU type 586
Oracle8i Enterprise Edition release 8.1.7.0.0 - Production
With the Partitioning option
```

```

JServer release 8.1.7.0.0 - Production
Windows 2000 Version 5.0 Service Pack 3, CPU type 586
Instance name: ora817

Redo thread mounted by this instance: 1

Oracle process number: 10

Windows thread id: 2124, image: ORACLE.EXE

*** 2003-01-28 02:13:21.520
*** SESSION ID:(8.1665) 2003-01-28 02:13:21.510
WAIT #1: nam='direct path write' ela= 0 p1=4 p2=1499 p3=1
WAIT #1: nam='direct path write' ela= 0 p1=4 p2=1501 p3=1
WAIT #1: nam='db file sequential read' ela= 0 p1=1 p2=3690 p3=1
WAIT #1: nam='db file sequential read' ela= 0 p1=1 p2=3638 p3=1
WAIT #1: nam='db file sequential read' ela= 12 p1=1 p2=3691 p3=1
WAIT #1: nam='db file sequential read' ela= 0 p1=1 p2=3692 p3=1
=====
PARSING IN CURSOR #2 len=36 dep=1 uid=0 oct=3 lid=0 tim=38025864
hv=1705880752 ad='39be068'
select file# from file$ where ts#=:1
END OF STMT
PARSE #2:c=0,e=0,p=0,cr=0,cu=0,mis=1,r=0,dep=1,og=0,tim=38025864
...
Пропущено порядка 87700 строк, ни одна из которых не содержит операции
глубины dep=0.
...
WAIT #1: nam='direct path read' ela= 0 p1=4 p2=3710 p3=1
WAIT #1: nam='direct path read' ela= 0 p1=4 p2=3711 p3=3
WAIT #1: nam='direct path read' ela= 1 p1=4 p2=3586 p3=1
WAIT #1: nam='direct path read' ela= 0 p1=4 p2=3587 p3=4
WAIT #1: nam='direct path read' ela= 0 p1=4 p2=3591 p3=1
WAIT #1: nam='direct path read' ela= 0 p1=4 p2=3592 p3=2
=====
PARSING IN CURSOR #1 len=32 dep=0 uid=5 oct=3 lid=5 tim=38037728
hv=3588977815 ad='39b3e88'
select count(*) from dba_source
END OF STMT
UNMAP #1:c=0,e=3,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=38037728
WAIT #1: nam='SQL*Net message from client' ela= 2 p1=1111838976 p2=1 p3=0
FETCH #1:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=0,tim=38037730
WAIT #1: nam='SQL*Net message to client' ela= 0 p1=1111838976 p2=1 p3=0

```

И что еще хуже – непонятно, где тот запрос, который подсчитывает записи в DBA_SOURCE? Продолжительность запроса составила более 10 секунд (я сидел и смотрел, как он выполнялся), и вернул он всего одну строку, а единственная строка FETCH в файле трассировки сообщает о том, что запрос выполнен практически мгновенно и не вернул ни одной строки.

В примере 6.9 показаны те данные, которые мы ожидали, но не смогли увидеть в примере 6.8. Этот файл получен в результате запуска трасси-

ровки перед разбором запроса. Обратите внимание, что вместо двух вызовов базы данных (UNMAP и FETCH в примере 6.8) здесь мы видим целых *пять* вызовов:

1. Вызов PARSE для запроса к DBA_SOURCE, который имел место до того, как была включена трассировка в первом примере, поэтому данная строка не была передана в первый файл трассировки.
2. Вызов EXEC для запроса, который также начался до того, как была включена трассировка в первом примере, из-за чего и эта строка не попала в первый файл трассировки.
3. Вызов FETCH, на который ушла большая часть времени отклика запроса. В первом примере этот вызов начался раньше, чем была включена трассировка, поэтому соответствующая строка тоже отсутствует в примере 6.8.
4. Вызов UNMAP освобождает сегмент сортировки, используемый одним из рекурсивных представлений.
5. Последний вызов FETCH, удостоверяющий отсутствие данных, доступных курсору. Как видите, данный вызов выборки не возвращает строк.

И наконец, обратите внимание, что включение трассировки до начала запроса награждает нас данными STAT для сеанса, что само по себе приятно.

Пример 6.9. Данный файл сформирован в результате трассировки того же самого подсчета строк DBA_SOURCE, но на этот раз атрибут трассировки был установлен самим сеансом. Благодаря тому, что трассировка уже была включена в момент начала вызова FETCH, строка FETCH попала в данные трассировки

```
Dump file C:\oracle\admin\ora817\udump\ORA01588.TRC
Tue Jan 28 10:23:25 2003
ORACLE V8.1.7.0.0 - Production vsnsta=0
vsnsql=e vsnxtr=3
Windows 2000 Version 5.0 Service Pack 3, CPU type 586
Oracle8i Enterprise Edition release 8.1.7.0.0 - Production
With the Partitioning option
JServer release 8.1.7.0.0 - Production
Windows 2000 Version 5.0 Service Pack 3, CPU type 586
Instance name: ora817

Redo thread mounted by this instance: 1

Oracle process number: 9

Windows thread id: 1588, image: ORACLE.EXE

*** 2003-01-28 10:23:25.791
*** SESSION ID:(8.1790) 2003-01-28 10:23:25.781
APPNAME mod='SQL*Plus' mh=3669949024 act='' ah=4029777240
=====
```

```

PARSING IN CURSOR #1 len=69 dep=0 uid=5 oct=42 lid=5 tim=40966100
hv=589283212 ad='394821c'
alter session set events '10046 trace name context forever, level 8'
END OF STMT
EXEC #1:c=0,e=2,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=40966101
WAIT #1: nam='SQL*Net message to client' ela= 0 p1=1111838976 p2=1 p3=0
*** 2003-01-28 10:23:36.267
WAIT #1: nam='SQL*Net message from client' ela= 1046 p1=1111838976 p2=1 p3=0
=====
PARSING IN CURSOR #2 len=37 dep=1 uid=0 oct=3 lid=0 tim=40967147
hv=1966425544 ad='3afe9c4'
select text from view$ where rowid=:1
END OF STMT
PARSE #2:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=40967147
EXEC #2:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=40967147
WAIT #2: nam='db file sequential read' ela= 5 p1=1 p2=1669 p3=1
FETCH #2:c=0,e=5,p=1,cr=2,cu=0,mis=0,r=1,dep=1,og=4,tim=40967152
STAT #2 id=1 cnt=1 pid=0 pos=0 obj=59 op='TABLE ACCESS BY USER ROWID VIEW$ '
=====
PARSING IN CURSOR #1 len=32 dep=0 uid=5 oct=3 lid=5 tim=40967154
hv=3588977815 ad='39b3e88'
select count(*) from dba_source
END OF STMT
PARSE #1:c=1,e=8,p=1,cr=2,cu=0,mis=1,r=0,dep=0,og=4,tim=40967155
EXEC #1:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=40967155
WAIT #1: nam='SQL*Net message to client' ela= 0 p1=1111838976 p2=1 p3=0
WAIT #1: nam='db file sequential read' ela= 2 p1=1 p2=53 p3=1
WAIT #1: nam='db file sequential read' ela= 2 p1=1 p2=642 p3=1
WAIT #1: nam='db file sequential read' ela= 0 p1=1 p2=62 p3=1
...
Пропущено порядка 6700 строк, ни одна из которых не содержит операции
глубины dep=0.
...
WAIT #1: nam='direct path read' ela= 0 p1=4 p2=1944 p3=2
WAIT #1: nam='direct path read' ela= 0 p1=4 p2=1834 p3=1
WAIT #1: nam='direct path read' ela= 0 p1=4 p2=1835 p3=4
WAIT #1: nam='direct path read' ela= 0 p1=4 p2=1839 p3=1
WAIT #1: nam='direct path read' ela= 1 p1=4 p2=1840 p3=2
FETCH #1:c=1449,e=3669,p=6979,cr=879863,cu=10,mis=0,r=1,dep=0,og=4,tim=40970824
UNMAP #1:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=40970824
WAIT #1: nam='SQL*Net message from client' ela= 0 p1=1111838976 p2=1 p3=0
FETCH #1:c=0,e=0,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=0,tim=40970824
WAIT #1: nam='SQL*Net message to client' ela= 0 p1=1111838976 p2=1 p3=0
WAIT #1: nam='SQL*Net message from client' ela= 951 p1=1111838976 p2=1 p3=0
XCTEND rlbk=0, rd_only=1
STAT #1 id=1 cnt=1 pid=0 pos=0 obj=0 op='SORT AGGREGATE '
STAT #1 id=2 cnt=436983 pid=1 pos=1 obj=0 op='VIEW DBA_SOURCE '
STAT #1 id=3 cnt=436983 pid=2 pos=1 obj=0 op='SORT UNIQUE '
STAT #1 id=4 cnt=436983 pid=3 pos=1 obj=0 op='UNION-ALL '
STAT #1 id=5 cnt=436983 pid=4 pos=1 obj=0 op='NESTED LOOPS '
STAT #1 id=6 cnt=405 pid=5 pos=1 obj=0 op='NESTED LOOPS '

```

```

STAT #1 id=7 cnt=22 pid=6 pos=1 obj=22 op='TABLE ACCESS FULL USER$ '
STAT #1 id=8 cnt=425 pid=6 pos=2 obj=18 op='TABLE ACCESS BY INDEX ROWID OBJ$ '
STAT #1 id=9 cnt=3200 pid=8 pos=1 obj=34 op='INDEX RANGE SCAN '
STAT #1 id=10 cnt=436983 pid=5 pos=2 obj=64 op='TABLE ACCESS BY INDEX
ROWID SOURCE$ '
STAT #1 id=11 cnt=437387 pid=10 pos=1 obj=109 op='INDEX RANGE SCAN '
STAT #1 id=12 cnt=0 pid=4 pos=2 obj=0 op='NESTED LOOPS '
STAT #1 id=13 cnt=1 pid=12 pos=1 obj=0 op='NESTED LOOPS '
STAT #1 id=14 cnt=1 pid=13 pos=1 obj=0 op='FIXED TABLE FULL X$JOXFT '
STAT #1 id=15 cnt=0 pid=13 pos=2 obj=18 op='TABLE ACCESS BY INDEX ROWID OBJ$ '
STAT #1 id=16 cnt=0 pid=15 pos=1 obj=33 op='INDEX UNIQUE SCAN '
STAT #1 id=17 cnt=0 pid=12 pos=2 obj=22 op='TABLE ACCESS CLUSTER USER$ '
STAT #1 id=18 cnt=0 pid=17 pos=1 obj=11 op='INDEX UNIQUE SCAN '

```

Включение трассировки SQL во время выполнения любого длительно-го вызова базы данных приводит к потере данных, вроде той, которую мы только что рассмотрели. Важно уметь распознать такую ошибку сбора данных. Ведь если попытаться диагностировать причину ухудшения производительности на основе данных с такой ошибкой, то можно попасть в бездонную яму, т. к. придется иметь дело с большими объемами избыточно учтенного времени.

Подобную ошибку можно выявить, обратив внимание на то, что сумма значений `ela` для последовательности событий ожидания (строки WAIT) значительно превышает границы общей продолжительности вызова базы данных (значение `e`), породившего эти события ожидания. В примере 6.8 об ошибке говорит тот факт, что более 87 700 строк WAIT #1 были переданы за гораздо больший интервал времени, чем отмеченная фактическая продолжительность (`e=3` сантисекунды) вызова UNMAP #1, который следует непосредственно за этими строками.

Профилактика – единственное лекарство, которое я могу порекомендовать от такого рода ошибок сбора данных. Избегайте включения трассировки SQL в процессе выполнения длительного вызова базы данных. Если полученный файл трассировки содержит такую ошибку, то лучше всего повторить процедуру сбора данных заново.



Включая расширенную трассировку SQL из стороннего сеанса, приложите все усилия к тому, чтобы не сделать этого во время выполнения длительного вызова базы данных. Если все же не удастся избежать такой ситуации, то, вероятно, наилучшим выходом будет применение одного из приемов, описанных в главе 8.

Избыточные данные о вызовах при включении трассировки

В некоторых случаях отмеченная в файле трассировки длительность вызова базы данных превышает тот период времени, на который была включена трассировка. Такое явление наблюдается, когда время начала вызова базы данных (значение `tim - e`) меньше времени начала сбора данных, т. е. когда `tim - e < t0`. Мы несколько раз наблюдали подобные случаи, когда трассировка была включена сторонним сеансом в ходе

длительного выполнения блока PL/SQL. (Отличие этого случая от только что рассмотренного заключается во включении трассировки именно в ходе долго выполняющегося блока PL/SQL, а не просто в ходе выполнения длительного вызова базы данных). Единственная часть файла трассировки, которая может пострадать от феномена избыточного времени, – это первые отмеченные в файле трассировки операции с определенной рекурсивной глубиной (значением поля *dep*).

Подобное явление не так просто заметить, если несколько тысяч строк WAIT (не содержащих полей *tim*) предшествуют строке первого вызова базы данных, содержащей это поле. В главе 5 вы узнали, как выполнить обратный отсчет времени, начиная со значения первого поля *tim* файла через значения всех полей *ela* и до достижения первой строки. Однако этот метод подвержен заметному накоплению систематической ошибки, как это было показано при описании отсчета времени в главе 5.

Гораздо более удачный способ вычисления «виртуального» значения *tim* для первой строки WAIT файла трассировки заключается в том, чтобы ввести функции преобразования, которые позволяют преобразовывать значения полей *tim* Oracle к значениям системного времени и обратно. Соотношение между величиной *tim* и системным временем можно установить, сделав следующие шаги:

1. Выполните приведенные ниже команды в SQL*Plus для системы, соотношение времен для которой вы хотите установить:

```
alter system set events '10046 trace name context forever, level 8';
execute sys.dbms_system.ksdddt;
exit
```

2. Исследуйте полученные данные трассировки. Он будет содержать такие строки:

```
*** 2003-01-28 14:30:56.513
EXEC #1:c=0,e=483,p=0,cr=0,cu=0,mis=0,r=1,dep=0,og=4,tim=1043785856513829
```

3. По этой информации вы можете установить прямое соответствие указанного значения поля *tim* указанной временной метке. В приведенном примере обратите внимание на совпадение составляющих секунд и миллисекунд двух значений (выделено жирным шрифтом). В изучаемой нами системе отношение между значениями времени очень просто: каждое значение поля *tim* – это просто количество микросекунд, прошедших с момента начала эры UNIX (00:00:00 UTC – 1 января 1970). В примере 6.10 приведена программа, которая служит мне инструментом для преобразования значений *tim* в системное время и обратно.

Пример 6.10. Программа, преобразующая значения tim Oracle в значения системного времени и обратно

```
#!/usr/bin/perl

# $Header: /home/cvs/cvm-book1/sqltrace/tim.pl,v 1.3 2003/02/05 05:06:58
```

```

cvm Exp $
# Cary Millsap (cary.millsap@hotsos.com)

# Вывод системного времени, которое соответствует определенному значению tim
use strict;
use warnings;
use Date::Format qw(time2str);
use Date::Parse qw(str2time);

my $usage = "Usage: $0 wall-time\\n          $0 tim-value\\n\\t";
my $arg = shift or die $usage;      # значение tim или системного времени
# printf "arg =%s\\n", $arg;
if ($arg =~ /^[0-9]+$/ ) {
    # входной аргумент - это значение tim
    my $sec = substr($arg, 0, length($arg)-6);
    my $msec = substr($arg, -6);
    # printf "sec =%s\\n", $sec;
    # printf "msec=%s\\n", $msec;
    printf "%s\\n", time2str("%T.$msec %A %d %B %Y", $sec);
}
else {
    # входной аргумент - это значение системного времени
    my $frac = ($arg =~ /\d+:\d+\.\d+/) ? $1 : 0;
    if ((my $l = length $frac) >= 6) {
        # если length(frac) >=6, то выполнить округление
        $frac = sprintf "%06.0f", $frac/(10**($l-6));
    } else {
        # иначе дополнить справа нулями
        $frac .= ('0' x (6-$l));
    }
    printf "%s\\n", str2time($arg), $frac;
}

```

Рассмотрим простой пример файла трассировки, начальные строки которого содержат данные о событиях, которые произошли до момента начала сбора данных:

```

*** 2003-02-24 04:28:19.557
WAIT #1: ... ela= 20000000 ...
EXEC #1:c=10000000,e=30000000,...,tim= 1046082501582881

```

Нелегко понять, что происходит, если не преобразовать значения времени к сопоставимому формату. Возьмем из примера 6.10 программу для преобразования значения tim в моей Linux-системе в более удобочитаемое значение абсолютного времени и увидим, что вызов исполнения завершился только через 2,025881 секунды после включения трассировки:

```

$ perl tim.pl 1046082501582881
04:28:21.582881 Monday 24 February 2003

```

Но фактическая продолжительность вызова исполнения составляет 30 секунд (e=30000000). То есть часть продолжительности этого вызова

базы данных относится к периоду, предшествующему временной метке, выведенной в начале файла трассировки. Я уже говорил, что эта временная метка не всегда соответствует тому моменту времени, в который на самом деле был установлен атрибут трассировки. Придется отдельно отслеживать время включения трассировки (назовем его t_0). Это проще всего сделать, отметив время в поле `tim` при выполнении команды на включение трассировки SQL.

Итак, вы определили, что в файле трассировки учтено избыточное время, и следующая задача состоит в том, чтобы избавиться от него. На рис. 6.7 показано, как это сделать. Здесь трассировка SQL включается в момент t_0 , во время некоторого вызова разбора внутри долго выполняющегося блока PL/SQL. При этом часть продолжительности e вызова разбора относится к интересующему нас интервалу наблюдения, а другая его часть – к периоду времени, предшествующему t_0 . Лишнее время легко вычислить, т. к. значение t_0 в единицах `tim` известно. Не вошедшее в период наблюдения время T можно вычислить так:

$$T = t_0 - (t - e)$$

Если первые несколько строк, переданные в файл трассировки, не содержат значения `tim`, то можно вычислить значение t для начала файла, преобразовав начальное значение временной метки (в строке `***`) в соответствующее значение `tim`, как было показано выше. Помните, что временная метка сопоставлена моменту завершения операции, которая следует за данной строкой в файле трассировки. Задача сводится к ситуации, уже рассмотренной на рис. 6.7, в которой вам известны t , e , t_0 (фактически и все промежуточные значения `ela` в случае, если длительность одного из событий ожидания также включает в себя t_0).

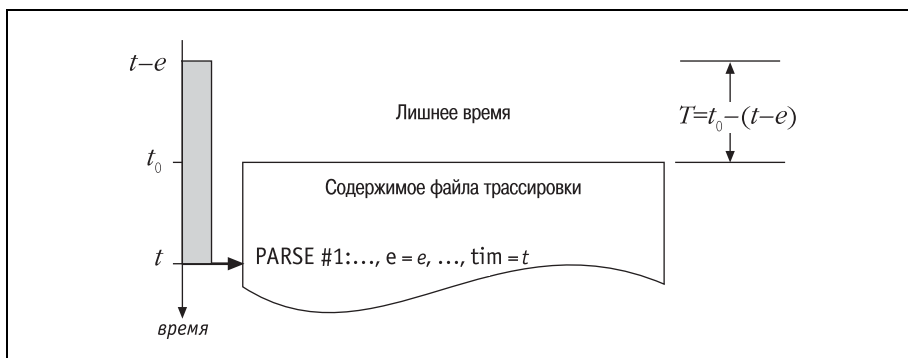


Рис. 6.7. Если трассировка SQL включается сторонним сеансом в момент времени t_0 , то трассировка может начаться во время выполнения вызова базы данных. Тогда файл трассировки будет содержать избыточное время, потраченное вызовом базы данных до момента включения трассировки SQL

Пропуск времени при отключении трассировки

При завершении сеанса, для которого была включена расширенная трассировка SQL, в файле трассировки будет учтено все время около момента завершения сеанса. Так, если сеанс отключает собственную трассировку командой ALTER SESSION SET EVENTS, в файле трассировки учитывается все время сеанса, прошедшее до момента выполнения этой команды. Если же трассировка отключается сторонним сеансом, то это отключение произойдет, скорее всего, в ходе выполнения события ожидания или вызова базы данных сеанса. В таком случае некоторые нужные данные о сеансе могут отсутствовать в файле трассировки.

Например, трассировка указанного сеанса отключается в момент времени `tim=1043788733690992`. Однако в конце файла трассировки содержится лишь следующее:

```
*** 2003-01-28 15:18:43.688
WAIT #1: nam='SQL*Net message from client' ela= 24762690 p1=1650815232 p2=1 p3=0
STAT #1 id=1 cnt=1 pid=0 pos=0 obj=0 op='MERGE JOIN '
STAT #1 id=2 cnt=1 pid=1 pos=1 obj=0 op='SORT JOIN '
STAT #1 id=3 cnt=1 pid=2 pos=1 obj=0 op='FIXED TABLE FULL X$KSUSE '
STAT #1 id=4 cnt=1 pid=1 pos=2 obj=0 op='SORT JOIN '
STAT #1 id=5 cnt=9 pid=4 pos=1 obj=0 op='FIXED TABLE FULL X$KSUPR '
=====
PARSING IN CURSOR #1 len=39 dep=0 uid=5 oct=3 lid=5
                        tim=1043788723689828 hv=364789794 ad='512c8b5c'
select 'missing time at tail' from dual
END OF STMT
PARSE #1:c=0,e=871,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=4,tim=1043788723689794
EXEC #1:c=0,e=72,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=4,tim=1043788723690030
WAIT #1: nam='SQL*Net message to client' ela= 5 p1=1650815232 p2=1 p3=0
FETCH #1:c=0,e=118,p=0,cr=1,cu=2,mis=0,r=1,dep=0,og=4,tim=1043788723690276
WAIT #1: nam='SQL*Net message from client' ela= 445 p1=1650815232 p2=1 p3=0
FETCH #1:c=0,e=2,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=0,tim=1043788723690992
WAIT #1: nam='SQL*Net message to client' ela= 4 p1=1650815232 p2=1 p3=0
```

Обратите внимание на выделенную часть завершающего значения в поле `tim`: файл трассировки содержит сведения о том, что произошло до момента времени `...23,690992` (в секундах) и фактически на 4 микросекунды позже, но нет записей о том, что происходило между моментами `...23,690992` и `...33,690992`. Не учтено ровно 10 секунд.

На рис. 6.8. показано, как сложилась такая ситуация. Трассировка SQL была выключена в момент t_1 , в ходе выполнения события ожидания z . Но ядро Oracle передает в файл трассировки строку события ожидания только по завершении этого события. Трассировка была выключена раньше, чем завершилось событие ожидания, никаких данных о нем в файле трассировки нет. Часть длительности события ожидания относится к периоду времени после t_1 , но та его часть, которая произошла до t_1 , остается неучтенной.

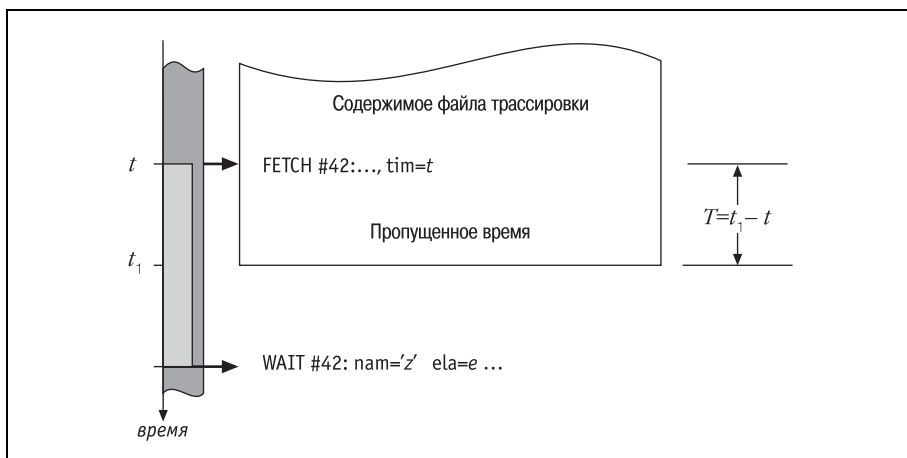


Рис. 6.8. Если трассировка SQL выключается сторонним сеансом в момент времени t_1 , то трассировка может завершиться в ходе выполнения некоторого события. В этом случае время, затраченное на данное событие, не появится в файле трассировки, в результате чего часть времени будет утеряна

В данном случае вычислить пропущенное время просто, ведь значение t_1 в единицах `tim` известно. Это делается по формуле:

$$T = t_1 - t$$

Опять-таки проще всего отслеживать t_1 , пометив момент времени, когда выполняется команда на отключение трассировки SQL. При отключении трассировки необходимо определить имя события, которое не было завершено на момент t_1 . Из стороннего сеанса это можно узнать посредством такой команды SQL:

```
select event from v$session_wait
where sid=:sid and state='WAITING'
```

Разработанный Hotsos сборщик данных *Sparky* выполняет подобный запрос непосредственно перед командой на включение и выключение трассировки. Если запрос не возвращает имени события, то все пропущенное время T следует отнести к процессорному времени. Если же запрос возвращает имя события, то по крайней мере некоторая часть пропущенного времени соответствует событию, имя которого возвратил запрос. При этом, как видно на рис. 6.8, некоторая часть пропущенного времени все равно может быть израсходована процессором.



Возможно, некоторая часть пропущенного времени израсходована на выполнение инструкций ядра Oracle, для которых отсутствуют средства измерения (подробно мы поговорим об этом в главе 7).

Может быть, удастся приблизительно определить, какую часть времени T следует отнести на счет загрузки процессора, а какую – сопоставить событию ожидания. Однако наша практика показала, что в случае, если запрос `V$SESSION_WAIT` возвращает имя события, сопоставление всего значения T этому событию будет хорошим приближением.

Наличие строк `WAIT` в конце файла трассировки несколько усложняет вычисление пропущенного времени, т. к. приходится осуществить еще одну операцию по отсчету времени. В данном случае вам придется получить значение t , отсчитывая время вперед от значения последнего поля `tim` файла через значения полей `ela`.

Неполные данные рекурсивного SQL

Включение и отключение трассировки SQL из стороннего сеанса также может привести к усечению данных трассировки, необходимых для определения природы рекурсивных отношений в SQL. Включение трассировки SQL после выполнения рекурсивных операций, но до завершения соответствующих родительских операций, приводит к отсутствию данных о потомках в файле трассировки. Например, если выполняется приведенный ниже код PL/SQL, то полученный файл трассировки будет отображать ряд рекурсивных отношений между различными элементами блока и самим блоком:

```
alter session set events '10046 trace name context forever, level 8';
declare
    cursor lc is select count(*) from sys.source$;
    cnt number;
begin
    open lc;
    fetch lc into cnt;
    close lc;
    open lc;
    fetch lc into cnt;
    close lc;
end;
/
```

Однако можно коренным образом изменить данные трассировки, убрав команду `ALTER SESSION` и включив трассировку из стороннего сеанса (например, при помощи `DBMS_SUPPORT.START_TRACE_IN_SESSION`) в ходе выполнения блока. Сделав так, вы обнаружите, что ядро опустит массу сведений для всех рекурсивных дочерних операций, выполнение которых началось прежде, чем была включена трассировка. Включение трассировки SQL из стороннего сеанса может привести к отсутствию в файле трассировки дочерних вызовов базы данных для всех рекурсивных родительских операций, перечисленных в файле трассировки.

Как и в рассмотренном ранее случае с отсутствием данных о вызовах базы данных при включении трассировки, лучшим лекарством от по-

Аналогично *отключение* трассировки SQL из стороннего сеанса означает возможность отсутствия в файле трассировки *родительских* вызовов базы данных для всех рекурсивных ($dep > 0$) операций, отмеченных в файле трассировки. Представьте себе, что в примере 6.11 трассировка уже была включена на момент начала приведенного фрагмента кода, а затем была выключена сторонним сеансом в момент, помеченный значком ❶. Результат такой трассировки показан в примере 6.12.

```

=====
PARSING IN CURSOR #2 len=37 dep=1 uid=0 oct=3 lid=0 tim=1033174180230513
                                hv=1966425544 ad='514bb478'

select text from view$ where rowid=:1
END OF STMT

PARSE #2:c=0,e=107,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=1033174180230481
BINDS #2:
    bind 0: dty=11 mxl=16(16) mal=00 scl=00 pre=00 oacflg=18 oacfl2=1
                                size=16 offset=0
        bfp=0a22c34c bln=16 avl=16 flg=05
        value=00000AB8.0000.0001

EXEC #2:c=0,e=176,p=0,cr=0,cu=0,mis=0,r=0,dep=1,og=4,tim=1033174180230878
FETCH #2:c=0,e=89,p=0,cr=2,cu=0,mis=0,r=1,dep=1,og=4,tim=1033174180231021
❶
STAT #2 id=1 cnt=1 pid=0 pos=0 obj=62 op='TABLE ACCESS BY USER ROWID VIEW$'
=====
PARSING IN CURSOR #1 len=85 dep=0 uid=5 oct=3 lid=5 tim=1033174180244680
                                hv=1205236555 ad='50cafbec'

select object_id, object_type, owner, object_name from dba_objects
                                where object_id=:v

END OF STMT

PARSE #1:c=10000,e=15073,p=0,cr=2,cu=0,mis=1,r=0,dep=0,og=0,
                                tim=1033174180244662

```

В примере 6.11 очевидно наличие рекурсивных отношений между вызовами базы данных, т. к. в нем присутствуют три операции со значением глубины `dep=1` (они выделены жирным шрифтом в примерах 6.11 и 6.12). Проблема примера 6.12 в том, что трассировка была отключена прежде, чем ядро Oracle передало какие-либо сведения о рекурсивном родителе глубины `dep=0` для наших операций. Обратите внимание на операцию глубины `dep=0` в примере 6.11 (она также выделена жирным шрифтом), которая и служит родителем, но в примере 6.12 трассировка была завершена прежде, чем информация о родителе попала в файл трассировки.

Пример 6.12. В конце этого файла трассировки нет вызова базы данных, который бы следовал за событиями глубины `dep=1`, выступая как их родитель

```
...
=====
PARSING IN CURSOR #2 len=37 dep=1 uid=0 oct=3 lid=0 tim=1033174180230513
hv=1966425544 ad='514bb478'
select text from view$ where rowid=:1
END OF STMT
PARSE #2: c=0, e=107, p=0, cr=0, cu=0, mis=0, r=0, dep=1, og=4, tim=1033174180230481
BINDS #2:
  bind 0: dty=11 mxl=16(16) mal=00 scl=00 pre=00 oacflg=18 oacfl2=1
                                     size=16 offset=0

  bfp=0a22c34c bln=16 avl=16 flg=05
  value=00000AB8.0000.0001
EXEC #2: c=0, e=176, p=0, cr=0, cu=0, mis=0, r=0, dep=1, og=4, tim=1033174180230878
FETCH #2: c=0, e=89, p=0, cr=2, cu=0, mis=0, r=1, dep=1, og=4, tim=1033174180231021
STAT #2 id=1 cnt=1 pid=0 pos=0 obj=62 op='TABLE ACCESS BY USER ROWID VIEW$ '
Конец файла
```

Из усеченных данных примера 6.12 можно узнать о том, что присутствуют три рекурсивных операции SQL, для которых где-то существует родитель, но кто он – неизвестно. Такие вызовы базы данных остаются «сиротами». Отключение трассировки SQL из стороннего сеанса приводит к возможному отсутствию в файле трассировки родительских вызовов базы данных для всех рекурсивных (`dep > 0`) операций, упомянутых в этом файле

И вновь лучшее лекарство – это профилактика. Вы естественным образом избежите таких ошибок, если будете выбирать время остановки сбора данных, основываясь на наблюдениях за пользовательскими операциями, как это и должно быть.

Упражнения

1. Системные администраторы и администраторы баз данных могут бесконечно спорить о том, имеет ли установка параметра Oracle `TIMED_STATISTICS` значительное влияние на производительность при-

ложений. Обратитесь к Oracle *MetaLink*, чтобы определить, существуют ли какие-либо ошибки, приводящие к неприемлемым отрицательным последствиям в случае вашей реализации. Затем придумайте пример для изучения влияния на время отклика значения `TIMED_STATISTICS=TRUE`, установленного для системы.

2. Файл трассировки приводит данные о фактической продолжительности, которая значительно меньше известной вам продолжительности интервала наблюдения. Объясните, какого рода ошибки могли привести к возникновению такого явления.

7

Измерение времени ядром Oracle

Вне зависимости от того, получаете ли вы временные статистики ядра Oracle из данных расширенной трассировки SQL, из фиксированных представлений `V$` или даже непосредственно из лежащего в их основе сегмента разделяемой памяти, вся временная статистика формируется с помощью небольшого набора функций операционной системы. Каким бы интерфейсом вы ни пользовались для доступа к этой статистике, для нее всегда действуют ограничения, присущие таймерам операционной системы, используемым для ее получения. В этой главе рассмотрены такие ограничения и описано их влияние на работу аналитика.

Управление процессами операционной системы

С точки зрения операционной системы ядро Oracle – это обычное приложение. В его работе нет ничего мистического – это просто огромная и весьма впечатляющая программа на языке C. Для полного понимания хронометрических данных, предоставляемых ядром Oracle, необходимо поближе познакомиться с возможностями самого ядра, в свою очередь предоставляемых ему операционной системой.

В этом разделе мы уделим основное внимание поведению операционных систем, основанных на UNIX. Изложенный здесь материал достаточно адекватно описывает поведение клонов UNIX, таких как Linux, Sun Solaris, HP-UX, IBM AIX и Tru64. Но сведения из этого раздела будут интересны и тем, кто работает с MS Windows. Те же, кто работает с какой-то другой ОС, должны будут обратиться за дополнительной информацией к руководству по этой ОС.

Я считаю, что лучшее описание функционирования процесса в контексте современной ОС дал Морис Бах (Maurice Bach) в своей книге «The Design of the Unix Operating System» (Архитектура операционной сис-

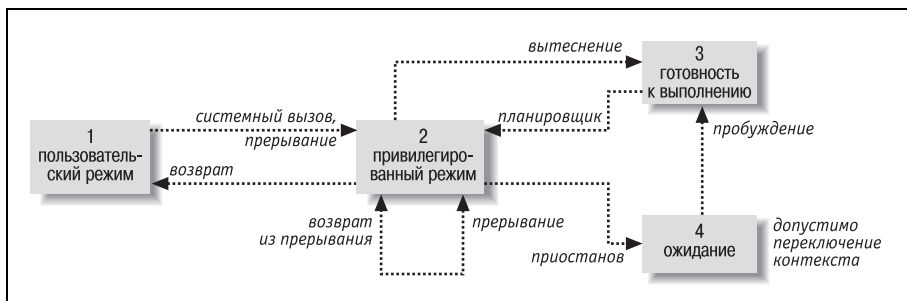


Рис. 7.1. Упрощенная диаграмма состояний процесса изображает основные состояния, принимаемые процессом в большинстве современных ОС с разделением времени [Bach (1986) 31]

темы Unix, [Bach 1986 (31)]. Рисунок 2.6 из этой книги, озаглавленный «Process States and Transitions» (Состояния процесса и переходы между ними) и воспроизведенный на рис. 7.1, будет отправной точкой моего рассказа. На этой схеме каждый *узел* (прямоугольник) представляет состояние, которое процесс может принимать в операционной системе. Каждое *ребро* (направленная линия) – это переход из одного состояния в другое. Можно воспринимать состояния как существительные, а переходы – как глаголы, которые вызывают переход процесса из одного состояния в следующее.

Большая часть процессов ядра Oracle проводит основную часть своего времени в состоянии *выполнения пользователем* (*user running*), называемом также *пользовательским*, или *непривилегированным*, режимом (*user mode*). Разбор SQL, сортировка строк, чтение блоков кэша буферов и преобразование типов данных – все эти операции Oracle выполняет в пользовательском режиме. Переход процесса из пользовательского режима в *привилегированный* (*kernel mode*) может быть инициирован одним из двух событий. Вам необходимо понимать оба варианта такого перехода, поэтому рассмотрим их более подробно.

Переход по системному вызову

Когда процесс, находящийся в пользовательском режиме, совершает вызов операционной системы, он переходит в привилегированный режим. К типичным системным вызовам относятся *read* и *select*. В привилегированном режиме процесс наделяется специальными полномочиями, которые позволяют ему манипулировать низкоуровневыми аппаратными компонентами и произвольными ячейками памяти. Например, в привилегированном режиме процесс может управлять устройствами ввода/вывода, такими как сокет и дисковые устройства [Bovet and Cesati (2001) 8].

Можно ожидать, что многие системные вызовы будут проводить в ожидании ответа от устройства очень много процессорных циклов. Например, вызов *read* для дисковой подсистемы сегодня обычно выполняет

ся за время порядка нескольких миллисекунд. Многие же нынешние процессоры способны выполнить миллионы инструкций за то время, пока будет выполняться единственная операция физического дискового ввода/вывода. Так что за время одного вызова чтения можно выполнить порядка миллиона инструкций процессора. Естественно, создатели эффективных системных вызовов чтения учитывают этот факт и организуют свой код таким образом, чтобы освободить процессор для других программ на то время, когда процесс чтения находится в ожидании.

Рассмотрим некий процесс ядра Oracle, выполняющий системный вызов чтения для получения блока данных Oracle с диска. После выдачи запроса к предположительно «медленному» устройству, код системного вызова `read` переведет вызывающий процесс в состояние *ожидания* (*sleep*), в котором процесс будет ожидать прерывания, оповещающего о завершении операции ввода/вывода. Такая учтивость со стороны процесса позволяет другому *готовому к выполнению* (*ready to run*) процессу занять те циклы процессора, которые в любом случае не были бы доступны процессу чтения.

Когда устройство ввода/вывода сигнализирует о готовности операции ввода/вывода *ожидającego* процесса к дальнейшей обработке, процесс активизируется, т. е. переходит в состояние *готовности к выполнению*. А с процессом, готовым к выполнению, может работать планировщик. Этот процесс, выбранный планировщиком для исполнения, будет возвращен в *привилегированный режим*, в котором и будет выполнена оставшаяся часть кода вызова `read` (например, передача данных, полученных от канала ввода/вывода, в оперативную память). Последняя инструкция подпрограммы `read` возвращает управление вызывающей программе (нашему процессу ядра Oracle), т. е. процесс переходит обратно в *пользовательский режим*. В этом состоянии процесс ядра Oracle продолжает потреблять время процессора в непривилегированном режиме до тех пор, пока не возникнет следующее прерывание и не будет сделан очередной системный вызов.



Надо сказать, что вызов `exit` тоже относится к системным, поэтому даже когда приложение заканчивает свою работу, *единственными* способами выхода из *пользовательского режима* являются *системный вызов* и *переход по прерыванию*.

Переход по прерыванию

Второй путь в схеме состояний процессов операционной системы образован переходами по *прерыванию* (*interrupt*). Прерывание – это механизм, посредством которого периферийное устройство ввода/вывода, системные часы или какое-то другое устройство может асинхронно прервать работу процессора [Bach (1986) 16]. Я уже говорил о том, почему периферийное устройство ввода/вывода может прервать процесс. Конфигурация большинства систем такова, что системные часы поро-

ждуют прерывание каждую сотую долю секунды (т. е. раз в сантисекунду). По получении прерывания от таймера каждый процесс системы, находящийся в *пользовательском* режиме, сохраняет свой *контекст* (образ того, чем занимался процесс) и выполняет специальную подпрограмму операционной системы – *планировщик* (*scheduler*). Планировщик определяет, следует ли позволить процессу продолжить выполнение или же произвести его *вытеснение*.

Прерывание обслуживания (*preemption*) переводит процесс из *привилегированного* состояния в состояние *готовности к выполнению*, освобождая тем самым путь для возвращения какого-то другого процесса из состояния *готовности к выполнению* в *пользовательский* режим [Bach (1986) 148]. Именно таким образом большая часть современных операционных систем реализует разделение времени. Любой процесс, находящийся в состоянии *готовности к выполнению*, обрабатывается именно так, как было описано: процесс становится доступным планировщику и т. д. Понимание вытеснения (приоритетного прерывания обслуживания) по таймеру необходимо для осознания материала, изложенного далее в главе при описании одной из основных причин «пропуска данных» в файле трассировки Oracle.

Другие состояния и переходы

Я уже упоминал о существовании более сложной схемы переходов между состояниями процесса, чем та, которая приведена на рис. 7.1. Действительно, обсудив четыре состояния процесса и семь переходов, представленных на рис. 7.1, Бах далее в своей книге приводит более сложную схему переходов состояний процессов [Bach (1986) 148]. На этой схеме подробно рассмотрены действия, выполняемые во время переходов: вытеснение, подкачка, ветвление и даже создание процессов-зомби. Тем, чьи приложения работают в Unix-системах, я бы настойчиво рекомендовал добавить в свою библиотеку книгу Баха.

Для рассмотрения оставшегося материала главы нам будем вполне достаточно рис. 7.1. Надеюсь, вы согласитесь с тем, что изображенные на нем состояния процессов упрощают понимание временных статистик Oracle.

Измерение времени ядром Oracle

Ядро Oracle выводит информацию о времени лишь нескольких типов. Данные расширенной трассировки SQL включают в себя четыре важных статистики. Все они содержатся в приведенных ниже двух строчках, сформированных ядром Oracle версии 9.0.1.2.0 в системе Solaris 5.6:

```
WAIT #34: nam='db file sequential read' ela= 14118 p1=52 p2=2755 p3=1
FETCH #34:c=0,e=15656,p=1,cr=6,cu=0,mis=0,r=1,dep=3,og=4,
tim=1017039276349760
```

Эти две соседние строки данных трассировки описывают один вызов выборки базы данных. Посмотрим, какие временные статистики в них присутствуют:

```
ela= 14118
```

Фактическое время выполнения системного вызова `db file sequential read` ядром Oracle составило 14 118 микросекунд (мкс, где 1 мкс = 0,000001 секунды).

```
c=0
```

Ядро Oracle сообщает, что на вызов выборки было потрачено 0 мкс процессорного времени.

```
e=15656
```

Ядро Oracle сообщает, что фактическая продолжительность вызова выборки составила 15 656 мкс.

```
tim=1017039276349760
```

Системное время завершения вызова – 1 017 039 276 349 760 (выражено в количестве микросекунд, прошедших после полуночи по всеобщему скоординированному времени (Coordinated Universal Time – UTC) 1 января 1970 года).

Общая фактическая продолжительность выборки включает в себя и общее время пользования процессором, и любое время, потраченное на выполнение событий ожидания Oracle. Статистики, собранные в двух строках файла трассировки, связаны между собой следующим отношением:

$$e \approx c + ela$$

В данном случае с точки зрения человека точность достаточно хороша: $15\,656 \approx 0 + 14\,118$, погрешность составляет 0,001538 секунды.

Каждый отдельный вызов базы данных (такой как *разбор*, *исполнение* или *выборка*) порождает всего одну строку вызова базы данных (PARSE, EXEC или FETCH) в файле трассировки (в том числе и *рекурсивные* вызовы, которые могут последовать за отдельным вызовом базы данных). В то же время один вызов базы данных может породить несколько строк WAIT, представляющих системные вызовы для каждой действующей строки курсора. Рассмотрим фрагмент файла трассировки, в котором присутствуют 6288 вызовов `db file sequential read`, выполненных одним вызовом выборки:

```
WAIT #44: nam='db file sequential read' ela= 15147 p1=25 p2=24801 p3=1
```

```
...
```

```
6,284 similar WAIT lines are omitted here for clarity
```

```
...
```

```
WAIT #44: nam='db file sequential read' ela= 105 p1=25 p2=149042 p3=1
```

```
WAIT #44: nam='db file sequential read' ela= 18831 p1=5 p2=115263 p3=1
```

```
WAIT #44: nam='db file sequential read' ela= 114 p1=58 p2=58789 p3=1
```

```
FETCH #44:c=7000000,e=23700217,p=6371,cr=148148,cu=0,mis=0,r=1,dep=1,og=4,
```

```
tim=1017039304454213
```

Таким образом, корректное соотношение, связывающее значения c , e и ela для определенного вызова базы данных, должно содержать *сумму* значений ela , полученных в контексте данного вызова базы данных:

$$e \approx c + \sum_{\text{db call}} ela$$

Данное соотношение служит основой любых измерений времени отклика ядром Oracle.

Как программное обеспечение измеряет само себя

Понять, каким образом ядро Oracle осуществляет измерение собственных характеристик, не слишком сложно. Основой этого раздела послужили наши исследования ядер Oracle8i и Oracle9i, работающих в ОС Linux. Программное обеспечение Oracle в разных операционных системах может использовать разные системные вызовы. Для того чтобы разобраться с этим, необходимо с помощью программы, выполняющей трассировку системных вызовов определенного процесса, посмотреть, что делает система Oracle. Например, Linux предоставляет для трассировки системных вызовов инструментальное средство `strace`. В других операционных системах такие программы называются иначе: `truss` для ОС Sun Solaris (в действительности `truss` изначально был инструментом трассировки системных вызовов для Unix), `sotrace` для IBM AIX, `tusc` для HP-UX и `strace` для Windows. В дальнейшем я буду говорить о принятом в Linux `strace` как о семействе инструментов трассировки системных вызовов.



На момент написания этой книги инструментарий `strace` для различных операционных систем доступен по адресу <http://www.pugcentral.org/howto/truss.htm>.

Программа `strace` проста в употреблении. Например, можно непосредственно наблюдать за деятельностью ядра Oracle, выполнив такую команду:

```
$ strace -p 12417
read(7,
```

В данном случае `strace` сообщает о том, что приложение Linux с идентификатором процесса ID 12417 (в моей системе это процесс ядра Oracle) вызвала функцию чтения `read` и ожидает выполнения этого вызова (поэтому в выводе отсутствует правая скобка).

Чрезвычайно полезно просматривать вывод `strace` и вывод трассировки Oracle SQL одновременно в двух окнах, чтобы видеть, *когда* строки появляются в обоих потоках вывода. Вызовы `write`, посредством которых ядро Oracle передает данные трассировки, естественно, появляются

ся в выводе `strace` именно тогда, когда и ожидается. Благодаря присутствию этих вызовов легче понять, в какой момент действия ядра Oracle порождают данные трассировки. В Oracle9i для Linux (и Oracle9i для некоторых других операционных систем) сопоставить вывод `strace` и вывод трассировки SQL очень просто, т. к. значения, возвращаемые `gettimeofday`, сразу же появляются в файле трассировки в качестве значений `tim`. Применяя таким образом одновременно `strace` и трассировку SQL, можно подтвердить или опровергнуть тот факт, что ваше ядро Oracle ведет себя так же, как приведенный в последующих разделах псевдокод.



Применение `strace` вызовет значительный эффект влияния измерителя (*measurement intrusion effect*) на производительность исследуемой программы. Этот эффект обсуждается ниже в данной главе.

Фактическая продолжительность

Ядро Oracle получает все свои временные статистики с помощью вызовов операционной системы, в которой оно выполняется. В примере 7.1 показано, как программа, подобная ядру Oracle, вычисляет продолжительность собственных действий.

Пример 7.1. Программа измеряет собственное время отклика

```
t0 = gettimeofday; # отметить момент времени, непосредственно предшествующий
                  # выполнению какого-то действия (do_something)

do_something;
t1 = gettimeofday; # отметить момент времени, следующий непосредственно
                  # за выполнением действия do_something
t = t1 - t0;       # t - это приблизительная длительность такого действия
```

Функция `gettimeofday` — это системный вызов, имеющийся в POSIX-совместимых системах. Обратившись к документации на ОС, вы узнаете, что `gettimeofday` возвращает вызывающей программе структуру данных языка C, содержащую количество секунд и микросекунд, прошедших с начала эры Unix — 00:00:00 UTC 1 января 1970 года.



Обычно сведения о таких системных вызовах доступны в документации на ОС. Например, в Unix-системах для получения информации о `gettimeofday` можно ввести в командной строке `man gettimeofday`. Получить определение POSIX можно по адресу http://www.unix-systems.org/single_unix_specification/.

Имейте в виду, что в своем псевдокоде я, чтобы не отвлекать вас, опустил ряд второстепенных деталей. Например, на самом деле функция `gettimeofday` возвращает не время, а 0 в случае успешного выполнения и -1 при неудаче. «Возвращаемое» время записывается в двухэлементную (составляющая секунд и составляющая микросекунд) структуру по адресу, указанному первым аргументом вызова `gettimeofday`. Я считаю, что приведе-

ние всех этих подробностей в псевдокоде лишь усложнило бы описание.

Попробуем изобразить выполнение примера 7.1 на оси времени, как показано на рис. 7.2. На рисунке значение `gettimeofday` $t_0 = 1492$ в момент начала выполнения функции под названием `do_something`. В момент окончания работы `do_something` значение `gettimeofday` $t_1 = 1498$. Следовательно, измеренная длительность исполнения `do_something` – это $t = t_1 - t_0 = 6$ тактов системных часов.



Чтобы упростить восприятие, я взял для времени значения в интервале 1492–1499. Конечно же, они не похожи на реальные значения секунд и микросекунд, которые `gettimeofday` возвращает в двадцать первом веке. Можно считать, что это последние несколько разрядов реальных значений времени.

Ядро `Oracle` выводит две разновидности фактической продолжительности: величина `e` отражает длительность отдельного вызова базы данных, а величина `ela` – протяженность последовательности инструкций, исполняемой процессом ядра `Oracle` и снабженной измерительными средствами (часто это системный вызов). Ядро выполняет такие вычисления, исполняя код, организованный приблизительно так же, как псевдокод в примере 7.2. Обратите внимание, что ядро задейству-

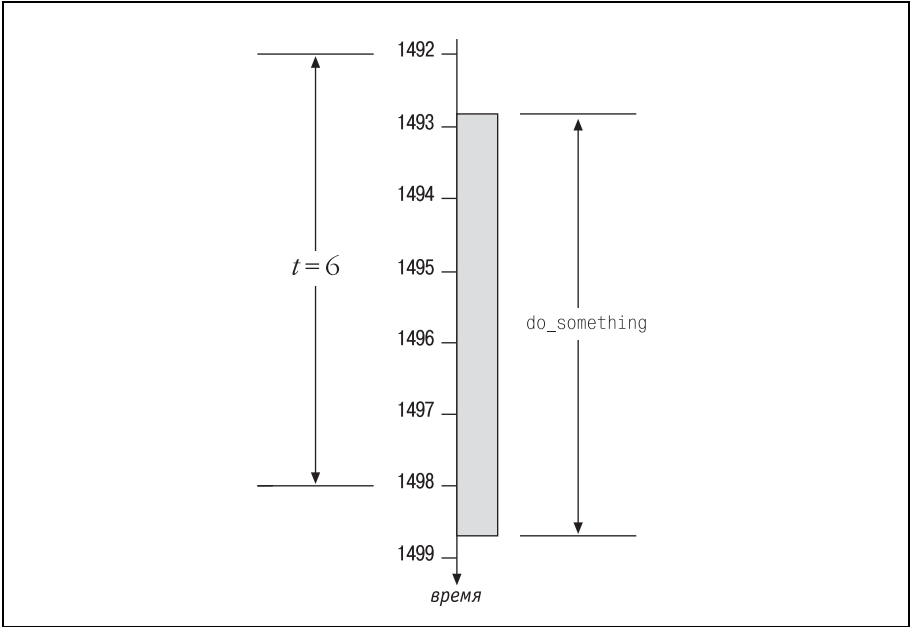


Рис. 7.2. Функция `do_something` начинает работу после такта 1492 и заканчивает ее после такта 1498, что в результате дает время отклика, равное 6 тактам системных часов

ет метод, приведенный в примере 7.1, в качестве основного стандартного блока для построения метрик *e* и *ela*.

Пример 7.2. Псевдокод, показывающий, как ядро Oracle измеряет собственное время исполнения

```

procedure dbcall {
    e0 = gettimeofday;      # отметить текущее время
    ...                    # выполнить вызов БД (возможен вызов wevent)
    e1 = gettimeofday;      # отметить текущее время
    e = e1 - e0;            # фактическая длительность вызова БД
    print(TRC, ...);        # вывести строку PARSE, EXEC, FETCH и т.д.
}

procedure wevent {
    ela0 = gettimeofday;    # отметить текущее время
    ...                    # выполнить событие ожидания
    ela1 = gettimeofday;    # отметить текущее время
    ela = ela1 - ela0;       # ela - это продолжительность события ожидания
    print(TRC, "WAIT...");  # вывести строку WAIT
}

```

Загрузка процессора

Ядро Oracle выводит не только фактическую продолжительность *e* для каждого вызова базы данных и *ela* для каждого системного вызова, но и общий объем процессорного времени, израсходованного каждым вызовом базы данных. В контексте диаграммы состояний и переходов, представленной на рис. 7.1, величина *s* может быть определена как приблизительное количество времени, в течение которого процесс находился в одном из двух режимов:

пользовательском
привилегированном

В POSIX-совместимых системах ядро Oracle получает информацию об использовании процессора от функции *getrusage* (как, например, в ОС Linux и многих других) или от аналогичной функции *times* в HP-UX и некоторых других ОС. Несмотря на то, что спецификации этих двух системных вызовов значительно отличаются друг от друга, я буду употреблять имя *getrusage*, подразумевая при этом обе функции. Каждая функция предоставляет вызывающей программе разнообразную статистику процесса, включая структуры данных, содержащие следующие четыре характеристики:

- Приблизительное время, проведенное процессом в *пользовательском* режиме.
- Приблизительное время, проведенное процессом в *привилегированном* режиме.
- Приблизительное время, проведенное потомками процесса в *пользовательском* режиме.

- Приблизительное время, проведенное потомками процесса в *привилегированном* режиме.

Каждая из этих величин выражается в микросекундах независимо от точности полученных данных.



Скоро вы узнаете, что, хотя стандарт POSIX и предписывает функции `getrusage` возвращать результат в микросекундах, точность информации в нем редко превышает доли сантисекунд.

Ядро Oracle вычисляет `c`, `e` и `ela`, выполняя код, устроенный приблизительно так же, как псевдокод, приведенный в примере 7.3. Обратите внимание, что этот пример основан на примере 7.2 – он включает в себя исполнение системного вызова `getrusage` и последующую обработку результатов. Аналогично вычислениям, производимым `gettimeofday`, ядро Oracle отмечает объем времени процессора, израсходованного процессом на момент начала вызова базы данных, а затем – на момент завершения вызова. Разность между этими двумя значениями (`c0` и `c1`) – это приблизительный объем процессорного времени, израсходованного вызовом базы данных. Чуть позже я расскажу о том, насколько приблизительно эта величина.

Пример 7.3. Псевдокод, показывающий, как ядро Oracle измеряет собственное время исполнения и использованное процессорное время

```
procedure dbcall {
    e0 = gettimeofday;           # отметить текущее время
    c0 = getrusage;              # получить статистику использования ресурса
    ...                          # выполнить вызов БД (возможен вызов wevent)
    c1 = getrusage;              # получить статистику использования ресурса
    e1 = gettimeofday;          # отметить текущее время
    e = e1 - e0;                 # фактическая длительность вызова БД
    c = (c1.utime + c1.stime)    - (c0.utime + c0.stime);
                                # общее время процессора,
                                # израсходованное dbcall
    print(TRC, ...);            # вывести строку PARSE, EXEC, FETCH и т.д.
}

procedure wevent {
    ela0 = gettimeofday;        # отметить текущее время
    ...                          # выполнить событие ожидания
    ela1 = gettimeofday;        # отметить текущее время
    ela = ela1 - ela0;           # ela – это продолжительность события ожидания
    print(TRC, "WAIT...");      # вывести строку WAIT
}
```

Неучтенное время

Анализируя файлы трассировки, практически в каждом вы обнаружите некоторое несоответствие фактического времени отклика и периода времени, учтенного в этом файле. В силу причин, о которых мы

поговорим чуть позже, неполный учет встречается чаще, чем учет лишнего времени. В этой книге в обеих ситуациях мы будем говорить о *неучтенном времени*. Если времени не хватает, будем говорить о положительной неучтенной длительности. Если же будет учтено какое-то лишнее время, то неучтенная длительность будет отрицательной. Наличие неучтенного времени в файлах трассировки Oracle может быть вызвано пятью разными явлениями:

- Влиянием измерителя
- Двойным учетом занятости процессора
- Ошибкой квантования
- Наличием времени, в течение которого процесс не выполняется
- Наличием кода ядра Oracle, не имеющего измерительных средств

В последующих разделах рассмотрено влияние каждого из этих факторов на количество неучтенного времени.

Влияние измерителя

Любая прикладная программа, пытающаяся измерить длительность выполнения собственных подпрограмм, подвержена ошибке, называемой *влиянием измерителя* [Malony et al. (1992)]. Эффект влияния измерителя – это разновидность ошибки, возникающей из-за того, что длительность выполнения измеряемой подпрограммы отличается от длительности выполнения той же подпрограммы, когда она не подвергается измерениям. В последние годы у меня *не было* причин полагать, что эффект влияния измерителя сколько-нибудь заметно сказывался на проанализированных мною измерениях времени отклика Oracle. Однако знание этого эффекта помогло мне опровергнуть ошибочные аргументы в отношении ненадежности временных характеристик функционирования Oracle.

Разберемся с эффектом влияния измерителя на примере. Пусть у нас есть такая программа *U*:

```
program U {
    # нет измерительных средств
    do_something;
}
```

Наша цель в том, чтобы определить, сколько времени заняло исполнение подпрограммы *do_something*. Для этого снабжаем необходимыми инструментами нашу программу *U* и получаем новую программу *I*:

```
program I {
    # есть измерительные средства
    e0 = gettimeofday; # измерение
    do_something;
    e1 = gettimeofday; # измерение
    printf("e=%.6f sec\n", (e1-e0)/1E6);
}
```

Мы ожидаем, что новая программа I выведет длительность выполнения `do_something`. Но выводимое значение лишь приблизительно равно времени выполнения `do_something`. Полученное значение $e_1 - e_0$, выраженное в секундах, включает в себя не только продолжительность `do_something`, но и продолжительность одного вызова `gettimeofday`. На рис. 7.3 наглядно показано, почему так происходит.

Эффект влияния измерителя сказывается на результатах измерений следующим образом:

- Время выполнения I превышает время выполнения U на величину длительности двух вызовов `gettimeofday`.
- Измеренная продолжительность `do_something` в программе I превышает фактическое время выполнения `do_something` на величину длительности одного вызова `gettimeofday`.

Этот эффект минимален в тех приложениях, где длительность одного вызова `gettimeofday` мала по отношению к продолжительности любой измеряемой подпрограммы, подобной `do_something`. Однако в системах с неэффективно реализованными вызовами `gettimeofday` (думаю, что так можно охарактеризовать версии HP-UX, предшествующие версии 10), эффект может быть значительным.

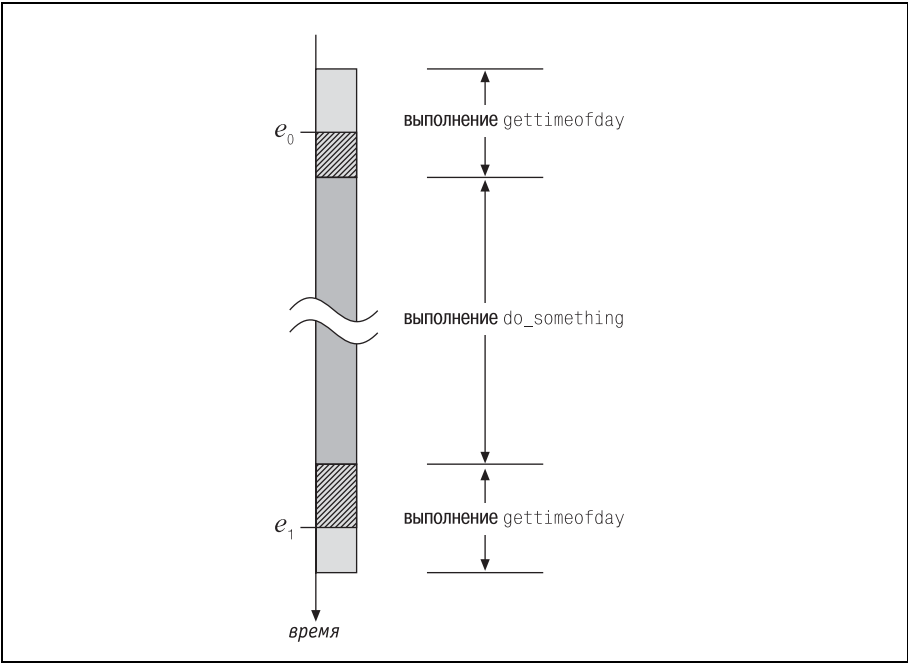


Рис. 7.3. Значение $e_1 - e_0$ лишь приблизительно равно длительности исполнения `do_something`; она также включает в себя полную длительность одного вызова `gettimeofday` (затененная область)

Эффект влияния измерителя относится к *систематическим ошибкам*. Систематическая ошибка является результатом погрешности эксперимента, которая постоянна от измерения к измерению [Lilja (2000)]. Постоянство влияния измерителя позволяет оценить его воздействие на получаемые данные. Например, для того чтобы измерить эффект влияния измерителя ядра Oracle, вызванный вызовами `gettimeofday`, необходимо знать две величины:

- Количество вызовов таймера, которые ядро Oracle выполняет для определенной операции.
- Ожидаемая продолжительность отдельного вызова таймера.

Зная частоту и среднюю продолжительность вызовов таймера для ядра Oracle, можно количественно оценить их влияние на измеряемую величину. Влияние измерителя – это, вероятно, одна из причин потери времени, с которой мы сталкиваемся при отсчете времени в Oracle9i в главе 5.

Получить эти две характеристики несложно. Для того чтобы определить, сколько раз ядро Oracle вызывает таймер для заданного набора операций базы данных, можно обратиться к инструменту `strace` данной платформы. Ожидаемая продолжительность одного вызова таймера вычисляется с помощью программы, аналогичной коду, приведенному в примере 7.4. Этот код измеряет время выполнения нескольких последовательных вызовов `gettimeofday`, а затем вычисляет их среднюю продолжительность для указанного объема выборки.

Пример 7.4. Вычисление эффекта влияния измерителя – вызовов `gettimeofday`

```
#!/usr/bin/perl

# $Header: /home/cvs/cvm-book1/measurement\040intrusion/mef.pl,v 1.4 2003/
03/19 04:38:48 cvm Exp $
# Cary Millsap (cary.millsap@hotsos.com)
# Copyright (c) 2003 by Hotsos Enterprises, Ltd. All rights reserved.

use strict;
use warnings;
use Time::HiRes qw(gettimeofday);

sub fnum($;$) {
    # возвращает строковое представление числового значения в формате
    # ${precision}f с указанными разделителями
    my ($text, $precision, $separator) = @_;
    $precision = 0 unless defined $precision;
    $separator = "," unless defined $separator;
    $text = reverse sprintf "%.${precision}f", $text;
    $text =~ s/(\d\d\d)(?=\d)(?! \d*\.)/$1$separator/g;
    return scalar reverse $text;
}

my ($min, $max) = (100, 0);
my $sum = 0;
```

```

print "How many iterations? "; my $n = <>;
print "Enter 'y' if you want to see all the data: "; my $all = <>;
for (1 .. $n) {
    my ($s0, $m0) = gettimeofday;
    my ($s1, $m1) = gettimeofday;
    my $sec = ($s1 - $s0) + ($m1 - $m0)/1E6;
    printf "%.06f\n", $sec if $all =~ /y/i;
    $min = $sec if $sec < $min;
    $max = $sec if $sec > $max;
    $sum += $sec;
}
printf "gettimeofday latency for %s samples\n", fnum($n);
printf "\t%.06f    seconds minimum\n", $min;
printf "\t%.06f    seconds maximum\n", $max;
printf "\t%.09f seconds average\n", $sum/$n;

```

В Linux, где я обычно провожу свои исследования (800 МГц Intel Pentium), этот код обычно дает значение задержки gettimeofday приблизительно 2 мкс:

```

Linux$ mef
How many iterations? 1000000
Enter 'y' if you want to see all the data: n
gettimeofday latency for 1,000,000 samples
    0.000001    seconds minimum
    0.000376    seconds maximum
    0.000002269 seconds average

```

Эффект влияния измерителя во многом зависит от реализации операционной системы. Например, на моем ноутбуке под управлением MS Windows 2000 (тоже 800 МГц Intel Pentium) время выполнения gettimeofday составляет в среднем почти 6 мкс, т. е. эффект влияния измерителя в 2,5 раза превышает эффект для Linux-сервера:

```

Win2k$ perl mef.pl
How many iterations? 1000000
Enter 'y' if you want to see all the data: n
gettimeofday latency for 1,000,000 samples
    0.000000    seconds minimum
    0.040000    seconds maximum
    0.000005740 seconds average

```

Экспериментируя подобным образом с системными вызовами, вы начнете понимать, в рамках каких ограничений приходится действовать разработчикам ядра в корпорации Oracle. Именно эффектом влияния измерителя объясняется то, что разработчики стремятся создавать инструменты для измерения времени только для тех событий, длительность которых достаточно велика по сравнению с длительностью измерения. Компромиссное решение заключается в предоставлении полезной информации о времени без ухудшения производительности исследуемого приложения.

Двойной учет занятости процессора

Еще одна неточность соотношения:

$$e \approx c + \sum_{\text{db call}} ela$$

вызвана двойным учетом процессорного времени в правой части выражения. В терминологии ядра Oracle величина c содержит все время, проведенное в пользовательском и привилегированном режимах. Каждое значение ela включает в себя общее время, проведенное внутри измеряемой последовательности инструкций ядра Oracle. Когда такая последовательность инструкций, снабженная измерительными средствами, занимает ресурсы процессора, это будет учтено дважды.

Представьте себе, например вызов базы данных Oracle, выполняющий чтение диска (рис. 7.4). На рисунке исполнение вызова базы данных начинается в момент времени e_0 . В течение периода времени A вызов потребляет процессорное время в пользовательском режиме. В момент времени ela_0 процесс ядра Oracle совершает вызов `gettimeofday`, который предваряет выполнение события ожидания Oracle. В зависимости от операционной системы, выполнение системного вызова `gettimeofday` может на несколько микросекунд переводить процесс ядра Oracle в привилегированный режим, а затем возвращать его в пользовательский режим.

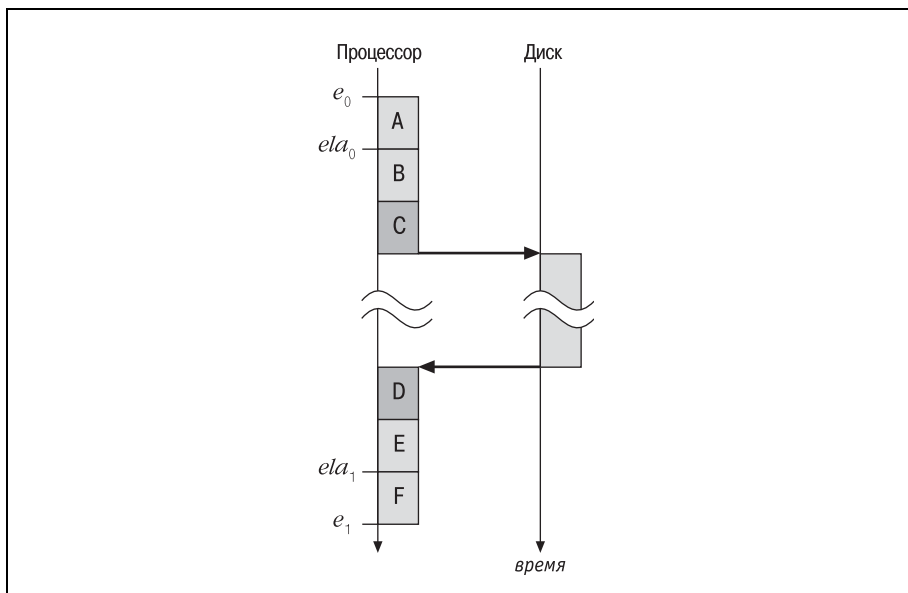


Рис. 7.4. Этот вызов базы данных расходует время процессора как в пользовательском, так и в привилегированном режимах, а затем ожидает чтения с диска



Некоторые реализации ядра Linux позволяют исполнять весь системный вызов `gettimeofday` в пользовательском режиме, что приводит к значительному улучшению производительности (один из примеров находится по адресу http://www-124.ibm.com/linux/patches/?patch_id=597).

Процесс в течение периода *B* расходует процессорное время, находясь в пользовательском режиме, а затем переходит в привилегированный режим до завершения периода *C*. По завершении периода *C* процесс ядра переходит в состояние ожидания и ждет результата запроса от диска.

После завершения запроса диск отправляет сигнал прерывания, вызывающий пробуждение процесса ядра Oracle, вследствие чего тот переходит в состояние готовности к выполнению. На рис. 7.4 видно, что в этот момент времени процессор простаивает, а процесс сразу же обрабатывается планировщиком и переводится в привилегированный режим. В этом режиме процесс Oracle обрабатывает результаты вызова дискового ввода/вывода, в частности, копирует данные из канала ввода/вывода в пользовательскую область памяти процесса Oracle, на что уходит время *D*.

По завершении периода *D* вызов чтения диска завершается, и процесс Oracle переходит в пользовательский режим на период времени *E*. В момент ela_1 процесс Oracle фиксирует время завершения дискового чтения вызовом `gettimeofday`. Затем процесс Oracle приступает к выполнению оставшихся инструкций (также в пользовательском режиме), чтобы завершить вызов базы данных. Наконец в момент e_1 обработка вызова базы заканчивается.

В результате этих действий ядро Oracle сформирует следующую статистику для вызова базы данных:

$$e = e_1 - e_0$$

$$ela = ela_1 - ela_0 = B + C + \text{Disk} + D + E$$

$$c = A + B + C + D + E + F$$

Чуть позже я подробно поясню, как именно вычисляется *c*. Значение *c* будет приблизительно равно сумме длительностей *A*, *B*, *C*, *D*, *E* и *F*. Наверное, вы уже поняли, где возникает двойной учет. Как значение ela , так и *c*, включают в себя длительности *B*, *C*, *D* и *E*. Те участки использования процессора, которые оказались в пределах события ожидания, учтены дважды.

Насколько серьезна проблема двойного учета? К счастью, на практике ею обычно можно пренебречь. Наш опыт анализа более чем тысячи файлов трассировки в *hotsos.com* показывает, что длительности, помеченные как *B*, *C*, *D* и *E* на рис. 7.4, обычно бывают небольшими. Дело в том, что время отклика (т. е. значение ela) измеряемых событий ожидания в Oracle8i и Oracle9i, как правило, значительно превышает время использования процессора. В некоторых редких случаях (один

из них мы вскоре рассмотрим) двойной учет проявляется в небольших фрагментах данных трассировки, но в общей схеме учета времени отклика Oracle эффект *двойного учета процессорного времени*, как правило, можно не принимать в расчет.

Ошибка квантования

Несколько лет назад один приятель убеждал меня в том, что расширенная трассировка SQL в Oracle бесперспективна для диагностики производительности. Он аргументировал это тем, что в эпоху гигагерцевых процессоров разрешающая способность в одну сантисекунду тогдашнего ядра Oracle8i практически бесполезна. Однако во многих сотнях проектов в этой области расширенная трассировка SQL работала практически безупречно (даже эта идея с сантисекундами из Oracle8i себя оправдывала). Достоверность расширенной трассировки SQL трудно оценить, не понимая, что такое *разрешающая способность измерений и ошибка квантования*.

Разрешающая способность измерительной системы

Во времена моего детства одним из преимуществ жизни в эпоху прогресса было изобретение электронного будильника. Цифровые часы чрезвычайно просты для восприятия, даже для маленького мальчика, который еще не умеет определять время по стрелочным часам. Тяжело ошибиться, когда видишь перед собой большие красные цифры «7:29». Имея дело с аналоговыми часами, ребенок может перепутать пять и семь часов, но отличие цифры 5 от 7 очевидно.

Но и у электронных часов есть свои проблемы. Как, посмотрев на электронные часы, показывающие «7:29», определить, сколько осталось до 7:30? Проблема именно в том, что никак. Глядя на электронные часы, нельзя сказать, сколько сейчас времени: 7:29:00, 7:29:59 или где-то посередине. Что касается стрелочных часов, то даже если на них нет секундной стрелки, всегда можно определить доли минуты по тому, насколько минутная стрелка близка к следующему делению на циферблате.

Все значения времени, измеряемые цифровыми компьютерами, получены от *интервальных таймеров*, которые ведут себя как старые электронные часы у нашего изголовья. Интервальный таймер – это устройство, которое тикает через определенные интервалы времени. Значения времени, полученные от интервального таймера, могут обладать интересными особенностями. Для того чтобы осознанно принимать решения на основе данных, полученных от интервальных таймеров Oracle, необходимо осознавать ограниченность их возможностей.

Разрешающая способность интервального таймера – это период времени между двумя последовательными тиками. Разрешающая способность таймера – это величина, обратная *частоте* таймера. То есть тай-

Принцип неопределенности Гейзенберга и анализ компьютерной производительности

Проблема измерения продолжительности компьютерных событий посредством дискретных часов аналогична знаменитому *принципу неопределенности* квантовой физики. Сформулированный Вернером Гейзенбергом в 1926 году принцип неопределенности утверждает, что произведение неопределенности местоположения частицы на неопределенность скорости ее движения и на массу частицы не может быть меньше определенной величины, называемой постоянной Планка [Hawking (1988) 55]. Следовательно, для очень маленьких частиц невозможно одновременно точно знать и где находится частица, и с какой скоростью она перемещается.

Аналогично, некоторые вещи в вычислительной системе тяжело измерить очень точно, особенно при помощи программных часов. Уменьшение разрешающей способности приводит к более точным измерениям, но применение такого разрешения для программных часов может вызвать значительное ухудшение производительности исследуемого приложения. Далее в главе будет рассмотрен такой пример (при обсуждении разрешающей способности системной функции `getrusage`).

В дополнение к тому влиянию, которое разрешающая способность оказывает на результат измерения временных характеристик приложения, на время исполнения пользовательской программы также воздействует и эффект влияния измерителя. Совокупное влияние такого незапланированного поведения измерительных средств на производительность приложения создает то, что аналитики по производительности Oracle могут назвать аналогом эффекта Гейзенберга.

мер, тикающий с частотой 1 ГГц (приблизительно 10^9 тиков в секунду), имеет разрешающую способность приблизительно $1/10^9$ секунд, или около 1 наносекунды. Чем больше значение разрешающей способности таймера, тем менее точную информацию о продолжительности измеряемого события он может сообщить. Но для некоторых таймеров (в особенности реализованных программно) значительное уменьшение разрешающей способности может настолько увеличить накладные расходы системы, что это повлияет на характеристики производительности измеряемого события.

По мере передачи временной статистики от оборудования через различные уровни прикладного программного обеспечения каждый из таких уровней может как оставить ее точность неизменной, так и ухудшить ее. Например:

- Разрешение результата системного вызова `gettimeofday` по стандарту POSIX равно одной микросекунде. Однако многие процессоры Intel Pentium содержат аппаратный счетчик временных меток, который обеспечивает разрешение в одну наносекунду. Например, в Linux вызов `gettimeofday` преобразует значение из наносекунд (10^{-9} секун-

ды) в микросекунды (10^{-6} секунды), выполняя целочисленное деление значения в наносекундах на 1000, что означает потерю трех последних разрядов.

- Статистика `e` в `Oracle8i` обладает разрешающей способностью в одну сантисекунду. Однако большинство современных операционных систем предоставляют информацию `gettimeofday` с точностью до микросекунды. Ядро `Oracle8i` преобразует значение из микросекунд (10^{-6} секунды) в сантисекунды (10^{-2} секунды), выполняя целочисленное деление значения в микросекундах на 10 000, что означает потерю четырех последних разрядов значения [Wood (2003)].

То есть фактически каждая величина `e` и `ela`, выведенная ядром `Oracle8i`, представляет собой системную характеристику, значение которой точно определить невозможно, известно лишь, что оно находится в определенном диапазоне. Такой диапазон значений представлен в табл. 7.1. Например, про величину `e=2` в `Oracle8i` можно сказать, что на самом деле она может представлять любую длительность e_a в следующем диапазоне:

$$2,0000 \text{ сантисекунд} \leq e_a \leq 2,9999 \text{ сантисекунд}$$

Таблица 7.1. Одно значение статистики `e` или `ela` в `Oracle8i` соответствует диапазону фактических значений времени

Статистика <code>e</code> , <code>ela</code> (сантисекунд)	Минимально возможное значение <code>gettimeofday</code> (сантисекунд)	Максимально возможное значение <code>gettimeofday</code> (сантисекунд)
0	0,000 000	0,999 999
1	1,000 000	1,999 999
2	2,000 000	2,999 999
3	3,000 000	3,999 999
...

О важности тестирования системы

Инструмент, подобный `strace`, позволяет проверить предположения о точности измерения времени ядром Oracle для конкретной системы. У нас есть один файл трассировки, сформированный версией Oracle 9.2.0.1.0 для Compaq OSF1, в котором `e` имеет разрешение 3333.3 мкс, `e` – разрешение 1000 мкс, а `ela` – 10000 мкс. Эти данные вызывают резонный вопрос о том, действительно ли значения `e` и `ela` для данной платформы получены от одного системного вызова `gettimeofday`. (Если бы это было так, т. е. значения `e` и `ela` для данной платформы были бы результатами одного системного вызова, почему бы они могли иметь очевидно различную точность измерения?) При наличии трассировки системного вызова на этот вопрос было бы очень просто ответить. В противном случае нам остается только гадать.

Определение ошибки квантования

Ошибка *квантования* обозначается буквой E и определяется как разность между реальной продолжительностью события e_a и его измеренной продолжительностью e_m . То есть

$$E = e_m - e_a$$

Вернемся к примеру 7.1, точнее, к его отображению на шкале времени на рис. 7.5. На этом рисунке каждое деление соответствует такту интервального таймера, подобного таймеру `gettimeofday`. В момент начала `do_something` таймер имел значение $t_0 = 1492$, а в момент завершения `do_something` — $t_1 = 1498$. То есть измеренная продолжительность вызова `do_something` равна $e_m = t_1 - t_0 = 6$. Однако если измерить длину отрезка, соответствующего продолжительности e_a на рисунке, то окажется, что реальная длительность исполнения `_something` (e_a) равна 5,875 такта. Точное значение длительности можно получить, измерив линейкой высоту отрезка e_a на рисунке, а вот приложение не может получить точное значение e_a , имея в своем распоряжении только интервальный таймер с указанным размером такта. Ошибка квантования составляет $E = e_m - e_a = 0,125$ такта или около 1,7% реальной длительности (5,875 тактов).

Теперь рассмотрим случай, в котором длительность исполнения `do_something` близка к разрешению таймера, как это показано на рис. 7.6. В левой части рисунка длительность `do_something` покрывает всего одно

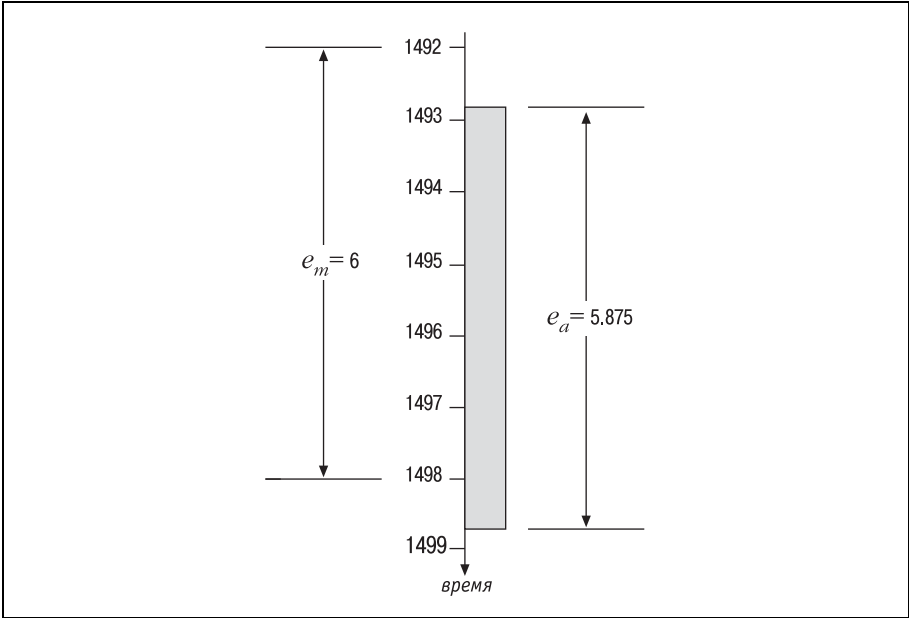


Рис. 7.5. Интервальный таймер с удовлетворительной точностью измеряет длительности событий, охватывающих множество тактов его часов

деление шкалы, так что его измеренная продолжительность $e_m = 1$. Однако фактическая продолжительность события составляет всего $e_a = 0,25$ (это значение можно проверить, измерив длину отрезка на рисунке). В данном случае ошибка квантования вычисляется как $E = 0,75$, т. е. составляет целых 300% реальной длительности события (0,25 такта). В правой части рисунка выполнение вызова `do_something` не захватывает ни одного такта системных часов, и его измеренная продолжительность $e_m = 0$, в то время как реальная длительность $e_a = 0,9375$. Теперь ошибка квантования $E = -0,9375$ и составляет -100% реальной длительности события (0,9375 такта).

Упрощенно ошибку квантования можно описать следующим образом:

Точность любого измерения, выполненного интервальным таймером, не превышает единицы его разрешающей способности.

Если формулировать строже, то разность двух отсчетов цифровых часов равна сумме фактической продолжительности и ошибки квантования, точное значение которой определить невозможно, известен лишь диапазон таких значений: приблизительно от -1 такта до $+1$ такта системных часов. Если обозначить разрешение некоторого таймера x переменной r_x , то получится следующее соотношение:

$$x_m - r_x < x_a < x_m + r_x$$

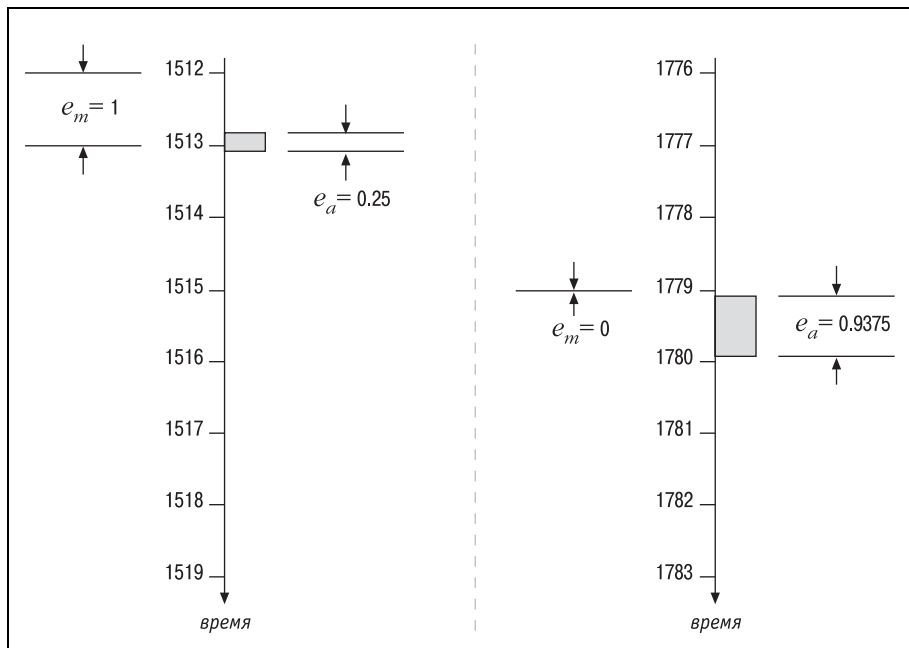


Рис. 7.6. Интервальный таймер неточно измеряет длительности событий, которые не захватывают ни одного такта или продолжаются в течение всего нескольких тактов его часов

Ошибка квантования E , свойственная любому дискретному (цифровому) измерению, – это равномерно распределенная *случайная переменная* (см. главу 11) с диапазоном значений $-r_x < E < r_x$, где r_x – это разрешающая способность интервального таймера.

Любое значение фактической продолжительности, выводимое ядром Oracle (или каким-то другим программным обеспечением), следует воспринимать с учетом точности измерений. Например, если в файле трассировки Oracle8i встречается значение $e=4$, то *не стоит* думать, что реальная длительность какого-то события равнялась 4 сантисекундам. На самом деле такое значение показывает, что если разрешение таймера составляет не более 1 сантисекунды, то реальная продолжительность события находится в диапазоне от 3 до 5 сантисекунд. И это наиболее точная из доступных вам оценок.

Если собранные значения малы, то такая погрешность может привести к курьезным результатам. Например, нельзя даже будет корректно сравнить длительности событий, измеренные длительности которых приблизительно равны. Несколько таких курьезных ситуаций изображено на рис. 7.7. Представим, что таймер тикает с интервалом в одну

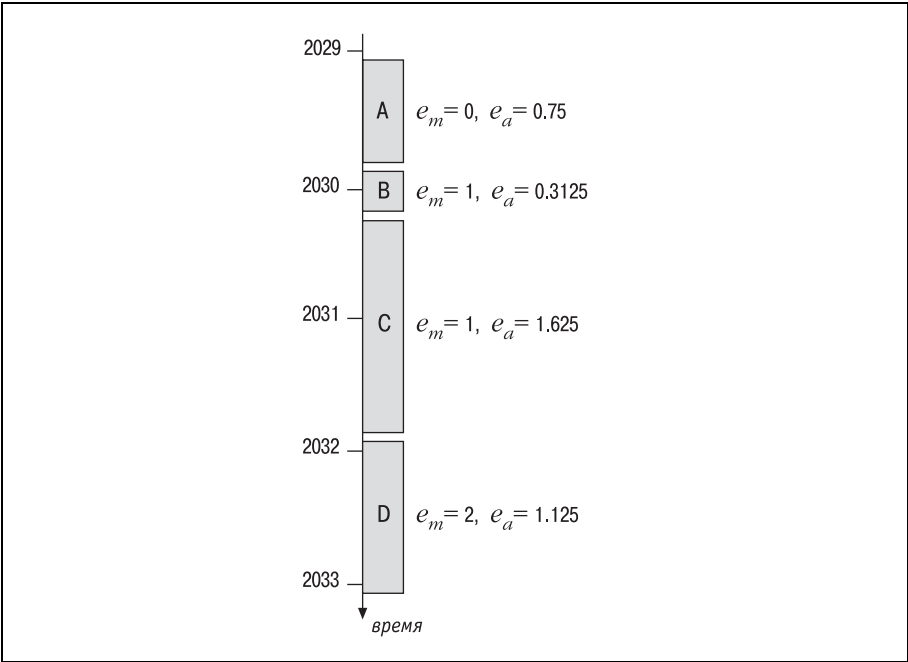


Рис. 7.7. Точность любого измерения длительности при помощи интервального таймера не превышает единицы разрешающей способности такого таймера. Имейте в виду, что события, измеренная длительность которых равна n тактам, в действительности могут быть более продолжительными, чем события с измеренной длительностью в $n + 1$ такт

сантисекунду. Это имитирует работу Oracle8i, где значащие разряды, следующие за сотыми долями секунды, отсекаются. На рисунке видно, что фактическая продолжительность события *A* превышает фактическую продолжительность события *B*, но их измеренные продолжительности находятся в обратном отношении. Событие *C* заняло больше времени, чем *D*, при этом *D* имеет большую измеренную продолжительность. Обобщая, можно сказать, что любое событие с измеренной продолжительностью $n + 1$ может иметь фактическую длительность, которая больше, равна или *даже меньше*, чем у другого события с измеренной продолжительностью n . И вы не можете знать, какое отношение имеет место в данном случае.

Интервальный таймер может выполнять измерения только с точностью до ± 1 такта системных часов, но на практике это ограничение не уменьшает полезность применения таких таймеров. При больших объемах выборок положительные и отрицательные ошибки квантования постепенно компенсируют друг друга. Например, сумма ошибок квантования для случая, изображенного на рис. 7.6, равна:

$$E_1 + E_2 = 0,75 + (-0,9375) = 0,1875$$

В отличие от значительных относительных величин отдельных ошибок их сумма оказывается гораздо меньше и составляет 16% от суммы реальных длительностей событий. Проанализировав в *hotsos.com* несколько сотен файлов трассировки SQL, полученных от сотен различных систем Oracle, мы выявили тенденцию взаимного погашения положительной и отрицательной ошибок квантования в файлах в несколько сотен строк. В большинстве случаев величина суммарной ошибки не выходит за пределы $\pm 10\%$ общего времени отклика, измеренного в файле трассировки.

Сложности измерения процессорного времени

Возможно, вы обратили внимание на то, что системные вызовы `gettimeofday` дают существенно большую точность, чем `getrusage`. Несмотря на то, что псевдокод в примере 7.3 создает ощущение, что `gettimeofday` и `getrusage` делают практически одно и то же, в реальности эти две функции работают совершенно по-разному. И в результате имеют абсолютно разную точность.

Как работает `gettimeofday`

Разобраться, как работает `gettimeofday`, гораздо легче, чем сделать это для `getrusage`. Будем рассматривать в качестве примера Linux на процессоре Intel Pentium. Как я уже говорил, процессор Intel Pentium имеет аппаратный счетчик временных меток TSC, который обновляется с каждым тактом аппаратных часов. Например, процессор с частотой 1 ГГц обновляет счетчик приблизительно миллиард раз в секунду [Bovet and Cesati (2001) 139–141]. Подсчитывая количество тактов,

зарегистрированных счетчиком с тех пор, когда пользователь установил время командой `date`, ядро Linux может определить, сколько тактов прошло с момента начала эпохи Unix. Возвращаемый `gettimeofday` результат – это полученное число, усеченное до микросекунд (для обеспечения соответствия функции `gettimeofday` стандарту POSIX).

Как работает `getrusage`

Операционная система может учитывать процессорное время, израсходованное процессом в пользовательском и привилегированном режиме, двумя способами:

Опрос (Polling)

Операционная система может содержать специальный код, позволяющий каждому выполняющемуся процессу через фиксированные интервалы времени обновлять собственную таблицу `rusage`. На каждом интервале каждый работающий процесс может обновлять собственную статистику расходования ресурсов процессора, считая, что он использовал процессор в течение всего интервала в том режиме, в каком процесс находится в текущий момент.¹

Событийно обусловленные измерения (Event-based instrumentation)

Операционная система может содержать специальный код, который при каждом переходе процесса в *пользовательский* или *привилегированный* режим осуществляет вызов таймера высокой точности. При каждом выходе процесса из такого состояния ОС может повторно вызывать таймер и выводить величину (в микросекундах) разности между двумя вызовами в структуру `rusage` процесса.

В большинстве операционных систем (по крайней мере, по умолчанию) применяется опрос. Так, Linux обновляет несколько атрибутов для каждого процесса, в том числе использованное до указанного момента процессорное время, при каждом прерывании по таймеру [Bovet and Cesati (2001) 144–145]. Некоторые операционные системы поддерживают событийно обусловленные измерения. Например, такая возможность имеется в Sun Solaris под названием «*учет микросостояний*» (*microstate accounting*) [Cockroft (1998)].

При учете микросостояний ошибка квантования ограничена одной единицей разрешения таймера на каждое переключение состояния. В случае применения таймера с высоким разрешением (подобного `gettimeofday`) общая ошибка квантования для статистики использования процессора, полученная в результате учета микросостояний, оказывается совсем небольшой. Но повышение точности достигается за счет увеличения эффекта влияния измерителя. Однако, как вы сейчас

¹ Другими словами, если в момент обновления процесс находится, например, в пользовательском режиме, то считается, что и в течение всего интервала он находился в пользовательском режиме. – *Примеч. науч. ред.*

увидите, в случае применения опроса ошибка квантования может быть значительно больше.

Независимо от того, каким способом получена информация о расходовании ресурсов, операционная система предоставляет эту информацию любому процессу, которому она необходима, посредством системного вызова, подобного `getrusage`. Стандарт POSIX требует, чтобы в качестве единицы измерения в функции `getrusage` выступали микросекунды, но для систем, получающих данные `rusage` путем опроса, реальное разрешение получаемых данных зависит от частоты прерываний по таймеру.

В большинстве систем частота составляет 100 прерываний в секунду или 1 прерывание в сантисекунду (в текстах операционных систем значения часто выражены в миллисекундах: 1 сантисекунда = 10 миллисекунд = 0,010 секунды). Частота прерываний по таймеру во многих системах устанавливается параметром, но большинство системных администраторов оставляет ее равной 100 прерываниям в секунду. Если попросить систему обслуживать прерывания чаще 100 раз в секунду, то точность измерения времени будет выше, но при этом пострадает производительность. Даже если обслуживать прерывания всего в десять раз чаще, накладные расходы на работу планировщика в привилегированном режиме возрастут в десять раз. Такое решение нельзя считать разумным компромиссом.

Если операционная система соответствует стандарту POSIX, то определить разрешение ее планировщика поможет следующая программа на Perl [Chiesa (1996)]:

```
$ cat clkres.pl
#!/usr/bin/perl
use strict;
use warnings;
use POSIX qw(sysconf _SC_CLK_TCK);
my $freq = sysconf(_SC_CLK_TCK);
my $f = log($freq) / log(10);
printf "getrusage resolution %.${f}f seconds\n", 1/$freq;
$ perl clkres.pl
getrusage resolution: 0.01 seconds
```

Если системные часы имеют разрешение в одну сантисекунду, то `getrusage` может возвращать значение в микросекундах, но такие значения никогда не будут содержать корректную информацию в разрядах младше, чем сотые доли секунды.

Причина, по которой я вам все это объясняю, заключается в том, что ошибка квантования статистики `s` для Oracle в корне отличается от ошибки квантования статистик `e` и `ela`. Вернемся к утверждению:

Точность любого измерения, выполненного интервальным таймером, не превышает единицы его разрешающей способности.

Проблема статистики s заключается в том, что величина, возвращаемая функцией `getrusage`, – это *на самом деле не продолжительность*. То есть «продолжительность» использования процессора, возвращаемая `getrusage`, не вычисляется как разность пары измерений интервального таймера.

- В системах, где ведется учет микросостояний, объем использования процессора вычисляется как сумма очень большого количества коротких временных интервалов.
- В системах, получающих информацию `rusage` путем опроса, объем использования процессора – это приблизительная оценка продолжительности, полученная в процессе опроса.

Получается, что в любом случае ошибка квантования, присущая статистике s , может быть *гораздо более серьезной*, чем просто один такт системных часов. Проблема актуальна даже в системах, применяющих учет микросостояний. В системах, которые это не делают, все обстоит еще хуже.

На рис. 7.8 изображена типичная ситуация при организации опроса, в которой ошибки определения времени занятости процессора в пользовательском режиме приводят к учету лишнего времени при вычислении времени отклика вызова базы данных. Диаграмма последовательности состояний на этом рисунке приводит время занятости процессора в пользовательском режиме и время системного вызова, потребленные вызовом базы данных. На оси ЦПУ отмечены прерывания системного таймера с интервалом в 1 миллисекунду. Размер рисунка не позволяет

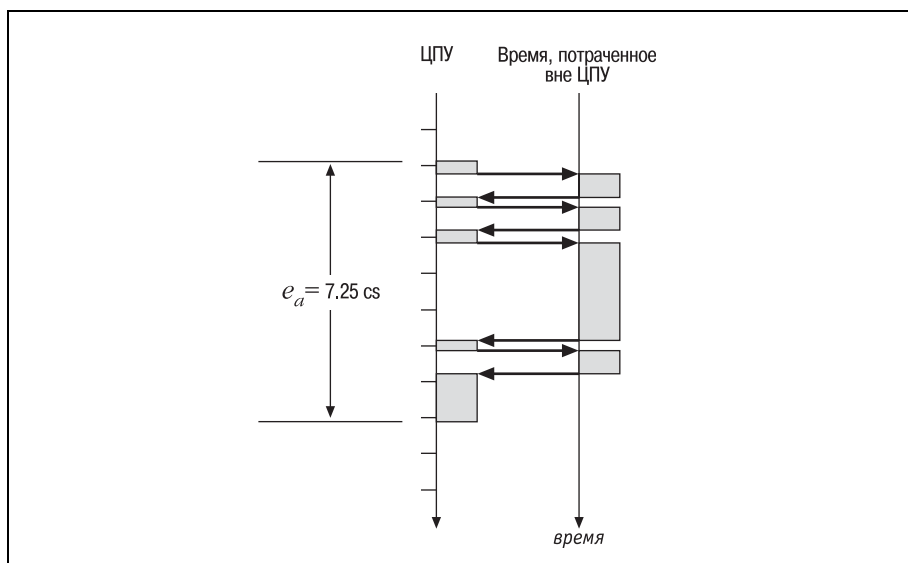


Рис. 7.8. Подсчет времени ЦПУ методом опроса в `getrusage` может привести к сопоставлению излишнего времени отклика отдельному вызову базы данных

отобразить на оси «Время, потраченное вне ЦПУ» 10 000 тактов, которые проходят между каждой парой тактов на оси ЦПУ.

Можно предположить, что в результате операций, изображенных на рис. 7.8, ядро Oracle9i сформирует данные трассировки, подобные приведенным в примере 7.5. Я вычислил ожидаемые значения статистик `e` и `ela`, измеряя длины отрезков на рисунке. Благодаря высокой точности таймера `gettimeofday`, с помощью которого я выполнял измерения, ошибка квантования таких измерений `e` и `ela` пренебрежимо мала.

Пример 7.5. Временные статистики Oracle9i, которые могли бы быть получены в результате событий, изображенных на рис. 7.8

```
WAIT #1: ...ela= 6250
WAIT #1: ...ela= 6875
WAIT #1: ...ela= 32500
WAIT #1: ...ela= 6250
FETCH #1:c=60000,e=72500,...
```

Реально затраченное вызовом базы данных процессорное время составляет 2,5 сантисекунды. Это значение я получил, физически измерив длины соответствующих отрезков на рис. 7.8. Однако `getrusage` получает свои данные об использовании процессора из структуры расходования ресурсов процессом, обновляемой в результате опроса по прерываниям таймера. При каждом прерывании планировщик процессов операционной системы добавляет одну полную сантисекунду (10 000 мкс) процессорного времени любому процессу, исполняемому в данный момент времени. Соответственно, `getrusage` сообщит о том, что вызов базы данных на рис. 7.8 занял шесть полных сантисекунд времени процессора. Проверить результат очень просто – посмотрите на рис. 7.8 и сосчитайте количество тактов, пришедшихся на моменты использования процессора.

На рисунке все кажется разумным, но обратите внимание на неучтенное время:

$$\begin{aligned}\Delta &= e - \left(c + \sum_{\text{db call}} ela \right) \\ &= 72500 - (60000 + (6250 + 6875 + 32500 + 6250)) \\ &= -39375\end{aligned}$$

Если значение неучтенного времени меньше нуля, значит, данные трассировки содержат отрицательную величину «упущенного времени». Другими словами, вызову базы данных приписаны лишние 39 375 мкс. Это число выглядит настораживающе большим, но не забывайте о том, что на самом деле это всего около 4 сантисекунд. Реально же вызов базы данных затратил в пользовательском режиме всего 25 000 мкс времени процессора (опять-таки, это значение я получил мошенически, измерив длины отрезков на рис. 7.8).

Выявление ошибки квантования

Ошибка квантования $E = e_m - e_a$ — это разность между реальной продолжительностью события e_a и его измеренной продолжительностью e_m . У вас нет возможности узнать реальную продолжительность события, следовательно, нельзя и обнаружить ошибку квантования, основываясь на отдельном значении. Однако можно доказать наличие ошибки квантования, исследуя *группы* родственных статистик. Мы уже рассматривали пример, в котором удалось выявить ошибку квантования. В примере 7.5 наличие ошибки квантования удалось определить, заметив, что:

$$c + \sum_{\text{db call}} ela > e$$

Ошибку квантования легко выявить, исследуя вызов базы данных и выполняемые им события ожидания в системе с *низкой загрузкой*, где минимизировано влияние других факторов, способных нарушить отношение $e \approx c + \sum ela$.

Рассмотрим фрагмент файла трассировки Oracle8i, который демонстрирует эффект ошибки квантования:

```
WAIT #103: nam='db file sequential read' ela= 0 p1=1 p2=3051 p3=1
WAIT #103: nam='db file sequential read' ela= 0 p1=1 p2=6517 p3=1
WAIT #103: nam='db file sequential read' ela= 0 p1=1 p2=5347 p3=1
FETCH #103:c=0,e=1,p=3,cr=15,cu=0,mis=0,r=1,dep=2,og=4,tim=116694745
```

Данный вызов выборки инициировал ровно три события ожидания. Мы знаем, что приведенные значения c , e и ela должны быть связаны таким приблизительным равенством:

$$e \approx c + \sum_{\text{db call}} ela$$

В системе с низкой загрузкой величина, на которую отличаются левая и правая части приблизительного равенства, указывает на общую ошибку квантования, присутствующую в пяти измерениях (одно значение c , одно значение e и три значения ela):

$$\begin{aligned} E &\approx e_m - \left(c_m + \sum_{\text{db call}} ela_m \right) \\ &= 1 - (0 + (0 + 0 + 0)) \\ &= 1 \end{aligned}$$

С учетом того, что отдельному вызову `gettimeofday` в большинстве систем соответствует лишь несколько микросекунд ошибки, вызванной влиянием измерителя, получается, что ошибка квантования вносит

значительный вклад в «разность» длиной в одну сотисекунду в данных трассировки.

Следующий фрагмент файла трассировки Oracle8i демонстрирует простейший вариант избыточного учета продолжительности, в результате которого возникает отрицательная величина неучтенного времени:

```
WAIT #96: nam='db file sequential read' ela= 0 p1=1 p2=1691 p3=1
FETCH #96:c=1,e=0,p=1,cr=4,cu=0,mis=0,r=1,dep=1,og=4,tim=116694789
```

В данном случае $E = -1$ сотисекунда:

$$\begin{aligned} E &\approx e_m - \left(c_m + \sum_{\text{db call}} ela_m \right) \\ &= 0 - (1 + (0)) \\ &= -1 \end{aligned}$$

При наличии «отрицательной разности» (подобного только что рассмотренному) невозможно все объяснить эффектом влияния измерителя, ведь этот эффект может быть причиной появления только *положительных* значений неучтенного времени. Можно было бы подумать, что имел место двойной учет использования процессора, но и это не соответствует действительности, т. к. нулевое значение `ela` свидетельствует о том, что время занятости процессора вообще не учитывалось для события ожидания. В данном случае ошибка квантования имеет преобладающее влияние и приводит к излишнему учету времени для выборки.

В Oracle9i разрешение временной статистики улучшено, но и эта версия отнюдь не защищена от воздействия ошибки квантования, что видно в предложенном ниже фрагменте файла трассировки для $E > 0$:

```
WAIT #5: nam='db file sequential read' ela= 11597 p1=1 p2=42463 p3=1
FETCH #5:c=0,e=12237,p=1,cr=3,cu=0,mis=0,r=1,dep=2,og=4,tim=1023745094799915
```

В данном случае $E = 640$ мкс:

$$\begin{aligned} E &\approx e_m - \left(c_m + \sum_{\text{db call}} ela_m \right) \\ &= 12237 - (0 + (11597)) \\ &= 640 \end{aligned}$$

Некоторая часть этой ошибки, несомненно, является ошибкой квантования (невозможно, чтобы общее время использования процессора данной выборкой *действительно равнялось нулю*). Несколько микросекунд следует отнести на счет эффекта влияния измерителя.

Наконец, рассмотрим пример ошибки квантования $E < 0$ в данных трассировки Oracle9i:

```

WAIT #34: nam='db file sequential read' ela= 16493 p1=1 p2=33254 p3=1
WAIT #34: nam='db file sequential read' ela= 11889 p1=2 p2=89061 p3=1
FETCH #34:c=10000,e=29598,p=2,cr=5,cu=0,mis=0,r=1,dep=3,og=4,
tim=1017039276445157

```

Теперь $E = -8784$ мкс:

$$\begin{aligned}
 E &\approx e_m - \left(c_m + \sum_{\text{db call}} ela_m \right) \\
 &= 29598 - (10000 + (16493 + 11889)) \\
 &= -8784
 \end{aligned}$$

Возможно, в данном случае имел место двойной учет использования процессора. Также вероятно, что именно ошибка квантования внесла основной вклад в полученное время вызова выборки. Избыточный учет 8784 микросекунд говорит о том, что фактический общий расход процессорного времени вызовом базы данных составил, вероятно, всего около $(10000 - 8784)$ мкс = 1,216 мкс.

Диапазон значений ошибки квантования

Величину ошибки квантования, содержащейся во временных статистиках Oracle, нельзя измерить напрямую. Зато можно проанализировать статистические свойства ошибки квантования в данных *расширенной* трассировки SQL. Во-первых, величина ошибки квантования для конкретного набора данных трассировки ограничена сверху. Легко представить ситуацию, в которой ошибка квантования, вносимая такими характеристиками продолжительности, как e и ela , будет максимальной. Наибольшего значения данная ошибка достигает в том случае, когда в последовательности значений e и ela все отдельные ошибки квантования имеют максимальную величину и их знаки совпадают.

На рис. 7.9 показан пример возникновения описанной ситуации: имеется восемь очень непродолжительных системных вызовов, причем все они попадают на такты интервального таймера. Фактическая длительность каждого события близка к нулю, но измеренная длительность каждого такого события равна одному такту системного таймера. В итоге суммарная фактическая продолжительность всех вызовов близка к нулю, а общая измеренная продолжительность равна 8 тактам. Для такого набора из $n = 8$ системных вызовов ошибка квантования по существу равна $n r_x$, где r_x — это разрешение интервального таймера, с помощью которого измеряется характеристика x .

Думаю, вы обратили внимание, что изображенный на рис. 7.9 случай выглядит надуманно и изобретен исключительно для прояснения вопроса. В реальной жизни подобная ситуация чрезвычайно маловероятна. Вероятность, что n ошибок квантования будут иметь одинаковые знаки, равна всего $0,5^n$. Вероятность того, что $n=8$ последовательных

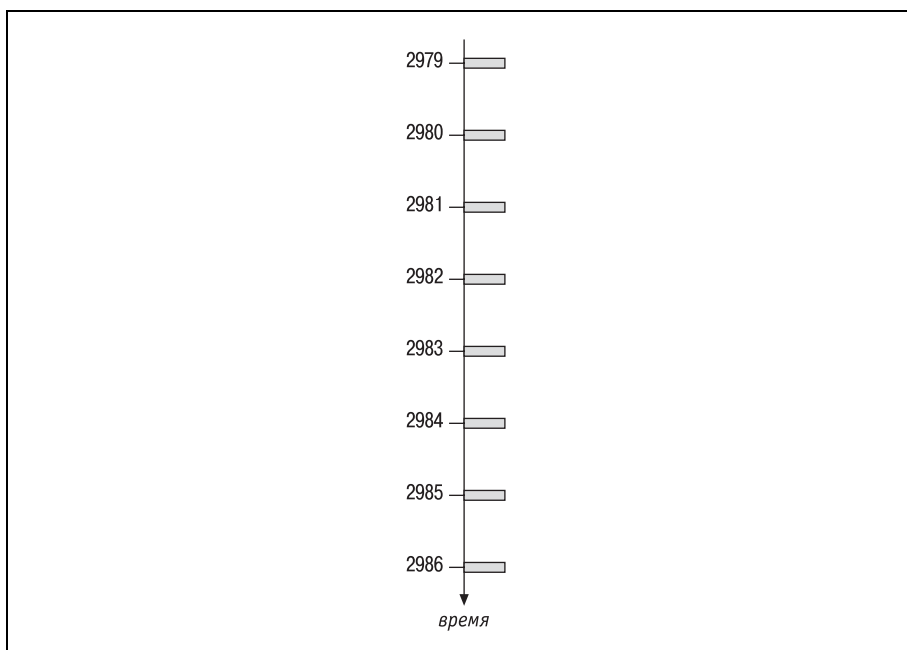


Рис. 7.9. Наихудший вариант накопления ошибки квантования для последовательности измеренных продолжительностей

ошибок квантования будут отрицательными, равна всего 0,00390625 (т. е. приблизительно четыре шанса из тысячи). Для 266 значений шанс совпадения знаков у всех ошибок квантования меньше, чем один из 10^{80} .

Для больших наборов значений длительностей совпадение знаков всех ошибок квантования практически невозможно. Но это не единственное, в чем состоит надуманность ситуации, изображенной на рис. 7.9. Она также предполагает, что *абсолютная величина* каждой ошибки квантования максимальна. Шансы наступления такого события еще более иллюзорны, чем у совпадения всех знаков ошибок. Например, вероятность того, что величина каждой из n имеющихся ошибок квантования превышает 0,9, равна $(1 - 0,9)^n$. Вероятность того, что величина каждой из $n = 266$ ошибок квантования превысит 0,9, составляет всего 1 из 10^{266} .

Вероятность того, что все n ошибок квантования имеют одинаковый знак и абсолютная величина всех из них больше t , чрезвычайно мала и равна произведению рассмотренных ранее вероятностей:

$$P(\text{значения всех } n \text{ ошибок квантования больше } t \text{ или меньше } -t) = (0,5)^n (1 - t)^n$$

Ошибки квантования для продолжительностей (например, значений `e` и `ela` в Oracle) – это случайные числа в диапазоне:

$$-r_x < E < r_x$$

где r_x — это разрешение интервального таймера, с помощью которого измеряется характеристика x (x — это e или ela).

Так как положительные и отрицательные ошибки квантования возникают с равной вероятностью, средняя ошибка квантования для выбранного набора статистик стремится к нулю даже для больших файлов трассировки. Опираясь на теорему Лапласа (Pierre Simon de Laplace, 1810), можно предсказать вероятность того, что ошибки квантования для статистик e и ela будут превышать указанное пороговое значение для файла трассировки, содержащего определенное количество статистик.

Я начал работать над вычислением вероятности того, что *общая* ошибка квантования файла трассировки (включая ошибку, вносимую статистикой c) будет превышать заданную величину, однако мое исследование еще не завершено. Мне предстоит получить распределение ошибки квантования для статистики c , что, как я уже говорил, осложняется особенностями получения этой статистики в процессе опроса. Результаты этих изысканий планируется воплотить в одном из будущих проектов.

К счастью, относительно ошибки квантования есть и оптимистические соображения, которые позволяют не слишком расстраиваться по поводу невозможности определения ее величины:

- Во многих сотнях файлов трассировки Oracle, проанализированных нами в *hotsos.com*, общая продолжительность неучтенного вре-

Что означает «один шанс из десяти в [очень большой] степени»?

Для того чтобы представить себе, что такое «один шанс из 10^{80} », задумайтесь над следующим фактом: ученые утверждают, что в наблюдаемой вселенной содержится всего около 10^{80} атомов (по данным <http://www.sunspot.noao.edu/sunspot/pr/answerbook/universe.html#q70>, <http://www.nature.com/nsu/020527/020527-16.html> и др.). Это означает, что если бы вам удалось написать на каждом атоме нашей вселенной 266 равномерно распределенных случайных чисел от -1 до $+1$, то лишь на *одном* из этих атомов можно было бы ожидать наличия всех 266 чисел с одинаковым знаком.

Представить вторую упомянутую вероятность — «один шанс из 10^{266} » — еще труднее. На этот раз представим себе три уровня вложенных вселенных. То есть что каждый из 10^{80} атомов нашей вселенной сам по себе является вселенной, состоящей из 10^{80} вселенных, каждая из которых в свою очередь содержит 10^{80} атомов. Теперь у нас достаточно атомов для того, чтобы представить себе возможность возникновения ситуации с вероятностью «один из 10^{240} ». Даже во вселенных третьего уровня вложенности вероятность появления атома, для которого все 266 его случайных чисел по абсолютной величине больше 0,9, составит один из 100 000 000 000 000 000 000 000 000 000.

мени в случае *корректного сбора данных* (см. главу 6) чрезвычайно редко превышала 10% общего времени отклика.

- Несмотря на то, что и ошибка квантования, и двойной учет использования процессора могут привести к такому результату, файл трассировки *чрезвычайно* редко содержит отрицательное неучтенное время, абсолютная величина которого превышала бы 10% общего времени отклика.
- В случаях, когда неучтенное время оценивается более чем в 25% времени отклика для корректно собранных данных трассировки, такой объем неучтенного времени *почти всегда* объясняется одним из двух явлений, описанных в последующих разделах.
- Наличие ошибки квантования не лишает нас возможности правильно диагностировать основные причины проблем производительности при помощи файлов расширенной трассировки SQL в Oracle (даже в файлах трассировки Oracle8i, в которых вся статистика приводится с точностью лишь до сотых долей секунды).
- Ошибка квантования становится еще менее значимой в Oracle9i благодаря повышению точности измерений.

В некоторых случаях влияние ошибки квантования способно привести к утрате доверия к достоверности данных трассировки Oracle. Наверное, ничто не может так подорвать боевой дух, как подозрение в недостоверности данных, на которые вы полагаетесь. Думаю, что лучшим средством, призванным укрепить веру в получаемые данные, должно служить четкое понимание влияния ошибки квантования.

Время «невыполнения»

Давайте сделаем небольшой мысленный эксперимент для того, чтобы познакомиться с четвертой причиной наличия неучтенного времени в корректно собранных данных трассировки Oracle. Представьте себе программу *P*, которая ровно десять секунд занимает процессор в пользовательском режиме и формирует вывод на терминал. Пусть такая программа циклически запускается на компьютере с одним процессором. Если тот, кто запускает ее, является единственным пользователем данной системы, то он должен ожидать, что время отклика для каждого выполнения *P* составит 10 секунд.

Наблюдая за использованием процессора на данной однопроцессорной машине в ходе многократного выполнения *P* одним пользователем, можно заметить, что процессор все это время будет работать со 100% загрузкой. Но что будет, если добавить второй экземпляр циклично выполняющейся программы *P* в однопроцессорную систему? В рамках любого одного десятисекундного интервала времени однопроцессорный компьютер может предоставить только десять секунд процессорного времени. То есть невозможно ожидать полного выполнения двух экземпляров программы, требующих по десять секунд процессорного

времени, в течение одного десятисекундного промежутка времени. Можно предположить, что время отклика для каждого из экземпляров *P* возрастет приблизительно до 20 секунд. Столько времени потребуется одному процессору для того, чтобы предоставить по десять секунд своего времени каждому из конкурирующих процессов, если он будет распределять его равномерно и небольшими порциями двум параллельным процессам.

Измерения в эксперименте

Предположим, что мы добавили в наш код измерительные средства, подобные применяемым в ядре Oracle, как это показано в примере 7.6.

Пример 7.6 Программа P оснащена средствами для вывода собственного времени отклика и использованного процессорного времени

```
e0 = gettimeofday;
c0 = getrusage;
P;                                # помните о том, что P не выполняет системных вызовов
c1 = getrusage;
e1 = gettimeofday;
e = e1 - e0;
c = (c1.stime + c1.utime) - (c0.stime + c0.utime);
printf "e=%.0fs, c=%.0fs\n", e, c;
```

Теперь попробуем предсказать значения выводимых временных статистик для каждого указанного уровня параллелизма в однопроцессорной системе (табл. 7.2). Ожидаем, что программа *P* будет потреблять одинаковый общий объем процессорного времени независимо от занятости системы. Но, конечно же, учитывая, что ресурсы процессора по мере увеличения уровня параллелизма разделяется между возрастающим количеством пользователей, можно ожидать замедления выполнения программы и увеличения периодов времени, прошедших прежде, чем программе удастся получить необходимые ей десять секунд времени процессора.

Таблица 7.2. Ожидаемый результат выполнения программы P, потребляющей десять секунд процессорного времени в пользовательском режиме и измеряющей время выполнения для различных уровней параллелизма

Количество пользователей, параллельно запускающих P	Выводимые временные статистики
1	e=10s, c=10s
2	e=20s, c=10s
3	e=30s, c=10s
4	e=40s, c=10s

Мы видим то, что и ожидали, но при этом в некоторых строках таблицы возникает проблема «недостающего времени». Помните, какой была наша модель производительности системы? Длительность выполне-

ния программы должна быть приблизительно равна сумме времени использования процессора и времени, потраченного на исполнение измеряемых «событий ожидания»:

$$e \approx c + \sum_{\text{db call}} ela$$

Однако уже для двух пользователей мы имеем несоответствие:

$$20 \approx 10 + 0$$

Мы имеем право подставить ноль вместо значения `ela`, т. к. знаем, что наша программа не исполняет «событий ожидания», длительность которых могла бы быть измерена. Единственное, что делает наша программа, — это расходует некоторое количество процессорного времени и выводит результат (и даже функция `printf` должна быть исключена из круга подозреваемых, т. к. она вызывается вне области действия таймеров). Очевидно, что соотношение $c + \sum ela = 10$ является очень плохим приближением для $e = 20$. На что ушло «потерянное время»? Из табл. 7.2 видно, что ситуация ухудшается с увеличением количества пользователей. Что же мы сделали не так, добавив в P измерительные средства?

Еще раз о состояниях процессов и переходах

Для того чтобы упростить ответ на вопрос, обратимся к рис. 7.1. Вспомним о том, что даже когда процесс спокойно выполняется в пользовательском режиме, в большинстве систем каждую сотую долю секунды происходит прерывание по системному таймеру. Такое регулярно повторяющееся прерывание переводит каждый работающий процесс в привилегированный режим. Перейдя в привилегированный режим, процесс сохраняет текущий *контекст*, а затем выполняет подпрограмму планировщика (см. раздел «Переход по прерыванию» выше в этой главе). Если присутствует процесс в состоянии готовности к исполнению, то политика планирования системы может потребовать приостановки обслуживания (вытеснения) исполняемого процесса и предоставления готовому к исполнению процессу возможности использовать процессор.

Обратите внимание на то, что при этом происходит с процессом, который выполнялся изначально. Когда он прерывается, то сразу же переходит в привилегированный режим. Заметьте, что у процесса нет никакой возможности выполнить какой-либо код, с тем чтобы узнать, в какое время произошел такой переход. Когда обслуживание процесса приостанавливается, он переходит в состояние готовности к выполнению, ожидая, когда планировщик продолжит его выполнение. Когда наконец наступает его время (возможно, это будет уже через каких-то 10 миллисекунд), процесс проводит в привилегированном режиме столько времени, сколько необходимо для восстановления его контекста, а затем возвращается в пользовательский режим, ровно в то место, на котором остановился.

Как вытеснение процесса влияет на получаемые хронометрические данные? Время, которое процессор проводит в привилегированном режиме, пока планировщик готовится к вытеснению процесса, учитывается как время ЦПУ, использованное процессом. А вот время, проведенное в состоянии готовности к исполнению, *не* считается временем ЦПУ, использованным процессом. Однако когда процесс завершает свою работу, разность $e = e1 - e0$, естественно, включает в себя *все* время, проведенное процессом во *всех* состояниях на диаграмме состояний процесса. В результате все время, проведенное в состоянии готовности к выполнению, учитывается в значении e , но не учитывается ни в объеме процессорного времени, ни в каких-то других характеристиках, измеряемых приложением. Все обстоит так, как будто процесс ударили по голове, а затем разбудили, и при этом нет никакой информации о том, что происходило, пока он был без сознания.

Именно это и происходит с каждым процессом по мере добавления параллельных процессов, как показано в табл. 7.2. Естественно, чем больше процессов находится в состоянии готовности к ожиданию, тем дольше каждому процессу приходится дожидаться своей очереди на использование процессора. Чем дольше ожидание, тем больше общая продолжительность выполнения программы. Для трех и четырех пользователей неучтенное время увеличивается пропорционально. Все просто: один и тот же пирог (процессорное время) делится между все большим и большим количеством едоков (пользователей, исполняющих программу P). Так что «чинить» измерительные инструменты программы P незачем. Достаточно понять, как оценить количество неучтенного времени, которое можно отнести на счет времени, в течение которого процесс не выполнялся.

Наличие и точный размер такого промежутка времени чрезвычайно важны для аналитика по производительности Oracle. Величина этого промежутка позволяет определить на основе данных расширенной трассировки SQL, чем вызваны проблемы производительности – излишней подкачкой или же длительным ожиданием в очереди на использование процессора.

Код ядра Oracle без измерительных средств

Последняя причина недостачи времени в файле трассировки, о которой мы поговорим, – это наличие в ядре Oracle *кода без измерительных средств*. Как вы уже знаете, для вызовов базы данных Oracle предоставляет такие данные, как s и e – общее время использования процессора и общая продолжительность вызова соответственно. Для тех участков кода ядра Oracle, которые могут значительно увеличивать время отклика, не расходуя много процессорного времени, корпорация Oracle предоставляет такой инструмент, как «событие ожидания», характеризующееся продолжительностью и отличительным именем исполняемого сегмента кода.

В главе 12 приведены некоторые сегменты кода, снабженные измерительными средствами, для нескольких распространенных версий ядра Oracle, начиная с 7.3.4. Обратите внимание, что с увеличением номера версии значительно увеличивается и количество таких сегментов. Например, в версии 9.2.0 имеется 146 дополнительных, по сравнению с версией 8.1.7, измеряемых системных вызовов. Конечно, некоторые из этих событий представляют новые возможности продукта. Возможно, что часть новых имен в `V$EVENT_NAME` относится к сегментам кода, которые присутствовали, но не были оснащены измерительными средствами в более ранних версиях ядра Oracle.

Эффект

Если корпорация Oracle оставляет какую-то последовательность инструкций ядра без измерительных средств, то неучтенное время может проявиться в одном из двух вариантов:

Пропущенное время внутри вызова базы данных

Любой код, не содержащий измерительных средств и выполняемый в контексте вызова базы данных, приводит к расхождению значений общей продолжительности вызова базы данных (e) и суммы значений $c + \sum ela$ для данного вызова. Данные файла трассировки не позволяют отличить это явление от рассмотренной ранее проблемы наличия времени, проведенного вне выполнения. В тех системах, где интенсивность свопинга невелика, значительное несоответствие частей (Δ) данного равенства в рамках всего файла трассировки говорит о недостатке измерительных средств:

$$\Delta = \sum_{dep=0} e - \left[\sum_{dep=0} c + \sum_{\substack{\text{within} \\ \text{calls}}} ela \right]$$

Для того чтобы осознать проблему, представьте себе, что продолжительность пятисекундного чтения файла базы данных, выполняемого вызовом выборки, не измеряется. Фактическая продолжительность выборки (e) составила бы 5 секунд, при этом ни общее время использования процессора (c), ни длительность событий ожидания (значения ela) не были бы настолько велики, чтобы составить общую фактическую продолжительность вызова.

Пропущенное время между вызовами базы данных

Не снабженный измерительными средствами код, присутствующий вне контекста вызова базы данных, нельзя выявить тем же способом, что аналогичный код внутри вызова базы данных. Тут есть два варианта. Во-первых, последовательность событий между вызовами, не перемежающаяся вызовами базы данных, является сигналом наличия неизмеряемого кода в ядре Oracle. Во-вторых, неизмеряемые вызовы можно обнаружить, исследуя значения ста-

тики `tim` в файле трассировки. Если «соседние» значения `tim` отличаются друг от друга значительно сильнее, чем это можно было бы объяснить на основании присутствующих в выводе строк вызовов базы данных и событий ожидания, то можно считать, что проблема обнаружена. Данные файла трассировки в такой ситуации демонстрируют большие значения Δ в формуле, где R обозначает известное время отклика, которое, предположительно, должно быть учтено в файле трассировки:

$$\Delta = R - \sum_{dep=0} e$$

В качестве примера данной проблемы можно привести ошибку Oracle с номером 2425312. Речь идет о случае, когда целые вызовы базы данных, выполненные посредством PL/SQL RPC, не выводят никаких данных трассировки. В результате в файле трассировки может возникнуть огромный промежуток неучтенного времени.

На практике ситуация, в которой неизмеряемый системный вызов занимал бы значительную часть фактической продолжительности программы, может не встретиться никогда. Мы в *hotsos.com* сталкивались с подобным явлением не чаще, чем в пяти случаях из тысячи файлов трассировки. Одним из вариантов неизмеряемой активности базы данных является ошибка 2425312 в Oracle MetaLink. Такую ошибку можно увидеть при трассировке приложений Oracle Forms, содержащих (на стороне клиента) код PL/SQL. Могут встретиться и другие случаи серьезного влияния неизмеренного времени на ваши исследования, но они будут редкими.

Запись трассировки

Применяя трассировку SQL, вы встретите по крайней мере один неизмеряемый системный вызов – вызов `write`, посредством которого ядро Oracle записывает вывод трассировки SQL в файл трассировки. Влияние этого вызова на производительность обычно невелико. С помощью утилиты `strace` нетрудно разобраться, каким образом ядро Oracle записывает каждую строку данных в файл трассировки. Из нескольких сотен файлов расширенной трассировки SQL, собранных нами в *hotsos.com* к моменту написания этой книги, менее чем в 1% случаев наблюдается накопление неучтенного времени, которое можно объяснить медленной записью в файл трассировки. Однако для того чтобы снизить риск существенного ухудшения производительности приложения за счет самого факта применения трассировки, необходимо следовать приведенными ниже рекомендациями:

- Обратитесь к Oracle *MetaLink*, чтобы проверить, не подвержена ли система ошибкам ядра Oracle, способным замедлить запись файла трассировки. Например, ошибка 2202613 влияет на производительность записи файла трассировки для некоторых выпусков

MS Windows 2000. Ошибка 1210242 приводит к необоснованному снижению производительности Oracle при включении трассировки.

- Разместите свои каталоги `USER_DUMP_DEST` и `BACKGROUND_DUMP_DEST` на достаточно производительных устройствах ввода/вывода. Не записывайте данные трассировки в корневую файловую систему или на самое старое и медленное дисковое устройство системы. Несмотря на то, что результатом диагностического процесса может быть значительное улучшение производительности, ни один аналитик не захочет, чтобы его обвинили даже в кратковременном ее снижении.
- При трассировке сохраняйте как можно более низкой нагрузку, конкурирующую с вводом/выводом файла трассировки. Например, избегайте трассировки нескольких сеансов одновременно на одном устройстве ввода/вывода. Исключения составляют прикладные программы, формирующие несколько файлов трассировки, это могут быть параллельные операции или любая программа, распределяющая нагрузку между несколькими серверными процессами Oracle.

Дополнительные накладные расходы на запись файлов трассировки не должны удерживать вас от применения расширенной трассировки SQL в качестве средства диагностики производительности. Эти расходы окупятся в дальнейшем. В большинстве случаев они оказываются незначительными, но даже если ухудшение производительности оказывается действительно невыносимым, то в любом случае *однократные* затраты на трассировку программы стоят того, если диагностика приводит к одному из следующих результатов:

- Вы сможете исправить исследуемую программу, в результате чего будут сэкономлены вычислительные мощности и значительно уменьшено время отклика для конечного пользователя.
- Вы сможете доказать, что исследуемая программа работает настолько хорошо, насколько это возможно, и дальнейшие инвестиции в оптимизацию производительности бесполезны.

Упражнения

1. Установите в системе утилиту трассировки системных вызовов, подобную `strace`. Примените ее для трассировки процесса ядра Oracle. В отдельном окне наблюдайте за выводом трассировки SQL (при помощи `tail -f` или аналогичной команды). Какие вызовы для измерения времени выполняет ядро Oracle в системе? В какой последовательности выполняются вызовы измерения времени? Похоже ли поведение системы на то, что описано в примере 7.2?
2. Выполните в своей системе программу из примера 7.4. Каков в среднем эффект влияния измерителя (одного вызова `gettimeofday`) в вашей системе?
3. Программа на Perl из примера 7.7 сохраняет значения, возвращенные «скорострельной» последовательностью системных вызовов

`times`. Она просматривает список сохраненных значений и печатает значение лишь в том случае, если оно отличается от предыдущего значения в списке. Какую информацию запуск такой программы предоставляет о разрешающей способности учета ресурсов процессора в вашей системе?

Пример 7.7. Программа на Perl, выполняющая быструю последовательность системных вызовов `times`

```
#!/usr/bin/perl
use strict;
use warnings;
use IO::File;
autoflush STDOUT 1;
my @times = (times)[0];
while ((my $t = (times)[0]) - $times[0] < 1) {
    push @times, $t;
}
print scalar @times, " distinct times\n";
my $prior = '';
for my $time (@times) {
    print "$time\n" if $time ne $prior;
    $prior = $time;
}
```

4. У нас в *hotsos.com* скопились миллионы строк данных трассировки Oracle8i такого вида:

```
FETCH #1:c=1,e=0,p=0,cr=0,cu=0,mis=0,r=10,dep=0,og=3,tim=17132884
```

Поясните, откуда они возникают.

5. В Oracle9i приведенная ниже строка появляется так же часто, как строка из предыдущего упражнения:

```
PARSE #7:c=10000,e=2167,p=0,cr=0,cu=0,mis=1,r=0,dep=1,og=0,
tim=1016096093915846
```

Объясните почему. Чем данное явление отличается от описанного в предыдущем упражнении?

6. Напишите программу для проверки мысленного эксперимента из примера 7.6. Поясните имеющиеся существенные различия в результатах программы и мысленного эксперимента, представленных в табл. 7.2.
7. Выполните в своей системе трассировку клиентских программ, использующих разные интерфейсы Oracle, а именно:
- Вызовы PL/SQL RPC из кода PL/SQL клиентских приложений Oracle Forms
 - Вызовы Java RMI, которыми обмениваются виртуальные машины клиента и сервера

Создают ли они аномально большие объемы неучтенного времени?

8

Данные фиксированных представлений Oracle

Вероятно, до того как к вам в руки попала эта книга, вас гораздо больше интересовало содержимое представлений `V$`, чем исходные данные трассировки. Всех нас явно или неявно учили, что компетентный специалист по производительности Oracle обязан как можно больше знать о *фиксированных представлениях* Oracle. Эти фиксированные представления – суть псевдотаблицы с именами, начинающимися на `V$` или `GV$`, а еще лучше на `X$`. Складывается впечатление, что где-то есть подсобное хозяйство, единственной целью которого является выпуск все новых и новых плакатов с изображением запутанных отношений между более чем 500 представлений, описанных в `V$FIXED_VIEW_DEFINITION`.

Люди, интересующиеся курсами *hotsos.com*, бывают удивлены тем, как мало времени мы уделяем на этих курсах рассмотрению фиксированных представлений Oracle. Разумеется, в фиксированных представлениях содержатся полезные данные, которые могут иногда пригодиться в деле повышения производительности. Но с 1999 года, в сотнях случаев успешного решения проблем производительности нашей командой, мы использовали данные расширенной трассировки SQL в корректно определенной области – и ничего больше.

В 2000 г. в *hotsos.com* выполнялись параллельно два исследовательских проекта. Первый имел целью создание оптимального метода повышения производительности на основе данных расширенной трассировки SQL. Целью второго было создание оптимального метода повышения производительности, основанного на данных фиксированных представлений. Результаты меня удивили. Приступая к этим проектам, я был уверен в преимуществе метода, основанного на фиксированных представлениях Oracle, над любым другим, базирующимся на «простых» данных трассировки. Но при работе с данными фиксиро-

ванных представлений трудности возникали одна за другой. Недостатки, присущие этим данным, требовали множества времени на поиск обходных путей только для того, чтобы сохранить паритет по качеству анализа с методом на основе трассировочных данных.

Однажды, в июне 2000 года, разрабатывая анализатор на базе фиксированных представлений, я в n -ый раз просматривал файл расширенной трассировки Oracle, пытаюсь подтвердить или опровергнуть правильность очередного обходного маневра размером в сотню строк. До этого момента мы применяли средства анализа трассировочных файлов только для оценки программ анализа фиксированных представлений. Но в тот день мы выдвинули анализатор файла трассировки на роль главного инструмента анализа. Мы закрыли проект анализатора фиксированных представлений и никогда больше к нему не возвращались. Эта глава начинается с описания ряда трудностей, связанных с данными фиксированных представлений. Затем рассматриваются некоторые часто применяемые запросы к фиксированным представлениям и дается оценка их сильных и слабых сторон.

Изъяны данных фиксированных представлений

Ценность фиксированных представлений Oracle неоспорима. Вскоре мы рассмотрим несколько примеров удачного использования запросов к представлениям $V\$$. Например, каждой строке данных, выводимой ядром Oracle в файл трассировки, могут соответствовать несколько тысяч операций, о которых вы никогда не узнаете, если только не обратитесь к данным $V\$$. Однако у фиксированных представлений Oracle есть и недостатки, о которых, возможно, не подозревают многие аналитики по производительности. Ниже описаны те из них, с которыми мы столкнулись, попытавшись воспользоваться фиксированными представлениями в качестве основного источника данных при диагностике проблем производительности.

Избыток источников данных

По данным фиксированных представлений можно построить приблизительный профиль ресурсов заданного сеанса. Как это сделать, рассказывается в данной главе. Однако профиль ресурсов лишь указывает на то, какие данные действительно будут нужны: не изучив его, невозможно определить направление дальнейшего поиска. Следовательно, единственный способ обрести уверенность в том, что собрана вся информация, которая может понадобиться, состоит в том, чтобы учесть все потенциально полезные данные для выбранных диапазонов времени и операций. Сделать это с помощью фиксированных представлений практически невозможно.

Недостаток подробностей

Некоторые типы детальных данных, с большим трудом получаемые из документированных фиксированных представлений Oracle, легко извлекаются из файлов расширенной трассировки SQL. Например, по данным фиксированных представлений Oracle очень сложно:

- Оценить тенденции продолжительности отдельных операций ядра Oracle
- Сопоставить отдельные вызовы ввода/вывода соответствующим устройствам
- Сопоставить потребление ресурсов отдельным вызовам БД
- Выявить рекурсивные отношения между вызовами БД

Подавляющее большинство фиксированных представлений Oracle предоставляет только агрегированную статистику в рамках сеанса (например, `V$SESSTAT`) либо экземпляра (например, `V$SYSSTAT`). Скрывая подробности, агрегированная статистика неоправданно усложняет анализ.

Замечательным исключением являются `X$TRACE` и `V$SESSION_WAIT`, предоставляющие данные по ходу выполнения. Однако использование представления `X$TRACE`, по крайней мере, в Oracle9i Release 2, представляется нецелесообразным, т. к. это представление недокументировано, ненадежно и не поддерживается. Представление `V$SESSION_WAIT`, конечно же, поддерживается, но для того чтобы получить с его помощью данные того же уровня детализации, как и из файла расширенной трассировки Oracle7, пришлось бы опрашивать его с частотой более 100 раз в секунду. С помощью SQL сделать это невозможно (см. раздел «Эффект влияния измерителя при опросе»). А уровень детализации расширенной трассировки Oracle9i потребовал бы опрашивать `V$SESSION_WAIT` 1 000 000 раз в секунду.

Эффект влияния измерителя при опросе

Опрос фиксированных представлений Oracle с помощью SQL создает исключительно сильный эффект влияния измерителя на систему. Получить подробную статистику выполнения в реальном времени просто невозможно. Пример 8.1 иллюстрирует данную проблему. На нашем восьмисотмегагерцевом сервере под Linux типичная скорость не превышает 50 выполнений в секунду для 50-строчного запроса к `V$SESSION`:

```
$ perl polling.pl --username=system --password=manager
sessions      50
polls         1000
elapsed       21.176
user-mode CPU 14.910
kernel-mode CPU 0.110
polls/sec     47.223
```

Приговор: SQL непригоден для опроса даже небольших представлений V\$ со скоростью 100 раз в секунду.

Пример 8.1. Программа на Perl, демонстрирующая фундаментальное ограничение метода опроса с помощью SQL. Обратите внимание, что разбор выполняется однократно и применяется не построчная выборка, а выборка массивом

```
#!/usr/bin/perl

# $Header: /home/cvs/cvm-book1/polling/polling.pl, v1.6 2003/04/23 03:49:37
# Cary Millsap (cary.millsap@hotsos.com)

use strict;
use warnings;
use DBI;
use DBD::Oracle;
use Getopt::Long;
use Time::HiRes qw(gettimeofday);

my @dbh;      # список дескрипторов соединений с БД
my $dbh;      # дескриптор соединения "основного" сеанса
my $sth;      # дескриптор команды Oracle

my $hostname = "";
my $username = "/";
my $password = "";
my %attr = (
    RaiseError => 1,
    AutoCommit => 0,
);
my %opt = (
    sessions    => 50,      # количество сеансов Oracle
    polls       => 1_000,   # количество опросов объекта v$
    hostname    => "",
    username    => "/",
    password    => "",
    debug       => 0,
);

# Получить параметры и аргументы командной строки.
GetOptions(
    "sessions=i"    => \$opt{sessions},
    "polls=i"       => \$opt{polls},
    "debug"         => \$opt{debug},
    "hostname=s"    => \$opt{hostname},
    "username=s"    => \$opt{username},
    "password=s"    => \$opt{password},
);

# Заполнить v$session "фоновыми" соединениями.
for (1 .. $opt{sessions}) {
    push @dbh, DBI->connect("dbi:Oracle:$opt{hostname}", $opt{username},
        $opt{password}, \%attr);
}
```

```

    print "." if $opt{debug};
}
print "$opt{sessions} sessions connected\n" if $opt{debug};

# Выполнить запрос трассировки.
$dbh = DBI->connect("dbi:Oracle:$opt{hostname}", $opt{username},
$opt{password}, \%attr);
$stmt = $dbh->prepare(q(select * from v$session));
my $t0 = gettimeofday;
my ($u0, $s0) = times;
for (1 .. $opt{polls}) {
    $sth->execute();
    $sth->fetchall_arrayref;
}
my ($u1, $s1) = times;
my $t1 = gettimeofday;
$dbh->disconnect;
print "$opt{polls} polls completed\n" if $opt{debug};

# Вывести результаты теста.
my $ela = $t1 - $t0;
my $usr = $u1 - $u0;
my $sys = $s1 - $s0;
printf "%15s %8d\n", "sessions", $opt{sessions};
printf "%15s %8d\n", "polls", $opt{polls};
printf "%15s %8.3f\n", "elapsed", $ela;
printf "%15s %8.3f\n", "user-mode CPU", $usr;
printf "%15s %8.3f\n", "kernel-mode CPU", $sys;
printf "%15s %8.3f\n", "polls/sec", $opt{polls}/$ela;

# Закрыть "фоновые" соединения.
for my $c (@dbh) {
    $c->disconnect;
    print "." if $opt{debug};
}
print "$opt{sessions} sessions disconnected\n" if $opt{debug};

__END__

=head1 NAME

polling - test the polling rate of SQL upon V$SESSION

=head1 SYNOPSIS

polling
  [--sessions=I<s>]
  [--polls=I<p>]
  [--hostname=I<h>]
  [--username=I<u>]
  [--password=I<p>]
  [--debug=I<d>]

=head1 DESCRIPTION

```

B<polling> устанавливает I<s> соединений Oracle, а затем выдает I<p> запросов к B<V\$SESSION>. Выводит статистику производительности для опросов, включая фактическую продолжительность, время использования процессора в пользовательском и привилегированном режимах и количество опросов в секунду. Программа удобна для демонстрации возможностей механизма опроса в Oracle.

=head2 Options

=over 4

=item B<--sessions=>I<s>

Количество соединений Oracle, которые создаются перед началом опроса. Значение по умолчанию 50.

=item B<--polls=>I<p>

Количество запросов, которые будут исполнены. Значение по умолчанию 1000.

=item B<--hostname=>I<u>

Имя хоста Oracle. Значение по умолчанию "" (пустая строка).

=item B<--username=>I<u>

Имя схемы Oracle, к которой подключается B<polling>. Значение по умолчанию "/".

=item B<--password=>I<p>

Пароль Oracle, который B<polling> будет использовать для подключения. Значение по умолчанию "" (пустая строка).

=item B<--debug=>I<d>

Если значение равно 1, то B<polling> создает дамп своих внутренних структур данных в дополнение к обычному выводу. Значение по умолчанию 0.

=back

=head1 EXAMPLES

Использование B<polling> будет аналогично следующему примеру:

```
$ perl polling.pl --username=system --password=manager
      sessions      50
      polls        1000
      elapsed      15.734
      user-mode CPU   7.111
      kernel-mode CPU 0.741
      polls/sec     63.557
```

=head1 AUTHOR

Cary Millsap (cary.millsap@hotsos.com)

=head1 COPYRIGHT

Copyright (c) 2003 by Hotsos Enterprises, Ltd. All rights reserved.

Сложность правильного выбора операций

В данных представлений V\$, как правило, нет атрибута принадлежности к сессии. Чтобы понять, к чему это приводит, представьте, что профиль ресурсов показывает, что время отклика расходуется в основном на ожидание освобождения защепок. Представление V\$LATCH указывает на активное использование двух различных защепок в период выполнения исследуемой пользовательской операции. Какая из защепок отвечает за ее время отклика? Это может быть и одна, и вторая, и даже обе. Как узнать, отвечает ли за данную активность тот сеанс, за которым вы наблюдаете, или какой-то другой сеанс, случайно совпавший с вашим по времени? Ответы на эти вопросы только на основе данных из V\$ за выбранный промежуток времени требуют гораздо большего времени, чем получение ответа на этот же вопрос от данных расширенной трассировки SQL.

Схожие аргументы применимы и ко второму пути. Ядро Oracle сообщает о событии ожидания освобождения защепок только в том случае, когда запрос на получение защепок прошел спин-фазу¹, но был отвергнут, вследствие чего процесс ядра совершает системный вызов, освобождаящий процессор для другого, произвольно выбранного процесса. Если попытка получения защепок оказалась успешной, в файл трассировки ничего не попадает, даже если процессу ядра Oracle потребовалось для этого множество спин-итераций [Millsap (2001c)].

Сочетание данных расширенной трассировки SQL и хорошего инструмента для работы с данными V\$, такого как тестовый инструмент Томаса Кайта (описанного ниже в этой главе), предоставляет куда больше возможностей, чем каждое из этих средств в отдельности.

Сложность выбора временной области

Еще одной причиной, побудившей меня прекратить большой проект *hotsos.com* по диагностике на основе фиксированных представлений, стала неизлечимая проблема с получением данных в заданной временной области. В том случае, когда граница интервала наблюдения попадает в середину события, важно знать, какая часть события должна быть включена в этот интервал, а какая — отброшена. Например, если в момент времени *t* сделан запрос к V\$SESSION_WAIT и обнаружено выполняющееся событие `db file scattered read`, то как определить, сколь-

¹ Когда процесс, пытаясь получить защепок, обнаруживает, что она занята другим процессом, он несколько раз выполняет серию простых процессорных инструкций и повторяет попытку. Этот итеративный процесс называется *спином* (*spin*). Спин можно рассматривать как активное ожидание: с точки зрения операционной системы процесс продолжает выполняться, потребляя процессорное время, но фактически процесс ждет освобождения защепок. После нескольких неудачных спин-итераций процесс освобождает процессор и переходит в состояние ожидания. — *Примеч. науч. ред.*

ко времени прошло с начала его выполнения? Оказывается, это можно выяснить с точностью 0,01 секунды, только если вы в состоянии выполнять опрос со скоростью 100 и более раз в секунду.

Еще одна неприятность возникает, когда сеанс завершает соединение раньше, чем получены все необходимые данные фиксированных представлений в конце предполагаемого интервала наблюдения. Если не успеть опросить различные представления `V$`, содержащие информацию о сеансе, до того как соединение прервется, нужные вам данные будут потеряны навсегда. Опять-таки, опрос с высокой частотой способен помочь в решении данной проблемы, но для этого надо обращаться к разделяемой памяти Oracle средствами, отличными от SQL.

Чувствительность к переполнению и другие ошибки

Еще одна слабая сторона фиксированных представлений – их чувствительность к ошибкам переполнения. Дело в том, что n -разрядная переменная счетчика может принимать только $2^n - 1$ различных значений. Когда n -битное беззнаковое целое в ядре Oracle достигает значения $2^n - 1$, при следующем приращении его значение обнуляется. Ошибки переполнения приводят к тому, что в определенный момент «накопленные» значения статистики оказываются меньше, чем некоторое время назад. Если разработчик ядра выбрал для переменной счетчика целое со знаком, то некоторые значения после определенного порога становятся отрицательными. Восстановить данные после переполнения несложно, но это еще один момент, требующий внимания при анализе данных `V$` и никогда – при анализе данных расширенной трассировки SQL.

Ситуация усугубляется наличием проблем со статистикой CPU used by this session, включая ошибки Oracle с номерами 2327249, 2707060, 1286684 и другие. Если нельзя доверять системным средствам измерения основных составляющих времени отклика оптимизируемой системы, то и результат всей работы находится под вопросом.

Отсутствие данных о длительности вызовов БД

Посмотрите определения представлений `V$` и, я уверен, вы нигде не найдете эквивалента статистики `e`. Не зная фактической продолжительности вызова БД, невозможно даже обнаружить наличие неучтенного времени, которое должно быть сопоставлено данному вызову. Разумеется, если невозможно судить о наличии такого неучтенного времени, то невозможно его и измерить. Как рассказывается в главах 6, 9 и 12, измерение неучтенного времени пользовательской операции – ключ к обнаружению, например, свопинга, по полученным от Oracle данным, *независимым* от операционной системы.

Думаю, вы согласитесь со мной в том, что отсутствие в представлениях `V$` данных о продолжительности вызовов БД приводит к курьезным последствиям. Некоторые аналитики рассматривают «проблему поте-

рянного времени» в файлах трассировки как доказательство того, что данные представлений $V\$_$ для анализа производительности имеют приоритетное значение. Но не забывайте, что данные фиксированных представлений и расширенной трассировки SQL получены с помощью одних и тех же системных вызовов (как рассказывалось в главе 7). Следовательно, для данных из $V\$_$ характерны те же самые проблемы «потерь времени», которым, якобы, подвержены файлы расширенной трассировки. Утверждение о том, что «потерянное время» свидетельствует о большей достоверности данных из $V\$_$, по сравнению с данными расширенной трассировки, столь же разумно, как и совет зажмуриться для большей безопасности, оказавшись наедине с голодным медведем.

Отсутствие согласованности чтения

Последней проблемой, похоронившей наши амбиции по созданию «самого главного анализатора $V\$_$ » (как будто недостаточно перечисленных), оказалась проблема согласованности чтения. Причина ее в том, что Oracle получает данные о производительности не из стандартных таблиц, а путем обращений к разделяемой памяти. Вследствие этого фиксированные представления не используют принятую в Oracle стандартную модель согласованного чтения, в которой для создания согласованного образа блока на определенный момент времени применяют блоки отката.



Корпорация Oracle не может допустить, чтобы доступ к фиксированным представлениям сопровождался накладными расходами на поддержание согласованности чтения. Ведь в таком случае непомерные издержки на обращение к этим представлениям сделали бы их практически бесполезными.

В Oracle есть два пути для получения данных $V\$_$: можно самостоятельно считывать их из разделяемой памяти, а можно с помощью SQL обращаться за ними к предоставленным Oracle фиксированным представлениям. Способ с самостоятельным обращением к разделяемой памяти имеет значительное преимущество, позволяя избежать огромных дополнительных расходов на обработку SQL, что особенно актуально для сервера Oracle, и без того отягощенного проблемами с производительностью. Однако ни один из способов не обеспечивает согласованного чтения данных о производительности. Когда мы обращаемся к представлению $V\$_$, результат не соответствует состоянию системы на определенный момент времени. Полученные данные «размазаны» по всему интервалу времени выполнения запроса.

Чтение большой порции данных из памяти не является атомарной операцией. Для создания образа сегмента памяти, обеспечивающего согласованное чтение, необходимо либо заблокировать этот сегмент на время выполнения запроса, либо применить более сложный механизм со-

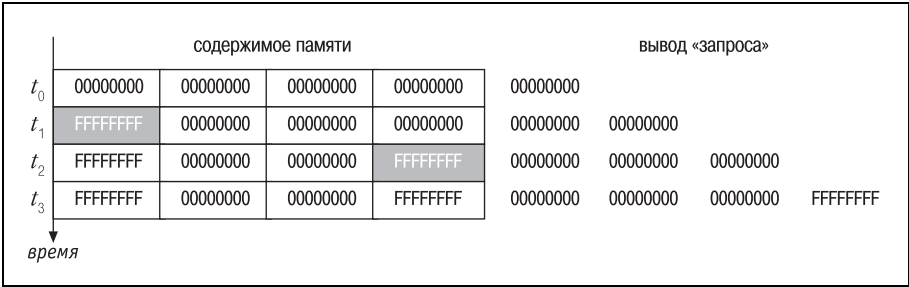


Рис. 8.1. Проблема, вызванная несогласованностью чтения: в выходном потоке может быть представлено состояние памяти, никогда в действительности не существовавшее

гласованного чтения, подобный тому, который ядро Oracle задействует для реальных таблиц. В противном случае полученные запросом данные могут показать состояние системы, никогда не имевшее места в действительности (рис. 8.1). Чтение сегмента памяти началось в момент времени t_0 и завершилось в момент t_3 . Темные ячейки показывает фрагменты памяти, содержимое которых изменилось за указанный период. Светло-серые ячейки показывают фрагменты, копируемые в выходной поток в указанный момент времени. Вследствие того, что операция чтения большой области памяти не обладает свойством атомарности, выходной поток может представлять такое состояние памяти, которое никогда не существовало на самом деле.

Важность проблемы согласованного чтения возрастает при увеличении длительности запроса. Предположим, что при выборке данных о 2000 сеансов Oracle простым запросом к `V$SESSION` выполняется последовательность шагов, показанная в табл. 8.1. Результат запроса представляет собой не моментальную копию, а набор строк, каждая из которых соответствует слегка отличающемуся состоянию системы, имевшему место на протяжении 40 секунд времени выполнения запроса.

Таблица 8.1. Последовательность событий, произошедших за время выполнения запроса к `V$SESSION`

Время	Событие
0:00:00,00	select sid from v\$session; имеются 2000 сеансов
0:00:00,01	Получена первая строка результата
0:00:00,12	Сеанс с номером 1297 завершен
0:00:00,26	Область разделяемой памяти, хранившая информацию о сеансе 1297, больше не содержит этих сведений; следовательно, никаких данных о сеансе 1297 (который был активен в 10:00:00,00) не получено
0:00:00,40	Получена последняя строка результата

Разумеется, результат запроса, не гарантирующего согласованности чтения, не может считаться достоверным. Все еще более усложняется, если включить в запрос дополнительные источники данных. Предположим, требуется получить копию оперативных данных, содержащих информацию из всех перечисленных фиксированных представлений:

```
V$BH  
V$DB_OBJECT_CACHE  
V$FILESTAT  
V$LATCH  
V$LIBRARYCACHE  
V$LOCK  
V$OPEN_CURSOR  
V$PARAMETER  
V$PROCESS  
V$ROLLSTAT  
V$ROWCACHE  
V$SESSION  
V$SESSION_EVENT  
V$SESSION_WAIT  
V$SESSTAT  
V$SQL  
V$SQLTEXT  
V$TIMER  
V$TRANSACTION  
V$WAITSTAT
```

Хотелось бы верить, что все полученные в одном запросе данные действительно соответствуют единому моменту времени. Однако это не так. Для фиксированных представлений с небольшим количеством сравнительно редко изменяющихся данных данная проблема не очень критична. Но для представлений с тысячами строк простые команды `SELECT` могут дать странные результаты. Еще хуже дело обстоит, если по такому длинному списку фиксированных представлений строится моментальная копия. Если бы это были настоящие таблицы Oracle, можно было бы, наверное, применить такой способ объединения нескольких запросов в атомарное событие:

```
set transaction readonly;  
select * from v$bh;  
select * from v$db_object_cache;  
...  
select * from v$waitstat;  
commit;
```

Но для фиксированных представлений `V$` такой метод неприменим, т. к. это не настоящие таблицы. Каким бы способом вы ни извлекали информацию для своей моментальной копии, полученные данные всегда будут размазаны по интервалу выполнения всего набора запросов. Время получения состояния из первой строки `V$BH` будет отличаться от времени обращения к последней строке `V$WAITSTAT` на величину длительности выполнения всех этих команд. В приведенном примере эта длительность наверняка превысит секунду. Ни одна программа не может просканировать гигабайты (и даже сотни мегабайт) в одной атомарной операции.

Очень трудно согласовать по времени данные из нескольких источников, даже если они входят в одну моментальную копию. А если собранные данные дополняются статистикой операционной системы, проблема еще более усложняется.

Справочник по фиксированным представлениям

Несмотря на недостатки, фиксированные представления Oracle во многих случаях представляют ценность для аналитика по производительности. В этом разделе описан ряд представлений, понимание которых необходимо аналитику. Все представленные здесь описания объектов относятся к версии Oracle 9.0.1.0.0.

V\$SQL

Не исключено, что самое важное из фиксированных представлений для аналитика по производительности – это `V$SQL`. Оно показывает несколько важных характеристик команд `SQL`, чьи заголовки в данный момент находятся в разделяемом пуле. Представление содержит следующие столбцы:

```
SQL> desc v$sql
```

Name	Null?	Type
SQL_TEXT		VARCHAR2(1000)
SHARABLE_MEM		NUMBER
PERSISTENT_MEM		NUMBER
RUNTIME_MEM		NUMBER
SORTS		NUMBER
LOADED_VERSIONS		NUMBER
OPEN_VERSIONS		NUMBER
USERS_OPENING		NUMBER
EXECUTIONS		NUMBER
USERS_EXECUTING		NUMBER
LOADS		NUMBER
FIRST_LOAD_TIME		VARCHAR2(19)
INVALIDATIONS		NUMBER
PARSE_CALLS		NUMBER

DISK_READS	NUMBER
BUFFER_GETS	NUMBER
ROWS_PROCESSED	NUMBER
COMMAND_TYPE	NUMBER
OPTIMIZER_MODE	VARCHAR2(10)
OPTIMIZER_COST	NUMBER
PARSING_USER_ID	NUMBER
PARSING_SCHEMA_ID	NUMBER
KEPT_VERSIONS	NUMBER
ADDRESS	RAW(4)
TYPE_CHK_HEAP	RAW(4)
HASH_VALUE	NUMBER
PLAN_HASH_VALUE	NUMBER
CHILD_NUMBER	NUMBER
MODULE	VARCHAR2(64)
MODULE_HASH	NUMBER
ACTION	VARCHAR2(64)
ACTION_HASH	NUMBER
SERIALIZABLE_ABORTS	NUMBER
OUTLINE_CATEGORY	VARCHAR2(64)
CPU_TIME	NUMBER
ELAPSED_TIME	NUMBER
OUTLINE_SID	NUMBER
CHILD_ADDRESS	RAW(4)
SQLTYPE	NUMBER
REMOTE	VARCHAR2(1)
OBJECT_STATUS	VARCHAR2(19)
LITERAL_HASH_VALUE	NUMBER
LAST_LOAD_TIME	VARCHAR2(19)
IS_OBSOLETE	VARCHAR2(1)

Представление V\$SQL позволяет ранжировать команды SQL по объему выполняемой работы или по любому другому критерию эффективности (см. раздел «Обнаружение неэффективного SQL» ниже в этой главе). Запрос к V\$SQLTEXT_WITH_NEWLINES обеспечивает получение полного текста команды SQL, а не только первых 1000 байт, хранящихся в V\$SQL. SQL_TEXT:

```
select sql_text from v$sqltext_with_newlines
where hash_value=:hv and address=:addr
order by piece
```

Можно даже посмотреть, как применение переменных связывания влияет на эффективность команд SQL:

```
select count(*), min(hash_value), substr(sql_text,1,:len) from v$sql
group by substr(sql_text,1,:len)
having count(*)>=:threshold
order by 1 desc, 3 asc
```

В этом запросе :len указывает длину префикса текста SQL, по которому определяется «схожесть» двух различных команд. Например, при

:len=8, строки `select salary,...` и `select s.program,...` считаются похожими, т. к. первые восемь символов в них совпадают. Обычно интересные результаты получаются при таких значениях, как 32, 64 и 128. Значение `:threshold` определяет допустимое количество похожих команд в библиотечном кэше. Как правило, значение `:threshold` равно трем или больше, потому что наличие всего двух схожих команд SQL в библиотечном кэше не вызывает проблем. Если система выходит из-под контроля во время работы с разделяемым SQL, то можно увеличить `:threshold`, чтобы поначалу сосредоточиться на исправлении лишь нескольких неразделяемых команд.

V\$SESS_IO

V\$SESS_IO – это простое фиксированное представление, позволяющее измерить логический и так называемый физический ввод/вывод, порождаемый сеансом:

SQL> desc v\$sess_io

Name	Null?	Type
-----	-----	-----
SID		NUMBER
BLOCK_GETS		NUMBER
CONSISTENT_GETS		NUMBER
PHYSICAL_READS		NUMBER
BLOCK_CHANGES		NUMBER
CONSISTENT_CHANGES		NUMBER

Информация в V\$SESS_IO хорошо коррелирует со статистическими данными расширенной трассировки SQL:

BLOCK_GETS

Соответствует статистике `su` необработанных трассировочных данных.

CONSISTENT_GETS

Соответствует статистике `cg` необработанных трассировочных данных.

PHYSICAL_READS

Соответствует статистике `p` необработанных трассировочных данных.

Количество операций логического ввода/вывода (LIO) равно сумме значений `BLOCK_GETS` и `CONSISTENT_GETS`. Когда сеанс Oracle потребляет очень много процессорного времени, а регистрируемые события ожидания возникают лишь изредка, возникает ощущение, что трассировка сеанса «замирает». Периодическое исполнение приведенного ниже запроса позволяет следить, выполняет ли сеанс вызовы LIO в те моменты, когда не генерируются данные трассировки:

```
select block_gets, consistent_gets from v$sess_io where sid=:sid
```

V\$SYSSTAT

Фиксированное представление V\$SYSSTAT – одно из первых, которые мне довелось применять. Его структура проста:

```
SQL> desc v$sysstat
```

Name	Null?	Type
-----	-----	-----
STATISTIC#		NUMBER
NAME		VARCHAR2(64)
CLASS		NUMBER
VALUE		NUMBER

Каждая строка V\$SYSSTAT содержит одну из статистик уровня экземпляра. Большинство из них – это счетчики выполнения операций с момента последнего старта экземпляра. Строки V\$SYSSTAT подвержены влиянию ошибок переполнения.

Денормализованная структура V\$SYSSTAT позволяет легко, не устанавливая соединений, определить, что происходило в системе с момента последнего запуска экземпляра. Приведенный ниже запрос, выполненный в Oracle9i, показывает примерно 250 значений, описывающих деятельность экземпляра на протяжении его жизни:

```
select name, value from v$sysstat order by 1
```

Следующий запрос выводит ряд значений статистики, имеющих отношение к разбору:

```
select name, value from v$sysstat where name like 'parse%'
```

V\$SESSTAT

Как уже говорилось в главе 3, при сборе диагностических данных область операций, как правило, не должна распространяться на всю систему. Представление V\$SESSTAT содержит те же статистические данные, что и V\$SYSSTAT, но применительно к сеансам:

```
SQL> desc v$sesstat
```

Name	Null?	Type
-----	-----	-----
SID		NUMBER
STATISTIC#		NUMBER
VALUE		NUMBER

Каждая строка V\$SESSTAT содержит счетчик, показывающий, сколько раз выполнялось приращение соответствующей статистики с момента создания сеанса.

Представление V\$SESSTAT в отличие от V\$SYSSTAT не денормализовано, поэтому для получения наименования статистики необходимо соединение с V\$STATNAME. Приведенный запрос показывает все статистики, собранные для сеанса с момента его рождения:

```
select name, value
from v$statname n, v$sesstat s
where sid=:sid and n.statistic#=s.statistic#
and s.value>0
order by 2
```

Следующий запрос выводит приблизительное количество процессорного времени (в сотых долях секунды), израсходованного данным сеансом:

```
select name, value
from v$statname n, v$sesstat s
where sid=:sid and n.statistic#=s.statistic#
and n.name='CPU used by this session'
```

V\$SYSTEM_EVENT

Фиксированное представление V\$SYSTEM_EVENT содержит агрегированную статистику о выполнении кода, оснащенного средствами измерения, с момента последнего запуска экземпляра:

```
SQL> desc v$system_event
```

Name	Null?	Type
-----	-----	-----
EVENT		VARCHAR2(64)
TOTAL_WAITS		NUMBER
TOTAL_TIMEOUTS		NUMBER
TIME_WAITED		NUMBER
AVERAGE_WAIT		NUMBER
TIME_WAITED_MICRO		NUMBER

Каждая строка V\$SYSTEM_EVENT содержит информацию о возникновении определенного события за время жизни экземпляра.



Можно заметить, что в V\$SYSTEM_EVENT нет столбца MAX_WAIT. Этот полезный столбец при желании можно добавить к определению V\$SYSTEM_EVENT, следуя инструкциям, приведенным в [Lewis (2001b) 577–581].

Oracle7 и Oracle8i позволяют получить статистику потребления всех ресурсов, за исключением ЦПУ, с помощью такого запроса:

```
select event, total_waits, time_waited/100 t
from v$system_event
order by 3 desc
```

В Oracle9i ту же статистику, выраженную в микросекундах, дает следующий запрос:

```
select event, total_waits, time_waited_micro/1000000 t
from v$system_event
order by t desc
```

V\$SESSION_EVENT

Еще раз напомним, что при сборе диагностических данных область операций, как правило, не должна включать в себя всю систему. Представление V\$SESSION_EVENT позволяет получать диагностические данные о выполнении различных участков кода ядра для определенного сеанса:

```
SQL> desc v$session_event
```

Name	Null?	Type
-----	-----	-----
SID		NUMBER
EVENT		VARCHAR2(64)
TOTAL_WAITS		NUMBER
TOTAL_TIMEOITS		NUMBER
TIME_WAITED		NUMBER
AVERAGE_WAIT		NUMBER
MAX_WAIT		NUMBER
TIME_WAITED_MICRO		NUMBER

Каждая строка V\$SESSION_EVENT содержит информацию о выполнении определенного участка кода ядра Oracle («событий ожидания») для заданного сеанса с момента его рождения. Таким образом, V\$SESSION_EVENT содержит агрегированные значения данных, получаемых при расширенной трассировке SQL:

EVENT

Имя события ожидания Oracle. Обратите внимание, что все значения EVENT соответствуют значениям `nam` в строках WAIT расширенной трассировки.

TOTAL_WAITS

Количество строк WAIT, содержащих `nam='x'`, где `x` – значение поля EVENT данной строки.

TIME_WAITED

Сумма значений `ela` для всех строк WAIT, содержащих `nam='x'`, где `x` – значение поля EVENT данной строки.

Фиксированное представление V\$SESSION_EVENT не содержит записей о потреблении сеансом процессорного времени. За этими данными придется обращаться к V\$SESSTAT.

Следующий запрос выводит данные о событиях ожидания, произошедших за время существования указанного сеанса Oracle8i:

```
select event, total_waits, time_waited/100 t
from v$session_event
where sid=:sid
order by t desc
```

Данные о событиях ожидания, произошедших за время существования указанного сеанса Oracle9i, можно получить таким запросом:


```
select event, total_waits, time_waited_micro/1000000 t
from v$session_event
where sid=:sid
order by t desc
```

V\$SESSION_WAIT

Если вы спросите кого-нибудь, что такое «интерфейс ожидания», то, скорее всего, вам расскажут о V\$SESSION_WAIT. В отличие от фиксированных представлений V\$SYSTEM_EVENT и V\$SESSION_EVENT, V\$SESSION_WAIT не накапливает данные об имевших место событиях. Зато оно позволяет увидеть, чем занимается указанный сеанс в данный момент:

SQL> desc v\$session_wait

Name	Null?	Type
-----	-----	-----
SID		NUMBER
SEQ#		NUMBER
EVENT		VARCHAR2(64)
P1TEXT		VARCHAR2(64)
P1		NUMBER
P1RAW		RAW(4)
P2TEXT		VARCHAR2(64)
P2		NUMBER
P2RAW		RAW(4)
P3TEXT		VARCHAR2(64)
P3		NUMBER
P3RAW		RAW(4)
WAIT_TIME		NUMBER
SECONDS_IN_WAIT		NUMBER
STATE		VARCHAR2(19)

Строки V\$SESSION_WAIT содержат информацию о текущем состоянии сеанса. Предлагаемая V\$SESSION_WAIT статистика включает:

SEQ#

Каждый раз при завершении события ядро Oracle последовательно увеличивает это значение.

WAIT_TIME

В начале измеряемого события ожидания ядро Oracle обнуляет значение WAIT_TIME. Оно остается нулевым вплоть до завершения данного события, когда ядро присваивает ему одно из значений, приведенных в табл. 8.2. Учтите, что в качестве единицы измерения выступает сотая доля секунды, даже в Oracle9i. Столбца WAIT_TIME_MICRO нет, по крайней мере, в версиях до 9.2.0.2.1 включительно, хотя значения WAIT_TIME получены из представленных в микросекундах значений представления X\$.

SECONDS_IN_WAIT

В начале измеряемого события ожидания ядро Oracle обнуляет значение SECONDS_IN_WAIT. Сам сеанс никогда не изменяет это значение,

Первая неудачная попытка документирования V\$SESSION_WAIT

Для того чтобы понять информацию об измерительных средствах ядра, представленную в этой книге и впервые опубликованную в 1992 г., потребовалось много лет. Ранняя документация по Oracle, посвященная этой новой возможности, не всегда способствовала прояснению ситуации. Например, в документе «Oracle7 Server Tuning guide» (Руководство по настройке сервера Oracle7) приводился такой результат запроса к V\$SESSION_WAIT [Oracle (1996)]:

```
SQL> SELECT sid, event, wait_time
       2     FROM v$session_wait
       3     ORDER BY wait_time, event;
SID EVENT                                WAIT_TIME
-----
...
205 latch free                          4294967295
207 latch free                          4294967295
209 latch free                          4294967295
215 latch free                          4294967295
293 latch free                          4294967295
294 latch free                          4294967295
117 log file sync                       4294967295
129 log file sync                       4294967295
  22 virtual circuit status              4294967295
```

Далее следовал совет: «Необычно большие значения времени ожидания для нескольких последних событий означают, что сеансы в данный момент находятся в состоянии ожидания этого события [sic]. Как видите, в настоящий момент несколько сеансов ожидают освобождения зацепок и синхронизации журнальных файлов». Если бы эти выводы были верны, то показанные в данном примере события находились бы в состоянии ожидания уже 1,36193 года. Тут что-то не так.

Проблема возникла из-за отсутствия столбца STATE в списке выбора запроса. Дело в том, что десятичное целое -1, представленное в виде 32-битного шестнадцатеричного числа, выглядит так: ffffffff. Запишите его как 32-битное беззнаковое целое и получите $2^{32} - 1$ или 4 294 967 295.

На самом деле показанные значения WAIT_TIME представляют собой -1. Это значение соответствует значению WAITED SHORT TIME (табл. 8.2) в столбце STATE. Каждое «необычно большое время ожидания» на самом деле представляет завершившееся событие, причем завершившееся настолько быстро, что измерения показали нулевую продолжительность в сотых долях секунды.

На раннем этапе множество авторов совершало подобные ошибки, пытаясь объяснить, как следует интерпретировать новые данные о «событиях» и «ожиданиях». Их оправдывает то, что они были пионерами, привлечшими множество сторонников новой технологии. Но безусловно, ошибки, подобные описанной здесь, особенно в официальной документации Oracle, затормозили распространение замечательных диагностических возможностей Oracle.

пока не наступит следующее измеряемое событие ожидания, и сеанс не установит снова значение в ноль. Приблизительно раз в три секунды процесс записи в журнал (LGWR) увеличивает значение SECONDS_IN_WAIT на 3. Учтите, что единицей измерения служат секунды, а не санти- и микросекунды.

События, вызывающие тайм-аут, несколько усложняют дело. Например, для события ожидания enqueue тайм-аут возникает через каждые две секунды даже для тех блокировок, которые длятся значительно больше. При каждом тайм-ауте ядро Oracle увеличивает SEQ#, но не переустанавливает значение SECONDS_IN_WAIT.

STATE

В начале измеряемого события ожидания STATE принимает значение WAITING. Это значение сохраняется вплоть до завершения события, после чего ядро устанавливает одно из значений, перечисленных в табл. 8.2.

Таблица 8.2. Описание значений столбцов STATE и WAIT_TIME представления V\$SESSION_WAIT

STATE	WAIT_TIME	Описание
WAITED UNKNOWN TIME	-2	Параметр сеанса TIMED_STATISTICS имел при завершении события значение FALSE, поэтому фактическая продолжительность неизвестна.
WAITED SHORT TIME	-1	Событие ожидания завершено, но оно началось и закончилось в течение одного такта gettimeofday.
WAITING	0	Событие ожидания обрабатывается и пока не завершено.
WAITED KNOWN TIME	$t \geq 0$	Событие ожидания завершилось, затратив $t = t_1 - t_0$ сантисекунд фактического времени (см. главу 7).

Приведенный запрос отображает данные о событиях ожидания, выполняющихся в данный момент в данной системе:

```
select sid, event, wait_time/100 t, seconds_in_wait w, state
from v$session_wait
order by 1
```

Следующий запрос выводит гистограмму, показывающую, какими операциями заняты в настоящий момент сеансы системы:

```
select event, count(*) from v$session_wait
where state='WAITING'
group by event
order by 2 desc
```



Не указывайте условие WAIT_TIME=0 в запросе к V\$SESSION_WAIT, если на самом деле подразумеваете STATE='WAITING'. У некоторых аналитиков вошло в привычку считать, что утверждения WAIT_TIME=0

и `STATE='WAITING'` эквивалентны, поскольку в Oracle7 и Oracle8i дело обстоит именно так. Однако в Oracle9i эти два предиката не равнозначны.

Ядро Oracle9i вычисляет `WAIT_TIME` как `round(x$ksusecst.ksusstim/10000)`, а значение `STATE` — как результат `DECODE` от неокругленного значения `KSUSSTIM`. Следовательно, поле `WAIT_TIME` может содержать ноль при ненулевом исходном значении. Поэтому в ядре Oracle9i возможны ситуации, когда поле `WAIT_TIME` равно нулю, а поле `STATE` имеет значение, отличное от `WAITING`.

Полезные запросы к фиксированным представлениям

Практически у каждого администратора базы данных есть набор запросов к V\$, помогающий ему в деле анализа производительности. Этот раздел посвящен некоторым из моих (и, я уверен, ваших тоже) излюбленных запросов. Вполне вероятно, что некоторые из отчетов, на данные которых вы сейчас полагаетесь, заставляют вас делать неправильные выводы. Практически каждый запрос к V\$ может быть подвергнут подозрению на возможность ошибочной интерпретации.

Инструменты от Тома Кайта

Одно из моих любимых средств для работы с фиксированными представлениями — это тестовый инструментарий Тома Кайта (Tom Kyte), позволяющий прикладному программисту сравнить производительность двух конкурирующих подходов к разработке приложения. Полное описание имеется на сайте <http://asktom.oracle.com/~tkyte/runstats.html>. По этому адресу находятся инструкции по применению простых инструментов, включая примеры, демонстрирующие исключительно плохую масштабируемость приложений, не использующих связывание переменных (http://asktom.oracle.com/pls/ask/f?p=4950:8:::::F4950_P8_DISPLAYID:2444907911913).

Тестовый инструментарий Тома особенно полезен разработчикам приложений Oracle на ранних стадиях разработки. Как правило, программисты пишут код, который впоследствии будет выполняться в сильно загруженных системах. Однако системы, на которых программисты пишут этот код, обычно загружены гораздо меньше. По крайней мере, характер нагрузки у них иной, чем у систем пользователей. Тестовый инструментарий Тома измеряет степень использования приложением тех ресурсов Oracle, которые хуже всего масштабируются (в первую очередь это, пожалуй, зацепки Oracle). Интерпретация результатов проста: чем меньше сериализуемых ресурсов требует некоторое решение, тем больше шансов, что оно будет масштабироваться, став частью промышленной системы. Лучшее в инструментах Тома то, что они настолько просты, что разработчики действительно будут применять их.

С того момента как программисты начинают думать в терминах потребления ресурсов, они начинают писать более масштабируемый код.

Определения фиксированных представлений

Поиск сведений о фиксированных представлениях V\$ в публикациях об Oracle может оказаться нелегким делом. Иногда необходимая информация просто нигде не опубликована. Иногда она отыскивается, но оказывается неверной. Не следует слепо доверять публикациям по Oracle, относящимся к ядру, т. к. оно быстро развивается. К счастью, в том, что касается фиксированных представлений, ядро в некотором роде самодокументировано. Секрет заключается в правильном использовании V\$FIXED_VIEW_DEFINITION. Самое сложное здесь – запомнить это имя:

```
SQL> desc v$fixed_view_definition
```

Name	Null?	Type
VIEW_NAME		VARCHAR2(30)
VIEW_DEFINITION		VARCHAR2(4000)

Из представления V\$FIXED_VIEW_DEFINITION я, например, узнал точные определения столбцов STATE и WAIT_TIME представления V\$SESSION_WAIT. Это можно сделать при помощи нескольких простых шагов. Начнем с выполнения следующего запроса, возвращающего определение представления V\$SESSION_WAIT:

```
SQL> select * from v$fixed_view_definition
2 where view_name='V$SESSION_WAIT';
```

```
VIEW_NAME
```

```
VIEW_DEFINITION
```

```
V$SESSION_WAIT
```

```
select sid,seq#,event,p1text,p1,p1raw,p2text,p2,p2raw,p3text, p3,p3raw,wait
_time,seconds_in_wait,state from gv$session_wait where inst_id =
USERENV('Instance')
```

Кстати, заметьте, что значение VIEW_NAME для этого представления хранится в верхнем регистре. Итак, теперь вы знаете, что V\$SESSION_WAIT – это просто отображение GV\$SESSION_WAIT. Хотя пока это вам мало о чем говорит. На следующем шаге выясним определение GV\$SESSION_WAIT:

```
SQL> desc gv$session_wait
```

Name	Null?	Type
INST_ID		NUMBER
SID		NUMBER
SEQ#		NUMBER
EVENT		VARCHAR2(64)
P1TEXT		VARCHAR2(64)
P1		NUMBER
P1RAW		RAW(4)

P2TEXT	VARCHAR2(64)
P2	NUMBER
P2RAW	RAW(4)
P3TEXT	VARCHAR2(64)
P3	NUMBER
P3RAW	RAW(4)
WAIT_TIME	NUMBER
SECONDS_IN_WAIT	NUMBER
STATE	VARCHAR2(19)

```
SQL> select * from v$fixed_view_definition
2  where view_name='GV$SESSION_WAIT';
```

```
VIEW_NAME
```

```
VIEW_DEFINITION
```

```
GV$SESSION_WAIT
```

```
select s.inst_id,s.indx,s.ksusseq,e.kslednam, e.ksledp1,s.ksussp1,s.ksussp
1r,e.ksledp2, s.ksussp2,s.ksussp2r,e.ksledp3,s.ksussp3,s.ksussp3r, round(s.
ksusstim / 10000), s.ksusewtm, decode(s.ksusstim, 0, 'WAITING', -2, 'WAITED
UNKNOWN TIME', -1, 'WAITED SHORT TIME', 'WAITED KNOWN TIME') from x$ksus
ecst s, x$ksled e where bitand(s.kspsflg,1)!=0 and bitand(s.ksuseflg,1)!=0
and s.ksusseq!=0 and s.ksussopc=e.indx
```

Вуаля! Здесь можно заметить операцию округления при вычислении `WAIT_TIME`. Из приведенного текста также видно, в каких единицах выражена величина, названная здесь `X$KSUSECST.KSUSSTIM`. Нам известно, что `WAIT_TIME` выражается в сотых долях секунды, также мы знаем, что ядро Oracle, чтобы получить сантисекунды, делит исходное значение на 10^4 . Следовательно, в одной секунде содержится 10^4 единиц, причем $10^4 / 10^2 = 10^2$. Отсюда получаем 10^6 единиц `KSUSSTIM` в одной секунде. Другими словами, ядро Oracle измеряет время ожидания в микросекундах, а внешний API (`V$SESSION_WAIT`) представляет его в сантисекундах.

Обнаружение неэффективного SQL

Написанный Джеффом Холтом сценарий *htopsql.sql*, приведенный в примере 8.2, — это тот инструмент, посредством которого мы в *hotsos.com* можем быстро выяснить, какая команда SQL из находящихся в данный момент в библиотечном кэше вносит наибольший вклад в загрузку системы. Этот запрос напрямую не связан с временем отклика, но для большинства команд существует сильная корреляция между показателем `LIO` и полным временем выполнения SQL-запроса. Новые столбцы `CPU_TIME` и `ELAPSED_TIME`, появившиеся в Oracle9i, представляют в `V$SQL` сведения, ранее доступные только в данных трассировки SQL.

Пример 8.2. Сценарий, оценивающий эффективность команд SQL, находящихся в данный момент в разделяемом пуле

```
rem $Header: /usr/local/hostos/RCS/htopsql.sql,v 1.6 2001/11/19 22:31:35
rem Автор: jeff.holt@hotsos.com
```

```

rem          Copyright (c) 1999 by Hotsos Enterprises, Ltd.
rem          All rights reserved.
rem  Применение: Сценарий выводит неэффективный код SQL, вычисляя отношение
rem               количества логических чтений к числу обработанных строк.
rem               Для просмотра первой страницы следует нажать клавишу Return.
rem               Самые неэффективные конструкции должны быть выведены на
rem               первой странице, после чего пользователь может нажать ^C,
rem               затем [Return], когда первая страница будет полностью
rem               выведена. Хэш-значения SQL на самом деле являются
rem               идентификаторами команд и используются в качестве входных
rem               параметров хэш-функции для определения того, находится ли
rem               команда в разделяемом пуле. Сценарий приводит только
rem               идентификаторы команд. Для вывода текста интересующих вас
rem               команд используйте hsqltxt.sql.
rem  Замечания: Данные будут возвращены для команд SELECT, INSERT, UPDATE
rem               и DELETE. Для PL/SQL-блоков строки не возвращаются, т.к. их
rem               операции чтения учтены в содержащихся в них командах SQL.
rem               Знать, какая программа PL/SQL исполняет неэффективную
rem               команду, полезно, но это знание имеет ценность, лишь когда
rem               вам известно, в чем заключается ошибка в команде.

col stmtid      heading 'Stmt Id'          format 9999999999
col dr          heading 'PIO blks'         format 999,999,999
col bg          heading 'LIOs'             format 999,999,999
col sr          heading 'Sorts'           format 999,999
col exe         heading 'Runs'             format 999,999,999
col rp          heading 'Rows'             format 9,999,999,999
col rpr         heading 'LIOs|per Row'     format 999,999,999
col rpe         heading 'LIOs|per Run'     format 999,999,999

set termout    on
set pause      on
set pagesize   30
set pause      'More: '
set linesize   95

select hash_value stmtid
       ,sum(disk_reads) dr
       ,sum(buffer_gets) bg
       ,sum(rows_processed) rp
       ,sum(buffer_gets)/greatest(sum(rows_processed),1) rpr
       ,sum(executions) exe
       ,sum(buffer_gets)/greatest(sum(executions),1) rpe
  from v$sql
 where command_type in ( 2,3,6,7 )
 group by hash_value
 order by 5 desc
/

set pause off

```

Результат запроса отсортирован по количеству вызовов ЛЮ, приходящихся на одну строку результата. Эта величина служит грубой оцен-

кой эффективности команды. Например, представленные ниже данные должны вызвать вопрос: «Почему для получения пяти строк приложению требуется более 174 миллионов обращений к памяти?»

```
SQL> @htopsql
More:

      Stmt Id      PIO blks      LIOs Rows      LIOs
      per Row      Runs      LIOs
      per Run
-----
2503207570      39,736  871,467,231  5 174,293,446      138  6,314,980
1647785011 10,287,310  337,616,703  3 112,538,901  7,730,556      44
4085942203      45,748  257,887,860  8  32,235,983      138  1,868,753
3955802477      10,201  257,887,221  8  32,235,903      138  1,868,748
1647618855      53,136   5,625,843  0   5,625,843  128,868      44
3368205675      35,666   3,534,374  0   3,534,374      1   3,534,374
3722360728      54,348   722,866    1   722,866      1   722,866
497954690      54,332   722,779    0   722,779      1   722,779
  90462217     361,189   4,050,206  8   506,276     137   29,564
 299369270      1,268    382,211    0   382,211     42,378      9
...
```

Приведенные здесь данные помогли в 1999 году найти ту единственную команду SQL, которая потребляла почти 50% общего процессорного времени в системе оперативной обработки. Однако определение эффективности команд по количеству LIO на строку может, как и в случае с любым отношением, привести к неверным выводам. Рассмотрим, к примеру, такую команду SQL:

```
select cust, sum(bal)
from colossal_order_history_table
where cust_id=:id
group by cust
```

Этот запрос может вполне оправданно опросить множество блоков Oracle (даже с использованием индекса по первичному ключу CUST_ID¹), но вернет, как правило, только одну строку. Согласно отчету *htopsql.sql*, такой запрос неэффективен, хотя на самом деле такой негативный вывод может оказаться ошибочным.

Многие аналитики используют запросы, аналогичные *htopsql.sql*, в качестве первого шага в каждом проекте по повышению производительности. Однако попытка основать метод повышения производительности на каком-либо запросе к V\$SQL чревата ошибками определения временной области и области операций. Как и в большинстве данных, получаемых из фиксированных представлений V\$, в случае применения V\$SQL трудно контролировать принадлежность данных определенному

¹ Очевидно, что речь идет не о первичном ключе, а об обычном индексе. В противном случае сумма с группировкой по CUST_ID не понадобилась бы. – *Примеч. науч. ред.*

множеству программ на определенном интервале времени. Рассмотрим следующие ситуации:

- Команда SQL, требующая первоочередного внимания, не выполнялась со времени расчета итогов последнего месяца. Команда отсутствует в библиотечном кэше и, следовательно, не попадет в сегодняшний отчет по V\$SQL.
- Команда SQL, признанная «самой неэффективной», входит в программу загрузки данных, которая в вашей компании никогда не будет использована повторно.
- Команда SQL, признанная «самой неэффективной», выполняется между полуночью и тремя часами утра. Поскольку допустимое время выполнения продолжается до 6:00, а все ночные пакетные задания заканчиваются задолго до этого срока, медленное выполнение неэффективной команды SQL никого не беспокоит.
- Команда SQL, производительность которой в наибольшей степени препятствует увеличению чистой прибыли, движению денежных потоков и возврату инвестиций, не попала в верхние строчки ни одного из отчетов по V\$SQL. Она находится где-то в середине, т. к. ни один из ее статистических показателей особо не выделяется, но с точки зрения бизнеса именно она в наибольшей степени снижает экономическую эффективность системы.

Не обладая знаниями о *бизнесе*, невозможно решить, действительно ли критична для этого бизнеса производительность команды, расположившейся в верхних строчках отчета. Я уверен, что из всех фиксированных представлений V\$SQL – самое ценное для аналитика по производительности. Однако эффект от применения этого представления в проекте повышения производительности существенно меньше, чем от Метода R.

Обнаружение причин зависания сеанса

Время от времени все мы попадаем в такую ситуацию: звонит, скажем, Нэнси и сообщает, что «ее сеанс повис». Она уже спросила своих коллег, не остановлена ли система, но у них все работает прекрасно. Возможно, Нэнси позвонила вам потому, что во время «обеденного совещания»¹ на прошлой неделе вы рассказывали пользователям, почему им не следует перезагружать их персональные компьютеры, когда случается нечто подобное. Зная, как определить идентификатор сеанса Нэнси (глава 6), нетрудно узнать, что с ним происходит. Предположим, мы выяснили, что ID сеанса Нэнси равен 42. Тогда причину зависания поможет выяснить следующий запрос:

¹ A *brown-bag lunch* – перерыв, во время которого сотрудники съедают принесенные с собой завтраки, разговаривая, естественно, о работе.

```
SQL> col sid format 999990
SQL> col seq# format 999,990
SQL> col event format a26
SQL> select sid, seq#, event, state, seconds_in_wait seconds
2   from v$session_wait
3  where sid=42
```

SID	SEQ#	EVENT	STATE	SECONDS
42	29,786	db file sequential read	WAITED SHORT TIME	174

Означает ли это, что сеанс Нэнси завис в ожидании ввода/вывода? Нет, в действительности этот запрос говорит об обратном. Последнее событие ожидания, которое выполнил процесс ядра для сеанса Нэнси, представляло собой операцию файлового чтения, которая завершилась примерно 174 секунды назад (плюс-минус 3 секунды). Более того, время выполнения этой операции было меньше разрешающей способности таймера Oracle. (Для Oracle8i это означает, что фактическое время операции чтения составило менее 0,01 секунды.) Так чего же ждет сеанс Нэнси?

Ответ заключается в том, что ее сеанс (на самом деле это выделенный ей серверный процесс Oracle) либо занимается вычислениями, потребляя процессорное время, либо стоит в очереди на выполнение, ожидая следующей возможности заняться вычислениями и потратить процессорное время. Посмотреть, чем занят сеанс, можно с помощью ряда последовательных запросов к V\$SESS_IO:

```
SQL> col block_gets format 999,999,999,990
SQL> col consistent_gets format 999,999,999,990
SQL> select to_char(sysdate, 'hh:mi:ss') "TIME",
2   block_gets, consistent_gets
3  from v$sess_io where sid=42;
```

TIME	BLOCK_GETS	CONSISTENT_GETS
05:20:27	2,224	22,647,561

SQL> /

TIME	BLOCK_GETS	CONSISTENT_GETS
05:20:44	2,296	23,382,994

Включив в запрос текущее время, можно получить представление о скорости, с которой система Oracle способна обрабатывать вызовы LIO. Показанная здесь система обработала 735505 вызовов LIO примерно за 17 секунд, что соответствует скорости 43265 операций LIO в секунду. По этим данным вы начинаете понимать, в чем тут дело. Ее программа потратила на выполнение более 22 000 000 операций LIO почти девять минут к тому моменту, как вы начали смотреть на нее. Теперь надо выяснить, какие команды выполняются в этой программе, чтобы

впоследствии их исправить. Это можно сделать посредством запроса с соединением к представлениям `V$OPEN_CURSOR` и `V$SQL`. Я бы предпочел воспользоваться данными расширенной трассировки SQL, будь у меня такая возможность, но если вы лишены такой роскоши, грамотное применение фиксированных представлений поможет решить проблему.

Обнаружение причин зависания системы

Иногда, отвечая на звонок Нэнси, которая еще не успела изложить свою проблему, вы замечаете, что вам уже звонят по второй линии. Затем в течение двух минут вы выслушиваете жалобы четырех пользователей и получаете еще семь голосовых сообщений. Что делать? Если ваша система допускает выполнение запросов, я бы посоветовал следующий:

```
SQL> break on report
SQL> compute sum of sessions on report
SQL> select event, count(*) sessions from v$session_wait
      2 where state='WAITING'
      3 group by event
      4 order by 2 desc;
```

EVENT	SESSIONS
-----	-----
SQL*Net message from client	211
log file switch (archiving needed)	187
db file sequential read	27
db file scattered read	9
rdbms ipc message	4
smon timer	1
pmon timer	1

sum	440

Приведенный результат показывает наличие 440 сеансов. Во время выполнения этого запроса более 200 процессов ядра Oracle заблокировано на операции чтения сокета `SQL*Net`, в то время как их пользовательские приложения заняты выполнением операций в промежутке между вызовами базы данных. Вероятно, многие из этих 211 процессов находятся в неактивном состоянии, пока пользователи работают с не-Oracle приложениями, разговаривают с коллегами или вышли освежиться. Хуже то, что 187 сеансов заблокированы в ожидании события `log file switch (archiving needed)`. Это сообщение говорит о том, что процесс ARCH не успевает за формированием оперативного журнала.

Оставшиеся несколько пользователей продолжают работать (36 заняты чтением файлов базы данных), но каждый, кто попытается выполнить вызов `COMMIT`, вынужден будет ждать завершения события `log file switch (archiving needed)`. Чем дольше эта проблема будет оставаться нерешенной, тем больше пользователей застынут в ожидании этого со-

бытия. Администратор системы, на которой был получен вышеописанный результат, пренебрег предупреждениями о скором переполнении выделенной процессу ARCH файловой системы.

Построение приблизительного профиля ресурсов для сеанса

Сделанная на скорую руку программа *vprof*, приведенная в примере 8.3, предназначена для сбора временных характеристик Oracle для заданного сеанса на определенном интервале времени. Я написал ее не для промышленного применения (я считаю, что она для этого не подходит), а для иллюстрации некоторых трудностей использования SQL-запросов к фиксированным представлениям с целью получения диагностических данных в заданных областях. Думаю, *vprof* может применяться в образовательных целях для пояснения следующих моментов:

- Посредством соединения представлений `V$SESSTAT` и `V$SESSION_EVENT` можно получить приближенную оценку полного времени отклика пользовательской операции.
- Попытки получения временных характеристик пользовательской операции из фиксированных представлений Oracle наталкиваются на трудность определения временной области.
- Диагностика причины ухудшения производительности выбранной пользовательской операции требует значительно больших усилий, чем просто создание профиля ресурсов этой операции.

Пример 8.3. Программа на Perl, с помощью SQL строящая приближенный профиль ресурсов сеанса на заданном интервале времени

```
#!/usr/bin/perl

# $Header: /home/cvs/cvm-book1/sqltrace/vprof.pl,v 1.8 2003/04/08 14:27:30
# Cary Millsap (cary.millsap@hotsos.com)
# Copyright (c) 2003 by Hotsos Enterprises, Ltd. All rights reserved.

use strict;
use warnings;
use Getopt::Long;
use DBI;
use Time::HiRes qw(gettimeofday);
use Date::Format qw(time2str);

sub nvl($;$) {
    my $value = shift;
    my $default = shift || 0;
    return $value ? $value : $default;
}

# получить параметры командной строки
my %opt = (
    service      => "",
    username     => "/",
```

```

        password      => "",
        debug         => 0,
    );
    GetOptions(
        "service=s"   => \${opt}{service},
        "username=s"  => \${opt}{username},
        "password=s"  => \${opt}{password},
        "debug"       => \${opt}{debug},
    );

    # получить sid из командной строки
    my $usage = "Usage: $0 [options] sid\n\t";
    my $sid = shift or die $usage;

    # установить соединение с Oracle и подготовить SQL для моментального снимка
    my %attr = (RaiseError => 1, AutoCommit => 0);
    my $dbh = DBI->connect(
        "dbi:Oracle:\${opt}{service}", \${opt}{username}, \${opt}{password}, \%attr
    );
    my $sth = $dbh->prepare(<<'END OF SQL', {ora_check_sql => 0});
    select
        'CPU service' ACTIVITY,
        value TIME,
        (
            select
                value
            from
                v$sesstat s,
                v$statname n
            where
                sid = ?
                and n.statistic# = s.statistic#
                and n.name = 'user calls'
        ) CALLS
    from
        v$sesstat s,
        v$statname n
    where
        sid = ?
        and n.statistic# = s.statistic#
        and n.name = 'CPU used by this session'
    union
    select
        e.event ACTIVITY,
        e.time_waited TIME,
        e.total_waits CALLS
    from
        v$session_event e
    where
        sid = ?
    END OF SQL

```

```

# ждать сигнала и получить снимок потребления ресурсов для момента t0
print "Press <Enter> to mark time t0: "; <>;
my ($sec0, $msec0) = gettimeofday;
$sth->execute($sid, $sid, $sid);
my $h0 = $sth->fetchall_hashref("ACTIVITY");

# ждать сигнала и получить снимок потребления ресурсов для момента t1
print "Press <Enter> to mark time t1: "; <>;
my ($sec1, $msec1) = gettimeofday;
$sth->execute($sid, $sid, $sid);
my $h1 = $sth->fetchall_hashref("ACTIVITY");

# построить таблицу профиля
my %prof;
for my $k (keys %$h1) {
    my $calls = $h1->{$k}->{CALLS} - nvl($h0->{$k}->{CALLS}) or next;
    $prof{$k}->{CALLS} = $calls;
    $prof{$k}->{TIME} = ($h1->{$k}->{TIME} - nvl($h0->{$k}->{TIME})) / 100;
}

# вычислить неучтенную продолжительность
my $interval = ($sec1 - $sec0) + ($msec1 - $msec0)/1E6;
my $accounted = 0; $accounted += $prof{$_}->{TIME} for keys %prof;
$prof{"unaccounted-for"} = {
    ACTIVITY => "unaccounted-for",
    TIME     => $interval - $accounted,
    CALLS    => 1,
};

# вывести отладочные данные, если требуется
if ($opt{debug}) {
    use Data::Dumper;
    printf "t0 snapshot:\n%s\n", Dumper($h0);
    printf "t1 snapshot:\n%s\n", Dumper($h1);
    print "\n\n";
}

# вывести профиль ресурсов
print "\nResource Profile for Session $sid\n\n";
printf "%24s = %.06d\n", "t0", time2str("%T", $sec0), $msec0;
printf "%24s = %.06d\n", "t1", time2str("%T", $sec1), $msec1;
printf "%24s = %15.6fs\n", "interval duration", $interval;
printf "%24s = %15.6fs\n", "accounted-for duration", $accounted;
print "\n";
my ($c1, $c2, $c4, $c5) = (32, 10, 10, 11);
my ($c23) = ($c2+1+7+1);
printf "%-${c1}s %-${c23}s %-${c4}s %-${c5}s\n",
    "Response Time Component", "Duration (seconds)", "Calls", "Dur/Call";
printf "%-${c1}s %-${c23}s %-${c4}s %-${c5}s\n",
    "-x$c1, -x$c23, -x$c4, -x$c5;
for my $k (sort { $prof{$b}->{TIME} <=> $prof{$a}->{TIME} } keys %prof) {
    printf "%-${c1}s ", $k;
    printf "%${c2}.2f ", $prof{$k}->{TIME};

```

```

    printf "%7.1f%% ",    $prof{$k}->{TIME}/$interval*100;
    printf "%${c4}d ",    $prof{$k}->{CALLS};
    printf "%${c5}.6f\n",
        ($prof{$k}->{CALLS} ? $prof{$k}->{TIME}/$prof{$k}->{CALLS} : 0);
}
printf "%-${c1}s %${c23}s %${c4}s %${c5}s\n",
    "-x${c1}, "-x${c23}, "-x${c4}, "-x${c5};
printf "%-${c1}s %${c2}.2f %7.1f%%\n",
    "Total", $interval, $interval/$interval*100;

# завершить работу
$dbh->disconnect;

__END__

```

=head1 NAME

vprof - create an approximate resource profile for a session

=head1 SYNOPSIS

```

vprof
  [--service=I<h>]
  [--username=I<u>]
  [--password=I<p>]
  [--debug=I<d>]
  I<session-id>

```

=head1 DESCRIPTION

B<vprof> использует запросы из B<V\$SESSTAT> и B<V\$SESSION_EVENT> для создания приблизительного профиля ресурсов для сеанса Oracle, идентификатор B<V\$SESSION.SID> которого получен из I<session-id>. Границы интервала наблюдения определяются интерактивно – пользователь вводит моменты времени I<t0> и I<t1>, где I<t0> – это время начала интервала наблюдения, а I<t1> – время окончания интервала наблюдения.

=head2 Options

=over 4

=item B<--service=>I<h>

Имя сервера Oracle, к которому будет подключаться B<vprof>. По умолчанию – "" (пустая строка), в этом случае B<vprof> будет использовать для соединения, например, псевдоним Oracle TNS по умолчанию.

=item B<--username=>I<u>

Имя схемы Oracle, к которой будет подключаться B<vprof>. По умолчанию – "".

=item B<--password=>I<p>

Пароль Oracle, который B<vprof> будет использовать для подключения. По умолчанию – "" (пустая строка).

=item B<--debug=>I<d>

Если значение равно 1, то B<vprof > выводит дамп своих внутренних структур данных в дополнение к обычному выводу. Значение по умолчанию - 0.

=back

=head1 EXAMPLES

При использовании B<vprof> вы получите результат, подобный приведённому ниже, где я использовал B<vprof> для получения статистики по его собственному сеансу:

```
$ vprof --username=system --password=manager 8
Press <Enter> to mark time t0:
Press <Enter> to mark time t1:
```

Resource Profile for Session 8

	t0 = 14:59:12.596000			
	t1 = 14:59:14.349000			
interval duration =	1.753000s			
accounted-for duration =	1.670000s			
Response Time Component	Duration (seconds)		Calls	Dur/Call
SQL*Net message from client	1.38	78.7%	1	1.380000
CPU service	0.29	16.5%	1	0.290000
unaccounted-for	0.08	4.7%	1	0.083000
SQL*Net message to client	0.00	0.0%	1	0.000000
Total	1.75	100.0%		

=head1 AUTHOR

Cary Millsap (cary.millsap@hotsos.com)

=head1 BUGS

B<vprof> имеет ряд серьезных ограничений, в частности:

=over 2

=item -

Если событие ожидания выполняется в момент I<t0>, то профиль будет включать в себя избыточное время - отрицательное «неучтенное» время. Такая ситуация часто возникает для событий 'SQL*Net message from client'. Выполнение этого события ожидания происходит во время бездействия пользователя.

=item -

Если событие ожидания выполняется в момент I<t1>, то в профиле будет отсутствовать некоторый промежуток времени - положительное «неучтенное» время. Подобная ситуация может возникнуть, если момент I<t1> находится внутри интервала выполнения длительной программы.

=item -

Указанные ограничения могут сочетаться, в результате чего «неучтенная» продолжительность будет иметь небольшое значение. Может возникнуть иллюзия, что все обстоит отлично, в то время как на самом деле присутствуют две проблемы.

=item -

Если указанный sid не существует на момент времени I<t0>,то программа возвращает профиль, заполненный неучтенным временем.

=item -

Если сеанс с указанным sid завершился в промежутке между I<t0> и I<t1>, то полученный профиль ресурсов будет содержать только неучтенное время... Если только новый сеанс с указанным B<sid> (но, естественно, другим B<serial#>) не будет создан до наступления момента I<t1>. В этом случае результат будет выглядеть нормально, но на самом деле будет абсолютно ошибочным.

=back

=head1 COPYRIGHT

Copyright (c) 2000-2003 by Hotsos Enterprises, Ltd. All rights reserved.

В моей системе для сеанса с V\$SESSION.SID=8 вывод *vprof* выглядит так:

\$ perl vprof.pl --username=system --password=manager 8

Press <Enter> to mark time t0: ↵

Press <Enter> to mark time t1: ↵

Resource Profile for Session 8

	t0 = 09:08:00.823000			
	t1 = 09:08:01.103000			
	interval duration =	0.280000s		
	accounted-for duration =	0.280000s		
Response Time Component	Duration (seconds)		Calls	Dur/Call
CPU service	0.27	96.4%	1	0.270000
SQL*Net message from client	0.01	3.6%	1	0.010000
unaccounted-for	0.00	0.0%	1	0.000000
SQL*Net message to client	0.00	0.0%	1	0.000000
Total	0.28	100.0%		

Главное достоинство программы *vprof* состоит в том, что она выводит загрузку процессора, неучтенное время и реальные события ожидания Oracle в таблицу, формируя настоящий профиль ресурсов. Вывод *vprof* особенно интересен, когда вы экспериментируете со временем каждого из двух интерактивных вводов. Например, если выбрать время t_0 на несколько секунд раньше того, как диагностируемый сеанс начнет что-либо делать, то *vprof* выдаст большое количество отрицательного неучтенного времени, как показано ниже:

```
$ perl vprof.pl --username=system --password=manager 58
Press <Enter> to mark time t0: ↵
Press <Enter> to mark time t1: ↵

Resource Profile for Session 58
```

	t0 = 23:48:18.072254			
	t1 = 23:49:09.992339			
	interval duration = 51.920085s			
	accounted-for duration = 86.990000s			
Response Time Component	Duration (seconds)		Calls	Dur/Call
SQL*Net message from client	54.04	104.1%	2	27.020000
CPU service	31.98	61.6%	3	10.660000
db file sequential read	0.93	1.8%	29181	0.000032
async disk IO	0.03	0.1%	6954	0.000004
direct path read	0.01	0.0%	1228	0.000008
SQL*Net message to client	0.00	0.0%	2	0.000000
db file scattered read	0.00	0.0%	4	0.000000
direct path write	0.00	0.0%	2	0.000000
unaccounted-for	-35.07	-67.5%	1	-35.069915
Total	51.92	100.0%		

В момент t_0 выполнялось длительное событие SQL*Net message from client, при этом данные о его продолжительности еще не попали в V\$SESSION_EVENT. При наступлении момента t_1 длительное событие SQL*Net message from client целиком попадает в V\$SESSION_EVENT, но часть этой длительности относится к периоду до начала интервала наблюдения. Программа *vprof* вычисляет длительность интервала именно как $t_1 - t_0$, но общее время события Oracle, учтенное в период между моментами t_1 и t_0 , превышает значение $t_1 - t_0$, поэтому *vprof* вводит отрицательное псевдособытие unaccounted-for для того, чтобы скорректировать профиль.

Это хороший пример *ошибки сбора данных*, которая может подпортить диагностические данные (ошибки сбора данных подробно рассматривались в главе 6). Для того чтобы сделать вывод *vprof* более ценным, можно проверить V\$SESSION_WAIT на предмет выполнения незавершенного события в момент t_0 , а затем внести исправления в соответствии с полученным результатом. Нечто подобное мы делали в 2000 г. в одном большом проекте, где анализировали данные V\$ – после того, как нашли решение ряда проблем и, столкнувшись с описанными выше ограничениями, решили сократить убытки от проекта. Например, что делать, если в верхних строках профиля ресурсов находятся события ожидания enqueue? Как определить, какой блокировки *ожидала* (в прошедшем времени) исследуемая программа в процессе исполнения? Дальнейшая диагностика подобной проблемы при отсутствии корректно собранных (во временной области и области операций) данных – это недетерминированный процесс, который легко может привести к одной из катастроф, описанных в главе 1.

Исследование всех событий ожидания системы

Начиная с середины 1990-х годов из всех отчетов о производительности системы чаще всего, вероятно, прибегали к отчету о событиях в масштабе всей системы. Самая простая из осмысленных версий такого отчета выглядит примерно так:

```
SQL> col event format a46
SQL> col seconds format 999,999,990.00
SQL> col calls format 999,999,990
SQL> select event, time_waited/100 seconds, total_waits calls
      2  from v$system_event
      3  order by 2 desc;
```

EVENT	SECONDS	CALLS
-----	-----	-----
rdbms ipc message	13,841,814.91	3,671,093
pmon timer	3,652,242.44	1,305,093
smon timer	3,526,140.14	12,182
SQL*Net message from client	20,754.41	12,627
control file parallel write	2,153.49	1,218,538
db file sequential read	91.61	547,488
log file parallel write	55.66	23,726
db file scattered read	26.26	235,882
control file sequential read	8.12	365,643
control file heartbeat	3.99	1
latch activity	2.93	30
buffer busy waits	1.41	72
resmgr:waiting in end wait	0.93	44
latch free	0.80	39
resmgr:waiting in check	0.53	36
log file sync	0.28	19
process startup	0.22	6
rdbms ipc reply	0.14	9
db file parallel read	0.11	4
async disk IO	0.10	19,116
db file parallel write	0.09	24,420
SQL*Net more data to client	0.09	2,014
resmgr:waiting in check2	0.06	2
SQL*Net message to client	0.06	12,635
direct path read	0.05	5,014
log file sequential read	0.03	4
refresh controlfile command	0.00	1
log file single write	0.00	4
SQL*Net break/reset to client	0.00	23
direct path write	0.00	10
30 rows selected.		

Предполагается, что подобный отчет должен помочь аналитику сразу же определить природу проблемы производительности системы. Однако у этого отчета есть множество недостатков, препятствующих этому.

Такие отчеты могут быть полезны в решении *некоторых видов* проблем, но они не состояниии помочь решить многие проблемы, рассмотренные в этой книге:

- Проблемы пользовательских операций, характеристики производительности которых отличаются от средних значений для системы. По агрегированным данным нельзя восстановить составляющие. Забыв об этом, можно неумышленно вызвать ухудшение производительности важных пользовательских операций (см. главу 4).
- Проблемы с пользовательскими операциями, которые легко выявляются по длительностям событий SQL*Net message from client, учитываемых во времени отклика пользовательских операций. Как узнать, связана ли приведенная в отчете V\$SYSTEM_EVENT длительность события SQL*Net message from client с низкой производительностью сетевого ввода/вывода, или же дело в том, что приложение совершает слишком много вызовов базы данных? Данные V\$SYSTEM_EVENT не позволяют ответить на этот вопрос. Большие значения могут означать наличие проблем данного типа. Но такие же значения возникнут и в случае, если пользователи провели много времени, подключившись к системе и не совершая никаких полезных действий.

Ситуация с отчетами на основе данных V\$SYSTEM_EVENT возвращает нас к уже поднимавшемуся в главе 3 вопросу о том, имеет ли смысл решать разные задачи разными способами. Применение различных методов для решения различных проблем предполагает, что вы каким-то образом можете определить, что за задача стоит перед вами, до того, как начнете ее диагностику. В этом и заключается недостаток метода, способный привести к полному провалу проекта, о чем я говорил в главе 1.

В последующих разделах будет описано несколько причин, по которым отчеты на основе данных V\$SYSTEM_EVENT не помогают в решении некоторых типов проблем производительности.

Проблема «событий простоя»

Глядя на приведенный выше отчет о событиях в системе, неопытный аналитик будет пребывать в уверенности, что событие rdbms ipc message составляет главную проблему системы. Однако такой диагноз, скорее всего, неверен. Как известно аналитикам, знакомым с «интерфейсом ожидания Oracle», rdbms ipc message – это одно из так называемых *событий простоя*. С помощью этого события процессы Oracle DBWn, LGWR, CKPT и RECO указывают, что в данный период времени они ничем не заняты. По той же причине pmon timer, smon timer и SQL*Net message from client тоже рассматриваются как события простоя.

Стандартная рекомендация заключается в том, чтобы игнорировать события простоя. Однако такой совет таит в себе серьезную опасность: относя некоторые события к «простою», вы ограничиваете свои возможности по диагностированию целых классов проблем, как это показано

в ряде примеров главы 12. В исследованных нами начиная с 2000 г. пользовательских операциях событие SQL*Net message from client в значительной части случаев оказывало наибольшее влияние на время отклика для конечного пользователя.

Почему же тогда это событие рассматривается как «событие простоя»? Дело в том, что в профиле, соответствующем *всему экземпляру* в области операций и времени с *момента старта* во временной области, большинство сеансов фактически простаивают в ожидании ввода от пользователя. Все то время, пока вы, установив соединение с Oracle, вместо выполнения запросов к базе данных пьете кофе, относится на счет события SQL*Net message from client. Поэтому в отчете о событиях ожидания для всей системы действительно *необходимо* игнорировать все эти события простоя. Более совершенные программы формирования отчетов о событиях ожидания системы анализируют таблицу событий простоя, позволяющую полностью исключить такие события из отчета.

Проблема нормирования

Поизучав некоторое время простой отчет, построенный по представлению V\$SYSTEM_EVENT, можно заинтересоваться тем, как же эти данные относятся со временем работы экземпляра. Едва ли не самая оригинальная из виденных мной программ, предназначенных для ответа на этот вопрос, приведена в примере 8.4. Программа, написанная на SQL*Plus, осуществляет попытку формирования реального профиля ресурсов, показывающего общую продолжительность каждого события в процентах от полного времени работы экземпляра.

Пример 8.4. Программа на SQL, показывающая события ожидания в системе

```
/* $Header: /home/cvs/cvm-book1/sql/sysprof.sql,v 1.2 2003/04/24 05:19:20
Cary Millsap (cary.millsap@hotsos.com)
Copyright (c) 2002 by Hotsos Enterprises, Ltd. All rights reserved.
```

Программа формирует приблизительный профиль ресурсов системы. Однако имейте в виду, что само по себе понятие доли времени ожидания в общем времени работы экземпляра лишено смысла, т. к. не учитывает изменяющегося количества сеансов, существующих в экземпляре на протяжении его жизни.

```
*/
```

```
set echo off feedback on termout on linesize 75 pagesize 66
clear col break compute
undef instance_uptime cpu_consumption event_duration delta
```

```
/* вычислить полное время работы экземпляра */
col td format 999,999,999,990 new_value instance_uptime
select (sysdate-startup_time)*(60*60*24) td from v$instance;
```

```
/* вычислить общее время процессора, использованное ядром */
col cd format 999,999,999,990 new_value cpu_consumption
select value/100 cd from v$sysstat
where name = 'CPU used by this session';
```

```
/* вычислить общую продолжительность событий */
col ed format 999,999,999,990 new_value event_duration
select sum(time_waited)/100 ed from v$system_event;

/* вычислить неучтенное время */
col dd format 999,999,999,990 new_value delta
select &instance_uptime - (&cpu_consumption + &event_duration) dd
from dual;

/* вычислить значения для профиля ресурсов */
col e format a30 head 'Event'
col t format 99,999,990.00 head 'Duration'
col p format 990.9 head '%'
col w format 999,999,999,999,990 head 'Calls'
break on report
compute sum label TOTAL of w t p on report
select
  'CPU service' e,
  &cpu_consumption t,
  (&cpu_consumption)/(&instance_uptime)*100 p,
  (select value from v$sysstat where name = 'user calls') w
from dual
union
select
  'unaccounted for' e,
  &delta t,
  (&delta)/(&instance_uptime)*100 p,
  NULL w
from dual
union
select
  e.event e,
  e.time_waited/100 t,
  (e.time_waited/100)/(&instance_uptime)*100 p,
  e.total_waits w
from v$system_event e
order by t desc
/
```

Как вам нравится идея подсчета процентной доли продолжительности событий ожидания в общем времени работы экземпляра? Вот пример такого отчета:

Event	Duration	%	Calls
-----	-----	-----	-----
rdbms ipc message	13,848,861.00	369.6	3,672,850
pmon timer	3,653,991.35	97.5	1,305,718
smon timer	3,527,940.29	94.2	12,188
CPU service	89,365.37	2.4	12,807
SQL*Net message from client	23,209.05	0.6	12,655
control file parallel write	2,154.32	0.1	1,219,121
db file sequential read	91.66	0.0	547,493

log file parallel write	55.68	0.0	23,739
db file scattered read	26.66	0.0	236,079
control file sequential read	8.12	0.0	365,817
control file heartbeat	3.99	0.0	1
latch activity	2.93	0.0	30
buffer busy waits	1.41	0.0	72
resmgr:waiting in end wait	0.93	0.0	44
latch free	0.80	0.0	39
resmgr:waiting in check	0.53	0.0	36
log file sync	0.28	0.0	19
process startup	0.22	0.0	6
rdbms ipc reply	0.14	0.0	9
db file parallel read	0.11	0.0	4
async disk IO	0.10	0.0	19,116
SQL*Net more data to client	0.09	0.0	2,018
db file parallel write	0.09	0.0	24,436
SQL*Net message to client	0.06	0.0	12,663
resmgr:waiting in check2	0.06	0.0	2
direct path read	0.05	0.0	5,014
log file sequential read	0.03	0.0	4
SQL*Net break/reset to client	0.00	0.0	25
direct path write	0.00	0.0	10
log file single write	0.00	0.0	4
refresh controlfile command	0.00	0.0	1
unaccounted for	-17,398,633.00	-464.3	

TOTAL	3,747,082.32	100.0	7,472,020

Обратите внимание на долю, приходящуюся на событие `rdbms ipc message event`. Странно, не правда ли? Как может полная продолжительность всего лишь одного события составлять 369,6% от времени работы экземпляра? На самом деле это просто. Причина в том, что в системе, для которой построен этот отчет, существуют четыре процесса, сообщающие о событии `rdbms ipc message`, и каждый из них относит к этому событию почти 100% своего времени (моя тестовая система, в которой получен этот отчет, по большей части простаивает). Далее, что означают -17 398 633,00 секунд неучтенного времени? Это всего лишь артефакт, возникший вследствие попытки «подогнать» все учтенные длительности к общей продолжительности интервала наблюдений, составляющей 3 747 082,32 секунд (наш экземпляр был запущен примерно 43 дня назад).

Возможно, целесообразно было бы построить отчет, показывающий затраты каждого из ресурсов в процентах от его полного имеющегося объема? Идея неплохая, но и она чревата рядом сюрпризов. Вы уже видели, что «объем» события `rdbms ipc message` для системы равен времени работы экземпляра, умноженному на количество процессов, сообщающих о возникновении данного события. Рассмотрим еще несколько событий:

CPU service

Объем доступного процессорного времени в системе равен количеству ЦПУ, умноженному на время работы экземпляра.

*SQL*Net message from client*

Максимально возможный промежуток времени «между вызовами» в системе равен сумме продолжительностей всех сеансов, имевших место с момента запуска экземпляра. Это значение может быть рассчитано по данным аудита на уровне соединений.

db file scattered read

Объем операций чтения в системе равен произведению количества дисков на время работы экземпляра, правильно? Не торопитесь. Ядро Oracle относит к длительности событий ожидания не только время выполнения дисковых операций. Вспомните, в главе 7 мы выяснили, что кроме времени использования ресурса имеется также (наиболее значительное) время ожидания ресурса и время, проведенное в состоянии готовности к выполнению. Таким образом, максимально возможная доля события *db file scattered read* в загрузке системы *также* равна сумме продолжительностей всех сеансов, имевших место с момента запуска экземпляра.

Насколько я могу судить, нет нормирующего коэффициента, который позволил бы привести данные `V$SYSTEM_EVENT` к такому виду, чтобы сформировать из них корректный профиль ресурсов.

Бесконечный ресурс ожидания

В значительной степени проблема обусловлена принципом, который лучше всего проиллюстрировать с помощью небольшого мысленного эксперимента. Представьте себе, что сто пользователей стоят в очереди к персональному компьютеру с очень медленными процессором и диском, чтобы установить соединение с экземпляром Oracle. Когда пользователь достигает начала очереди, он открывает новый сеанс SQL*Plus, устанавливает соединение с Oracle, сворачивает окно приложения и выходит из комнаты. Предположим, что после того, как все 100 пользователей проделали эти действия, можно получить точные сведения о расходовании времени во всех ста сеансах Oracle в рамках минутного интервала.

Вы обнаружите, что ядро Oracle затратило 100 минут на ожидание 100 отдельных блокирующих вызовов чтения сокета SQL*Net. Профиль ресурсов системы за эту минуту покажет, что система затратила 100 минут фактического времени на «выполнение события». Как это может быть? В системе есть только один процессор и один диск. Откуда в ней взялись ресурсы, предоставленные 100 пользователям на 100 минут фактического времени? Ответ прост:

В любой системе ресурс ожидания бесконечен.

Конечно, наш пример иллюстрирует лишь небольшой частный случай, поскольку в нем рассматривается событие, известное как «событие простоя». Даже однопроцессорная система может ожидать миллионы таких событий одновременно, вообще не загружая диск или процессор.

Удивительно то, что данный пример так же хорошо будет работать, если мы выдвинем на главную роль событие, заведомо не относящееся к простоям. Представьте себе, что, проявив чудеса координированности, 100 пользователей смогли одновременно обратиться к разным блокам базы данных на одном очень медленном диске настольного ПК. Для простоты предположим, что наш медленный диск может выполнять запросы на чтение со скоростью один блок в секунду.

Сначала все 100 сеансов будут ожидать события чтения одного блока db file sequential read. Через одну секунду первый сеанс, чей запрос на чтение будет выполнен, перейдет к ожиданию SQL*Net message from client, а остальные 99 продолжат ожидание db file sequential read. Через две секунды уже два сеанса будут ожидать чтения сокета, а 98 сеансов – чтения файла. Наконец, через 100 секунд все 100 сеансов будут заняты ожиданием чтения сокета SQL*Net.

Таким образом, через 100 секунд продолжительность ожиданий чтения файла составит $1 + 2 + 3 + \dots + 100 = 5050$ секунд, а продолжительность ожиданий чтения сокета будет равна $99 + 98 + 97 + \dots + 0 = 4950$ секундам. Профиль ресурсов системы, выполняющей 100 чтений файла на таком 100-секундном интервале, будет выглядеть так:

Event	Duration	%	Calls
db file sequential read	5,050.00	5,050.0	100
SQL*Net message from client	4,950.00	4,950.0	99
unaccounted for	-9,900.00	-9,900.0	
TOTAL	100.00	100.0	200

Теперь выясняется, что наша однопроцессорная система с медленным диском на 100-секундном интервале обеспечила своим пользователям 5050 секунд дисковых операций. Как ей это удалось? Дело в том, что пользовательские сеансы получили лишь 100 секунд работы с диском. Оставшаяся часть «времени ожидания» (которое, как вы увидите в главе 9, фактически является *временем отклика* в терминах теории массового обслуживания) представляет собой *задержку в очереди* – время, затраченное в ожидании освобождения дискового устройства. Опять-таки, как видите, ресурс ожидания любой системы бесконечен.

События простоя в фоновых сеансах

Пользовательские сеансы (сеансы, для которых V\$SESSION.TYPE = 'USER'), как правило, относят время простоя своих пользователей к событию SQL*Net message from client. В тех системах Oracle, где за время жизни

экземпляра происходило множество регистраций пользователей, это время обычно оказывается первым в запросе к `V$SYSTEM_EVENT`, упорядоченном по убыванию значений в поле `TIME_WAITED`.

В то же время фоновые процессы Oracle (сеансы, для которых `V$SESSION.TYPE = 'BACKGROUND'`) сохраняют соединение на протяжении всей жизни экземпляра, практически не выполняя никаких действий, когда для них нет работы. Вследствие этого такие процессы вносят существенный вклад в «события простоя». Следующий запрос показывает почему:

```
SQL> col program format a23
SQL> col event format a18
SQL> col seconds format 99,999,990
SQL> col state format a17
SQL> select s.program, w.event, w.seconds_in_wait seconds, w.state
  2   from v$session s, v$session_wait w
  3   where s.sid = w.sid and s.type = 'BACKGROUND'
  4   order by s.sid;
```

PROGRAM	EVENT	SECONDS	STATE
oracle@research (PMON)	pmon timer	1,529,843	WAITING
oracle@research (DBWO)	rdbms ipc message	249	WAITING
oracle@research (LGWR)	rdbms ipc message	246	WAITING
oracle@research (CKPT)	rdbms ipc message	0	WAITING
oracle@research (SMON)	smon timer	1,790	WAITING
oracle@research (RECO)	rdbms ipc message	208,071	WAITING

6 rows selected.

Из полученного отчета видно, что сеанс PMON находился в ожидании события `pmon timer` приблизительно 17,7 суток (наш экземпляр, предназначенный для исследований, не очень загружен). Сеансы DBWO, LGWR, CKPT и RECO ожидают события `rdbms ipc message`. А у сеанса SMON имеется собственное событие таймера `smon timer`. Все эти события можно с полным правом отнести к «простоям», т. к. сообщаемые о них сеансы в буквальном смысле ничего не делают, находясь в ожидании вызова от некоторого коммуникационного устройства.

Однако игнорирование событий простоя – это не очень хорошее решение фундаментальной проблемы, вызванной неправильным выбором временной области и области операций для сбора данных. До тех пор, пока повышение производительности фонового сеанса не представляет для нас интереса, мы можем не обращать внимания на события `pmon timer`, `rdbms ipc message` и `smon timer`. Если же действительно необходимо повысить производительность фонового сеанса и в собранных в корректной области данных велик вклад одного из этих событий, то вопрос, на который надо дать ответ, звучит так:

Почему этот сеанс простаивает, в то время как мы пытаемся заставить его работать быстрее, чем он это делает сейчас?

Если так называемое событие простоя увеличивает время отклика для конечного пользователя, то об этом *действительно* стоит беспокоиться.

Еще раз о выборе области сбора данных

Почему я так долго утаивал от вас эти чудовищные сложности, вызванные «событиями простоя», и только сейчас рассказал о них? На самом деле я ничего не скрывал. Я рассказывал о событиях простоя в главе 5. Просто я называл их «событиями, произошедшими между вызовами базы данных» и никогда не говорил, что с ними связаны какие-то неприятности. События между вызовами не вызывают никаких проблем, если анализировать диагностические данные, собранные в *корректно выбранной области*. К потере данных приводит неправильный выбор области. Если же область выбрана корректно, то события между вызовами имеют такую же ценность для диагностики, как и все прочие события.

Правильный выбор области на этапе сбора данных проекта повышения производительности обеспечивает релевантность *всех* событий ожидания Oracle. Данные, собранные в корректно выбранной области, не содержат «событий простоя», которыми можно было бы пренебречь.

Выбор области сбора данных – ключ к экономически оправданному повышению производительности.

«Интерфейс ожидания» Oracle

Прошедшие на рубеже столетия конференции показали, что популярный прежде способ «настройки» Oracle претерпел кардинальные изменения. В 2001 г. объем материалов конференций Oracle, посвященных новому «интерфейсу ожидания», сравнялся с объемом материалов по традиционному подходу, основанному на расходовании ресурсов, или даже превысил его. Что же такое «интерфейс ожидания»?

Многие аналитики по производительности дают узкое определение интерфейса ожидания как набора из четырех новых фиксированных представлений, предъявленных широкой публике в Oracle 7.0.12:

```
V$SYSTEM_EVENT  
V$SESSION_EVENT  
V$SESSION_WAIT  
V$EVENT_NAME
```

Разумеется, эти фиксированные представления дают очень важные данные о производительности, но они не *заменяют* другой информации базы данных, как и не образуют законченного интерфейса для измерений производительности. Эти фиксированные представления лишь предоставляют аналитику по производительности дополнительную информацию, помогая ему превратить ненадежную модель времени

отклика $e = c + \Delta$, применявшуюся в 80-х годах, в современную модель, полностью учитывающую время отклика:

$$e = c + \sum_{\text{db call}} ela + \Delta$$

Эти новые фиксированные представления не содержат *никакой* информации ни о расходовании процессорного времени, ни о причинах, заставивших ядро Oracle его расходовать (вызовы LIO, сортировки, хеширования и т. д.). Но с этим все в порядке – такие данные уже имеются в V\$SESSTAT и V\$SYSSTAT. Новые фиксированные представления созданы для использования совместно с уже существующими.

Суженное определение интерфейса ожидания как набора из четырех V\$-таблиц приводит к необоснованным утверждениям об ограничениях, например таким:

Интерфейс ожидания Oracle не позволяет выявлять некоторые проблемы производительности: перегруженность процессора операциями LIO; наличие ожидающих процессора сеансов и сеансов, ожидающих подкачки страниц.

Конечно, вы обнаружите в V\$SESSION_EVENT не больше операций LIO, чем в V\$PROCESS имеется наименований оперативных журналов. Но, как вы могли видеть, фиксированные представления или расширенная трассировка SQL *позволяют* обнаружить загруженность процессора операциями LIO. Хорошее понимание данных расширенной трассировки помогает даже найти сеансы, стоящие в очереди к ЦПУ или ожидающие окончания свопинга.

Употребляя термин «интерфейс ожидания», убедитесь в том, что вы и ваш собеседник понимаете, о чем идет речь. Я обычно подразумеваю под ним *все* хронометрические данные о работе Oracle, описанные в главе 7. Однако если ваш собеседник имеет в виду узкое определение, то вам придется потратить дополнительные усилия, чтобы объяснить ему, что вы на самом деле говорите о совокупности «действий» и «ожиданий», которые доступны либо из представлений V\$SESSTAT и V\$SESSION_EVENT, либо из данных расширенной трассировки SQL.

Упражнения

1. Если в мысленном эксперименте из раздела «Бесконечный ресурс ожидания» предположить, что все 100 пользователей одновременно запросили выделения одной секунды процессорного времени, то как будет выглядеть профиль ресурсов на 100-секундном интервале?
2. Обращение к V\$SQL создает меньшую нагрузку на сервер, чем обращение к V\$SQLAREA. Опираясь на данные расширенной трассировки SQL, объясните причину этого.

3. Поэкспериментируйте с *vprof*. Проведите эксперименты с такими временными областями:
- Отметьте t_0 и подождите несколько секунд, прежде чем выполнять первое обращение к базе данных в исследуемом сеансе.
 - Отметьте t_0 непосредственно перед первым обращением к базе данных в исследуемом сеансе.
 - Отметьте t_0 в середине долго выполняющейся команды SQL в исследуемом сеансе.
 - Отметьте t_1 сразу после завершения последнего обращения к базе данных в исследуемом сеансе.
 - Отметьте t_1 через несколько секунд после завершения последнего обращения к базе данных в исследуемом сеансе.
 - Отметьте t_1 в середине долго выполняющейся команды SQL в исследуемом сеансе.
4. Опишите трудности, препятствующие формированию отчета о расходовании ресурсов в событиях ожидания ядра Oracle. Например, представьте себе систему с такими тремя характеристиками:
- С момента запуска экземпляра события ожидания дискового ввода/вывода заняли 1000000 секунд.
 - Экземпляр был запущен 500000 секунд назад.
 - Что можно сказать об использовании дисков с момента запуска экземпляра, если в системе имеется шесть дисков?

Какие выводы вы можете сделать, исходя из этих наблюдений?

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-078-2, название «Oracle. Оптимизация производительности» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.

9

Теория массового обслуживания для специалиста по Oracle

Специалисты могут бесконечно спорить о том, как наилучшим образом повысить производительность системы, пока не будет найден способ доказать чью-либо правоту. Один из путей проверки высказываемых гипотез о повышении производительности – это метод проб и ошибок. Проблема оптимизации производительности этим методом заключается в том, что в среднем он оказывается весьма затратным. Для опробования каждого сценария необходимо столько времени и денег, что часто компания может позволить себе проверить только некоторые из них. Зачастую время и деньги заканчиваются прежде, чем удастся найти подходящее решение.

На эффективность метода проб и ошибок можно надеяться только в том случае, если при выборе очередной попытки руководствоваться каким-то разумом. Обычно в ход идет сочетание имеющегося опыта, интуиции и везения. Однако именно опыт, интуиция и удача являются движущей силой таких вот нескончаемых споров:

Аналитик: На компьютере предыдущего клиента мы заменили процессор на более мощный, и на следующий день все заработало на 50% быстрее. Надо и сейчас срочно установить более мощный процессор, дабы в корне пресечь все проблемы производительности.

Другой аналитик: Я считаю, что это пустая трата времени и денег. В последних семи известных мне случаях замены процессора деньги ушли в песок, т. к. никакого реального улучшения не наступило. А недавно установка более мощного процессора даже привела к *замедлению* работы некоторых приложений.

Кто из них прав? Вполне вероятно, что в каждом из случаев дело обстояло именно так, как это описывают собеседники. Чей опыт окажется более полезным для решения *вашей* следующей задачи? Вместо

ответа я приведу утрированно комичный гипотетический диалог двух исполненных благих намерений, но некомпетентных аналитиков, которые пытаются определить, достаточно ли велик имеющийся у них стакан для того, чтобы вместить всю воду из кувшина.

Аналитик: Вчера мы вылили воду из кувшина в стакан, и, уверяю вас, все содержимое кувшина отлично туда поместилось. Считаю, что следует вылить воду в стакан.

Другой аналитик: Ничего у вас не получится. Последние семь раз, когда я видел, как люди выливают воду из кувшина в стакан, оказывалось, что вся вода не может туда поместиться. Совсем недавно один мой клиент вылил воду из кувшина в стакан, и результат был ужасен – вода залила все вокруг!

Вывод может быть только один – надо перестать угадывать. Измерьте, сколько воды в кувшине. Измерьте объем стакана. Если количество воды превышает объем стакана, не надо ничего никуда переливать. В противном случае – смело действуйте.

Если можно измерить количество воды в кувшине и объем стакана, то не придется проводить эксперимент для того, чтобы убедиться в его результате. Это самый простой пример математической *модели*. Преимущество модели заключается в возможности предсказания будущего без предварительной проверки на практике. Конечно, эту модель можно усложнить, введя в нее такие факторы, как вероятность проливания воды в зависимости от формы носика кувшина, сноровки переливающего и т. д. Разумнее всего выбрать самую простую из моделей, позволяющих получить результат, соответствующий вашим требованиям точности.



Возможно, аналогия будет более точной, если в этом примере заменить воду лунными камнями. Как и рабочая нагрузка приложения, лунный камень имеет нестандартную форму, которую сложно смоделировать, и их получение для проведения экспериментов является невероятно затратным.

Модели производительности

Компьютерные модели производительности сложнее описанной «водной» модели, но все же они не настолько сложны, чтобы быть недоступными. Сложность моделирования производительности в том, что построение модели требует определенных математических знаний. Но надо сказать, что самую сложную часть работы за нас уже сделали. За последнее столетие ученые создали раздел математики, получивший название *теории массового обслуживания*, который и занимается моделированием производительности систем, подобных нашим. В этой главе рассказано о том, как использовать некоторую определенную модель теории массового обслуживания для получения достоверных ответов на следующие вопросы:

- Насколько быстрее будет работать функция приложения f для n пользователей при добавлении в систему k процессоров? Что будет, если заменить существующие процессоры другими, работающими на p процентов быстрее?
- Насколько медленнее будет работать функция приложения f , если добавить к текущей рабочей нагрузке системы n пользователей?
- Сколько процессоров потребуется системе, если необходимо обеспечить завершение p процентов выполнений f в течение не более r секунд?
- Насколько быстрее будет работать функция приложения f для n пользователей, если удастся избавиться от p процентов кода в f ?
- Что лучше отвечает нашим требованиям – система с m действительно быстрыми процессорами? Или система с $m + n$ более медленными процессорами?

Я не буду приводить в этой книге вывод формул теории массового обслуживания. Если вы захотите понять, *почему* теория массового обслуживания эффективна, то найдете нужную информацию в соответствующих специализированных источниках. Я же в этой главе разъясню, *как* применять теорию массового обслуживания на практике в проектах повышения производительности Oracle. Я рассмотрю прошедшую «полевые испытания» модель теории массового обслуживания, реализованную в Microsoft Excel. Здесь будет описана как сама модель, так и способы ее применения.



Те, кого заинтересовала теория массового обслуживания, могут обратиться к замечательной справочной литературе по этой теме. Я лично предпочитаю следующие издания: [Gross and Harris (1998)], [Gunther (1998)], [Jain (1991)], [Allen (1994)] и [Kleinrock (1975)].

Массовое обслуживание

Во всех компьютерных приложениях есть инициаторы запросов, чего-то требующие, и поставщики информации, обеспечивающие ответ на такие запросы. Весь анализ производительности Oracle посвящен отношениям между поставщиками и потребителями, в особенности в условиях острой конкуренции за совместно используемые ресурсы.

Массовое обслуживание – это то, что происходит, когда потребитель требует выделения ему некоторого ресурса, который в данный момент занят обслуживанием другого запроса. Это просто: вы ждете, когда наступит ваша очередь. Существует множество разнообразных дисциплин распределения ресурсов. Одной из наиболее распространенных является уравнительная дисциплина обслуживания в порядке поступления требований, т. е. обслуживания по принципу «*первым пришел – первым обслужен*» (*first-come, first-served*). Находят также применение различ-

«Queueing» или «Queuing»

Теоретики массового обслуживания должны определиться, как писать по-английски то самое слово, которое обозначает «массовое обслуживание» – *queueing*. Сколько букв «е» должно быть в этом слове – две или одна. Утилита проверки орфографии моего текстового процессора (как и многие словари) оповестила меня о том, что слово «queueing» пишется как «queuing», с одним «е». Однако в сфере теории массового обслуживания (подробности можно найти по адресу <http://www2.uwindsor.ca/~hlynka/qfaq.html>) принято писать это слово как «queueing». К счастью, именно это написание рекомендуют два таких уважаемых словаря, как «Oxford English Dictionary» и «Oxford American Dictionary».

ные варианты дисциплины обслуживания с приоритетами, при этом на порядок обслуживания влияет, например, статус заявки. Вот некоторые примеры: первоочередное обслуживание собственных сотрудников, членов королевской семьи, наиболее пробивных личностей и, наоборот, обслуживание последними самых кротких и благонаправных.

«Дождавшись своей очереди», вы получаете запрошенную услугу, что, естественно, занимает некоторое время. Затем вы освобождаете место, и стоящий следом за вами человек получает услугу в соответствии со своим запросом. Люди выстраиваются в очереди к обеденным столикам, банковским клеркам, билетным кассам, эскалаторам, автострадам и «горячим линиям» поддержки программного обеспечения. Компьютерные программы стоят в очередях к таким ресурсам, как процессор, память, дисковый и сетевой ввод/вывод, блокировки и защелки.

Экономика очередей

Конечно же, стояние в очереди сопровождается отчетливым ощущением напрасно потраченного времени. Один из способов, которым поставщик услуг может ускорить продвижение очереди, заключается в увеличении количества или производительности ресурсов. Более быстрые ресурсы или большее их количество, или и то, и другое вместе сократят время пребывания в очереди. Но, разумеется, поставщики услуг переложат на вас стоимость улучшения ресурсов, подняв цены на свои услуги. В конце концов, это *вы* решаете, где находится разумный компромисс между скоростью и стоимостью обслуживания.

В жизни мы ежедневно занимаемся оптимизацией по критериям времени отклика и экономичности. Например, многие из нас тратят тысячи долларов на покупку автомобилей. Несмотря на то, что покупка и обслуживание велосипеда обойдется гораздо дешевле, чем автомобиля, мы покупаем машины отчасти из-за того, что они намного быстрее и обеспечивают значительно лучшее время отклика в наших поездках. (Хотя мы, американцы, известны своей склонностью ездить на автомобилях даже там, где велосипед был бы не только дешевле, но и значительно быстрее.)

Купив автомобиль, мы обнаруживаем, что опять необходима оптимизация. В обычных поездках машина, развивающая скорость 325 км/час, не дает преимуществ во времени по сравнению с автомобилем, способным передвигаться не быстрее 100 км/час, т. к. правила дорожного движения и соображения безопасности накладывают более строгие ограничения, чем мощность двигателя. По этой же причине даже обладатели сверхбыстрых машин планируют свои поездки так, чтобы не попасть в час пик. В некоторых случаях мы применяем стратегию, минимизирующую время обслуживания, в других – стремимся уменьшить задержку в очереди. Наибольшая выгода для вас и ваших пользователей имеет место, когда есть возможность уменьшить время обслуживания, или задержки в очереди, или и то и другое.

Наглядное представление очередей и массового обслуживания

Как говорилось в главе 1, *диаграмма последовательности* – это удобная форма представления структуры расходования времени пользовательской операцией, путешествующей по различным уровням технологического стека. На рис. 9.1 показана диаграмма последовательности для системы с одним процессором и одним диском. Выполнение пользовательского запроса приводит к расходованию ресурсов процессора и диска. Каждая линия на диаграмме обозначает наличие ресурса на данном отрезке времени. Каждый темный прямоугольник на оси вре-

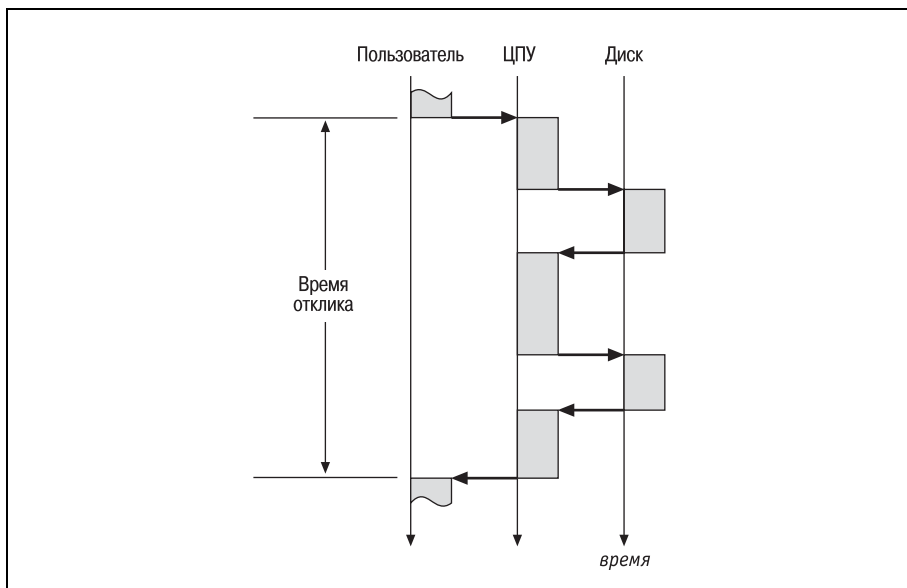


Рис. 9.1. Диаграмма последовательности наглядно показывает, как операция расходует время

мени ресурса показывает факт потребления запросом данного ресурса. Длина этого прямоугольника пропорциональна длительности использования ресурса. Участки временной оси, на которых отсутствуют темные прямоугольники, соответствуют периодам простоя. Диаграмма последовательности читается сверху вниз. Стрелка, направленная слева направо, означает запрос на обслуживание, направленная справа налево – предоставленную услугу. Время отклика равно длительности промежутка между инициированием и завершением запроса.

На рис. 9.1 пользователь приложения выполняет запрос на обслуживание к процессору. Процессор в момент получения запроса свободен, поэтому немедленно начинает его обслуживание. По ходу работы система обнаруживает, что для продолжения обработки требуется обращение к диску. Процессор отправляет диску запрос на обслуживание. Тот в момент получения запроса ничем не занят, поэтому сразу начинает его обслуживание. Выполнив запрос на обслуживание, диск возвращает процессору требуемый результат, и процессор продолжает обработку. После третьего обращения к процессору запрос пользователя выполнен, и процессор возвращает ему результат.

С точки зрения пользователя время отклика равно времени, прошедшему с момента отправки запроса до получения результата. Обратите внимание на то, что на рис. 9.1 показаны также и другие времена отклика. Например, длительность первого (т. е. самого верхнего) темного прямоугольника на оси *Диск* с точки зрения процессора представляет собой время отклика первого вызова дискового ввода/вывода.

Диаграмма последовательности особенно помогает понять, как конкуренция за совместно используемые ресурсы влияет на многозадачную систему. В частности, на рис. 9.2 показано, почему увеличение нагрузки в системе из предыдущего примера может привести к снижению производительности. В ненагруженной системе запросы на использование процессора исполняются без задержек (случай *а*).

Когда мы увеличиваем нагрузку на систему (случай *б*), некоторые запросы к процессору вынуждены ждать, т. к. процессор в момент их получения занят другой работой. На рис. 9.2 показаны две таких задержки в очереди. Второй запрос к процессору после того, как диск вернул управление, вынужден ждать, т. к. процессор занят выполнением задания, обозначенного светло-серым прямоугольником. И третий запрос к процессору ждет по той же причине. Увеличение общего времени отклика нагруженной системы (случай *б*) по сравнению с ненагруженной системой (случай *а*) в точности соответствует общему времени, проведенному запросами в очереди к занятому ресурсу.

Какого увеличения времени отклика можно ожидать, увеличивая нагрузку на систему? Для ответа на этот и многие другие важные вопросы разработан специальный математический аппарат: *теория массового обслуживания*.

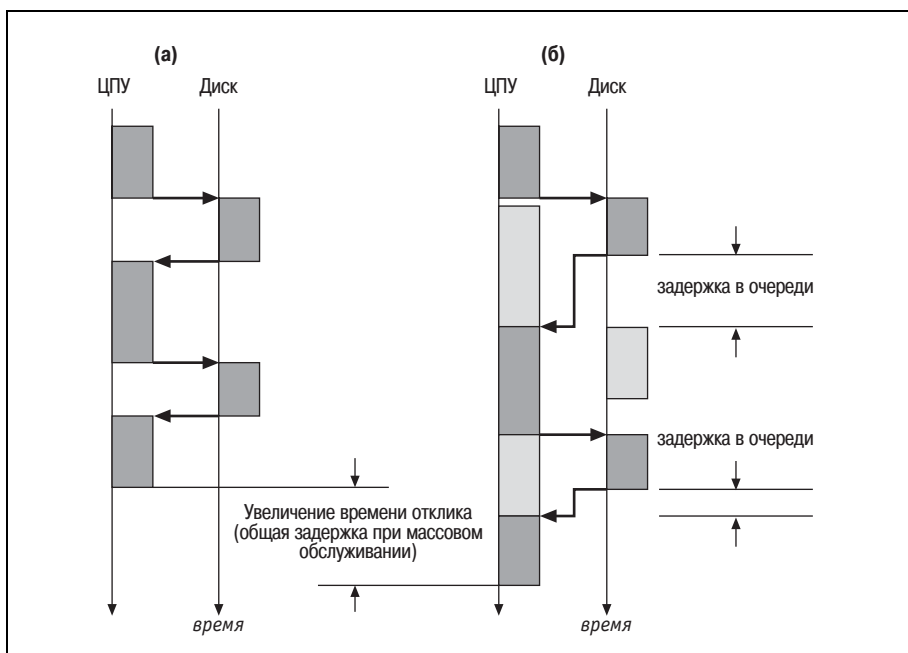


Рис. 9.2. Выполнение единственной функции приложения на ненагруженной системе приводит к простоям ЦПУ (случай а). Дополнительная нагрузка (случай б) приводит к сокращению простоев ЦПУ ценой увеличения времени отклика для исходной функции приложения

Теория массового обслуживания

Теория массового обслуживания – это раздел математики, изучающий поведение систем массового обслуживания. Диаграмма последовательности демонстрирует фундаментальное отношение теории массового обслуживания:

$$R = S + W$$

Время отклика равно сумме времени обслуживания и времени задержки в очереди. *Время обслуживания* – это время, действительно затраченное на использование запрошенного ресурса, а *задержка в очереди* – время, проведенное запросом в очереди к ресурсу.

На рис. 9.3 отношение $R = S + W$ представлено в виде графика. Время отклика, откладываемое по вертикальной оси, изменяется в соответствии с изменением степени загруженности системы на горизонтальной оси. Как вы уже поняли из примера диаграммы последовательности, время обслуживания остается неизменным при любой загруженности системы. Однако задержка в очереди ухудшается (т. е. растет) экспоненциально с ростом нагрузки. Сложение переменного времени задержки с постоянным временем обслуживания для всех возможных

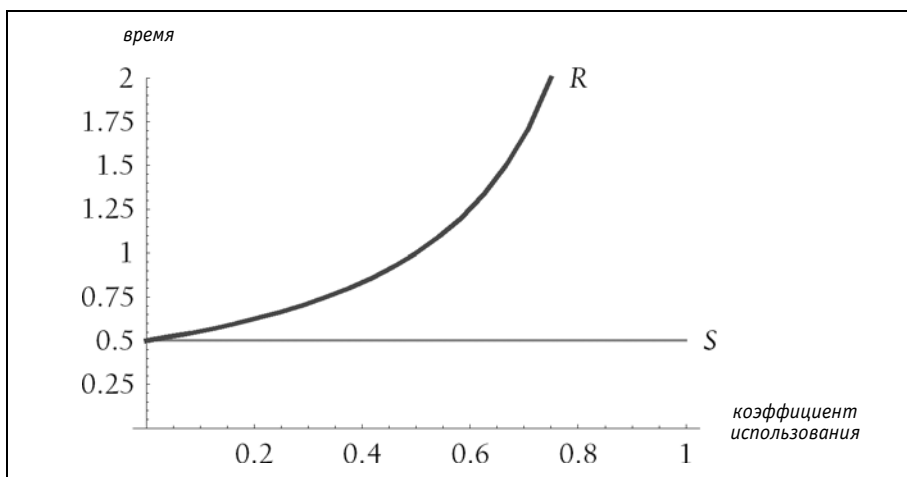


Рис. 9.3. Фундаментальное отношение теории массового обслуживания: $R = S + W$. Время обслуживания (S) постоянно при любых значениях нагрузки, но время отклика с ростом нагрузки увеличивается, т. к. задержка в очереди (расстояние от S до R) экспоненциально зависит от нагрузки

значений нагрузки дает известную кривую времени отклика, напоминающую по форме хоккейную клюшку (рис. 9.4).



Рис. 9.4. Хоккейная клюшка

Входные и выходные параметры модели

Особую ценность теория массового обслуживания представляет для аналитиков по производительности, позволяя предсказывать время отклика системы в гипотетических ситуациях. Правильно примененная модель массового обслуживания способна очень хорошо показать потенциальные проблемы с производительностью, не требуя затрат на натурную проверку различных конфигураций системы. Например, если планируемая замена процессора не в состоянии повысить производительность, гораздо дешевле выяснить это при помощи Excel, чем путем замены и тестирования реального оборудования.

Но не исключено, что еще важнее, каким образом знакомство с теорией массового обслуживания влияет на наше восприятие времени отклика. Становится более явным принципиальное различие между временем, потраченным на работу и на ожидание. Корректное применение теории массового обслуживания позволяет осознать взаимосвязи между разнообразными параметрами оптимизации и их зависимости. Становится понятно, что имеет отношение к нашим задачам, а что – нет.

Мы уже знакомы с фундаментальным отношением теории массового обслуживания: время отклика равно сумме времен обслуживания и задержки в очереди, или $R = S + W$. Мы знаем, что время отклика возрастает с увеличением нагрузки, и такое увеличение связано с изменением W , а не S . В последующих разделах рассказано, как формулы теории массового обслуживания применяются для прогнозирования характеристик производительности определенной конфигурации системы вне зависимости от реальности существования такой конфигурации (полный перечень формул теории массового обслуживания, встречающихся в книге, приведен в приложении D). Начнем с входных параметров, которые используются в формулах.

Поступающие и выполненные запросы

Большая часть необходимых нам сведений из теории массового обслуживания весьма проста для понимания. Систему массового обслуживания можно уподобить черному ящику, который принимает что-то на входе, обрабатывает это и формирует вывод, который предположительно представляет собой какое-то усовершенствование входных данных. Количество запросов, поступающих в систему, обозначается переменной A . Количество выполненных запросов, покидающих систему, обозначается переменной C . Для любой устойчивой системы очередей верно равенство $A = C$. То есть в устойчивой системе все, что попадает в ящик, из него возвращается, при этом оттуда не появляется ничего такого, что не было туда передано. Рассмотрим некоторый промежуток времени T , тогда *частота поступлений* (интенсивность входного потока системы) будет выражаться формулой $\lambda = A/T$ (λ – буква «лямбда» греческого алфавита), а *пропускная способность* (или *скорость выполнения*) системы $X = C/T$. Средний промежуток времени между двумя последовательными поступлениями запросов в систему $\tau = T/A = 1/\lambda$ (τ – буква «тау» греческого алфавита). Отношения между величинами A , T , λ и τ изображены на рис. 9.5.

Каналы обслуживания, коэффициент использования и устойчивость

Внутри системы массового обслуживания (того самого «черного ящика») может существовать один или несколько *каналов обслуживания*. Каждый канал обслуживания работает независимо от всех остальных каналов, обеспечивая обслуживание запросов, поступающих в опреде-

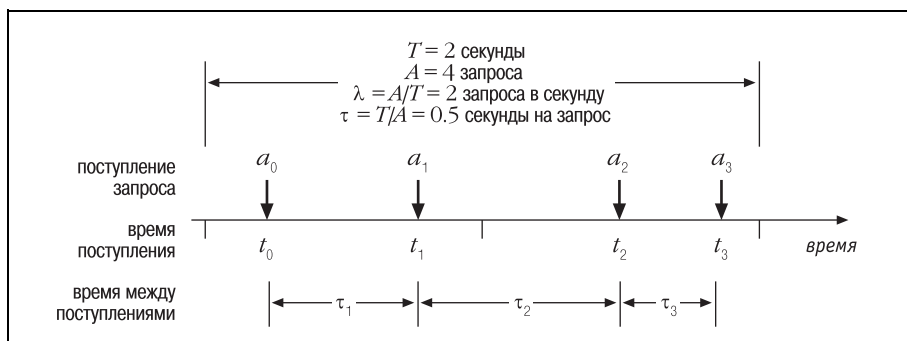


Рис. 9.5. На примере двухсекундного интервала представлены взаимоотношения фундаментальных параметров A , T , λ и τ , описывающих входные параметры системы массового обслуживания

ленную очередь. Например, симметричную многопроцессорную компьютерную систему (symmetric multiprocessing – SMP) с восемью процессорами, обслуживающую единую очередь на использование процессора, можно представить как одну систему с восемью параллельными каналами обслуживания. Количество параллельных каналов обслуживания внутри системы в разных источниках может обозначаться буквами m , s или s . В этой книге принято обозначение m , как в [Kleinrock (1975)].

Общее количество времени, потраченное каналами обслуживания внутри системы на реальное обслуживание запросов, обозначается переменной B (от англ. «busy» – занятый). Коэффициент использования системы на определенном интервале времени равен отношению времени занятости системы к указанному интервалу времени: $U = B/T$. Если внутри системы существует несколько каналов обслуживания (т. е. $m > 1$), то значение U может быть больше 1,0. Это может вызывать некоторое недоумение, до тех пор пока вы не нормализуете это отношение для получения величины среднего коэффициента использования в пересчете на один канал. Средний коэффициент использования одного канала будет равен $\rho = U/m$. Эту величину называют также интенсивностью трафика системы. Имейте в виду, что средний коэффициент использования в пересчете на один канал совсем не обязан быть равен среднему коэффициенту каждого канала. Например, для восьмиканальной системы значение ρ может быть равно 0,5 во множестве случаев, начиная с четырех каналов, работающих со 100% интенсивностью, и четырех – с нулевой, и заканчивая всеми восемью каналами, работающими ровно с 50% интенсивностью.

Система массового обслуживания называется устойчивой в том и только в том случае, если коэффициент ее использования в пересчете на один канал находится в диапазоне $0 \leq \rho < 1$. Работая с моделью массового обслуживания, можно представить себе гипотетические системы, в которых $\rho \geq 1$. Например, увеличивая частоту поступлений, можно

Почему буквы греческие

Одним из факторов, затрудняющих применение теории массового обслуживания в широких кругах специалистов Oracle, является кажущаяся сложность формул. Пугает присутствие греческих букв. При подготовке книги многие коллеги советовали мне преобразовать каждую греческую букву, используемую в главе, в некий аналог из латинского алфавита (если не ошибаюсь, мероприятие, на котором они умоляли меня об этом, называлось «вмешательство»).

Я решил придерживаться обозначений, принятых в литературе по теории массового обслуживания. Считаю, что браться за создание «новой» системы обозначений было бы слишком самонадеянно с моей стороны и, к тому же, это вряд ли помогло бы читателям. Даже если допустить, что мне удалось бы предложить нечто «более удобное» без внесения ошибок в обозначения, оказалось бы, что мои обозначения невозможно сопоставить любой другой информации в этой области, имеющейся в других источниках. К тому же, поверьте мне, даже в «латинском» варианте формулы не были бы красивыми. Если бы я преобразовал греческие буквы в латинские, то это привело бы лишь к усложнению жизни тех читателей, которые заинтересовались бы теорией – им пришлось бы изучать ее дважды на разных языках.

На самом деле греческий алфавит представляет собой не такую уж большую неприятность. В этой главе вам встретятся только буквы: λ (лямбда), μ (мю), ρ (ро), τ (тау), θ (тета) и Σ (сигма, этим же символом принято обозначать понятие суммы). В других главах встретится буква Δ (дельта) и, возможно, некоторые другие. Греческий алфавит, а также русские и английские названия его букв приведены в приложении В.

добиться того, что система сможет их обрабатывать только в том случае, если будет работать в диапазоне $\rho \geq 1$. Однако в реальности такие значения ρ недостижимы.

Время обслуживания и скорость обслуживания

Мы уже говорили о времени обслуживания для системы. Говоря более формально, *ожидаемое время обслуживания* системы (W) – это средний объем времени, в течение которого канал обслуживания бывает занятым для завершения обработки запроса, или $S = B/C$. Обычно вычисление времени обслуживания не составляет труда, т. к. в большинстве систем легко подсчитать количество выполненных запросов и время занятости канала. Иногда удобнее говорить не о *времени* обслуживания, а о *скорости* обслуживания. *Скорость обслуживания* – это то количество запросов, которое один канал обслуживания может обработать за единицу времени, $\mu = C/B$ или $\mu = 1/S$.

Задержка в очереди и время отклика

Как вы уже знаете, *ожидаемая задержка в очереди* (W) – это тот промежуток времени, который, как ожидается, должен пройти между мо-

ментом поступления запроса в систему и моментом начала его обслуживания. Следовательно, для вашего запроса задержка в очереди будет равна сумме ожидаемых значений времени обслуживания для запросов, которые находятся в очереди перед вашим. Возможность прогнозирования времени задержки в очереди – одно из замечательных преимуществ теории массового обслуживания. Величина задержки в очереди зависит не только от среднего времени обслуживания необходимого вам устройства, но и от количества запросов, которые уже будут стоять в очереди, когда в нее попадет ваш запрос.

Вывести формулу прогнозирования времени задержки в очереди без специальных знаний в данной области нелегко, но, к счастью, вся эта нелегкая работа уже была сделана до нас. Для системы массового обслуживания М/М/т (точное определение которой будет дано ниже) она выглядит следующим образом:

$$W = \frac{C(m, \rho)}{m\mu(1-\rho)},$$

где:

$$C(m, \rho) = P(\geq m \text{ jobs}) = \frac{(m\rho)^m}{m!} \frac{1}{(1-\rho) \sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \frac{(m\rho)^m}{m!}}$$

Сложнее всего было получить формулу для $C(m, \rho)$. Эта задача была решена в 1917 г. датским математиком Агнером Эрлангом [Erlang (1917)]. *Формула Эрланга*¹ служит для получения вероятности того, что запрос будет поставлен в очередь на предоставление обслуживания.



Не путайте C из формулы Эрланга с C , обозначающим количество обслуженных запросов системы. Обычно из контекста легко понять, какое из двух C имеется в виду, т. к. C , относящееся к формуле Эрланга, всегда записывается как функция с двумя аргументами.

Программная реализация формул Эрланга требует чуть большего объема математических знаний, чем тот, которым обладает большинство из нас. Однако в 1974 году ученый-исследователь Дэвид Ягерман (David Jagerman) разработал быстрый алгоритм, реализующий формулу Эрланга [Jagerman (1974)] и позволяющий без труда вычислить ожидаемое время задержки в очереди для системы. Я использовал алгоритм Ягермана в примере 9.1.

Пример 9.1. Программа на Visual Basic, вычисляющая формулу Эрланга по алгоритму Ягермана

```
Function ErlangC(m, Rho) As Double
' Erlang's C formula, adapted from [Gunther (1998), 65]
```

¹ В программных пакетах часто обозначается как Erlang C. – *Примеч. перев.*

```

Dim i As Integer
Dim traffic, ErlangB, eb As Double
' Jagerman's algorithm
traffic = Rho * m
ErlangB = traffic / (1 + traffic)
For i = 2 To m
    eb = ErlangB
    ErlangB = eb * traffic / (i + eb * traffic)
Next i
ErlangC = ErlangB / (1 - Rho + Rho * ErlangB)
End Function

```

Если известны ожидаемое время обслуживания системы и ожидаемое время задержки в очереди, то задача вычисления ожидаемого времени отклика становится тривиальной. Ожидаемое время отклика системы – это (как мы уже не раз говорили) сумма ожидаемого времени обслуживания и ожидаемой задержки в очереди, $R = S + W$.

На рис. 9.6 представлена m -канальная система массового обслуживания. Запросы поступают со средней скоростью λ запросов в единицу времени. Продолжительность времени между поступлением двух последовательных запросов равна τ . Каждый из m параллельных каналов выполняет обслуживание запросов со средней скоростью μ запросов в единицу времени, в среднем тратя на обслуживание одного запроса s единиц времени.

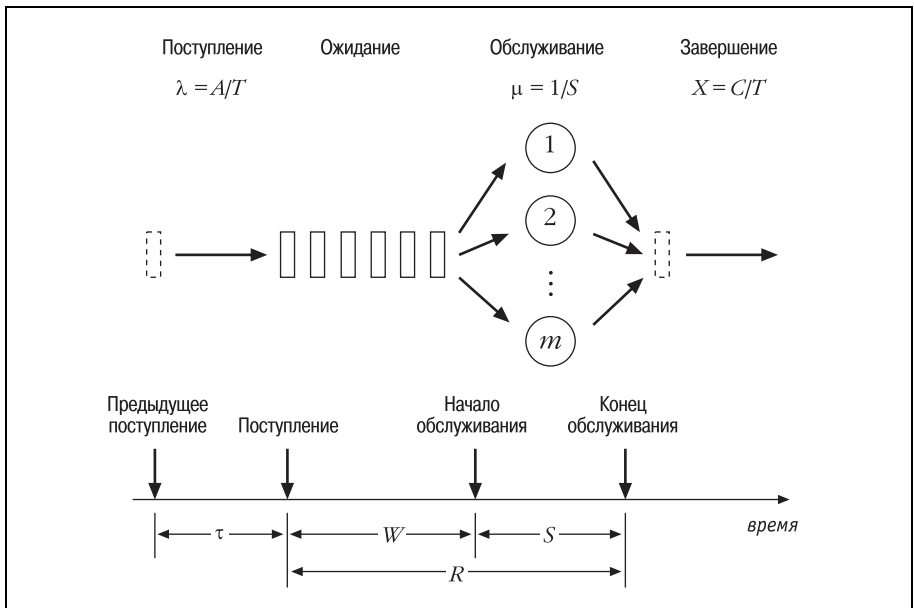


Рис. 9.6. Пример многоканальной системы массового обслуживания иллюстрирует фундаментальные соотношения между различными параметрами теории массового обслуживания (взято из [Jain (1991) 511])

Максимальная рабочая пропускная способность

Максимальная рабочая пропускная способность системы – это наибольшая частота поступлений, которые система может обработать без превышения допустимого значения пользовательского времени отклика r_{\max} . Максимальная рабочая пропускная способность системы, изображенной на рис. 9.7, равна величине λ_{\max} . По мере возрастания интенсивности поступления запросов в систему увеличивается среднее время задержки в очереди, и время отклика системы растет. Пропускная способность системы, при которой время отклика оказывается выше допустимого пользовательского предела, и является максимальной рабочей пропускной способностью, λ_{\max} . Это тот максимум производительности, которого можно требовать от системы, не опасаясь чрезмерно ухудшить среднее время отклика.

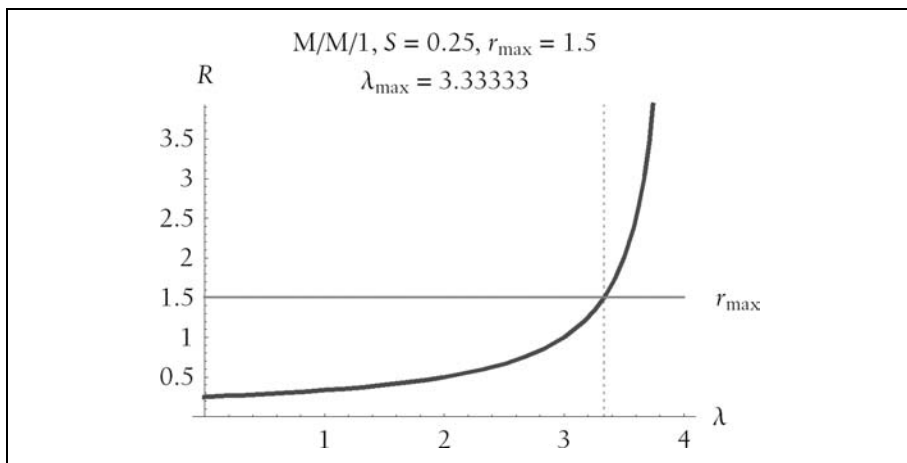


Рис. 9.7. Время отклика является функцией скорости поступления запросов. Определив среднее допустимое время отклика r_{\max} , получаем значение λ_{\max} – наибольшее значение пропускной способности, которое система может обеспечить без превышения порогового значения времени отклика

Для обеспечения удовлетворительного времени отклика необходимо, чтобы интенсивность поступления запросов в систему была меньше λ_{\max} , так что необходимо уметь вычислять эту величину. Аналитического решения для получения значения λ_{\max} не существует, но есть быстрый и простой способ оценки этого значения методом деления интервала пополам. Программа на Visual Basic, выполняющая такое вычисление, приведена в примере 9.2.



В примере 9.2 упоминаются такие объекты, как переменная q и функция *ResponseTime*, определенные в электронной книге Microsoft Excel – ее можно найти по адресу <http://www.oreilly.com/catalog/optoraclep>.

Пример 9.2. Программа на Visual Basic применяет метод деления интервала пополам для вычисления максимальной рабочей пропускной способности

```
Function LambdaMax(Rmax, q, m, mu) As Double
    ' Максимальная рабочая пропускная способность системы массового обслуживания
    ' ДОПУЩЕНИЕ: ResponseTime() – это монотонно возрастающая непрерывная функция
    Const error = 0.005
    ' разбиение интервала прекращается, когда
    ' модуль(lambda1-lambda0) <= error*lambda0

    Dim lambda0 As Double
    ' значение лямбда, для которого R < Rmax
    Dim lambda1 As Double
    ' значение лямбда, для которого R >= Rmax
    Dim lambdaM As Double
    ' арифметическое среднее для {lambda0, lambda1}
    ' Ищем интервал [lambda0, lambda1], для которого R(lambda0)<Rmax
    ' и R(lambda1)>=Rmax

    lambda0 = 0
    lambda1 = 1
    While ResponseTime(m, mu, Rho(lambda1 / q, m, mu)) < Rmax
        lambda0 = lambda1
        lambda1 = 2 * lambda1
    Wend
    ' Сужаем интервал итеративным разбиением пополам
    While Abs(lambda1 - lambda0) > error * lambda0
        lambdaM = (lambda0 + lambda1) / 2
        If ResponseTime(m, mu, Rho(lambdaM / q, m, mu)) < Rmax Then
            lambda0 = lambdaM
        Else
            lambda1 = lambdaM
        End If
    Wend
    LambdaMax = (lambda0 + lambda1) / 2
End Function
```

Интегральная функция распределения времени отклика

Если рассматривать максимальную рабочую пропускную способность в качестве меры производительности, то необходимо иметь в виду одну неувязку. Дело в том, что для пользователей не существует допустимого порога *среднего* времени отклика – на самом деле они думают о возможном допущении для *худшего* времени отклика. Слово «допустимый» скорее ассоциируется с некоторым предельным значением, чем с каким-то непонятным средним. Представим себе такой диалог:

Пользователь: Моя форма ввода заказа работает слишком медленно. Мы договаривались о том, что допустимое время отклика для обработки формы после ее заполнения должно быть полторы секунды, но на самом деле время отклика составляет менее 1,5 секунды лишь в 63,2% случаев.¹

¹ Я не случайно выбрал для этого примера число 63,2%. Это значение интегральной функции экспоненциального распределения в средней точке. То, что в диалоге названы именно эти цифры, означает, что пользователь наблюдает за поведением системы, среднее время отклика которой составляет приблизительно 1,5 секунды.

Администратор системы: На самом деле все именно так, как мы и договаривались. Речь шла лишь о том, что среднее время отклика для формы ввода заказа не должно превышать 1,5 секунд, а вы только что признали, что эта цель была достигнута.

Поставщик информации выполнил то, что было указано в соглашении об уровне обслуживания, но система не соответствует ожиданиям пользователя. На самом деле желание пользователя могло бы быть сформулировано, например, так:

Обработка формы после завершения ввода заказа должна завершаться не позднее чем в через 1,5 секунды, по меньшей мере, для 95 из каждых 100 исполнений.

Или, обобщая:

Функция f должна завершать свою работу не позднее чем через r секунд, по крайней мере, в p процентах своих исполнений.

С функциональной точки зрения подобное утверждение составляет основу соглашения между потребителем и поставщиком информации. Указывается, что для пользователей недопустимо возрастание времени отклика функции выше определенного значения. Отмечается и тот факт, что поставщик не может гарантировать, что абсолютно все пользователи останутся довольны производительностью данной функции, однако поставщик обещает свести неприятности к пренебрежимо малому проценту выполнений.

Как вы, наверное, догадываетесь, теория массового обслуживания предоставляет нам математические средства для вычисления тех величин, которые нам необходимы для создания подобных соглашений о качестве обслуживания. Одна из итоговых формул теории массового обслуживания предоставляет нам возможность определить минимальный уровень вложений в системные ресурсы, который необходим для удовлетворения требований к производительности системы в рамках экономических ограничений ее владельцев. *Интегральная функция распределения (Cumulative Distribution Function – CDF) времени отклика* позволяет вычислить вероятность $P(R \leq r)$ того, что при выполнении определенного запроса время отклика окажется не больше некоторого допустимого значения r . Возможно, это самая полезная характеристика, предоставляемая моделью массового обслуживания, т. к. она позволяет непосредственно измерить степень удовлетворенности пользователя временем отклика.

Интегральная функция распределения времени отклика вычисляется по сложной формуле. Для частного случая – системы массового обслуживания $M/M/m$ (о которой речь пойдет ниже) – она выглядит следующим образом [Gross and Harris (1998) 72–73]:

$$P(R \leq r) = F(r) = \frac{m(1-\rho) - W_q(0)}{m(1-\rho) - 1} (1 - e^{-\mu r}) - \frac{1 - W_q(0)}{m(1-\rho) - 1} (1 - e^{-(m\mu - \lambda)r}),$$

где $W_q(0)$ равно:

$$W_q(0) = 1 - \frac{(m\rho)^m p_0}{m!(1-\rho)},$$

а p_0 равно:

$$p_0 = \left(\sum_{n=0}^{m-1} \frac{(m\rho)^n}{n!} + \frac{(m\rho)^m}{m!(1-\rho)} \right)^{-1}, \quad \rho < 1$$

В примере 9.3 показан вариант решения этой задачи на старом добром Visual Basic.

Пример 9.3. Код на Visual Basic, вычисляющий интегральную функцию распределения времени отклика для системы массового обслуживания M/M/t

```
Function p0(m, Rho) As Double
    ' Вычислить P(zero jobs in system) [Jain (1991), 528]
    Dim i, n As Integer
    Dim t, term2, term3 As Double
    term2 = 1 / (1 - Rho)
    For i = 1 To m
        term2 = term2 * (m * Rho) / i
    Next i
    term3 = 0
    For n = 1 To m - 1
        t = 1
        For i = 1 To n
            t = t * (m * Rho) / i
        Next i
        term3 = term3 + t
    Next n
    p0 = (1 + term2 + term3) ^ (-1)
End Function

Function Wq0(m, Rho) As Double
    ' Вычислить Wq(0) [Gross & Harris (1998) 72]
    ' имейте в виду, что r = m*rho, c = m в обозначениях G&H
    Dim i As Integer
    Dim f As Double
    f = 1
    For i = 1 To m
        f = f * (m * Rho) / i
    Next i
    Wq0 = 1 - f * p0(m, Rho) / (1 - Rho)
End Function
```

```
Function CDFr(r, m, mu, Rho) As Double
    ' CDF времени отклика. Эта функция-оболочка необходима, т.к. формула
    в [Gross & Harris (1998), 73] содержит сингулярность.
    Const epsilon As Double = 0.000000001
    If (Abs(m * (1 - Rho) - 1) < epsilon) Then
```

```

CDFr = (CDFr2(r, m, mu, Rho - epsilon) + CDFr2(r, m, mu,
Rho + epsilon)) / 2
Else
    CDFr = CDFr2(r, m, mu, Rho)
End If
End Function

```

Как я понял, что CDF-формула Джейна ошибочна

Формула Джейна для интегральной функции распределения времени отклика [Jain (1991), 528, 531] довольно долго не давала мне покоя. Она содержит такое выражение:

$$1 - e^{-\mu r} - \frac{C(m, \rho)}{1 - m + m\rho} e^{-m\mu(1-\rho)r} - e^{-\mu r}$$

Я еще не понимал, о чем, собственно, в формуле идет речь, но наличие двух одинаковых членов ($-e^{-\mu r}$) казалось мне странным. Почему автор или редактор не сгруппировал одинаковые члены и не записал их как $-2e^{-\mu r}$? Оставалось предположить, что имела место какая-то типографская ошибка.

Я решил провести эксперимент. Выбрал произвольное целое значение для переменной m . В пакете Mathematica сгенерировал случайное время обслуживания s_1 из экспоненциального распределения со случайно выбранным средним значением $1/\mu$. Затем получил случайное время между последовательными поступлениями запросов t_1 из экспоненциального распределения со случайно выбранным средним значением $1/\lambda$. Применив известную нам формулу $R = S + W$, я вычислил ожидаемое время отклика системы, в которой время обслуживания и время между последовательными поступлениями запросов равнялись соответственно сгенерированным мною случайным числам. Имея на входе значения m , $\mu = 1/s_1$ и $\lambda = 1/t_1$, вычислить R было несложно.

Я повторил свой эксперимент несколько миллионов раз (для нескольких миллионов случайных значений времени обслуживания s_i , выбранных из экспоненциального распределения со средним значением $1/\mu$, и нескольких миллионов случайных значений интенсивности поступлений t_i , выбранных из экспоненциального распределения со средним значением $1/\lambda$). Сохранил результаты. Затем для случайно выбранного значения времени отклика r я просто сосчитал, какое количество значений времени отклика, полученных в ходе моих экспериментов, оказалось меньше r . Количество значений, не превышающих r , должно было приблизительно соответствовать значению интегральной функции распределения Джейна (именно так, по определению, она должна вести себя).

Но формула Джейна раз за разом выдавала результаты, *не совпадающие* с моим экспериментом. Зато формула Гросса и Хэрриса [Gross and Harris (1998) 72–73] всегда давала значения, *совпадающие* с моими результатами. Я пытался связаться с доктором Джейном, чтобы представить ему результаты моих исследований, но на момент написания этой книги ответа не получил.

```

Function CDFr2(r, m, mu, Rho) As Double
' CDF времени отклика, адаптированная из [Gross & Harris (1998), 73]
' Имейте в виду, что r = m*rho, c=m, lambda=rho*m*mu, t=r в обозначениях G&H
  Dim w As Double          ' значение Wq(0)
  Dim cdf1, cdf2 As Double  ' сложности формулы CDF
  If (Rho >= 1 Or r <= 0) Then
    CDFr2 = 0
    Exit Function
  End If
  w = Wq0(m, Rho)
  cdf1 = (m * (1 - Rho) - w) / (m * (1 - Rho) - 1) * (1 - Exp(-mu * r))
  cdf2 = (1 - w) / (m * (1 - Rho) - 1) * (1 - Exp(-(m * mu - Rho * m * mu) * r))
  CDFr2 = cdf1 - cdf2
End Function

```

Случайные величины

Одна из сложностей реальных систем массового обслуживания состоит в том, что невозможно точно предсказать время поступления запросов в систему (на самом деле, если бы запросы поступали в систему через равные интервалы времени и при этом не изменялось бы и время обслуживания, то при условии устойчивости системы *никаких* очередей бы не было). Например, мы можем считать, что запросы в телефонную систему поступают с частотой приблизительно 2,0 запроса в секунду, но неразумно ожидать поступления запросов с интенсивностью *ровно* один запрос каждые полсекунды. В большинстве реальных систем можно ожидать, что поступления запросов будут случайным образом распределены во времени. Мы предполагаем, что можно прогнозировать «среднее поведение» для большого количества запросов, но понимаем, что предсказать каждое отдельное время отклика *невозможно*.

Математическое ожидание

Описывая поведение непредсказуемого процесса, математики употребляют термин «случайная величина». *Случайная величина* – это просто функция со случайными значениями. *Математическое ожидание* (*expected value*) $E[X]$ случайной переменной X – это среднее из принимаемых X значений. Во многих трудах по статистике и теории вероятности прописная буква (как, например, X) обозначает случайную величину, обладающую рядом свойств, в том числе математическим ожиданием и распределением (о котором мы вскоре поговорим). В теории массового обслуживания прописные буквы часто обозначают как случайную величину, так и ее математическое ожидание. Читателям приходится по контексту догадываться о том, что именно подразумевается в каждом конкретном случае. Такие же обозначения приняты и в этой книге, например, технически более точная, но громоздкая формула $E[R] = E[S] + E[W]$ заменяется на $R = S + W$.

Функция плотности вероятности (pdf)

Несмотря на то, что как следует из ее названия, значением случайной переменной является случайное число, сам процесс появления таких значений обычно подчинен какому-то порядку. Например, в телефонной системе со средней интенсивностью поступления в 2,0 запроса в секунду, возможно получение 200 запросов в течение одной секунды, но такая ситуация крайне маловероятна. Математическая функция, моделирующая вероятность принятия случайной величиной определенного значения, называется *распределением* данной случайной величины. Если быть точнее, то вероятность того, что дискретная случайная величина X примет определенное значение x , называется *функцией плотности вероятности* (*probability density function – pdf*) случайной величины и обозначается как $f(x) = P(X = x)$ [Hogg and Tanis (1977) 51–58].

Использование pdf

Точно предсказать время поступления в систему следующего запроса невозможно, поэтому время между поступлениями последовательных запросов представляет собой случайную величину. Соответственно, интенсивность поступлений (значение, обратное предыдущему) также является случайной величиной. В 1909 г. Агнер Эрланг (Agner Erlang) показал, что интенсивность поступления вызовов в телефонной системе часто характеризуется *распределением Пуассона* [Erlang (1909)]. Точнее, если телефонные звонки поступают со средней интенсивностью $\lambda > 0$, то функция плотности вероятности для интенсивности поступления имеет вид:

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!}, \quad x = 0, 1, 2, \dots$$

Если телефонные вызовы поступают в среднем с частотой λ , равной 2 вызовам в секунду, то вероятность того, что в одну секунду будет получено 200 вызовов, равна $f(200) = 2,7575 \times 10^{-316}$. Другими словами, если процесс поступления телефонных вызовов действительно подчиняется распределению Пуассона для $\lambda = 2$, то вероятность получения пятидесяти четырех наборов из десятки, валета, дамы, короля и туза одной масти первым игроком при игре в покер (так называемого «royal flush») больше, чем вероятность существования секундного интервала, в течение которого в систему поступило бы 200 вызовов. Вероятность же того, что односекундный интервал будет заключать в себе ровно ноль, один, два, три или четыре вызова, значительно выше. Функция плотности вероятности для распределения Пуассона с параметром $\lambda = 2$ приведена на рис. 9.8. Кстати, интенсивность поступления запросов во многих компьютерных приложениях, включая и разнообразные компоненты Oracle, также подчиняется распределению Пуассона.

Необходимо отметить, что символ λ , обозначающий среднюю интенсивность поступления запросов в систему массового обслуживания,



Рис. 9.8. Функция плотности вероятности (pdf) для величины, подчиняющейся распределению Пуассона с $\lambda = 2$, отображает вероятность $P(A = x)$ того, что в течение секундного интервала наблюдения поступит ровно x запросов

конечно же, неслучайно применяется и для обозначения среднего значения распределения Пуассона. Немного забегаю вперед, скажу, что специфическая модель $M/M/t$ теории массового обслуживания (о которой мы вскоре поговорим) работает только в том случае, если процесс поступления запросов в систему подчиняется распределению Пуассона с параметром λ . Интенсивность поступлений в теории массового обслуживания обозначается буквой λ , потому что это и *есть* среднее значение распределения Пуассона.

Время обслуживания системы – это тоже случайная величина. Например, время, необходимое банковскому кассиру для подсчета денег клиента, можно оценить и спрогнозировать в среднем, но не в каждом конкретном случае. Невозможно предугадать даже, сколько времени потребуется процессору на выполнение операции ЛЮ в Oracle. *Логический ввод/вывод (ЛЮ)* – это операция, посредством которой ядро Oracle осуществляет выборку одного блока из кэша буферов базы данных. Например, процессор может в среднем обслужить 40 000 запросов ЛЮ в секунду (т. е. $\mu = 40\,000$), но скорость работы может значительно меняться от секунды к секунде. Элемент случайности вносят такие факторы, как тип и сложность блока Oracle (например, является ли блок блоком индекса или же таблицы), изменяющееся количество строк в каждом блоке Oracle и изменяющаяся ширина столбцов данных в таких блоках.

Почему важно понимать распределение вероятностей

Для того чтобы использовать математическое ожидание случайной величины в различных прогнозирующих формулах, необходимо знать, какому распределению она подчиняется. Например, можно сказать,

что во время обеденного перерыва клиенты входили в ресторан в среднем с частотой 2 клиента в минуту, так что среднее время между приходами составляет 30 секунд. Однако по среднему значению не восстановить весь ход событий. Если известен только средний интервал прибытия клиентов, то, например, нельзя ничего сказать о том, приходили ли клиенты поодиночке ровно раз в 30 секунд, или же они появлялись группами. Если известно только, что последовательные запросы поступают в среднем раз в 30 секунд, то, например, просто *нельзя* знать, какой из двух случаев из таблицы 9.1 имел место.

Таблица 9.1. Два существенно разных сценария, в обоих из которых среднее значение интервала между событиями τ равно 30 секундам

Интервалы времени	Количество событий	
	Случай I	Случай II
11:30–11:45	0	34
11:45–12:00	0	28
12:00–12:15	240	31
12:15–12:30	0	37
12:30–12:45	0	24
12:45–13:00	0	30
13:00–13:15	0	32
13:15–13:30	0	24
Среднее для 15-минутных интервалов	30	30

Если в действительности все было подобно случаю I, то нельзя ожидать от математической формулы достоверного предсказания событий, произошедших в период 13:00–13:15, сообщив ей, что «средняя интенсивность визитов составляла 120 в час». Для того чтобы модель массового обслуживания давала достоверные результаты, необходимо сообщить ей некоторую дополнительную информацию о свойствах случайных входных параметров, а не просто их средние значения. Модель также должна знать, каким образом *распределена* каждая случайная величина.

Теория массового обслуживания и «интерфейс ожидания»

Вы познакомились с формулами теории массового обслуживания. Теперь надо понять, как связаны с ними рабочие характеристики Oracle, а именно, как информация, полученная от так называемого «интерфейса ожидания», согласуется с теорией массового обслуживания? К сожалению, сама корпорация Oracle пришла к неверным выводам на эту тему. Статья 223117.1 Oracle MetaLink является тому примером:

Настройка производительности основывается на следующем фундаментальном отношении:

$$\text{Время отклика} = \text{Время обслуживания} + \text{Время ожидания}$$

Для базы данных Oracle «время обслуживания» измеряется при помощи статистики «время процессора, использованное данным сеансом», а «*время ожидания*» – *при помощи событий ожидания* (курсив автора).

Выделенная курсивом часть предложения ложна. На самом деле так называемое событие ожидания Oracle – это *не то*, о чем говорится в данном предложении.

Время ожидания Oracle

Путаница начинается уже с названия «событие ожидания». Термин выбран неудачно, т. к. заставляет пользователей думать, что продолжительность события ядра Oracle равна задержке в очереди. Но это не соответствует действительности. Из главы 7 вы уже знаете, что продолжительность события ожидания на самом деле включает в себя ряд разнообразных составляющих. Компоненты времени отклика для отдельного вызова чтения операционной системы представлены на рис. 9.9.

Время ожидания Oracle, измеренное при выполнении системного вызова, – это общее фактическое время, прошедшее с момента последней инструкции, предшествующей выполнению вызова ОС и до первой инструкции, следующей за возвратом вызова ОС. Все, что происходит в интервале между моментами времени t_0 и t_1 , – это время ожидания Oracle. На рис. 9.9 одно-единственное время ожидания Oracle включает в себя все перечисленные ниже составляющие:

s_{CPU}

Время работы процессора, потраченное на подготовку системного вызова. Для вызова дискового чтения основная часть этого времени проходит в привилегированном режиме. Однако некоторые системные вызовы могут расходовать процессорное время и в пользовательском режиме.

w_{disk}

Задержка в очереди к дисковому устройству, которая на рисунке включает в себя задержку при передаче запроса от процессора к дисковому устройству.

s_{disk}

Время работы дискового устройства, включая задержку поиска, задержку вращения и задержку передачи данных обратно с устройства ввода/вывода в память, к которой обращается процессор.

w_{CPU}

Задержка в очереди к процессору, процесс находится в состоянии готовности к исполнению.

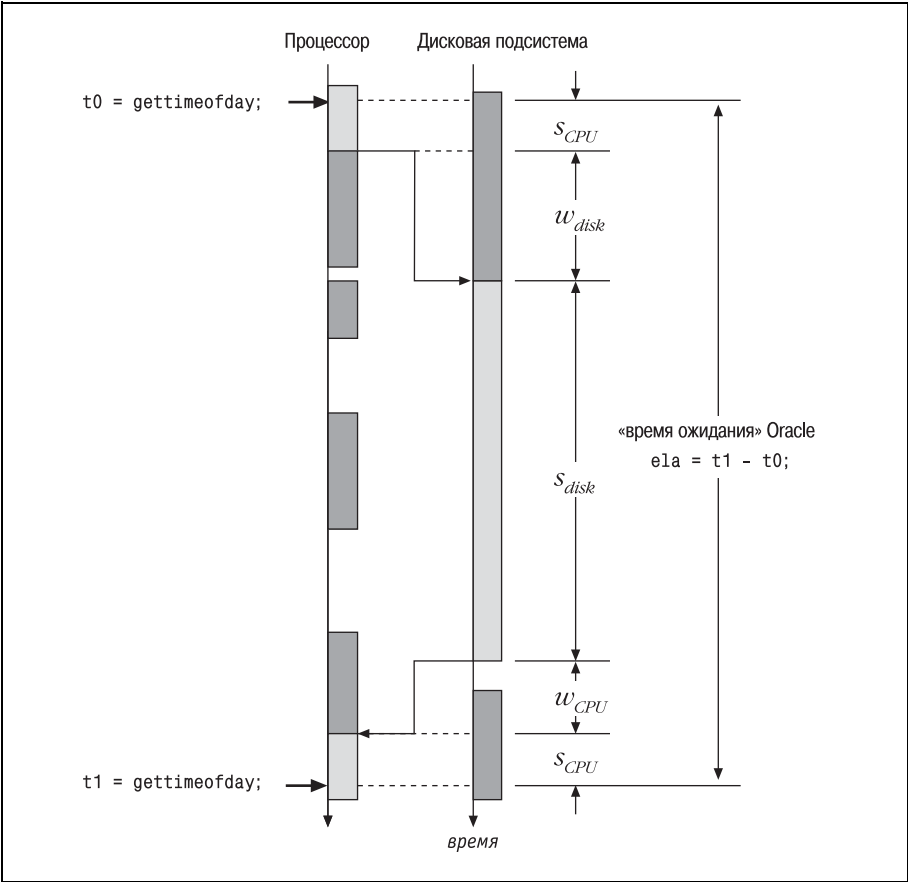


Рис. 9.9. *Время ожидания Oracle для системного вызова (подобного изображенному вызову дискового ввода/вывода) – это фактически время отклика, которое измеряется с точки зрения ядра Oracle. Длительность $ela = t_1 - t_0$ – это не задержка в очереди; она состоит из периодов обслуживания и задержек в очереди*

s_{CPU}

Еще один период работы процессора, необходимый для завершения системного вызова. Кроме того, для одних вызовов процессорное время может расходоваться в привилегированном, а для других – в пользовательском режиме.

Надеюсь, теперь вы отчетливо понимаете, что время ожидания Oracle для вызова чтения ОС не совпадает с величиной w_{disk} . Для остальных системных вызовов ситуация будет аналогичной.

Различные обозначения для понятий теории массового обслуживания

Прочитав несколько книг по теории массового обслуживания, можно еще сильнее запутаться в том, как же на самом деле измеряется время ожидания Oracle. Во всех источниках обычно употребляются одни и те же греческие буквы, под которыми подразумевается одно и то же, а вот формулы теории массового обслуживания авторы часто записывают по-разному (см. табл. 9.2).

Таблица 9.2. Пример представления формул теории массового обслуживания

Формула	Источник
$R = S + W$	Oracle MetaLink, [Gunther (1998) 84]
$T = S + T_q$	[Gross and Harris (1998) 11]
$W = 1/\mu + W_q$	[Gross and Harris (1998) 71]
$s_n = x_n + w_n$	[Kleinrock (1975) 198]

Я перегруппировал члены в правой части каждого из равенств так, чтобы все они представляли одну и ту же сущность. Другими словами, W (Гюнтер и я) – это в точности то же самое, что T_q и W_q (Гросс и Харрис), а также w_n (Клейнрок). Особенно сбивает с толку использование в разных книгах одних и тех же слов для обозначения абсолютно разных вещей, например:

Предполагаемое время ожидания устойчивой системы W равно времени обслуживания $1/\mu$ плюс задержка в очереди W_q [Gross and Harris (1998) 64].

Время отклика R равно сумме времени обслуживания S и времени, потраченного на ожидание в очереди W [Gunther (1998) 52].

Гросс и Харрис обозначают при помощи термина «время ожидания» то, что Гюнтер называет «временем отклика». Более того, разные авторы под термином «ожидание» подразумевают две абсолютно разные вещи. Если обратится к формуле $R = S + W$, то мы увидим, что Гросс и Харрис называют R ожиданием, а для Гюнтера ожидание – это W . По своей сути время ожидания Oracle ближе к определению Гросса и Харриса.

Итак, кто же прав? Выбор слов не имеет значения, важно лишь не смешивать подразумеваемые под ними понятия. Я выбрал обозначения, подобные используемым Джейном и Гюнтером, в основном потому, что именно с этих двух книг началось мое знакомство с теорией массового обслуживания. R , S и W можно называть любыми именами. Что *неправильно*, так это воспринимать время ожидания Oracle как одно из слагаемых правой части выражения $R = S + W$. На самом деле время ожидания Oracle – это время отклика для вызова операционной системы с точки зрения ядра Oracle. Ядро Oracle публикует значения времени ожидания во многих местах, в том числе в статистике `ela`

в строках WAIT расширенной трассировки и в следующих фиксированных представлениях:

```
V$SESSION_WAIT.WAIT_TIME  
V$SESSION_EVENT.TIME_WAITED  
V$SESSION_EVENT.AVERAGE_WAIT  
V$SESSION_EVENT.MAX_WAIT  
V$SESSION_EVENT.TIME_WAITED_MICRO  
V$SYSTEM_EVENT.TIME_WAITED  
V$SYSTEM_EVENT.AVERAGE_WAIT  
V$SYSTEM_EVENT.TIME_WAITED_MICRO
```

Каждая из этих статистик ссылается на некий промежуток времени, включающий задержку в очереди для запрашиваемого устройства, но кроме этого время ожидания Oracle включает в себя и множество других составляющих времени отклика. Точнее говоря, время ожидания Oracle – это *не* W из отношения $R = S + W$ теории массового обслуживания.

Модель массового обслуживания М/М/т

Модель М/М/т – это набор математических формул, которые позволяют прогнозировать производительность систем массового обслуживания, удовлетворяющих пяти специфическим условиям. Обозначение М/М/т на самом деле представляет собой сокращение более длинного М/М/т/∞/FCFS, полностью описывающего все пять условий:

М/М/т/∞/FCFS (*экспоненциальное время между поступлениями запросов*)

Время между поступлениями последовательных запросов является экспоненциально распределенной случайной переменной (смысл данного утверждения я поясню немного позже).

М/М/т/∞/FCFS (*экспоненциальное время обслуживания*)

Время обслуживания представляет собой экспоненциально распределенную случайную переменную.

М/М/т/∞/FCFS (*т однородных параллельных независимых каналов обслуживания*)

Существует t параллельных каналов обслуживания, при этом все они имеют одинаковые характеристики функциональности и производительности и все одинаково готовы к предоставлению обслуживания любому поступающему запросу. Например, в системе М/М/1 существует один канал обслуживания, а в системе М/М/32 – 32 параллельных канала обслуживания.

М/М/т/∞/FCFS (*отсутствие ограничений на длину очереди*)

На длину очереди не наложено никаких ограничений. Ни один запрос, попадающий в очередь, не покидает ее, не получив соответствующего обслуживания.

M/M/m/∞/FCFS (обслуживание в порядке поступления)

Дисциплина очереди – обслуживание в порядке поступления запросов. (First-Come, First-Served – FCFS). Система предоставляет запросам обслуживание в том порядке, в котором они были получены.

Системы M/M/m

Пять приведенных условий удачно вписываются в реальность систем Oracle. В нашей повседневной жизни нередко встречаются системы массового обслуживания, например:

- Агентство по продаже авиабилетов, в котором шесть билетных кассиров обслуживают клиентов, стоящих в одной длинной извилистой очереди. Это система M/M/6.
- Четырехрядная платная автомобильная дорога, на которой один пункт оплаты обслуживает легковые машины и грузовики, выбираемые из начала очередей, которые стоят на каждой полосе. Это четыре разных системы M/M/1, в каждой из которых средняя интенсивность поступлений выбрана соответствующим образом для каждого ряда отдельно.
- Симметричная многопроцессорная компьютерная система, в которой 12 процессоров обслуживают запросы, выбираемые из начала очереди запросов, готовых к исполнению. В связи с недостаточной масштабируемостью операционной системы (к которой я еще вернусь) для такой системы подходит модель M/M/m, в которой m находится в диапазоне $0 < m < 12$.

Почему «М» обозначает экспоненциальность

Может показаться удивительным, что теоретики массового обслуживания обозначают экспоненциальное распределение не буквой «Е», а буквой «М». Дело в том, что буква «Е» уже занята – именно ею обозначается распределение Эрланга (Erlang). Математикам пришлось выбирать для экспоненциального распределения букву, отличную от «Е», и они выбрали «М», т. к. экспоненциальное распределение обладает уникальным марковским свойством или свойством отсутствия последействия (memoryless – «отсутствия памяти»¹). Другие модели обозначаются буквами «G» (general) – произвольное, «D» (deterministic) – детерминированное и «H_k» (k-stage hyperexponential) – гиперэкспоненциальное распределение порядка k.

¹ Марковское свойство системы состоит в том, что будущее состояние системы зависит только от ее состояния в настоящем и не зависит от ее состояний в прошлом. Переформулировав, можно сказать, что для любых двух непересекающихся промежутков времени число событий, наступающих на одном из них, не зависит от числа событий, наступающих на другом. – *Примеч. перев.*

Очевидно, что все эти примеры удовлетворяют условиям m, ∞ и FCFS модели М/М/ m/∞ /FCFS. Для проверки же соответствия требованию М/М необходим дополнительный анализ. В следующем разделе мы поговорим о том, что означает экспоненциальность распределения случайной переменной.

Не-М/М/т системы

Прежде чем переходить к изучению критерия М/М, отметим, что не все системы массового обслуживания представляют собой системы М/М/ m . Например, следующие системы *не* являются системами М/М/ m :

- Агентство по продаже авиабилетов, в котором пять билетных каскиров обслуживают клиентов, при этом клиенты бизнес-класса и первого класса могут проходить вне очереди. Эта система нарушает принцип FCFS, соблюдение которого необходимо для того, чтобы система относилась к разряду М/М/ m .
- Массив из шести независимых компьютерных дисков, каждый из которых обслуживает запросы ввода/вывода из начала выделенной для него очереди. Эта система нарушает условие М/М/6, заключающееся в том, что все участвующие в обслуживании параллельные каналы одинаково готовы к обслуживанию любого прибывающего запроса. Например, запрос на чтение диска D может быть выполнен только диском D , вне зависимости от того, свободны ли остальные диски. Такую систему можно смоделировать на основе шести независимых систем М/М/1.
- Получение защепок Oracle. Защелки Oracle выделяются запросам не в порядке их поступления [Millsap (2001c)]. Поэтому система получения защепок Oracle нарушает FCFS-принцип М/М/ m .

Многие из систем, удовлетворяющих условиям $\dots/m/\infty$ /FCFS, нарушают принцип М/М из-за того, что процессы поступления запросов и их обслуживания не подчиняются экспоненциальному распределению. В следующих разделах я расскажу о том, как проверить, подчиняются ли данные экспоненциальному распределению.

Экспоненциальное распределение

Я уже упоминал, что в начале 1900-х годов Агнер Эрланг заметил, что интенсивность поступления запросов в телефонную систему «подчиняется распределению Пуассона». Выражение «подчиняется такому распределению» означает, что функция плотности вероятности исследуемой случайной величины имеет определенную математическую форму – в данном случае является пуассоновской функцией. Если нам известна функция плотности вероятности случайной величины, то мы можем вычислить вероятность того, что данная случайная величина примет любое интересующее нас конкретное значение. Говорят, что случайная величина имеет *экспоненциальное распределение* с пара-

метром $\theta > 0$ (греческая буква «тета»), если ее функция плотности вероятности имеет такой вид:

$$f(x) = \frac{1}{\theta} e^{-x/\theta}, \quad 0 \leq x < \infty$$

Функция плотности вероятности случайной величины с экспоненциальным распределением представлена на рис. 9.10.



Рис. 9.10. Функция плотности вероятности экспоненциально распределенной с параметром $\theta = 0.5$ случайной величины

Во многих реально существующих системах время между поступлениями запросов и время обслуживания распределяются экспоненциально, но для того чтобы достоверно смоделировать систему $M/M/t$, необходимо проверить процессы обслуживания и поступления запросов в такой системе. В этой книге содержится вся информация, необходимая для проверки соответствия рабочих характеристик конкретной системы модели $M/M/t$.

Распределение Пуассона и экспоненциальное распределение

Агнер Эрланг заметил, что интенсивность поступления вызовов в телефонную систему подчиняется распределению Пуассона. Я уже отмечал, что многие процессы поступления запросов в компьютерных системах, в том числе и в Oracle-системах, также подчиняются распределению Пуассона. Почему же тогда я решил предложить вашему вниманию $M/M/t$ -модель массового обслуживания, которая работает лишь тогда, когда время между поступлениями запросов и время обслуживания характеризуются *экспоненциальным* распределением? Почему я не выбрал модель, в которой процессы поступления и обслуживания запросов подчиняются распределению Пуассона?

Дело в том, что на самом деле я *выбрал* модель, в которой процессы поступления и обслуживания запросов подчиняются распределению Пуассона. Экспоненциальное и пуассоновское распределения связаны обратной зависимостью [Gross and Harris (1998) 16–22]:

- Для того чтобы система удовлетворяла условию первого «М» из «М/М/т», *время между поступлениями запросов* должно иметь экспоненциальное распределение. Если вы помните, среднее время между поступлениями запросов τ – это величина, обратная средней частоте поступления ($\tau = 1/\lambda$). Частота поступления имеет распределение Пуассона с параметром λ в том и только в том случае, если соответствующее время между поступлениями подчиняется экспоненциальному распределению с параметром $\theta = \tau = 1/\lambda$.
- Для того чтобы система удовлетворяла условию второго «М» из «М/М/т», *ее время обслуживания* должно быть распределено экспоненциально. Естественно, среднее время обслуживания S обратно пропорционально средней скорости обслуживания ($S = 1/\mu$). Скорость обслуживания имеет распределение Пуассона с параметром μ в том и только в том случае, если соответствующее время обслуживания подчиняется экспоненциальному распределению с параметром $\theta = S = 1/\mu$.

Поэтому некоторые авторы говорят о модели М/М/т как о модели, в которой интенсивность поступления и время обслуживания распределены по закону Пуассона.

Проверка на соответствие экспоненциальному распределению

Две буквы «М» в обозначении модели массового обслуживания М/М/т означают, что мы можем использовать модель лишь в том случае, если время между поступлениями запросов и время их обслуживания имеют экспоненциальное распределение. То есть мы можем применять М/М/т для моделирования производительности системы, *только* если гистограммы времени между поступлениями запросов в систему и скорости обслуживания имеют такой же вид, как и кривая функции плотности вероятности при экспоненциальном распределении, приведенная на рис. 9.11.

Вопрос в том, как определить, достаточно ли «похожи» множество промежутков времени между поступлениями или интенсивность обслуживания на то, что изображено на рис. 9.10? Для определения степени соответствия набора значений некоторому распределению статистики применяют *критерий согласия хи-квадрат* (*chi-square goodness-of-fit test*). Программа на Perl в примере 9.4 проверяет степень соответствия набора чисел, хранящегося в файле, экспоненциальному распределению. Она выносит вердикт: «Accept» (соответствует), «Almost suspect» (почти сомнительна), «Suspect» (вызывает сомнения) или «Reject» (не соответствует), основываясь на процедуре, рекомендованной

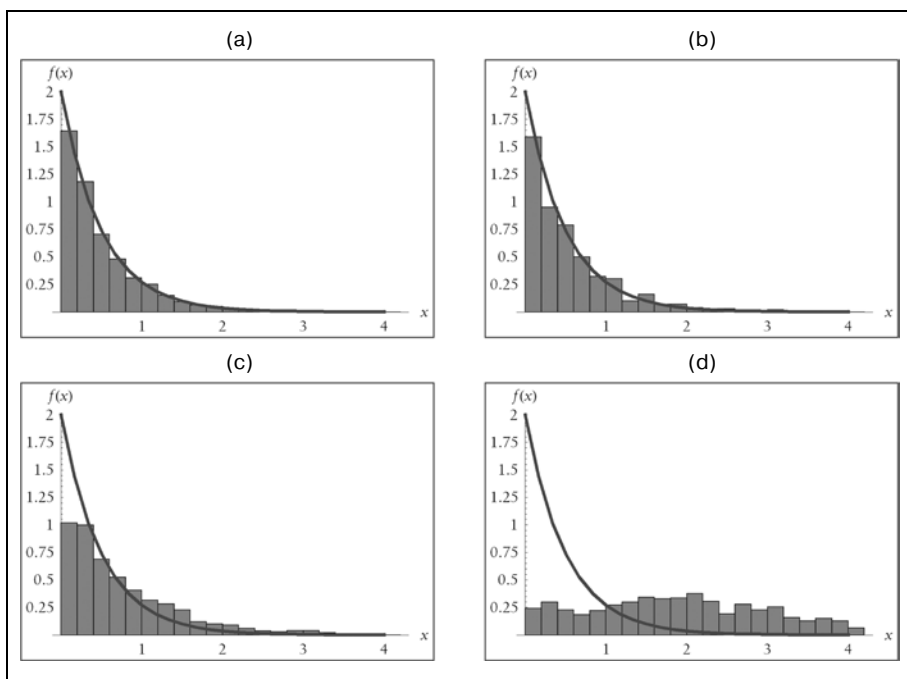


Рис. 9.11. Примеры случайных данных, в различной степени удовлетворяющих экспоненциальному распределению с параметром $\theta = 0,5$

в [Knuth (1981) 43–44]. Если данных слишком мало для выполнения теста «хи-квадрат», программа выдаст соответствующее сообщение. Если же по отношению к измеренным значениям времени между поступлениями запросов и времени обслуживания будет вынесен вердикт «Accept» или «Almost suspect», то можно не без оснований считать, что модель $M/M/t$ даст достоверные результаты.

Модель $M/M/t$ сформирует достоверные прогнозы лишь в том случае, если и время между поступлениями запросов, и время обслуживания являются экспоненциально распределенными случайными величинами. Другие модели массового обслуживания могут предложить точные прогнозы для систем, в которых эти характеристики не подчиняются экспоненциальному распределению. Я сосредоточился на модели $M/M/t$ потому, что она во многих случаях очень удачно подходит для анализа производительности Oracle. При работе над проектами повышения производительности Oracle обычно удается выделить подмножества рабочей нагрузки, удовлетворяющие условиям M/M , например:

- Легко проверить экспоненциальность распределения времени между поступлениями запросов и времени обслуживания при выполнении пакетных заданий. Хороший диспетчер очереди пакетных заданий записывает время постановки задания в очередь, время нача-

ла и завершения выполнения заданий для последующего анализа. Время между поступлениями заданий – это просто разность времен постановки в очередь для данного и предыдущего по отношению к нему задания. Время обслуживания задания – это разность времени завершения и времени начала выполнения задания. Получив 50 или более значений времени между поступлениями заданий и 50 или более значений времени обслуживания, можно определить, подчиняется ли данное подмножество пакетных заданий М/М-условиям модели М/М/т.

Важно применять модель М/М/т только для подмножества пакетных данных, демонстрирующего соответствующее поведение. Например, если рассматривать время между поступлениями пакетных заданий на протяжении 24-часового периода, то они вероятнее всего не будут распределены экспоненциально – ночной интервал наверняка будет значительно больше дневного. Аналогично, время обслуживания всех пакетных заданий также вряд ли будет распределено экспоненциально. А вот время обслуживания всех заданий, выполненных в течение менее одной минуты, скорее всего будет подчиняться экспоненциальному распределению.

- Логический ввод/вывод Oracle (LIO) удобно использовать в качестве единицы измерения запросов на обслуживание. В Oracle невозможно напрямую измерить интервалы между поступлениями или время обслуживания LIO, но интуиция и успешный опыт применения М/М/т для моделирования производительности LIO указывают на то, что в действительности время между поступлениями и время обслуживания LIO распределены экспоненциально. Исполнение любой бизнес-функции можно представить в терминах количества выполненных LIO, так что результат применения модели массового обслуживания можно выразить в терминах времени отклика и производительности бизнес-функции. Будет просто замечательно, если вы сможете думать о функциях приложения в терминах количества исполняемых LIO!

Программа для проверки экспоненциальности распределения

Все хорошие книги по теории массового обслуживания сообщают своим читателям, что прежде чем применять модель М/М/т, необходимо убедиться в экспоненциальном характере распределения времени между поступлениями запросов и времени обслуживания в моделируемой системе. Трудность в том, что большая часть этих хороших книг не дает никакого практического совета о том, как в этом бы убедиться. Такую задачу может выполнить программа на Perl, приведенная в примере 9.4. Ее идея подсказана мне работой [Allen (1994) 224–225]. В реализации я руководствовался в основном книгой [Knuth (1981) 38–45] с дополнительным привлечением средств пакета Mathematica, [Olkin et al. (1994)], [CRC (1991)] и ресурсов <http://www.cpan.org>.

Для того чтобы начать работу с программой, загрузите исходный текст в систему, где установлен Perl. В Unix (Linux, HP-UX, Solaris, AIX и т. д.), вы, вероятно, назовете этот файл *mdist*. В Windows он, вероятно, будет назван *mdist.pl*. Возможно, что в Unix-системе придется отредактировать первую строку кода, указав там точную ссылку на исполняемый файл Perl (может быть, например, исполняемый файл называется */usr/local/bin/perl*). Затем наберите в командной строке `perldoc mdist` (или `perldoc mdist.pl`) для обращения к странице руководства для данной программы.

Пример 9.4. Программа на Perl, оценивающая, соответствует ли распределение случайной переменной экспоненциальному

```
#!/usr/bin/perl

# $Header: /home/cvs/cvm-book1/mdist/mdist.pl,v 1.7 2002/09/05 23:03:57 cvm
# Cary Millsap (cary.millsap@hotsos.com)
# Copyright (c) 2002 by Hotsos Enterprises, Ltd. All rights reserved.

use strict;
use warnings;

use Getopt::Long;
use Statistics::Distributions qw(chisqr distr chisqrprob);

my $VERSION = do { my @r=(q$Revision: 1.23 $="/\d+/g); sprintf "%d"."%02d"
x$#r,@r };
my $DATE = do { my @d=(q$Date: 2003/10/24 20:24:37 $="/
\d{4}\D\d{2}\D\d{2}/g); sprintf $d[0] };
my $PROGRAM = "Test for Fit to Exponential Distribution";

my %OPT;

sub x2($$) {
    my ($mu, $p) = @_;
    # Параметр p, который ожидает &Statistics::Distributions::chisqr distr,
    # - это дополнение того, что мы находим в [Knuth (1981) 41],
    # Mathematica или [CRC (1991) 515].
    return chisqr distr($mu, 1-$p);
}

sub CDFx2($$) {
    my ($n, $x2) = @_;
    # Параметр p, который &Statistics::Distributions::chisqrprob
    # возвращает, - это дополнение того, что мы находим
    # в [Knuth (1981) 41], Mathematica или [CRC (1991) 515].
    return 1 - chisqrprob($n, $x2);
}

sub mdist(%) {
    my %arg = (
        list      => [],           # список значений для проверки
        mean      => undef,        # математическое ожидание
        # (среднее) распределения
```

```

quantiles => undef,          # количество квантилей для проверки
@_,                          # входные аргументы подменяют
                              # значения по умолчанию
);

# Заполняем список. Если список не содержит хотя бы 50 наблюдений, то
# тест хи-квадрат не действителен [Olkin et al. (1994) 613].
my @list = @{$arg{list}};
die "Not enough data (need at least 50 observations)\n" unless @list >= 50;

# Вычисляем количество квантилей и количество ожидаемых наблюдений
# для каждого квантиля. Если в каждом квантиле не ожидается хотя бы
# 5 наблюдений, то квантилей избыточное количество [Knuth (1981) 42]
# и [Olkin et al. (1994) 613].
my $quantiles = $arg{quantiles} ? $arg{quantiles} : 4;
my $m = @list/$quantiles;      # ожидаем, что квантили имеют
                              # одинаковые области

die "Too many quantiles (using $quantiles quantiles requires at least
    ". 5*$quantiles ." observations)\n" unless $m >= 5;

# Определяем степени свободы среднего значения и хи-квадрат. Если
# среднее значение не было передано, то оцениваем его. Но если
# оценивать среднее значение, то будет потеряна дополнительная степень
# свободы хи-квадрат.
my $mean = $arg{mean};
my $n_loss = 1;                # теряем одну степень свободы
                              # для угадывания квантилей

if (!defined $mean) {
    my $s = 0; $s += $_ for @list;    # суммируем полученные
                                      # наблюдениями значения
    $mean = $s/@list;                # вычисляем среднее значение
                                      # для выборки
    $n_loss++;                       # теряем дополнительную степень
                                      # свободы для оценки среднего значения
}

my $n = $quantiles - $n_loss;        # степени свободы хи-квадрат
die "Not enough quantiles for $n_loss lost degrees of freedom (need at
    least ". ($n_loss+1) ." quantiles)\n" unless $n >= 1;

# Выводим все вычисленные значения.
if ($OPT{debug}>=1) {
    print "list      = (", join(", ", @list), ")\n";
    printf "quantiles = %d\n", $quantiles;
    printf "mean      = %s\n", $mean;
}

# Вычисляем внутренние границы квантилей. N.B.: Определение границ
# квантиля отличает эту проверку на экспоненциальность распределения
# от проверки на другой тип распределения. Если входной список распределен
# экспоненциально, то мы ожидаем, что в каждом квантиле будет по $m
# наблюдений, при этом границы будут определяться следующим образом:
#  $-\text{mean} \cdot \log(1 - i / \text{quantiles})$  для каждого  $i = 1.. \text{quantiles} - 1$ .
```

```

my @q;                                     # список внутренних границ квантилей
for (my $i=1; $i<=$quantiles-1; $i++) {
    $q[$i] = -$mean*log(1-$i/$quantiles);
}

# Вычисляем частоту наблюдаемых значений [Knuth (1981) 40]. Если
# назначить $Y[0]=undef, то содержимое массива начнется с $Y[1],
# что упрощает индексацию массива.
my @Y = (undef, (0) x $quantiles);
for my $e (@list) {
    print "e=$e\n" if $OPT{debug}>=3;
    for (my $i=1; $i<=$quantiles; $i++) {
        print "  i=$i (before): q[$i]=$q[$i]\n" if $OPT{debug}>=3;
        if ($i == $quantiles) { $Y[-1]++; print "    Y[-1]->$Y[-1]\n" if
$OPT{debug}>=3; last }
        if ($e <= $q[$i])      { $Y[$i]++; print "    Y[$i]->$Y[$i]\n" if
$OPT{debug}>=3; last }
    }
}

# Заполняем список, содержащий частоту ожидаемых значений в квантилях
# [Knuth (1981) 40]. В нашем тесте использование для этой цели
# структуры данных является излишней сложностью, но при проверке
# на другие распределения в других приложениях это могло бы упростить
# задачу (мы же могли просто везде использовать $m вместо
# $np[$anything]).
my @np = (undef, ($m) x $quantiles);

# Выводим содержимое структуры данных при отладке.
if ($OPT{debug}>=1) {
    print "mean = $mean\n";
    print "q = (", join(", ", @q[1 .. $quantiles-1]), ")\n";
    print "Y = (", join(", ", @Y[1 .. $quantiles] ), ")\n";
    print "np = (", join(", ", @np[1 .. $quantiles] ), ")\n";
}

# Вычисляем статистику хи-квадрат [Knuth (1981) 40].
my $V = 0;
$V += ($Y[$_] - $np[$_])**2 / $np[$_] for (1 .. $quantiles);

# Вычисляем результат verdict как функцию от попадания V в строку
# соответствующей степени свободы в статистической таблице хи-квадрат
# [Knuth (1981) 43-44].
my $verdict;
my $p = CDFx2($n, $V);
if ($p < 0.01) { $verdict = "Reject" }
elsif ($p < 0.05) { $verdict = "Suspect" }
elsif ($p < 0.10) { $verdict = "Almost suspect" }
elsif ($p <= 0.90) { $verdict = "Accept" }
elsif ($p <= 0.95) { $verdict = "Almost suspect" }
elsif ($p <= 0.99) { $verdict = "Suspect" }
else { $verdict = "Reject" }

```



```

    # Возвращает хеш, содержащий результат и ключевые статистики.
    return (verdict=>$verdict, mean=>$mean, n=>$n, V=>$V, p=>$p);
}

%OPT = (
    mean          => undef,
    quantiles     => undef,
    debug         => 0,
    version       => 0,
    help          => 0,
);

GetOptions(
    "mean=f"      => \$OPT{mean},
    "quantiles=i" => \$OPT{quantiles},
    "debug=i"     => \$OPT{debug},
    "version"     => \$OPT{version},
    "help"        => \$OPT{help},
);

if ($OPT{version}) { print "$VERSION\n"; exit }
if ($OPT{help})    { print "Type 'perldoc $0' for help\n"; exit }
my $file = shift; $file = "&STDIN" if !defined $file;
open FILE, "<$file" or die "can't read '$file' (!)";
my @list;
while (defined (my $line = <FILE>)) {
    next if $line =~ /^#/;
    next if $line =~ /^\\s*$/;
    chomp $line;
    for ($line) {
        s/[~0-9.]/ /g;
        s/~/\\s*/g;
        s/\\s*/ /g;
    }
    push @list, split(/\\s+/, $line);
}
close FILE;
print join(" ", @list), "\\n" if $OPT{debug};

my %r = mdist(list=>[@list], mean=>$OPT{mean}, quantiles=>$OPT{quantiles});
print " Hypothesis: Data are exponentially distributed with mean $r{mean}\\n"
;
printf "Test result: n=%d V=%.2f p=%0.3f\\n", $r{n}, $r{V}, $r{p};
print "      Verdict: $r{verdict}\\n";

__END__

=head1 NAME

mdist - проверка данных на соответствие экспоненциальному распределению
с определенным средним значением

=head1 SYNOPSIS

mdist [--среднее=I<m>] [--квантили=I<q>] [I<file>]

```

=head1 DESCRIPTION

`B<mdist>` проверяет, подчиняется ли случайная величина экспоненциальному распределению со средним значением `I<m>`. Такая информация может быть полезна, например, для того, чтобы определить, является ли имеющийся перечень собранных значений времени обслуживания или времени между последовательными запросами подходящим для модели `M/M/m` теории массового обслуживания.

`B<mdist>` читает в файле `I<file>` список значений, полученных в результате наблюдений. Если входной файл не указан, то `B<mdist>` берет входные данные из `STDIN`. Входные данные должны содержать значения как минимум 50 наблюдений.

Программа выводит проверяемое предположение, результаты теста и вердикт:

```
$ perl mdist.pl 001.d
```

```
  Гипотеза: Данные распределены экспоненциально со средним
              значением 0.000959673232
```

```
  Результат теста: n=2 V=0.72 p=0.302
```

```
  Вердикт: Асепт
```

Статистики результатов теста включают в себя [Knuth (1981) 39–45]:

```
=over 4
```

```
=item I<n>
```

Степени свободы из теста хи-квадрат.

```
=item I<V>
```

Статистика «хи-квадрат».

```
=item I<p>
```

Вероятность того, что `I<V>` попадет в распределение хи-квадрат с `I<n>` степенями свободы.

```
=back
```

`B<mdist>` использует для экспоненциального распределения тест хи-квадрат с `I<q>` квантилями, созданный на основе [Allen (1994) 224–225] и [Knuth (1981) 39–40]. Аллен предложил делить данные на квантили и проверять, соответствует ли частота в каждом квантиле ожидаемому экспоненциальному распределению. Кнут предложил общую идею применения теста хи-квадрат, который выносит вердикт "Accept", "Almost suspect", "Suspect" или "Reject".

```
=head2 Options
```

```
=over 4
```

```
=item B<--среднее=>I<m>
```

Гипотетическое математическое ожидание случайной величины (т.е. гипотетическое среднее значение экспоненциального распределения). Если `I<m>` не указано, то `B<mdist>` использует среднее значение входной выборки и соответствующим образом корректирует степени свободы хи-квадрат.

```
=item B<--квантили=>I<q>
```

Количество квантилей, которое будет использоваться в критерии согласия

хи-квадрат. Количество квантилей должно быть не меньше 2, если среднее указано, и не меньше 3, если среднее будет вычисляться. Квантилей не должно быть слишком много, т.к. отношение количества наблюдений к $I_{<q}$ должно быть не меньше 5. По умолчанию $I_{<q} \geq 4$.

=back

=head1 AUTHOR

Cary Millsap (cary.millsap@hotsos.com)

=head1 BUGS

Вместо того чтобы оценивать среднее значение распределения, вычислив выборочное среднее, вероятно, следует использовать прием минимальной оценки хи-квадрат, описанный в [Olkin et al. (1994) 617-623].

=head1 SEE ALSO

Allen, A. O. 1994. Computer Performance Analysis with Mathematica. Boston MA: AP Professional

CRC 1991. Standard Mathematical Tables and Formulae, 29ed. Boca Raton FL: CRC Press

Knuth, D. E. 1981. The Art of Computer Programming, Vol 2: Seminumerical Algorithms. Reading MA: Addison Wesley

Olkin, I.; Gleser, L. J.; Derman, C. 1994. Probability Models and Applications, 2ed. New York NY: Macmillan

Wolfram, S. 1999. Mathematica. Champaign IL: Wolfram

=head1 COPYRIGHT

Copyright (c) 2002 by Hotsos Enterprises, Ltd. All rights reserved.

Поведение систем М/М/т

Прелесть М/М/т в том, что применение модели делает возможным проведение экспериментов с такими параметрами, манипулирование которыми в реальной жизни обошлось бы слишком дорого. В этом разделе мы поговорим об интересных особенностях поведения М/М/т. Эти особенности помогут понять, как избежать проблем с производительностью, которым подвержены реальные многоканальные системы массового обслуживания. В результате вы, вероятно, начнете гораздо глубже разбираться в системе Oracle.

Масштабируемость многоканальной системы

Безусловно, два процессора лучше, чем один. Но почему? И при каких условиях? Попробуем найти ответ при помощи диаграммы последовательности (рис. 9.12). В случае *a* первый вызов дискового ввода/вывода не может сразу же вернуться на единственный процессор системы, т.к. процессор занят выполнением другой работы. Поэтому образуется очередь к процессору, что, естественно, увеличивает время откли-

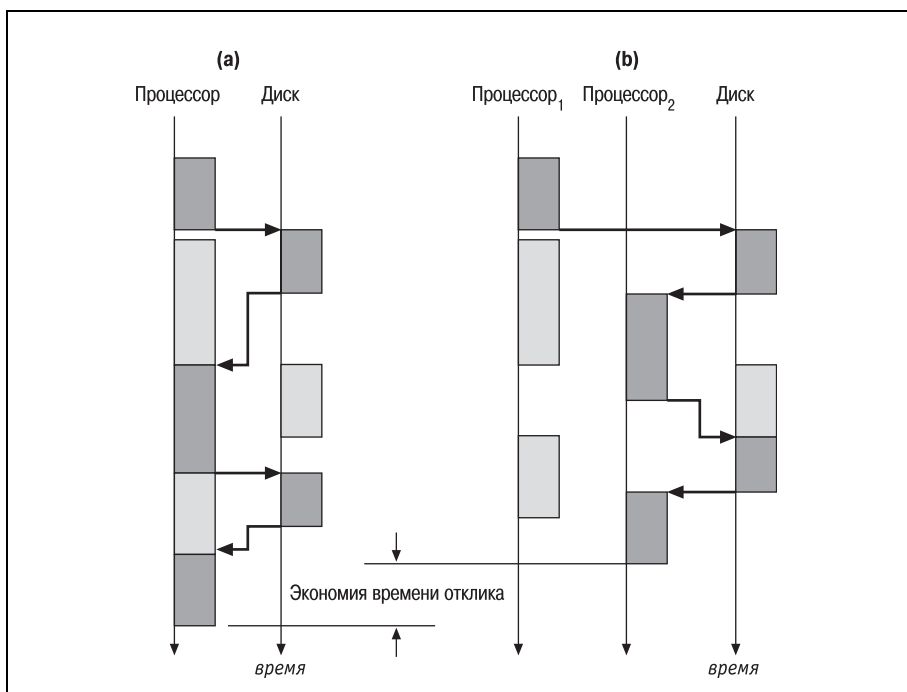


Рис. 9.12. Увеличение количества каналов обслуживания (в данном случае процессоров) улучшает время отклика в нагруженных системах за счет уменьшения задержки в очереди

ка. В случае *b* система имеет два процессора. Когда вызов дискового ввода/вывода заканчивается, то оказывается, что *процессор1* занят, но *процессор2* готов к работе и может обслужить запрос, что приводит к исчезновению задержки в очереди и улучшает время отклика для бизнес-функции. Обратите внимание на интересный эффект возникновения нового «узкого места» на диске в случае *b*.

Уменьшение задержки в очереди в системах с большим значением m четко проявляется в модели $M/M/m$. Влияние количества параллельных каналов обслуживания на производительность изображено на рис. 9.13. Несмотря на то, что время обслуживания (S) остается неизменным во всех системах, представленных на рисунке, время отклика (R) при большой интенсивности поступления запросов (λ) оказывается меньше в системах с большим количеством каналов обслуживания (m) за счет уменьшения задержки в очереди ($W = R - S$).

Так что же лучше – система с одним очень быстрым процессором или система с $m > 1$ более медленными процессорами? У большинства консультантов правильный ответ запрограммирован в их ДНК: «Это зависит». Основываясь на модели $M/M/m$, мы можем ответить на замечательный вопрос: «От чего зависит?».

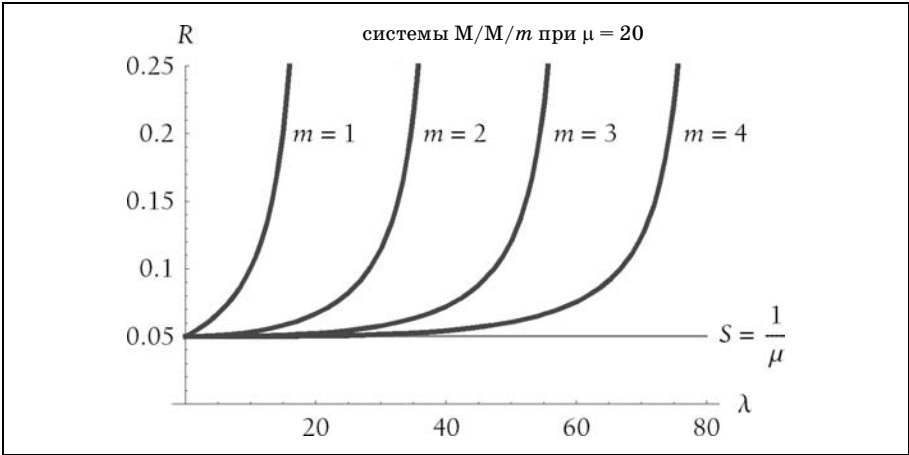


Рис. 9.13. Увеличение количества параллельных каналов обслуживания обеспечивает увеличение пропускной способности для обслуживания поступающих запросов, что снижает время отклика при высокой интенсивности поступления запросов

Рис. 9.14 ясно показывает, что все «зависит» только от одной переменной – интенсивности поступления запросов. При низкой интенсивности система с $m = 1$ быстрым процессором обеспечит наилучшую производительность. При высокой интенсивности поступления запросов система с $m > 1$ будет работать быстрее. Можно вычислить критическую точку λ_{eq} , изобразив кривые времени отклика при помощи инст-

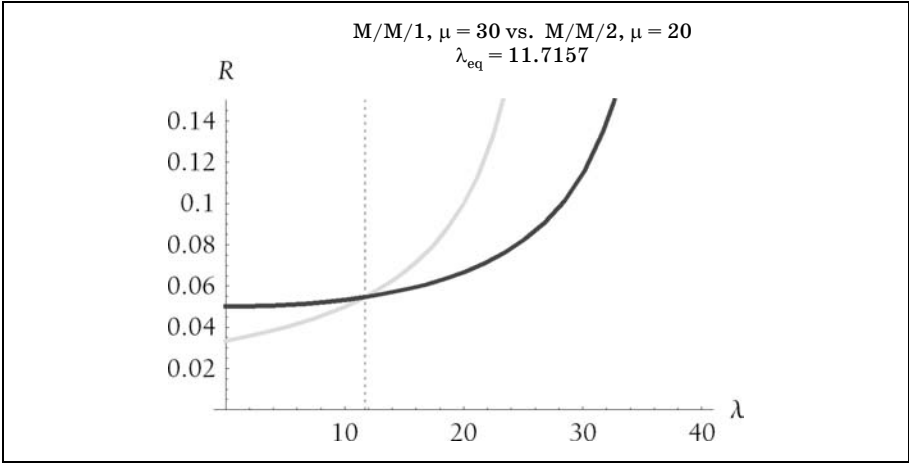


Рис. 9.14. Компьютер с одним быстрым процессором обеспечивает лучшее время отклика при низкой интенсивности поступления запросов, а компьютер с четырьмя более медленными процессорами – при высокой интенсивности поступления запросов

румента, подобного предлагаемому нами на <http://www.hotsos.com>. Те, кого интересует более точная информация, могут найти значение λ , при котором времена отклика двух систем совпадают, применяя метод деления интервала пополам (этот метод был использован в функции *LambdaMax* из примера 9.2). Или же можно вычислить λ_{eq} условно при помощи пакета *Mathematica*.

Люди, работавшие с системами обоих типов, так рассказывают о том, в чем проявляются на практике особенности, отраженные на рис. 9.14:

- Длительные ночные пакетные задания быстрее выполняются в системах с одним быстрым процессором. Такой результат часто удивляет пользователей, которые «усовершенствуют» систему за счет перехода от одного процессора к нескольким. Но когда однопоточное задание в одиночку выполняется в системе, интенсивность поступления запросов невысока. Задание будет быстрее выполняться на быстром процессоре. Многопроцессорная конфигурация не обеспечивает снижения задержки в очереди для системы с низкой интенсивностью поступления запросов, потому что в таких системах вообще нет очередей.
- Многопроцессорная система обеспечивает лучшее время отклика пользователям, работающим в оперативном режиме в часы наибольшей загрузки. Такой результат объясняется снижением задержки в очереди, которое обеспечивается многоканальностью системы. Многопроцессорные системы лучше приспособляются к высокой интенсивности поступления запросов (вызванной, например, большим количеством одновременно работающих пользователей), чем однопроцессорные системы.

Естественно, при покупке системы выбор определяется множеством как технических, так и нетехнических факторов, среди которых стоимость, надежность оборудования, дилерское обслуживание, гибкость обновлений и совместимость с другими системами. Но при этом обязательно должна учитываться и ожидаемая загрузка системы в часы пиковой нагрузки.

Излом

Наиболее интересной частью кривой производительности является ее «излом». Интуитивно можно сказать, что «излом кривой» – это то значение коэффициента использования, начиная с которого кривая начинает расти *вверх* быстрее, чем уходить *вправо* (вертикальная координата растет быстрее горизонтальной). К сожалению, такое определение нам не подходит, т. к. местоположение такой точки меняется в зависимости от того, как вычерчивается кривая. Обратимся к рис. 9.15. Приведенные схемы представляют собой два разных графика одной и той же кривой времени отклика. Графики отличаются только масштабом по вертикальной оси. Разница кажется непомерной: очевидно, что «эти две системы имеют различные изломы». Но вспомните, что речь

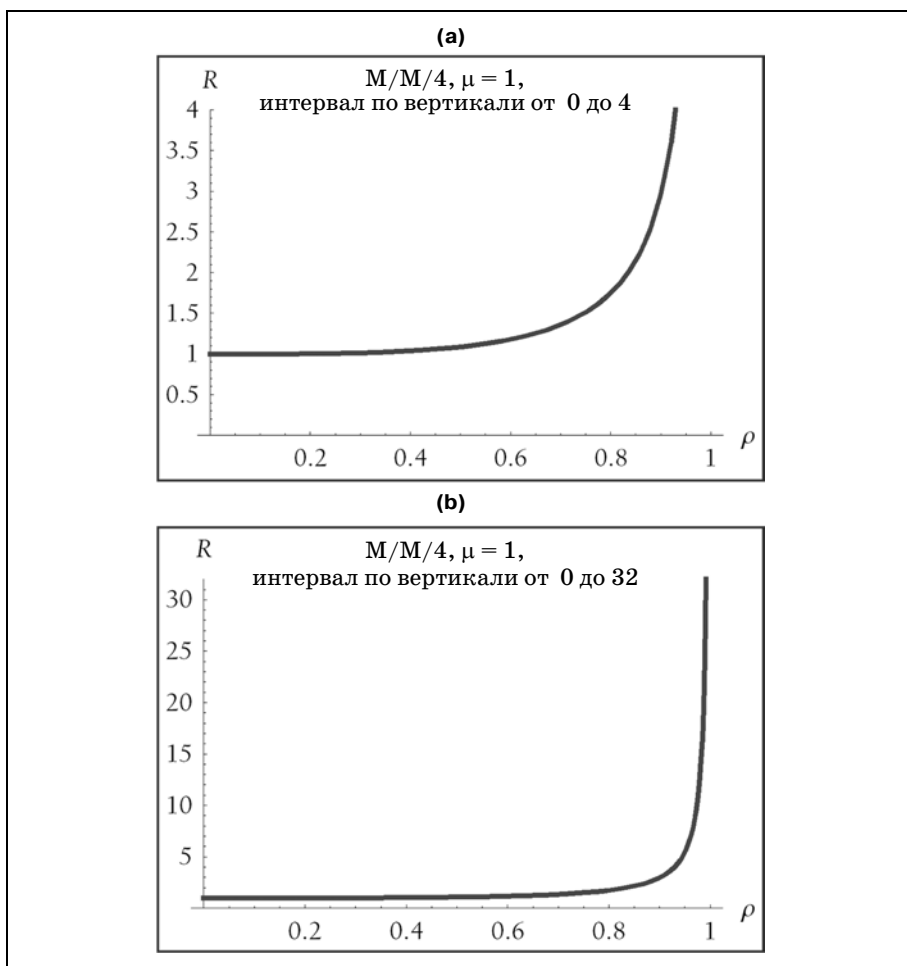


Рис. 9.15. Одна и та же кривая представлена в разных масштабах по вертикальной оси. На рисунке кажется, что излом на верхней кривой происходит в точке $\rho \approx 0,8$, а на нижней кривой – в точке $\rho = 0,9$

на самом деле идет не о двух системах, а о двух графиках *одной и той же* системы, изображенных в разных масштабах по вертикальной оси.

Итак, визуальное определение местоположения «излома» – это слишком ненадежный метод. Понятно, что настоящее определение «излома» системы не может зависеть от выбора единицы измерения осей при построении графика кривой времени отклика.



Один автор совершенно неправильно излагает свои взгляды по этому вопросу в статье «Performance Management: Myths & Facts» [Millsap (1999) 8] («Управление производительностью. Мифы и реальность», Oracle Magazine RE, август 2001). Факты,

которые он приводит, достоверны: кривая времени отклика *действительно* имеет излом, и точка излома *действительно* сдвигается вправо при увеличении количества каналов обслуживания.

Однако то определение, которое автор приводит для *излома* (значение, в котором вертикальная координата кривой начинает расти быстрее, чем горизонтальная), к сожалению, основывается лишь на зрительном восприятии, которое только что было развенчано. Тот способ, которым он предлагает находить точку излома (вычисление такого значения ρ , для которого $\partial R/\partial \rho = 1$ и т. д.), абсолютно ошибочен. Я уверен, что автор очень сожалел бы о своей ошибке, если бы знал о ней.

Более подходящее техническое определение «излома» выглядит следующим образом: излом кривой времени отклика имеет место при таком значении коэффициента использования ρ^* , при котором отношение R/ρ достигает своего минимума. Графически это означает, что излом приходится на такое значение коэффициента использования (ρ), при котором прямая, проведенная через начало координат, пересекает кривую времени отклика ровно в одной точке (т. е. касается ее), как это показано на рис. 9.16. Многие аналитики считают значение ρ^* оптимальным коэффициентом использования для системы $M/M/t$, т. к....

...обычно требуется одновременно минимизировать R (для удовлетворения пользователей) и максимизировать ρ (для того чтобы разделить стоимость ресурсов между многими пользователями) [Vernon(2001)].

Как вы видели, при значениях коэффициента использования, расположенных слева от излома ($\rho < \rho^*$), попусту растрачивается пропуск-

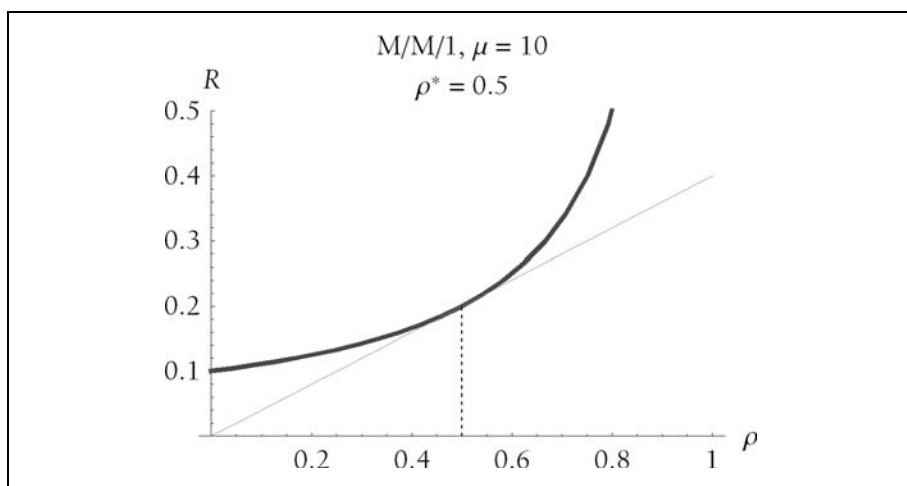


Рис. 9.16. Излом – это значение коэффициента использования ρ^* , при котором отношение R/ρ принимает наименьшее значение. Или, что эквивалентно, излом – это значение ρ , при котором прямая, проведенная через начало координат, является касательной к кривой времени отклика

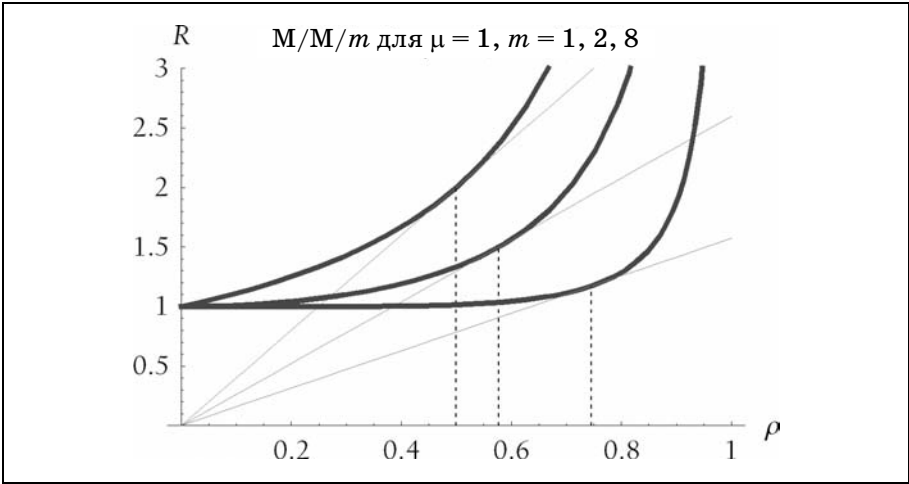


Рис. 9.17. Значение излома перемещается вправо по оси коэффициента использования (ρ) по мере увеличения количества параллельных каналов обслуживания (t)

ная способность системы. Ведь при низких значениях коэффициента использования можно увеличить нагрузку без существенного ухудшения времени отклика. Для значений, расположенных справа от излома ($\rho > \rho^*$), значительное ухудшение времени отклика для пользователей может возникнуть даже при малейшем изменении нагрузки.

Положение точки излома для системы М/М/ t зависит исключительно от значения t – количества параллельных каналов обслуживания. Мы уже знаем, что добавление параллельных каналов обслуживания позволяет системе работать с более высоким коэффициентом использования без существенного ухудшения времени отклика. На рис. 9.17 показано, как точка излома сдвигается вправо при увеличении значения t .

Положение излома для фиксированного значения t постоянно и не зависит от скорости обслуживания, что видно на рис. 9.18. На этом рисунке изменение скорости обслуживания (t) приводит к изменению времени отклика (обратите внимание на разные отметки на оси R), но вид кривой производительности и положение точки излома совпадают для всех систем М/М/1. Так же и все кривые М/М/2 имеют одну и ту же форму и одно и то же значение излома. Все кривые М/М/3 имеют одну форму и одну точку излома. И так далее.



Говорят, что две кривые f_1 и f_2 имеют *одинаковую форму*, если $f_1(x) = k f_2(x)$ для каждого значения x и некоторой постоянной k . То есть две кривые имеют одинаковую форму, если их можно сделать идентичными за счет увеличения для одной из них масштаба по вертикальной оси в некоторое постоянное количество раз.

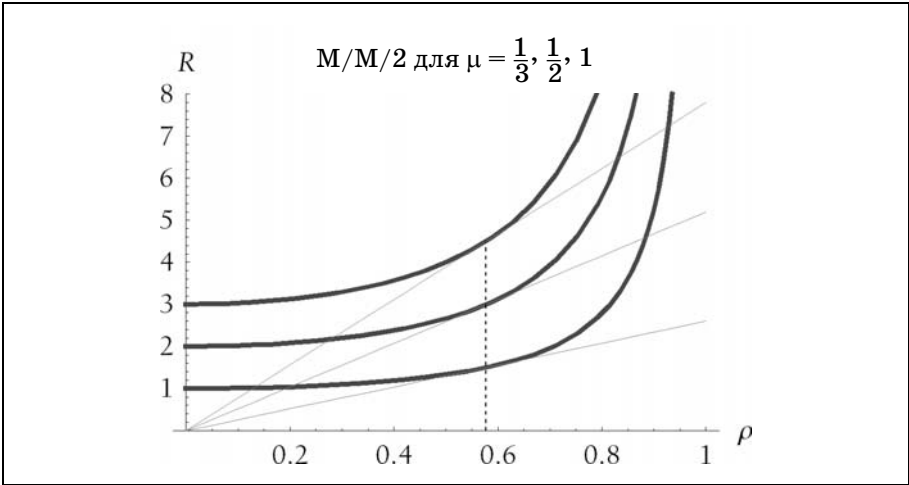


Рис. 9.18. Для определенного t положение излома постоянно и не зависит от значения μ . При изменении скорости обслуживания меняется вертикальное положение кривой, но ни на ее форму, ни на положение точки излома это не влияет

Все системы $M/M/t$ с фиксированным значением t имеют один и тот же коэффициент использования в точке излома, поэтому данные о положении излома для интересных значений t можно свести в одну таблицу (табл. 9.3).

Таблица 9.3. Значение коэффициента использования ρ^* , при котором в системе $M/M/t$ имеет место излом, зависит от значения t

t	ρ^*	t	ρ^*
1	0,5	32	0,86169
2	0,57735	40	0,875284
3	0,628831	48	0,885473
4	0,665006	56	0,893482
5	0,692095	64	0,899995
6	0,713351	80	0,910052
7	0,730612	96	0,917553
8	0,744997	112	0,923427
16	0,810695	128	0,928191
24	0,842207		

Данные табл. 9.4 позволяют чуть глубже познакомиться с производительностью систем $M/M/t$. В ней показано, как ухудшается среднее время отклика с увеличением коэффициента использования в различных системах $M/M/t$. Например, если в незагруженной системе $M/M/1$

среднее время отклика $R = S$ секунд, то при коэффициенте использования 50% среднее время отклика ухудшится до $R = 2S$ секунд (т. е. в два раза по сравнению с незагруженной системой). Когда значение коэффициента достигнет 75%, время отклика возрастет в четыре раза по отношению к незагруженной системе. Если же коэффициент использования достигнет 90%, то время отклика возрастет в *десять* раз! В системе М/М/8 время отклика возрастет до $R = 4S$ секунд, только когда коэффициент использования достигнет значения 96,3169%. Эти цифры подтверждают наши интуитивные предположения о влиянии дополнительных каналов обслуживания на масштабируемость времени отклика.

Таблица 9.4. Значения коэффициента использования для систем М/М/т, при которых время отклика в k раз превышает время обслуживания (т. е. при которых $R = kS$)

k	t				
	1	2	4	8	16
1	0,	0,	0,	0,	0,
2	0,5	0,707107	0,834855	0,909166	0,950986
3	0,666667	0,816497	0,901222	0,947673	0,97263
4	0,75	0,866025	0,929336	0,963169	0,980984
5	0,8	0,894427	0,944954	0,971569	0,985426
6	0,833333	0,912871	0,954907	0,976844	0,988184
7	0,857143	0,92582	0,961807	0,980467	0,990064
8	0,875	0,935414	0,966874	0,983109	0,991427
9	0,888889	0,942809	0,970753	0,985121	0,992462
10	0,9	0,948683	0,973818	0,986704	0,993273

Колебания времени отклика

Пользователи, работающие в оперативном режиме, выполняющие одну и ту же задачу вновь и вновь, чрезвычайно чувствительны к колебаниям времени отклика. Если бы у них был выбор, то большинство, вероятно, предпочло бы иметь постоянное двухсекундное время отклика онлайновой формы, чем такое же, в среднем двухсекундное, время отклика при 75% исполнений в течение одной секунды и 25% исполнений в течение семи секунд. Возможно, вы обратили внимание на то, что в системах с низкой загрузкой время отклика более или менее стабильно, но в сильно загруженных системах может очень сильно изменяться. Модель массового обслуживания объясняет, почему это происходит.

На рис. 9.19 изображено очень небольшое ухудшение времени отклика (от R_1 к R_2) многоканальной системы (т. е. $t > 1$) при увеличении коэффициента использования от значения ρ_1 до ρ_2 . Обратите внимание, что значения R_1 и R_2 настолько близки друг к другу, что перекры-

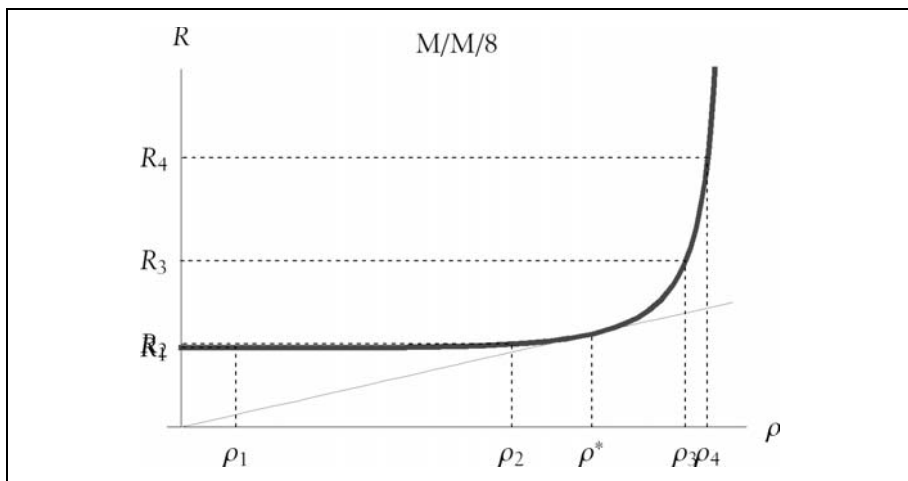


Рис. 9.19. Слева от излома время отклика нечувствительно даже к значительным изменениям коэффициента использования, но справа от излома даже минимальное его изменение приводит к значительному изменению времени отклика

ваются соответствующие им метки на вертикальной оси. Однако справа от излома даже очень небольшое изменение коэффициента использования от ρ_3 к ρ_4 приводит к серьезному ухудшению времени отклика (от R_3 к R_4).

Из предыдущего раздела нам уже известно, что многоканальная система может обеспечить большую устойчивость времени отклика при высокой интенсивности поступления запросов, чем одноканальная система. Отличная масштабируемость, показанная на рис. 9.19, служит еще одной иллюстрацией этого явления. Однако пропускная способность даже самой большой многоканальной системы не безгранична, и когда интенсивность поступления запросов превысит этот ограниченный предел мощности, производительность быстро ухудшится.

А вот производительность одноканальных систем уменьшается более равномерно, как показано на рис. 9.20. На рисунке видно, что время отклика быстрее ухудшается справа от точки излома. Однако в системах с небольшим количеством параллельных каналов (т. е. с небольшим значением m), такое ухудшение более равномерно распределено по всему диапазону значений коэффициента использования, чем в системах с большим количеством параллельных каналов обслуживания (большими значениями m).

Одноканальная система массового обслуживания имеет меньшую масштабируемость, чем аналогично загруженная система с несколькими каналами обслуживания. То есть ее точка излома находится ближе к началу области утилизации. Но в любом случае излом присутствует в любой системе. В хорошо масштабируемых многоканальных систе-

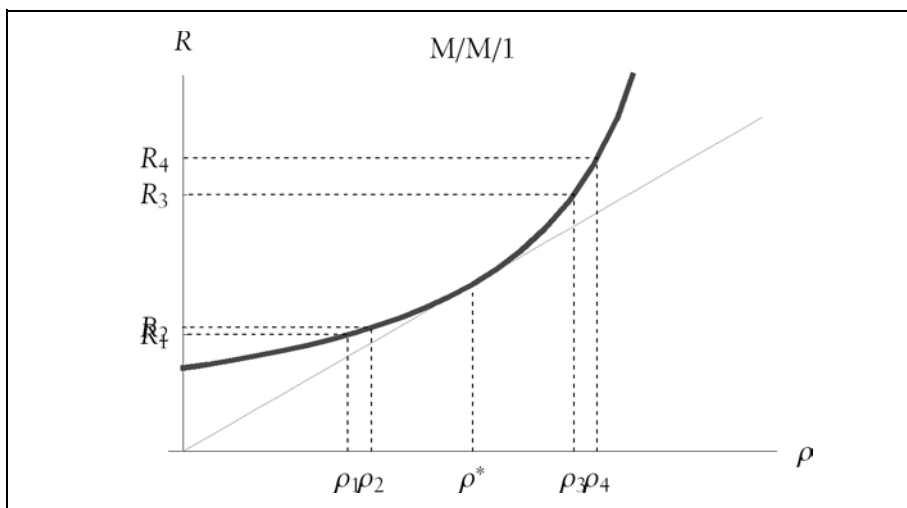


Рис. 9.20. *Время отклика ухудшается более плавно по всей оси коэффициента использования в одноканальных системах и в системах с небольшим t*

мах при переходе нагрузки за точку излома пользователи системы начинают ощущать значительные колебания времени отклика.

Чувствительность к параметрам

Электронные таблицы (Microsoft Excel и подобные) позволяют проверить различные ситуации типа «что, если» с такой быстротой, о которой Агнер Эрланг мог только мечтать. Такие проверки могут избавить от бесконечных нервоврепек при общении с реальными конечными пользователями. Меня много раз просили определить, почему потерпел неудачу проект повышения производительности. В ряде случаев модель массового обслуживания М/М/т позволила быстро и точно объяснить, почему усовершенствование оборудования не привело к ожидаемому положительному воздействию на общую производительность. Модель массового обслуживания часто позволяла привести аргументы такого вида:

...Поэтому переход на процессор, на 100% более быстрый, чем прежний, не привел к ожидаемому повышению производительности. Более того, даже если бы вы могли вчетверо увеличить количество этих более быстрых процессоров, то все равно не достигли бы ожидаемого повышения производительности. Единственным способом достижения требуемой производительности является снижение частоты использования данной функции или же уменьшение длины кода, выполняемого при ее вызове.

Применение модели массового обслуживания на более ранних этапах в этих ситуациях могло предотвратить катастрофу, которой и было вызвано мое присутствие.

Использование модели массового обслуживания $M/M/t$ в электронных таблицах учит нас тому, что каждый входной и каждый выходной параметр модели может служить предметом переговоров. Для того чтобы оптимизировать производительность системы, необходимо понимать, как параметры связаны друг с другом и какое экономическое воздействие оказывает выбор значения каждого из параметров. Рассмотрим перечень таких параметров и возможные варианты выбора:

λ

Интенсивность поступления запросов λ – это параметр, который может стать мощным рычагом в руках аналитика по производительности. Многие аналитики вообще не рассматривают возможность обсуждения рабочей нагрузки с конечными пользователями, считая, что объем работы, который требуется от системы, жестко фиксирован. Но во многих случаях основной причиной ухудшения производительности системы является именно бесполезная нагрузка, например:

- Приложение по расписанию отправляет уведомления по электронной почте всем пользователям каждые две минуты, при этом каждый из них читает уведомления лишь дважды в день. То есть интенсивность поступления уведомлений можно было бы снизить в 120 раз, с 30 в час до 0,25 в час.
- Система с восемью процессорами чрезвычайно замедляет работу с пользователями в период с 14:00 до 15:00. Основным элементом нагрузки для этого промежутка времени является набор из 16 отчетов (пакетных заданий), каждый из которых требует около 30 минут процессорного времени. Эти отчеты никто не будет читать до 8:00 следующего дня. Таким образом, если запланировать запуск 16 пакетных заданий в полночь (вместо пиковых рабочих часов), то интенсивность поступления запросов на обслуживание процессором уменьшится настолько, что будет сэкономлено около восьми часов процессорного времени в период с 14:00 до 15:00.
- Для того чтобы определить, сходится ли дебет с кредитом в различных бухгалтерских книгах, бухгалтеры каждый день формируют 14 отчетов о предварительном балансе. Каждый 200-страничный отчет требует выполнения 72 000 000 вызовов логического чтения Oracle (LIO), что занимает около 30 минут процессорного времени. К сожалению, пользователи не знают, что приложение поддерживает веб-страницу, на которой можно получить всю необходимую информацию о расхождениях в счетах, выполнив менее 100 операций LIO. То есть интенсивность поступления LIO для этих бизнес-операций можно снизить с миллиарда в день до приблизительно 1400.

Как быстрее всего сделать что-то? Надо найти способ не делать этого вообще. Часто оказывается, что этой цели вполне можно достичь, не потеряв никакой функциональности для бизнеса.

r_{\max}

Вполне возможно, что пользователи согласятся пойти на компромисс в том, что касается кратковременных повышений времени отклика, в особенности, если небольшие уступки могут сэкономить компании немалые деньги. Однако попытки убедить пользователей в том, что им следует смириться с низкой производительностью системы, вряд ли будут иметь успех. Так что если только это не жизненно необходимо, воздержитесь от предложения снизить r_{\max} .

 p

Как и в случае с r_{\max} , пользователи, скорее всего, не захотят обсуждать p (долю времен отклика, которые не должны превышать r_{\max}).

Коэффициент использования – не самоцель

Я с неохотой говорю о «хорошем коэффициенте использования процессора», потому что этот коэффициент – лишь косвенный показатель метрики, которая на самом деле *должна* нас интересовать – времени отклика. Однако, избежав некоторых заведомо *плохих* значений коэффициента использования, можно предотвратить проблемы с производительностью.

- В прикладных системах, ориентированных только на пакетную обработку, коэффициент использования процессора меньше 100%, – это плохо, если в очереди заданий что-то есть. Цель пользователя системы пакетной обработки состоит в достижении *максимальной пропускной способности*. Если какое-то задание ожидает своей очереди, то каждая секунда простоя процессора – это потерянное время, которое уже никогда не вернуть. Но будьте осторожны: длительная стопроцентная загрузка процессора часто приводит к «пробуксовке» планировщика операционной системы, что может *снизить* производительность.
- В исключительно интерактивных прикладных системах плохи те значения коэффициента загруженности процессора, которые долгое время остаются справа от точки излома. Цель пользователя такой системы – *минимизация времени отклика*. Когда коэффициент загруженности процессора превышает значение излома кривой времени отклика, колебания времени отклика становятся недопустимыми. Оставив в системе резервную мощность, системный инженер фактически приобретает стабильность времени отклика.
- В системах, где присутствует и пакетная, и интерактивная нагрузка, задача будет более сложной, т. к. цели пользователей противостоят друг другу. Поэтому в таких системах важно правильно определить *приоритеты*. Если интерактивное время отклика важнее, то следует убедиться в том, что коэффициент загруженности процессора не заходит слишком далеко вправо за точку излома. Если же важнее производительность пакетной обработки, то нет смысла терять значительную мощность процессора, чтобы обеспечить стабильность времени отклика для менее важных пользователей.

Способ определения оптимального соотношения пакетной и интерактивной нагрузки описан в [Millsap (2000b)].

q

Несмотря на то, что число q – изменяемый параметр системы массового обслуживания $M/M/m$, масштабирование прикладной системы Oracle для работы на нескольких серверах базы данных (даже в случае применения Oracle9i Real Application Clusters – RAC), технологически это сложное мероприятие, которое практически наверняка обойдется дороже, чем конфигурация, в которой база данных работает на одном хосте.

 m

Количество параллельных каналов обслуживания в системе это обсуждаемый параметр, зависящий от физических ограничений и ограничений масштабируемости, накладываемых главным образом операционной системой. Например, имеющаяся система может требовать, чтобы процессоры устанавливались в количествах, кратных двум, вплоть до 32, но мощность системы с 32 процессорами может оказаться такой же, что и у воображаемой системы с 24 идеально масштабируемыми процессорами. Такие данные можно получить в ходе проведения контрольных испытаний и бесед со специалистами поставщиков оборудования.

 μ

Скорость обслуживания μ – это параметр, возможность изменения которого является мощнейшим оружием аналитика по производительности. Первым побуждением множества аналитиков бывает улучшение (увеличение) μ за счет более быстрого аппаратного обеспечения. Например, установка процессоров, на 20% более быстрых, чем старые, должна привести к повышению скорости обслуживания процессором приблизительно на ту же величину, что может быть чрезвычайно полезно для приложений, ограниченных мощностью процессора.

Но есть еще одна очень важная вещь, о которой многие забывают – увеличения скорости обслуживания можно достичь за счет уменьшения объема кода, необходимого для выполнения конкретной бизнес-функции. В прикладных системах Oracle аналитики зачастую могут добиться впечатляющего сокращения кода за счет *оптимизации SQL*. Оптимизация SQL требует работы с сочетаниями определений схем, статистиками базы данных или экземпляра, параметрами конфигурации, поведением оптимизатора запросов или с исходным текстом приложения с тем, чтобы получить аналогичные выходные данные посредством выполнения меньшего количества инструкций на сервере базы данных.

Сокращение кода имеет ряд преимуществ по сравнению с модернизацией оборудования, а именно:

Стоимость

Усовершенствование оборудования часто означает не только увеличение стоимости самого оборудования, но и увеличение стоимости

поддержки и лицензионных выплат на программное обеспечение. Например, некоторые программные лицензии для систем с $(n + 1)$ процессорами стоят дороже, чем для систем с n процессорами. Уменьшение количества кода для неэффективных команд SQL обычно требует не более нескольких часов работы для каждой команды.

Эффект

Сокращение кода часто оказывается более мощным средством повышения производительности, чем усовершенствование оборудования. Например, удвоение скорости процессора возможно не чаще чем раз в два года. Что же касается сокращения кода, то часто удастся уменьшить его длину в 100 или более раз всего за несколько часов работы.

Риск

Сокращение кода практически не приводит к неожиданным негативным побочным эффектам, которые могут сопутствовать модернизации оборудования. Возможным побочным эффектом повышения пропускной способности является *ухудшение* производительности программ, которым не хватало ресурсов не того устройства, для которого была проведена модернизация [Millsap (1999)].

Даже если недолго поработать с моделью массового обслуживания, становится понятно, почему избавление от ненужной нагрузки приводит к впечатляющему воздействию на производительность всей системы. Существуют два способа устранения нагрузки: исключение бизнес-функций, которые необоснованно увеличивают затраты, и избавление от ненужных фрагментов кода. Оба эти способа на самом деле позволяют решить одну задачу – избавиться от ненужных затрат на разных уровнях технологического стека.

Использование М/М/т: пример с решением

Давайте на примере посмотрим, как можно использовать модель и интерпретировать полученные результаты. Рассмотрим поэтапно решение поставленной ниже задачи, применяя модель системы массового обслуживания М/М/т (рабочую книгу Microsoft Excel), которая доступна на нашем сайте <http://www.hotshots.com>. Задача по сути своей проста, но для ее решения требуется понимание теории массового обслуживания.

Задача формулируется следующим образом:

На выполнение важной команды SQL расходуется 0,49 секунды процессорного времени. Предположим, что в период пиковой загрузки системы каждый из 100 пользователей будет выполнять эту команду с частотой четыре раза в минуту. Сервер, работающий под Linux, допускает установку до 16 процессоров. Сколько процессоров потребуется данному серверу, если необходимо обеспечить время отклика не более одной секунды по крайней мере для 95% выполнений команды пользователями в период пиковой загрузки?

Попробуем ее решить.

Проверка на применимость модели М/М/т

В первую очередь необходимо проверить, можно ли применить модель массового обслуживания М/М/т/∞/FCFS (или, для краткости, М/М/т) к исследуемой нами системе:

М/М/т/∞/FCFS (экспоненциальное время между поступлениями запросов)

Если существует возможность измерить интервал между поступлениями запросов, то первым делом необходимо проверить, подчиняются ли такие интервалы экспоненциальному распределению. Сделаем это с помощью программы из примера 9.4. Если окажется, что время между поступлениями запросов распределено не экспоненциально, то следует рассмотреть более короткий временной промежуток. Например, в случае I (который воспроизведен в табл. 9.5) из уже рассмотренного примера с посетителями ресторана распределение времени между приходами клиентов отличается от экспоненциального на промежутке с 11:30 до 13:30. Но вот в промежутке с 12:00 до 12:15 оно с большой вероятностью имеет экспоненциальное распределение.

Таблица 9.5. Два существенно разных сценария, в обоих из которых среднее значение интервала между событиями равно 30 секундам

Интервалы времени	Количество приходов	
	Случай I	Случай II
11:30–11:45	0	34
11:45–12:00	0	28
12:00–12:15	240	31
12:15–12:30	0	37
12:30–12:45	0	24
12:45–13:00	0	30
13:00–13:15	0	32
13:15–13:30	0	24
Среднее для 15-минутных интервалов	30	30

Если реальное время между поступлениями запросов измерить невозможно (например, система еще не спроектирована), то придется подключить воображение. Почти для любой ситуации можно создать такое временное окно, в котором время между поступлениями запросов будет с большой вероятностью подчиняться экспоненциальному распределению (или, что то же самое, интенсивность запросов будет иметь распределение Пуассона). В нашем примере будем считать, что измерения подтверждают распределение поступлений запросов по закону Пуассона.

М/М/т/∞/FCFS (экспоненциальное время обслуживания)

Необходимо убедиться в том, что экспоненциально распределено время обслуживания. И здесь на помощь приходит программа из примера 9.4 (или какое-то другое средство). Если время обслуживания распределено не экспоненциально, то следует переопределить исследуемую единицу обслуживания. Допустим, вы рассматриваете в качестве единицы обслуживания отчеты, при этом время их обслуживания может составлять от 0,2 до 1392,7 секунды. Тогда время обслуживания, вероятнее всего, не будет распределено экспоненциально. Выберите в качестве единицы измерения определенный тип отчетов, время обслуживания которого не подвержено таким значительным колебаниям. Или же выберите более мелкую единицу измерения, например, для этой цели подойдут операции LIO.

В нашем примере будем считать, что измерения, проведенные в тестируемой системе, подтвердили экспоненциальное распределение времени обслуживания для «важной команды SQL».

М/М/т/∞/FCFS (т однородных параллельных независимых каналов обслуживания)

В формулировке задачи сказано, что в нашем случае в роли канала обслуживания выступает процессор компьютера, работающего под управлением операционной системы Linux. Linux полностью поддерживает симметричную многопроцессорную обработку (SMP), так что нам известно, что наши каналы обслуживания однородны, параллельны и независимы. Некоторые конфигурации операционных систем (например, использующие привязку задач к процессорам) нарушают ограничение на однородность процессоров. Далее вы узнаете о том, что в случае применения данной модели необходимо также учитывать несовершенство масштабирования. И наконец, мы используем модель М/М/т для прогнозирования производительности системы с более чем t процессорами.

М/М/т/∞/FCFS (отсутствие ограничений на длину очереди)

На длину очереди к процессору Linux не налагается никаких ограничений, поэтому можно не беспокоиться о составляющей «∞» определения М/М/т.

М/М/т/∞/FCFS (обслуживание в порядке поступления)

Стандартная политика планирования Linux близка к обслуживанию в порядке поступления. Несмотря на то, что алгоритм планирования разрешает использование приоритетов для процессов и выбор одной из нескольких политик (FCFS или циклическое обслуживание (round-robin)), модель М/М/т показала себя как пригодная для прогнозирования производительности систем Linux.

Вычисление необходимого количества процессоров

Наша задача состоит в том, чтобы определить количество процессоров, необходимое для достижения некоторого конкретного значения производительности. Условия задачи можно записать в рабочую книгу Excel *M/M/m*, как показано в табл. 9.6. Значения из столбца «Значение» следует внести в желтые ячейки листа «Multiserver Model» книги *MMm.xls*.

Таблица 9.6. Входные параметры модели массового обслуживания

Имя	Значение (value _a)	Описание
<i>jobunit</i>	stmt	Исследуемая единица работы определена в условии задачи как «важная команда SQL».
<i>timeunit</i>	sec	Допустимое время отклика и время обслуживания процессором команды SQL измеряются в секундах.
<i>queueunit</i>	system	В условии задачи говорится, что мы ищем количество процессоров, которое должно быть установлено в одной системе.
<i>serverunit</i>	CPU	Единицей обслуживания в нашей задаче является процессор.
λ	=100*4/60	«Предполагается, что каждый из 100 пользователей будет выполнять эту команду с частотой четыре раза в минуту в период пиковой загрузки системы.»
r_{\max}	1	Нашей целью является достижение значения времени отклика, меньшего чем 1 секунда («...обеспечить время отклика не более одной секунды ...»).
q	1	По условию задачи мы оцениваем работу одной системы.
m	1	Количество процессоров в системе – это то значение, которое мы ищем. В эту ячейку можно внести значение 1, т. к. мы скоро найдем подходящее значение.
μ	=1/0.49	«На выполнение важной команды SQL расходуется 0,49 секунды процессорного времени». Поэтому скорость обслуживания равна 1/0,49 команд в секунду.

Введя эти значения в книгу Excel, мы увидим, что конфигурация с одним процессором ($m = 1$) не может обработать исследуемую нагрузку (рис. 9.21).

Модель показывает, что если снабдить систему всего одним процессором, то при поступлении в систему 6,67 команды в секунду процессору придется работать с коэффициентом загрузки 326,7%. Естест-

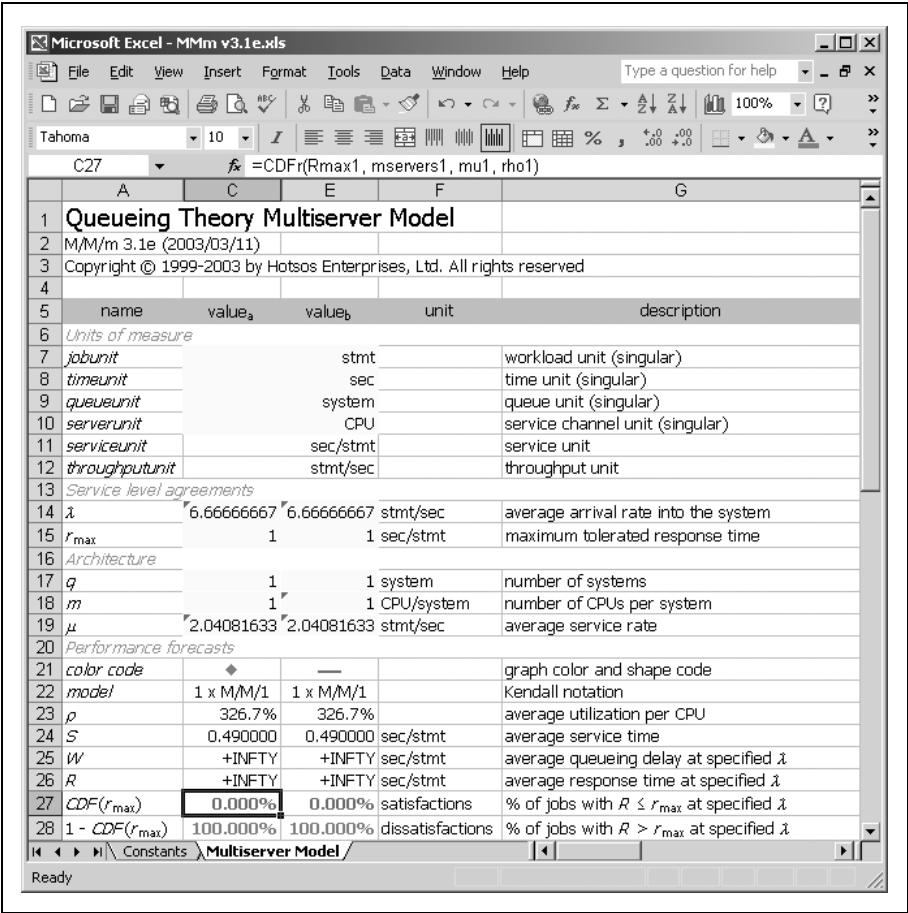


Рис. 9.21. Система с одним процессором сможет обеспечить время отклика, меньшее одной секунды, в 0% случаев при условии, что $\lambda = 6,667$ команд/сек, а $\mu = 2,041$ команд/сек. Заметьте, что высокая интенсивность поступлений запросов и сравнительно низкая скорость обслуживания приводят к тому, что коэффициент загрузки процессора достигает до 326,7% – такое значение возможно лишь в теории

венно, процессор не может быть загружен более чем на 100%. В реальности в такой системе коэффициент загрузки всегда будет равен 100%. Команды будут поступать быстрее, чем их можно будет обрабатывать, в связи с чем очередь будет постоянно и бесконечно расти. Но даже такой результат дает нам подсказку: для того чтобы у системы был хоть какой-то шанс справиться с заданной нагрузкой, в ней должно быть как минимум четыре процессора.

Электронная книга *MMm* для удобства содержит два столбца (*value_a* и *value_b*), позволяющие построчно сравнивать характеристики двух систем. Результат применения модели для $m = 4$ показан на рис. 9.22.

	A	C	E	F	G
1	Queueing Theory Multiserver Model				
2	M/M/m 3.1e (2003/03/11)				
3	Copyright © 1999-2003 by Hotsos Enterprises, Ltd. All rights reserved				
4					
5	name	value _a	value _b	unit	description
6	<i>Units of measure</i>				
7	jobunit		stmt		workload unit (singular)
8	timeunit		sec		time unit (singular)
9	queueunit		system		queue unit (singular)
10	serverunit		CPU		service channel unit (singular)
11	serviceunit		sec/stmt		service unit
12	throughputunit		stmt/sec		throughput unit
13	<i>Service level agreements</i>				
14	λ	6.66666667	6.66666667	stmt/sec	average arrival rate into the system
15	r_{\max}	1	1	sec/stmt	maximum tolerated response time
16	<i>Architecture</i>				
17	q	1	1	system	number of systems
18	m	1	4	CPU/system	number of CPUs per system
19	μ	2.04081633	2.04081633	stmt/sec	average service rate
20	<i>Performance forecasts</i>				
21	color code	◆	—		graph color and shape code
22	model	1 x M/M/1	1 x M/M/4		Kendall notation
23	ρ	326.7%	81.7%		average utilization per CPU
24	S	0.490000	0.490000	sec/stmt	average service time
25	W	+INFTY	0.418787	sec/stmt	average queueing delay at specified λ
26	R	+INFTY	0.908787	sec/stmt	average response time at specified λ
27	$CDF(r_{\max})$	0.000%	64.922%	satisfactions	% of jobs with $R \leq r_{\max}$ at specified λ
28	$1 - CDF(r_{\max})$	100.000%	35.078%	dissatisfactions	% of jobs with $R > r_{\max}$ at specified λ

Рис. 9.22. Использование 4-х процессоров при идеальной масштабируемости приводит к тому, что система соответствует требованиям к производительности в 64,9% случаев

Идеальная четырехпроцессорная система обеспечивает среднее время отклика 0,908787 секунд на команду. Однако радоваться еще рано, ведь в данной конфигурации время отклика меньше одной секунды всего в 64,922% случаев ее выполнения. А нам нужно, чтобы время отклика системы составляло менее одной секунды в 95% случаев выполнения при пиковой нагрузке.

Рассмотрим наилучшую теоретически достижимую производительность, которая достигается установкой максимально допустимого для компьютера количества процессоров – 16 (рис. 9.23). Как видите, у нас очень серьезные проблемы. Даже если количество процессоров $m = 16$, невозможно удовлетворить ожидания пользователей в отношении времени отклика, составляющего менее одной секунды более чем в 87%

Microsoft Excel - MMm v3.1e.xls

File Edit View Insert Format Tools Data Window Help Type a question for help

Tahoma 10

E27 $=CDF(r_{max2}, mservers2, mu2, rho2)$

	A	C	E	F	G
1	Queueing Theory Multiserver Model				
2	M/M/m 3.1e (2003/03/11)				
3	Copyright © 1999-2003 by Hotso's Enterprises, Ltd. All rights reserved				
4					
5	name	value _a	value _b	unit	description
6	<i>Units of measure</i>				
7	jobunit		stmt		workload unit (singular)
8	timeunit		sec		time unit (singular)
9	queueunit		system		queue unit (singular)
10	serverunit		CPU		service channel unit (singular)
11	serviceunit		sec/stmt		service unit
12	throughputunit		stmt/sec		throughput unit
13	<i>Service level agreements</i>				
14	λ	6.66666667	6.66666667	stmt/sec	average arrival rate into the system
15	r_{max}	1	1	sec/stmt	maximum tolerated response time
16	<i>Architecture</i>				
17	q	1	1	system	number of systems
18	m	4	16	CPU/system	number of CPUs per system
19	μ	2.04081633	2.04081633	stmt/sec	average service rate
20	<i>Performance forecasts</i>				
21	color code	◆	—		graph color and shape code
22	model	1 x M/M/4	1 x M/M/16		Kendall notation
23	ρ	81.7%	20.4%		average utilization per CPU
24	S	0.490000	0.490000	sec/stmt	average service time
25	W	0.418787	0.000000	sec/stmt	average queueing delay at specified λ
26	R	0.908787	0.490000	sec/stmt	average response time at specified λ
27	$CDF(r_{max})$	64.922%	87.008%	satisfactions	% of jobs with $R \leq r_{max}$ at specified λ
28	$1 - CDF(r_{max})$	35.078%	12.992%	dissatisfactions	% of jobs with $R > r_{max}$ at specified λ

Ready

Рис. 9.23. Даже при наличии 16 процессоров и идеальном масштабировании система соответствует исходным требованиям к производительности лишь в 87% случаев

выполнений команды. Можете протестировать модель самостоятельно: даже если бы мы могли довести конфигурацию до $m = 1000$, такая система *все равно* не смогла бы обеспечить запрошенную производительность в более чем в 87% случаев выполнения команды.

О чем говорит оптимистическая модель

Имейте в виду, что модель М/М/м позволяет строить гипотезы относительно систем, существование которых в реальности невозможно. Например, можно смоделировать идеально масштабируемую систему Linux с 1000 процессоров или даже идеально масштабируемую систему MS Windows NT с 4 процессорами. Вне зависимости от того, насколько абсурдными будут указанные входные значения, модель массового обслуживания честно расскажет о том, как будет работать идеально мас-

штабируемая гипотетическая система в данной конфигурации. На самом деле модель даже не знает, что именно моделируется: компьютерные системы, бакалейные магазины или же билетные кассы. Аналитик решает, возможно ли реальное создание гипотетической системы. Модель $M/M/t$ теории массового обслуживания (применяемая так, как здесь описано) является *оптимистической* в том смысле, что не учитывает никакие налагаемые реальной жизнью ограничения на масштабируемость, за исключением эффекта организации очередей.

Важно четко осознавать вышесказанное. Если данная модель теории массового обслуживания говорит о том, что нечто *может* быть сделано, вполне вероятно, что вам, тем не менее, *не удастся* это сделать. В конце концов ничто не мешает сказать модели, что можно втиснуть в компьютер под Linux 1000 процессоров, и все они будут без проблем работать на полную мощность. Однако если модель теории массового обслуживания уведомляет вас о том, что что-то сделать *невозможно*, этому совету можно полностью доверять. Если оптимистичная модель утверждает, что что-то реализовать невозможно, значит, это *так и есть*.



Модель $M/M/t$, реализованная в моей электронной книге Excel, *оптимистична*, т. к. не учитывает никаких препятствий для наращивания системы, возможных в реальной жизни, за исключением эффекта очередей. Следовательно, сообщение модели о том, что какое-то явление *возможно*, не означает, что вам обязательно удастся его реализовать. Однако утверждения системы о том, что нечто *невозможно*, является достаточным аргументом невозможности данного явления (при условии, что исходные данные верны).

Наша модель сообщила нам, что вне зависимости от того, какое количество процессоров мы добавим в свою гипотетическую конфигурацию, нам никогда не удастся добиться того, чтобы 95% выполнений завершались менее, чем через одну секунду. Конечно, это не очень хорошие новости для нашего проекта, но, согласитесь, гораздо лучше узнать горькую правду при помощи недорогой модели массового обслуживания, чем бесцельно вложить в проект огромные средства и стать свидетелем его краха.

Обсуждение параметров с клиентом

Что же нам делать теперь, когда мы узнали, что проект обречен? Менее очевидное достоинство теории массового обслуживания заключается в том, что она точно указывает, какие изменения системы (моделируемые изменением параметров $M/M/t$) могут оказать положительное воздействие на производительность. Параметры, значения которых допускают возможность обсуждения, были перечислены в разделе «Чувствительность к параметрам». В нашем примере следует проанализировать следующие допускающие изменение параметры:

Количество систем q и количество процессоров в системе t

Для начала исследуем интересный результат моделирования, чтобы понять, почему добавление процессоров не приводит к возрастанию интегральной функции распределения времени отклика более, чем до 87%. Модель показывает, что даже сто систем по сто процессоров в каждой не могут предложить удовлетворяющего пользователей времени отклика в более, чем 87% случаев.

Причина в том, что время обслуживания в модели M/M/ m – это случайная величина. Несмотря на то, что *среднее* время обслуживания постоянно, реальное время обслуживания для определенного типа функций меняется от одной функции к другой. Например, двум идентичным SQL-запросам с разными значениями переменных связывания в предикате инструкции *where* могут соответствовать несколько отличающиеся количества операций ЛЮ, из-за чего отличается и время обслуживания процессором. Увеличивая количество процессоров в системе, мы уменьшаем лишь составляющую средней задержки в очереди (W) для времени отклика ($R = S + W$). Если среднее время обслуживания само по себе достаточно близко к допустимому пределу, то случайных колебаний времени обслуживания будет достаточно для того, чтобы время отклика превысило указанный предел для значительного (в процентном отношении) количества выполнений.

На самом деле для нашей важной команды SQL не будет пользы от увеличения количества процессоров сверх шести.

Средняя интенсивность поступления запросов λ

Действительно ли *необходимо* каждому из 100 пользователей выполнять важную команду четыре раза в минуту? Такой вопрос звучит вполне законно. Часто оптимизация начинается с осознания того факта, что пользователи требуют от машины большего, чем они в принципе в состоянии потребить. Например, опрос технологического процесса и формирование сигналов для пользователей четыре раза в минуту не имеет смысла, если пользователю для успешной работы вполне хватает просмотра полученных предупреждений раз в час.

Если бы удалось соответственно снизить интенсивность поступления запросов в пиковое время для нашей задачи с

$$\lambda = 100 \frac{4}{60} \approx 6.666667$$

команд в секунду до

$$\lambda = 100 \frac{1}{60 \times 60} \approx 0.027778$$

команд в секунду, то нагрузка, порождаемая данной командой, уменьшилась бы в 240 раз. Результатом будет значительный выигрыш в масштабируемости. Системе потребуется всего один процес-

сор для обеспечения удовлетворяющего пользователей времени отклика в 87% исполнений команд.

Интенсивность запросов λ представляет собой отношение величин A/T , поэтому существуют два способа ее уменьшения. Во-первых, можно уменьшить числитель A , т. е. количество поступлений запросов в систему (это я уже предлагал). Во-вторых, можно увеличить знаменатель T , период времени, в течение которого система должна принять указанное количество запросов.

Уменьшение средней интенсивности поступления запросов может снизить стоимость системы, но не решит задачу до конца. Ни одно из описанных изменений не приведет к увеличению доли требуемого времени отклика для исследуемой команды до 95%.

Допустимый предел времени отклика r_{\max} и процентиль успеха p

Действительно ли для бизнеса необходимо, чтобы время отклика для той самой важной команды составляло менее одной секунды? Действительно ли это требуется в 95% случаев? Такие вопросы следует задавать шепотом, т. к. они относятся к самому сокровенному. Пользователи часто считают, что эти условия не подлежат обсуждению, но важно, чтобы они понимали, что жесткая позиция по данному вопросу повышает требования к стоимости системы, что может оказаться неприемлемым для бизнеса. Например, если согласие на 100 дополнительных секунд времени отклика в день будет означать экономию 100 000 долларов в год, то, вероятно, на такое предложение стоит откликнуться. Если компания готова пойти на снижение планки до обеспечения 95% успеха для $r_{\max} = 1,5$ секунды, то вам удастся обеспечить необходимую производительность для изначально запрошенной интенсивности поступлений ($\lambda \approx 6,666667$ команд в секунду) с помощью шести процессоров. Если же компания готова обсуждать и снижение интенсивности поступления запросов, то при значении $\lambda \approx 0,027778$ команд в секунду нашу задачу можно решить при помощи всего одного процессора.

Средняя скорость обслуживания μ

Именно этот параметр предоставляет широкие возможности регулирования производительности, не заставляя пользователей идти на компромисс в отношении потери производительности или функциональности. Вероятно, вы слышали, что настройка SQL может быть весьма эффективной, но хотелось бы знать, насколько именно эффективной, прежде чем углубляться в такую настройку. И на этот вопрос поможет ответить наша модель.

Так, если удастся уменьшить время обслуживания важной команды с 0,49 секунды до 0,3125 секунды, то скорость обслуживания для команды возрастет с $\mu = 2,04$ команд в секунду до $\mu = 3,2$ команд в секунду. Если удастся увеличить скорость обслуживания именно настолько, то исходное требование $r_{\max} = 1,0$ будет удовле-

творено в 95% выполнений команды при условии использования четырех процессоров. Если же можно повысить скорость обслуживания до $\mu = 10$ команд в секунду (например, оптимизировав SQL так, что среднее время обслуживания достигнет 0,1 секунды), то требования к производительности можно удовлетворить даже посредством *одного* процессора.

Я считаю, что главная ценность модели массового обслуживания М/М/т в том, что она показывает, в каком направлении следует действовать. В только что рассмотренном примере вы узнали, что не стоит даже *помышлять* об установке более чем шести процессоров для обслуживания важной команды. Узнали о том, что договоренность с пользователями о некотором увеличении значения r_{\max} позволит вам вздохнуть чуть свободнее. Но только снизив частоту выполнения пользователями важной команды (λ) или же уменьшив время обслуживания команд $S = 1/\mu$, вы получите систему, которая наилучшим образом соответствует требованиям к производительности.

Использование возможности подбора параметра в Microsoft Excel

Microsoft Excel предлагает средство под названием *Goal Seek* (Подбор параметра), позволяющее ответить на многие из поставленных вопросов. Оно предоставляет возможность рассматривать любые выходные значения модели в качестве своих входных параметров. Предположим, что требуется определить, какая средняя скорость обслуживания понадобится для того, чтобы довести удовлетворенность пользователей временем отклика до 95% при фиксированном значении m .

Решить задачу можно следующим образом. Сначала вводим для μ произвольное постоянное значение (на рис. 9.24 $\mu = 1$). Затем выбираем пункт меню *Сервис* → *Подбор параметра* (Tools → Goal Seek) для вызова диалогового окна выбора параметра (рис. 9.25). Наша задача состоит в том, чтобы довести значение $CDF(r_{\max})$ до 95%, изменяя μ . На рис. 9.26 в ячейке для μ показано значение, полученное в результате выполнения операции подбора параметра. Оказывается, что желаемого результата можно достигнуть при $\mu \approx 10$.

Анализ чувствительности

Качество прогнозирования времени отклика прямо зависит от качества «контура обратной связи», применяемого для уточнения прогноза. Получив старательно подготовленную формулировку задачи и модель массового обслуживания, практически каждый сможет получить правильные количественные показатели. Но при этом важно устоять перед искушением прекратить мыслительную деятельность сразу же после того, как модель «дала ответ». Получив ответ, следует потратить еще некоторое количество времени на чрезвычайно важное занятие – анализ чувствительности, которым пренебрегают 99% аналитиков, впер-

	A	C	E	F	G
1	Queueing Theory Multiserver Model				
2	M/M/m 3.1e (2003/03/11)				
3	Copyright © 1999-2003 by Hotsos Enterprises, Ltd. All rights reserved				
4					
5	name	value _a	value _b	unit	description
6	<i>Units of measure</i>				
7	jobunit		stmt		workload unit (singular)
8	timeunit		sec		time unit (singular)
9	queueunit		system		queue unit (singular)
10	serverunit		CPU		service channel unit (singular)
11	serviceunit		sec/stmt		service unit
12	throughputunit		stmt/sec		throughput unit
13	<i>Service level agreements</i>				
14	λ	6.66666667	6.66666667	stmt/sec	average arrival rate into the system
15	r_{\max}	1	1	sec/stmt	maximum tolerated response time
16	<i>Architecture</i>				
17	q	1	1	system	number of systems
18	m	4	1	CPU/system	number of CPUs per system
19	μ	2.04081633	1	stmt/sec	average service rate
20	<i>Performance forecasts</i>				
21	color code	◆	—		graph color and shape code
22	model	1 x M/M/4	1 x M/M/1		Kendall notation
23	ρ	81.7%	666.7%		average utilization per CPU
24	S	0.490000	1.000000	sec/stmt	average service time
25	W	0.418787	+INFTY	sec/stmt	average queueing delay at specified λ
26	R	0.908787	+INFTY	sec/stmt	average response time at specified λ
27	$CDF(r_{\max})$	64.922%	0.000%	satisfactions	% of jobs with $R \leq r_{\max}$ at specified λ
28	$1 - CDF(r_{\max})$	35.078%	100.000%	dissatisfactions	% of jobs with $R > r_{\max}$ at specified λ

Рис. 9.24. Чтобы определить, каким должно быть μ для $CDF(r_{\max}) \geq 95\%$, сначала устанавливаем значение μ в произвольную константу

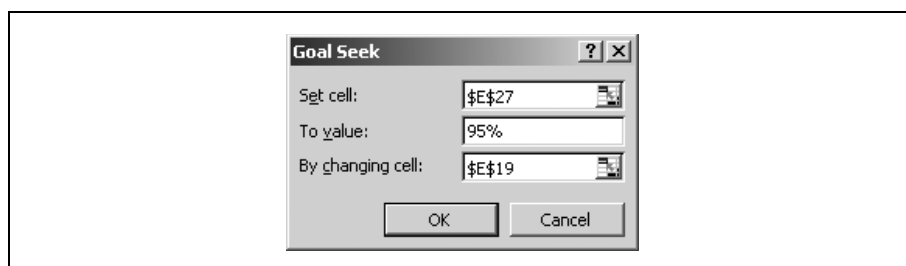


Рис. 9.25. Наша цель – довести значение $CDF(r_{\max})$ до 95%, изменяя значение μ

вые применяющих модель. Оставшаяся часть раздела посвящена некоторым ключевым моментам анализа ошибок все для того же примера.

	A	C	E	F	G
1	Queueing Theory Multiserver Model				
2	M/M/m 3.1e (2003/03/11)				
3	Copyright © 1999-2003 by Hotsos Enterprises, Ltd. All rights reserved				
4					
5	name	value _a	value _b	unit	description
6	<i>Units of measure</i>				
7	jobunit		stmt		workload unit (singular)
8	timeunit		sec		time unit (singular)
9	queueunit		system		queue unit (singular)
10	serverunit		CPU		service channel unit (singular)
11	serviceunit		sec/stmt		service unit
12	throughputunit		stmt/sec		throughput unit
13	<i>Service level agreements</i>				
14	λ	6.66666667	6.66666667	stmt/sec	average arrival rate into the system
15	r_{\max}	1	1	sec/stmt	maximum tolerated response time
16	<i>Architecture</i>				
17	q	1	1	system	number of systems
18	m	4	1	CPU/system	number of CPUs per system
19	μ	2.04081633	9.66807478	stmt/sec	average service rate
20	<i>Performance forecasts</i>				
21	color code	◆	—		graph color and shape code
22	model	1 x M/M/4	1 x M/M/1		Kendall notation
23	ρ	81.7%	69.0%		average utilization per CPU
24	S	0.490000	0.103433	sec/stmt	average service time
25	W	0.418787	0.229744	sec/stmt	average queueing delay at specified λ
26	R	0.908787	0.333177	sec/stmt	average response time at specified λ
27	$CDF(r_{\max})$	64.922%	95.028%	satisfactions	% of jobs with $R \leq r_{\max}$ at specified λ
28	$1 - CDF(r_{\max})$	35.078%	4.972%	dissatisfactions	% of jobs with $R > r_{\max}$ at specified λ

Рис. 9.26. После выполнения операции подбора параметра электронная таблица показывает, что необходимое значение $CDF(r_{\max})$ может быть достигнуто при $\mu \approx 10$

Наряду с выходными параметрами модели необходимо также учитывать следующие факторы:

Физические ограничения

Естественно, нельзя установить какую-то часть процессора. Количество процессоров в системе должно быть целым числом. Некоторые системы могут требовать установки процессоров парами. Разумеется, максимальное число процессоров, которые могут быть установлены в системе, тоже ограничено.

Несовершенство масштабируемости

Модель массового обслуживания ничего не знает о тех недостатках, которые не позволяют системе с m процессорами обеспечить увели-

чение производительности в m раз. Например, для получения шестикратной производительности сервера Linux, потребуется установка восьми или более процессоров. А в ОС Windows NT может оказаться, что независимо от того, сколько процессоров может физически вместить сервер, никакая конфигурация системы не позволит увеличить производительность более чем в два раза. Аналитик по производительности должен в своей работе учитывать несовершенство масштабируемости, препятствующее m -кратному приросту производительности реальной системы с m процессорами.

Дополнительная нагрузка

Формулировка задачи упоминает лишь об одной команде SQL, но на самом деле система почти наверняка будет выполнять и какие-то другие полезные действия. Поэтому результат моделирования для данного конкретного случая следует интерпретировать так: «Количество процессоров, необходимых для обеспечения требуемого времени отклика команды SQL примерно на 5 больше того количества, которое требуется для поддержки всей остальной нагрузки системы».

Другие узкие места

Формулировка задачи обращает наше внимание исключительно на мощность процессора, необходимую для обеспечения требований к времени отклика. А как обстоит дело с потенциальной нехваткой других ресурсов? Что, если на время отклика нашей команды SQL значительно влияют дисковая или сетевая задержки? Вы, конечно, можете смоделировать любой ресурс, участвующий в реализации некоторой функции, но часто самая простая модель оказывается и самой полезной. Если какой-то ресурс, кроме процессора, потребляет основную часть времени отклика функции, то разумно задать вопрос о причинах такого явления.

Изменения входных параметров

В школе принято считать, что предположения, сделанные в условии задачи, обязательно являются верными. Опытные профессионалы знают, что ошибка в постановке задачи может привести в дальнейшем к серьезным проблемам с проектом. Как же защитить проект от не заслуживающих доверия предположений, содержащихся в начальных условиях? Следует оценить влияние таких предположений на нашу модель. Что произойдет с результатом, если одно или несколько предположений, заложенных в формулировке задачи, окажутся неточными или просто со временем изменятся?

Общая производительность системы чрезвычайно чувствительна даже к небольшим изменениям средней скорости обслуживания (μ) и средней интенсивности поступлений запросов (λ). Это отражается в модели. По своей природе данные параметры исключительно сильно влияют на общую производительность, из-за чего погрешность в их оценке ставит под сомнение результат моделирования.

Например, завышение средней скорости обслуживания всего на 1% может привести к занижению среднего времени отклика в загруженной системе на 10% или даже более. Изменение величин μ и λ может быть вызвано следующими обстоятельствами:

- Некорректное тестирование. В качестве классического примера можно привести тесты, созданные для эмулирования поведения рабочей базы данных, но основанные на излишне упрощенных конфигурациях продукта или на нереально маленьких таблицах.
- Изменения объема данных, особенно для SQL-запросов, производительность которых линейно (или в большей степени) зависит от объема обрабатываемых данных [Millsap (2001a)].
- Изменения физического распределения данных, которые могут со временем повысить или понизить эффективность индекса (или его полезность для оптимизатора запросов Oracle) [Millsap (2002)].
- Изменения кода функции, такие как изменения команд SQL или объектов схемы, вызванные модернизацией оборудования, работами по повышению производительности или чем-то еще.
- Изменения объема бизнес-функции, вызванные приобретениями и поглощениями, неожиданными успехами или неудачами маркетинговых компаний и т. д.

Если неуправляемые изменения чувствительных входных параметров возможны, то следует позаботиться о том, чтобы выбранная конфигурация системы позволяла провести экономное повышение (или понижение) производительности. Ваши знания о чувствительности входных параметров помогут определить, с какими из них необходимо обращаться особенно аккуратно.

Резюме

Моделирование производительности – это сложная тема. Надеюсь, что эта глава помогла вам разобраться в технологии. И что еще важнее, надеюсь, вам стали понятны существующие ограничения. Мне не раз встречались аналитики, которые посвятили себя заведомо безуспешной борьбе с непреложными законами природы. Я построил эту главу так, чтобы вы не попали в эту ловушку. В заключение еще раз остановлюсь на основных моментах:

- Метод проб и ошибок – это неэффективный, дорогой и ненадежный метод оптимизации. Если система соответствует условиям применения математической модели, то решение, полученное на ее основе, будет гораздо более эффективным.
- Время отклика, по существу, является единственным параметром, который беспокоит конечных пользователей. Для пользователя время отклика – это продолжительность между отправкой запроса

и возвращением первого байта ответа на этот запрос. Для теоретика массового обслуживания время отклика равно сумме времени обслуживания и задержки в очереди. Мы можем уменьшить время отклика, уменьшив одну из составляющих: время обслуживания или же задержку в очереди.

- В нагруженных системах время отклика ухудшается из-за появления очередей. Бороться с этим можно, снижая нагрузку или же уменьшая время обслуживания. Аналитики по производительности часто забывают о том, что снижение рабочей нагрузки – вполне законная операция. Математическая модель массового обслуживания помогает аналитику выбрать компромиссное решение, позволяющее удовлетворить клиента и с точки зрения функциональности, и с точки зрения производительности.
- Модель массового обслуживания $M/M/m$ – это хорошо исследованная и хорошо проверенная модель прогнозирования производительности систем, в которых время между поступлениями запросов и скорость обслуживания распределены экспоненциально. Этим условиям соответствуют многие системы Oracle. В главе представлена полная реализация модели $M/M/m$ в Microsoft Excel, программа на Perl, проверяющая, подчиняется ли имеющаяся выборка данных экспоненциальному распределению, а также полный набор подробных пояснений относительно того, как применять модель в проекте Oracle.
- Одним из основных достоинств модели массового обслуживания является изменение нашего восприятия времени отклика. Модель выявляет четкую математическую зависимость между параметрами нагрузки, скорости обслуживания и длительности ожидания. Более того, она акцентирует внимание на том, что для оптимизации системы необходимо рассмотреть значения всех допускающих обсуждение параметров.
- Наш пример с решением иллюстрирует весьма распространенную ситуацию: несмотря на то, что система ощущает нехватку именно ресурсов процессора, увеличение мощности процессора не помогает удовлетворению требований к производительности системы. В данном случае модель выявляет то, что часто справедливо и в реальной жизни: наиболее экономически эффективный способ повышения производительности системы заключается в избавлении от лишней нагрузки. Существуют два основных метода избавления от лишней нагрузки: освобождение от ненужных бизнес-функций и сокращение кода функций.
- Модель массового обслуживания $M/M/m$ игнорирует ряд факторов, которые должен принимать в рассмотрение аналитик. Например, модель предполагает существование абсолютной масштабируемости для m каналов обслуживания. Модель во многом является оптимистичной. Если оптимистичная модель прогнозирует низкую

производительность системы, то и в реальности такая конфигурация даст низкую производительность. Однако если оптимистичная модель выносит положительный вердикт относительно производительности системы, это не обязательно означает, что в реальности такая система будет работать хорошо. Некоторые немоделируемые проблемы масштабируемости могут погубить проект.

Упражнения

1. Примените модель массового обслуживания $M/M/t$ для проверки всех результатов рассмотренного нами примера.
2. Проектировщики нового международного терминала в аэропорту Millsap International Airport (MQP) города Миллсап в Техасе обратились к вам за помощью в оптимизации времени обслуживания клиентов. Каждый из новых высокотехнологичных кассовых залов будет вмещать шесть билетных касс. Строители аэропорта просят помочь решить, следует ли им организовать одну длинную очередь для направления клиентов ко всем 6 кассам (конфигурация $M/M/6$) или же шесть коротких очередей, каждая из которых относилась бы только к определенному кассиру (шесть независимых систем $M/M/1$). Используйте модель массового обслуживания $M/M/t$ для определения того, какая конфигурация обеспечит наименьшее время отклика для клиентов.
3. Перед клиентом стоит выбор: какой из двух компьютеров купить? Модель H имеет один процессор, который способен выполнять 40 000 операций логического чтения (LIO) Oracle в секунду. Модель L имеет четыре процессора в симметричной многопроцессорной конфигурации, но каждый из этих процессор может выполнять всего 15 000 операций LIO в секунду. Какой компьютер следует приобрести?
4. Напишите программу для измерения интервалов времени между поступлениями запросов от вашего диспетчера очереди пакетных заданий. Подчиняются ли полученные значения экспоненциальному распределению? Удастся ли обнаружить экспоненциально распределенные подмножества данных, сузив круг программ, для которых собираются данные? Удастся ли обнаружить экспоненциально распределенные подмножества данных, сузив тот промежуток времени (время суток), для которого собираются данные?
5. Позволяет ли ваш диспетчер очереди пакетных заданий измерить реальное время обслуживания заданий? Например, можно ли для указанного задания определить, сколько процессорного времени было израсходовано? Если это возможно, то напишите программу для получения времени обслуживания от вашего диспетчера очереди пакетных заданий. Подчиняются ли полученные значения экспоненциальному распределению? Можно ли выделить экспоненци-

ально распределенные подмножества данных? Если диспетчер очереди пакетных заданий не измеряет реальное время обслуживания, то как можно получить такие данные?

6. Брокерская инвестиционная компания приобрела лицензию на Oracle Server и собирается спроектировать и создать клиентское приложение для обработки транзакций по инвестиционным сделкам. Код еще не написан, но бизнес-требования состоят в том, что конечный пользователь должен получать результат запроса или подтверждение выполнения транзакции в течение трех секунд после нажатия на кнопку, инициирующую соответствующее действие. Компания планирует обрабатывать 30 000 транзакций за рабочий день континентальной части США, при этом в пиковый период интенсивность предполагается равной 650 транзакциям за пять минут. Рабочий день в континентальной части США длится с 8:00 утра по восточному времени до 17:00 по тихоокеанскому времени (8:00 утра по тихоокеанскому времени – это 11:00 утра по восточному времени). Оперативный доступ к новой системе будут иметь приблизительно 2000 брокеров.

Все пользователи подключаются к серверу базы данных через сложную систему, включающую в себя локальную сеть, терминальные серверы и монитор обработки транзакций. Разработчики системы полагают, что после вычитания из трехсекундного значения (максимально допустимого общего времени отклика) времени, необходимого для организации представления данных на стороне клиента, времени отклика сети и времени работы монитора транзакций, на операции сервера Oracle останется порядка полутора секунд для каждой транзакции (табл. 9.7).

Таблица 9.7. Требования к времени отклика для инвестиционной брокерской компании

Максимально разрешенное время отклика (сек)	Этап выполнения
1,500	Время отклика сервера Oracle
1,500	Разбор, связывание, исполнение, выборка и т. д.
	Общее время отклика, не относящееся к Oracle
	Управление представлением данных на стороне клиента
	Время отклика сети
3,000	Монитор обработки транзакций
	Общее время отклика

7. Клиенту нужен совет по выбору архитектуры аппаратного и прикладного программного обеспечения. Укажите, какую конфигурацию оборудования следует выбрать и какие требования необходимо

предъявить к производительности транзакций при разработке SQL-кода приложения с тем, чтобы на выходе получить время отклика, удовлетворяющее компанию-заказчика.

Используйте модель массового обслуживания $M/M/t$ для моделирования времени отклика важной бизнес-функции вашей системы. Манипулируйте параметрами модели до тех пор, пока модель не будет достоверно прогнозировать реально существующее (подтвержденное измерениями) поведение функции. Например, если у вас есть система с 10 процессорами, обрабатывающая порядка 100 различных бизнес-функций, экспериментируйте с моделью так, как если бы лишь часть одного из процессоров была выделена для обслуживания одной конкретной функции. Что произойдет, если удвоить интенсивность поступлений запросов на исполнение данной функции? Что будет, если удастся на 10% повысить скорость обслуживания для данной функции? Как отразится на производительности функции выделение для нее вдвое большей процессорной мощности? Помимо усовершенствования оборудования, какими еще способами можно добиться увеличения мощности процессора, выделяемой для обслуживания данной бизнес-функции?

Часть III. Реализация

10

Работа с профилем ресурсов

Как рассказывалось в главе 1, задача оптимизации времени отклика регулярно возникает в повседневной жизни. Решение этой задачи основывается на соображениях здравого смысла, формализованных Джином Амдалом (Gene Amdahl): наибольшее сокращение времени отклика достигается уменьшением наиболее значительной его составляющей.

Вспомним формат профиля ресурсов, описанный в главе 1 и еще раз показанный в примере 10.1. Оптимизация времени отклика настолько «въелась в нас», что большинство людей (включая пользователей и руководителей, не имеющих подготовки в области анализа производительности) без труда понимают профиль ресурсов. Вне зависимости от уровня технической подготовки людям требуется не более десяти секунд на изучение примера 10.1, чтобы сделать правильный вывод:

Не знаю, что такое эти два «SQL*Net», но чем бы они ни были, они занимают приблизительно две трети общей продолжительности отчета. Из-за чего возникают SQL*Net message from client и SQL*Net more data from client?

Это и есть правильный путь решения задачи. Те руководители и пользователи, которым я показывал этот пример, всегда удивлялись, что профессиональные аналитики по производительности на три месяца углублялись в изучение ресурсов процессора и конкуренции за защелки, именно их считая источниками проблемы. (Пример 10.1 относится к той же проблеме производительности, связанной с платежной ведомостью, которая была описана в главе 1.) Ответ очевиден: аналитики этого проекта потратили три месяца на изучение ошибочных диагностических данных.

Пример 10.1. Формат профиля ресурсов одинаково понятен людям с разной технической подготовкой

Response Time Component	Duration		# Calls	Dur/Call
SQL*Net message from client	984.0s	49.6%	95,161	0.010340s
SQL*Net more data from client	418.8s	21.1%	3,345	0.125208s
db file sequential read	279.3s	14.1%	45,084	0.006196s
CPU service	248.7s	12.5%	222,760	0.001116s
unaccounted-for	27.9s	1.4%		
latch free	23.7s	1.2%	34,695	0.000683s
log file sync	1.1s	0.1%	506	0.002154s
SQL*Net more data to client	0.8s	0.0%	15,982	0.000052s
log file switch completion	0.3s	0.0%	3	0.093333s
enqueue	0.3s	0.0%	106	0.002358s
SQL*Net message to client	0.2s	0.0%	95,161	0.000003s
buffer busy waits	0.2s	0.0%	67	0.003284s
db file scattered read	0.0s	0.0%	2	0.005000s
SQL*Net break/reset to client	0.0s	0.0%	2	0.000000s
Total	1,985.4s	100.0%		

Мой любимый эпиграф к теме повышения производительности взят из учебника игры в гольф [Pelz (2000) 215]:

Нет ничего хуже, чем упорно решать неверно выбранную задачу, надеясь на улучшение, и в результате остаться ни с чем.

Профили ресурсов – превосходное средство, показывающее, *что* надо исправить, независимо от того, известно ли, *как* это сделать. Они помогают избежать типичной для настройки Oracle ловушки, когда исправляют то, что умеют исправлять, не заботясь о практических результатах.

Как работать с профилем ресурсов

Хотя люди в большинстве своем способны интуитивно понять формат профиля ресурсов, несколько формальных правил все же помогают более эффективно использовать представленную в нем информацию. Мы с коллегами, проанализировав, начиная с 2000 г., несколько сотен профилей ресурсов, обобщили наш подход в следующих основных правилах:

- Работайте с данными профиля ресурсов в порядке убывания вклада в общее время отклика.
- Исключите лишние вызовы, прежде чем пытаться уменьшить задержку на каждый вызов.
- Если после исключения избыточных обращений к ресурсу его вклад во время отклика все еще слишком велик, избавьтесь от ненужной конкуренции за этот ресурс.

Модель «трех подходов» Дэйва Энсора

Мой друг и коллега по издательству O'Reilly Дэйв Энсор (Dave Ensor) в своих публичных выступлениях отмечает, что существуют три подхода к решению проблемы времени отклика, вызванной дефицитом некоторого ресурса:

- *Коммерческий подход* заключается в увеличении ресурса. Такой подход действительно улучшает показатели чистой прибыли, возврата инвестиций и движения денежных потоков. К сожалению, эти улучшения имеют место для ваших поставщиков, но... не для вас.
- *Исследовательский подход* заключается в манипуляциях с конфигурациями, параметрами, оборудованием и всем тем, что может за счет «настроек» хоть как-то уменьшить время обработки запросов. При действительно больших задержках выполнения вызовов технические специалисты испытывают непреодолимое желание что-нибудь «настроить». Такая разновидность настройки – Самое Любимое Развлечение для исследователя, живущего в каждом из нас. Самое приятное здесь то, что это позволяет нам выглядеть очень занятыми людьми и избегать множества других менее интересных дел. К сожалению, такой подход ведет лишь к пустой трате времени при отсутствии сколько-нибудь заметного результата. Главным критерием остается закон Амдала.
- *Разумный подход* заключается в уменьшении количества обращений к ресурсу. Вопрос только в том, как это сделать.

Те, кто встречался с Дэйвом или слушал его выступления, поймут, почему я восхищаюсь сдержанностью, которую он проявил, давая названия этим трем подходам.

- Только исключив избыточные обращения к ресурсу и неоправданную конкуренцию за него, можно рассматривать вопрос об увеличении этого ресурса.

Эти правила постоянно помогают нам в быстрой и эффективной оптимизации. В следующих разделах они рассмотрены более подробно.

Работа в порядке убывания времени отклика

Очень легко работать с профилем ресурсов, отсортированным по убыванию вклада составляющих во время отклика. Список просматривается сверху вниз. Самый первый потребитель времени отклика и есть тот ресурс, от которого в наибольшей степени зависит повышение производительности. Вспомните закон Амдала: чем ниже компонент расположен в списке, тем слабее его вклад в уменьшение времени отклика в целом.

В примере 10.2 приведен профиль ресурсов, построенный для определенной пользовательской операции в системе с «очевидной проблемой дискового ввода/вывода». Задержка ввода/вывода одного блока в этой подсистеме временами превышает 0,600 секунды. Большинство инже-

неров сочтут задержку более 0,010 секунды на один блок неприемлемой. А задержки в этой системе в *шестьдесят раз* хуже этого порогового значения.

Пример 10.2. Профиль ресурсов, построенный для определенной пользовательской операции, в системе с известной проблемой производительности подсистемы ввода/вывода

Response Time Component	Duration		# Calls	Dur/Call
CPU service	37.7s	68.9%	214	0.175981s
unaccounted-for	8.4s	15.4%		
db file sequential read	5.5s	10.1%	568	0.009630s
db file scattered read	2.1s	3.8%	89	0.024157s
latch free	0.9s	1.6%	81	0.011605s
log file sync	0.1s	0.2%	3	0.026667s
SQL*Net more data to client	0.0s	0.0%	4	0.002500s
file open	0.0s	0.0%	12	0.001667s
SQL*Net message to client	0.0s	0.0%	58	0.000000s
Total	54.7s	100.0%		

Однако профиль ресурсов этой пользовательской операции ясно показывает, что улучшение времени отклика для нее надо начинать вовсе не с решения проблемы дискового ввода/вывода. Единственные компоненты времени отклика, на которые повлияет повышение производительности ввода/вывода – это db file sequential read и db file scattered read. Даже если удастся полностью *исключить* их из профиля ресурсов, это уменьшит время отклика примерно на 14%.



Занятно, что «проблема» подсистемы ввода/вывода *действительно* сказывается на производительности программы из примера 10.2. Данные трассировки свидетельствуют о том, что отдельные вызовы ввода/вывода для одного блока в данной пользовательской операции занимают до 0,620 секунды. Но даже эта информация несущественна. Возможный выигрыш от устранения любых проблем ввода/вывода для данной операции столь незначителен, что лучше потратить время, требуемое на анализ и устранение проблемы, на что-нибудь другое.

Почему правильный выбор так важен

Настало время проверить, насколько вы привержены методу R. Вы можете сказать: «Но ведь устранение такой серьезной проблемы ввода/вывода, безусловно, приведет к *некоторому* улучшению производительности системы...». Да, действительно, решение этой проблемы даст *некоторое* улучшение производительности. Но *необходимо* понимать, что решение этой проблемы ввода/вывода не даст ощутимого улучшения *данной* пользовательской операции (той, которая соответствует профилю ресурсов из примера 10.2 и ускорение которой отчаянно требуется бизнесу). Понимать это жизненно необходимо по двум причинам:

- Время и материалы, израсходованные на решение «проблемы ввода/вывода», – это ресурсы, которые уже нельзя вложить в повышение производительности пользовательской операции, представленной в примере 10.2. Если пользовательская операция выбрана правильно, то отвлечение на проблему ввода/вывода, по меньшей мере, непродуктивно.
- Решение проблемы ввода/вывода может в действительности *ухудшить* производительность пользовательской операции из примера 10.2. Это не только теоретическая возможность, мы встречались с этим явлением на практике (см. главу 12). Вот один из возможных случаев: представьте себе, что одновременно с операцией из примера 10.2 в системе выполняются еще несколько программ. И эти программы потребляют очень много процессорного времени, но в данное время они часто простаивают в очереди к медленному диску. Исключение задержки в очереди к диску для этих медленных процессов фактически приведет к обострению конкуренции за использование процессора, а это как раз та составляющая, которая преобладает во времени отклика рассматриваемой операции. Таким образом, решение проблемы ввода/вывода на самом деле ухудшит производительность выбранной пользовательской операции.

Да, решение проблемы ввода/вывода даст некоторый прирост производительности остальных программ. Но если выбор пользовательской операции для повышения производительности в примере 10.2 сделан правильно, то улучшение производительности второстепенных операций будет достигнуто ценой ее ухудшения для наиболее важной операции. Такой результат противоречит приоритетам бизнеса.

По обеим причинам, *если* выбор пользовательской операции из примера 10.2 сделан правильно, работа над «проблемой ввода/вывода» будет ошибкой. Если же пользовательская операция для повышения производительности выбрана неверно, то это значит, что вы ошиблись при выполнении действий, описанных в главе 2.

Возможные выгоды от небольших улучшений

Несмотря на вышесказанное, возможна ситуация, когда экономически наиболее оправданной будет работа не с самой первой составляющей профиля. Предположим, например, что проблема подсистемы ввода/вывода из примера 10.2 может быть «решена» простым отказом от запуска некоторого долго выполняющегося и требующего интенсивного обмена с диском отчета, ежедневно формируемого во время выполнения указанной пользовательской операции. Представьте себе, что решение заключается в исключении работы по созданию отчета, который, как вы выяснили, никто в компании даже не читает. В таком случае решение проблемы – простое удаление этого отчета – настолько элементарно, что неразумно было бы им не воспользоваться.

С точки зрения математики возврат инвестиций (Return Of Investment, ROI) от некоторой деятельности может быть высоким даже при небольшом значении возврата R , если инвестиции I тоже малы, вследствие чего отношение R/I велико. Иногда такое случается. Однако имейте в виду, что большое значение ROI – не единственная ваша цель. Мой профессор финансов Майкл Вецуйпенс (Michel Vetsuypens) однажды проиллюстрировал этот тезис, бросив в аудиторию пятицентовую монетку. Студент, поймавший ее, достиг тем самым почти бесконечного ROI – поймать монету практически ничего не стоило, а возврат составил пять центов. Несмотря на то, что это, наверное, был высочайший показатель ROI, достигнутый студентом за всю его жизнь, понятно, что общий эффект от добавления пяти центов к его капиталу был исчезающе мал. Из этой истории ясно, почему для вас так важно при определении целей повышения производительности наряду с ROI учитывать показатели *чистой прибыли и движения денежных потоков*, о чем рассказывалось в главе 2.



Эта история иллюстрирует также фундаментальный недостаток относительных величин: они скрывают абсолютные значения.

Исключение лишних вызовов

На курсах Hotsos Clinic бытует такая избитая шутка:

Вопрос: Как быстрее всего сделать нечто? (В качестве неизвестного «нечто» может выступать все что угодно: от выполнения вызовов базы данных до полета в другой город или посещения ванной.)

Ответ: Не делать.

Самый быстрый способ сделать *что бы то ни было* – избежать необходимости делать это. (Эта аксиома будет справедлива до тех пор, пока кто-нибудь не изобретет подходящий для человека способ путешествия во времени. Пока мы не найдем способ сделать продолжительность наших действий отрицательной, лучшее, что мы можем сделать – приравнять ее к нулю.)

Самый экономически эффективный способ повышения производительности системы обычно состоит в исключении бесполезной нагрузки. *Бесполезна* любая нагрузка, которая может быть исключена из системы без уменьшения ее ценности для владельца. Аналитики, мало знакомые с Методом R, часто бывают шокированы, обнаружив, что их системы вполне соответствуют такому правилу:

Во многих системах доля бесполезной нагрузки превышает 50%.

Это справедливо практически для каждой системы, измеренной мною начиная с 1989 г., и, скорее всего, для вашей системы тоже. Для этого утверждения есть основания: на протяжении 1980–90-х годов, когда многие аналитики Oracle проходили обучение, нас учили принципам,

поощрявшим расточительство. Например, популярное некогда мнение о предпочтительности высокого коэффициента попадания в кэш буферов привело к низкой эффективности многих приложений. Это заблуждение описано в ряде работ, в частности [Millsap (2001b; 2001c); Lewis (2001a); Vaidyanatha et al. (2001); McDonald (2000)].

Почему снижение нагрузки так полезно

Очевидно, что исключение бесполезной работы в первую очередь сказывается на производительности приложения, выполнявшего ее прежде. Однако не все осознают огромный сопутствующий эффект от снижения нагрузки. Избавляясь от лишнего запроса к ресурсу, мы уменьшаем вероятность образования очереди из остальных пользователей к этому ресурсу. Понять дополнительную выгоду от снижения нагрузки довольно просто. Представьте себе программу, использующую процессор почти без перерывов в течение примерно 14 часов (профиль ресурсов такой программы приведен в примере 1.4). Пусть эффективность программы увеличилась настолько, что теперь ей требуется всего десять минут процессорного времени. (Такое улучшение обычно достигается работой с планом выполнения критических команд SQL.)

Понятно, почему будет доволен пользователь отчета, выполняемого теперь за десять минут вместо четырнадцати часов. Но представьте себе и радость остальных пользователей, которых тоже коснулось это улучшение. До улучшения пользователи, претендовавшие на время процессора, конкурировали с процессом, занимавшим его *больше половины суток*. Теперь же отчет претендует лишь на десять минут процессорного времени. Вероятность образования очереди на обслуживание процессором уменьшилась во много раз. При таком 14-часовом периоде выгода для системы сопоставима с эффектом от установки второго процессора.



В описанном случае преимущество от сокращения нагрузки на самом деле будет даже больше, чем от дополнительного процессора, т. к. второй процессор потребует от операционной системы дополнительных действий по диспетчеризации процессорного времени. Кроме того, уменьшение нагрузки обойдется дешевле, чем установка второго процессора.

Сопутствующая выгода от уменьшения нагрузки может быть ошеломляющей. Математически это обосновано в главе 9.

Запросы и услуги в технологическом стеке

Как же исключить бесполезную нагрузку? Ответ зависит от уровня технологического стека. Понятие *технологического стека* системы было введено в главе 1, когда рассматривалось представление времени отклика пользовательской операции при помощи диаграммы последовательности. Технологический стек состоит из взаимодействующих друг с другом уровней, выступающих в роли поставщиков и потребителей, как показано на рис. 10.1. Отношения между ними просты.

Уровни получают запросы и предоставляют услуги, и все это требует времени (вследствие этого стрелки запросов и услуг наклонены вниз).

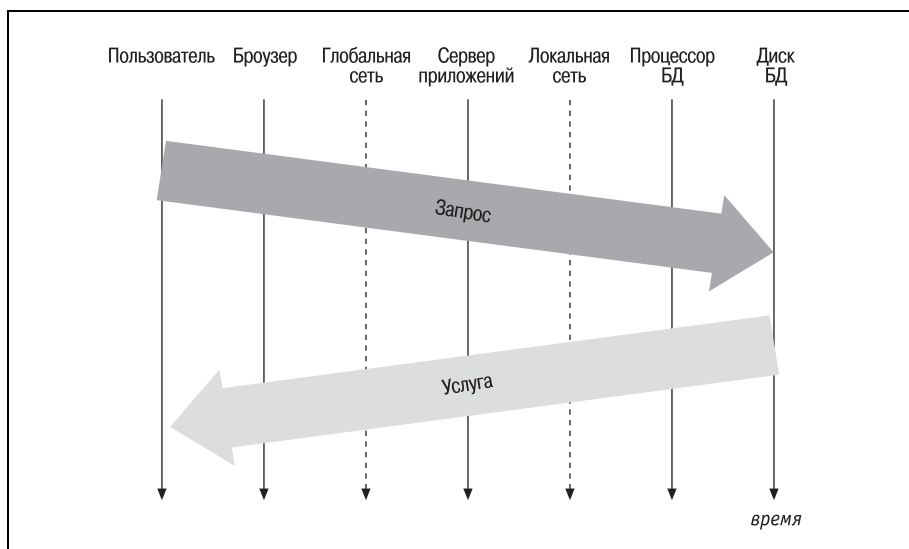


Рис. 10.1. Диаграмма последовательности показывает перемещение запросов и услуг между уровнями технологического стека с течением времени (ось времени направлена вниз)

Такое представление технологического стека помогает понять фундаментальную аксиому повышения производительности:

Почти всегда проблемы производительности бывают вызваны *чрезмерными запросами* к одному или нескольким ресурсам.

Практически в любом случае ухудшение производительности может быть устранено за счет уменьшения потребности в некотором ресурсе. Эта задача сокращения решается проходом «вверх» по технологическому стеку от перегруженного устройства. (Направление *вверх* в стеке соответствует направлению *влево* на диаграмме последовательности на рис. 10.1) Вопрос, которым следует задаться, занимаясь повышением производительности, звучит так:

Действительно ли оправдан столь высокий спрос на данный ресурс?

Рассмотрим профиль ресурсов, приведенный в примере 10.3. Без малого 97% времени отклика исследуемой пользовательской операции, составляющего 1,3 часа, расходуется на ожидание дискового ввода/вывода. Согласно профилю ресурсов, возможны два решения:

- Уменьшить количество вызовов, сделав так, чтобы оно не превышало 12165.
- Уменьшить среднюю длительность вызова, сделав так, чтобы она не превышала 0,374109 секунды.

Учтите, что улучшение любого из этих показателей линейно сказывается на вкладе данного компонента в значение времени отклика. Если, например, уменьшить количество вызовов вдвое, то и длительность уменьшится вдвое. Аналогично, если сократить вдвое среднюю продолжительность вызова, длительность также уменьшится вдвое. Несмотря на то, что сокращение количества вызовов и их продолжительности одинаково сказывается на результирующем значении, обычно бывает значительно легче добиться существенного уменьшения количества вызовов, чем снизить их среднюю продолжительность. Я специально выбрал для этой книги такую последовательность столбцов профиля ресурсов, чтобы лучшее решение первым бросалось в глаза.

Пример 10.3. Во времени отклика выбранной пользовательской операции доминируют вызовы чтения подсистемы дискового ввода/вывода

Response Time Component	Duration		# Calls	Dur/Call
db file scattered read	4,551.0s	96.9%	12,165	0.374109s
CPU service	78.5s	1.7%	215	0.365023s
db file sequential read	64.9s	1.4%	684	0.094883s
SQL*Net message from client	0.1s	0.0%	68	0.001324s
log file sync	0.0s	0.0%	4	0.010000s
SQL*Net message to client	0.0s	0.0%	68	0.000000s
latch free	0.0s	0.0%	1	0.000000s
Total	4,694.5s	100.0%		

Как исключить вызовы

Как уменьшить количество событий, выполняемых пользовательской операцией? Во-первых, выясните, что именно *делает* потребляемый ресурс. Что заставляет процесс Oracle, профиль которого приведен в примере 10.3, выполнять 12 165 вызовов многоблочного чтения? Затем выясните, можно ли обеспечить требуемую функциональность меньшим числом обращений к ресурсу. В главе 11 объясняется, как это сделать для нескольких часто встречающихся событий Oracle. Определите, можно ли ограничить потребность в перегруженном ресурсе, пройдя по всем уровням технологического стека. Например:

- Многие аналитики полагают, что, увеличивая размер кэша буферов базы данных (т. е. выделяя больше памяти системе Oracle), они уменьшают вероятность обращения к дисковым устройствам, выполняемого вслед за просмотром памяти.
- Однако поднявшись чуть выше по стеку, часто можно достичь гораздо лучших результатов, и не прибегая к увеличению объема памяти в системе. Оптимизируя план выполнения, следуя которому запрос извлекает строки из базы данных, часто удается избавиться даже от обращений к памяти.
- Поднявшись по стеку еще выше, можно обнаружить еще более простые пути достижения результата. Например, может выясниться,

что данная пользовательская операция может выполняться реже или не выполняться вообще (возможно, ее удастся чем-нибудь заменить), и это никак не скажется на ценности системы для бизнеса.

Думайте шире

Технические специалисты иногда ограничивают свою область деятельности нижними уровнями технологического стека, на которых они чувствуют себя наиболее комфортно. Такое поведение повышает риск упустить возможности значительного повышения производительности. В качестве примера приведу заказчика, с которым я работал в середине 1990-х. Бухгалтерия ежедневно выпускала трехфутовую стопку отчетов о предварительном балансе главной книги (General Ledger (GL) Aged Trial Balance). Выяснив причины столь частого запуска этого отчета, руководитель проекта внедрения главной книги из Oracle Consulting объяснил пользователям, как они могут получить нужные им данные менее затратным способом с помощью онлайн-формы. В результате мы смогли уменьшить нагрузку на систему на миллиарды команд в день, не затратив на «настройку» ни цента. Такое решение не только разгрузило систему, но и облегчило работу пользователей, которым гораздо удобнее работать с онлайн-формой, чем вылавливать нужные цифры из отчета толщиной в дюйм.

Сооснователь компании *hotsos.com*, Гэри Гудман (Gary Goodman), рассказывал о проектах внедрения приложений, которыми он руководил во время работы в Oracle Corporation. Один из приемов, практикуемых им в процессе внедрения, состоял в отключении всех прикладных отчетов в системе. Когда пользователи приходили с просьбой о недостающем отчете, специалисты подключали его обратно. По опыту Гэри, ни разу не пришлось подключить больше 80% отчетов, первоначально имевшихся в системе. Как по-вашему, какие 20% из *ваших* отчетов пользователи никогда не читают?

На уровне бизнес-требований вашего технологического стека вопрос, на который вы должны ответить, звучит так:

Действительно ли все предъявляемые бизнес-требования *оправданны*?

Не оправданные интересами бизнеса пользовательские операции следует просто отключить. В действительно необходимых пользовательских операциях постарайтесь исключить бесполезную работу (некоторые способы рассмотрены в главе 11). Данные диагностики производительности будут «подталкивать» вас в направлении снизу вверх, хотя, как правило, эффективнее продвигаться «сверху вниз». Например, прежде чем настраивать отчет, узнайте, нельзя ли от него отказаться. Не ограничивайте свою работу по оптимизации только техническими вопросами. Как было отмечено в главе 1, хороший аналитик по производительности должен хорошо разбираться в том, как соотносятся между собой требования бизнеса и вычислительные мощности, призванные обеспечивать выполнение этих требований.

Наконец, не забывайте о том, что с точки зрения бизнеса пользователи *не работают с системой*, они являются ее *частью*. Иллюстрацией этого может служить история, рассказанная моим коллегой Риком Минутеллой (Rick Minutella). Компания пригласила его оптимизировать производительность модернизированного недавно приложения ввода заказов. В табл. 10.1 показана разница в производительности до и после модернизации. На типичном «большом совещании» с участием финансового директора, пользователей, руководителей ИТ-отделов и поставщиков оборудования компания потребовала от корпорации Oracle восстановить производительность формы ввода заказов, которая губила весь бизнес.

Таблица 10.1. Производительность ввода заказов до и после модернизации

Показатель производительности	Значение до модернизации	Значение после модернизации
Скорость ввода заказов	10 вызовов/час	6 вызовов/час
Время отклика формы	5 сек/экран	60 сек/экран

Что и говорить, 60-секундное ожидание онлайн-овой формы ввода заказов – это слишком долго. Однако утверждение «производительность *формы ввода заказов* губит наш бизнес» попросту неверно. И вот почему. Если бизнес работает со скоростью шесть телефонных вызовов в час, то средняя продолжительность вызова составляет десять минут. В примере 10.4 приведен профиль ресурсов для такого вызова при работе с 60-секундной формой.

Пример 10.4. Профиль ресурсов процесса обработки заказа при неудовлетворительной производительности формы

Before optimizing the online order entry form				
Response Time Component	Duration		# Calls	Dur/Call
other	540s	90.0%	1	540s
wait for the order entry form	60s	10.0%	1	60s
Total	600s	100.0%		

Какое максимальное влияние на бизнес может оказать оптимизация формы? Ответ приведен в примере 10.5. Если *полностью исключить* время отклика формы, общее время обработки заказа уменьшится только до девяти минут.

Пример 10.5. Профиль ресурсов процесса обработки заказа при условии полного исключения времени отклика формы

Response Time Component	Duration		# Calls	Dur/Call
other	540s	100.0%	1	540s
wait for the order entry form	0s	0.0%	1	0s

Total

540s 100.0%

Беда в том, что этого недостаточно. Если оператор сможет принимать один заказ в среднем за 9 минут, то за час он примет в среднем только 6,67 заказа. Это все еще довольно далеко от требования обработки 10 заказов в час. Само по себе повышение производительности формы ввода не поможет этой компании увеличить скорость ввода заказов до указанного значения. Бизнес «убивает» вовсе не форма, а то, чем занимаются принимающие заказ операторы в оставшиеся 9 минут.

Для того чтобы улучшить производительность в данном случае, требуется выйти за рамки обычной «настройки системы». Куда же пропадает это «оставшееся» (other) время? Среди прочих возможностей выделим такие:

- Большая часть его может уходить на длительное ожидание при поиске идентификатора продукта, создавая клиенту неудобства. В этом случае следует найти способ сократить это время.
- «Остальное» время главным образом тратится на улучшение взаимоотношений с клиентом. Не исключено, что в этом случае имеет смысл нанять дополнительных операторов, чтобы при средней скорости приема заказов одним оператором шесть за час общая производительность соответствовала потребностям бизнеса.

Еще одно интересное предположение, которое следует проверить, – не группируются ли звонки по времени таким образом, что клиенты тратят много времени на ожидание ответа в часы наибольшей занятости. Уроки теории массового обслуживания, данные в главе 9, помогут вам более эффективно справиться с наплывом звонков, либо задействовав дополнительных операторов, либо уменьшая время обслуживания клиента. Здесь есть, о чем подумать. Главное – не ограничиваться при рассмотрении своей «системы» лишь характеристиками программного и аппаратного обеспечения. Бизнес требует, чтобы «систему ввода заказов» рассматривали шире, учитывая, что *все* участники процесса размещения заказа оказывают влияние на прибыль, возврат инвестиций и приток денежных средств.

Устранение конкуренции между процессами

Что делать, если все лишние вызовы в диагностируемой пользовательской операции уже исключены, а время отклика по-прежнему неприемлемо велико? Надо определить, насколько оправданны задержки в отдельных вызовах. Для того чтобы решить, допустима ли данная задержка, надо иметь представление о том, каких значений можно ожидать. Оказывается, эти значения состоят всего из нескольких составляющих. Наиболее важные с нашей точки зрения перечислены в табл. 10.2. Эти значения будут меняться с увеличением скорости работы оборудования, но на момент написания книги, в 2003 г., они представляют собой разумные ограничения для большинства существ-

вующих систем. В частности, значение LIO зависит от скорости работы процессора, все время стремительно увеличивающейся. Пояснения даны в сноске к табл. 10.2.

Таблица 10.2. Полезные константы для аналитика по производительности [Millsap and Holt (2002)]

Событие	Максимально допустимая задержка на одно событие	Количество событий в секунду при данной задержке
Logical read (LIO) ^a	20 мкс или 0,000 020 с	50000
Single-block disk read (PIO)	10 мс или 0,010 000 с	100
SQL*Net transmission via WAN	200 мс или 0,200 000 с	5
SQL*Net transmission via LAN	15 мс или 0,015 000 с	67
SQL*Net transmission via IPC	1 мс или 0,001 000 с	1000

^a Джонатан Льюис (Jonathan Lewis) опубликовал результаты экспериментов, согласно которым процессор, работающий на частоте 100 МГц, должен выполнять около 10 000 операций LIO в секунду [Lewis (2003)]. Следовательно, при частоте 1 ГГц ожидаемая производительность составит 100 000 LIO/сек, или 10 мкс на операцию. Указанное в таблице значение соответствует частоте процессора 500 МГц.

Задержки, превышающие ожидаемые значения из табл. 10.2, могут свидетельствовать о сбоях в оборудовании, но чаще они вызваны задержками в очередях к ресурсам. Чем вызваны длительные задержки в очереди? Наиболее вероятная причина (вы уже догадались?) – чрезмерный спрос на ресурс. А чем вызван этот чрезмерный спрос? Очевидно, одной или несколькими программами, конкурирующими за те же ресурсы, что и диагностируемая пользовательская операция.

Как справиться с проблемой задержек

В примере 10.6 большое время отклика пользовательской операции вызвано чрезмерными задержками в отдельных вызовах ввода/вывода. К моменту получения этого профиля ресурсов аналитик уже исключил избыточные вызовы дискового чтения, оставив всего восемнадцать действительно необходимых. Однако средняя задержка чтения, составляющая 2,023 секунды на вызов, далека от предельного значения 0,010 секунды, приведенного в табл. 10.2. По одному профилю ресурсов невозможно установить, длится ли каждый из 18 вызовов чтения 2,023 секунды или же какой-то один из них был намного продолжительнее остальных. (Помните, по агрегированным значениям нельзя восстановить составляющие... даже в профилях ресурсов.)

Пример 10.6. Профиль ресурсов пользовательской операции, во времени отклика которой преобладают недопустимо большие задержки дискового ввода/вывода

Response Time Component	Duration		# Calls	Dur/Call
db file sequential read	36.4s	98.9%	18	2.023048s
CPU service	0.4s	1.1%	4	0.091805s
SQL*Net message from client	0.0s	0.0%	3	0.004295s
SQL*Net message to client	0.0s	0.0%	3	0.000298s
Total	36.8s	100.0%		

Не вызывает сомнений только одно: в этой пользовательской операции по крайней мере один вызов дискового чтения выполняется аномально долго. Следующая последовательность шагов поможет нам добраться до истоков проблемы:

1. Какой блок или блоки участвуют в медленных операциях ввода/вывода? Ответ содержится в файле расширенной трассировки SQL. В главах 5 и 6 рассказано, как найти нужную информацию.
2. Выяснив, на каких блоках замедляется операция чтения, можно обратиться к администратору дисковой подсистемы за помощью в определении устройств, на которых они находятся.
3. Определив, на каких именно устройствах возникает проблема, узнайте, не слишком ли расточительно обращаются с ними программы, конкурирующие с диагностируемой операцией. Если это так, то исключение бесполезной нагрузки уменьшит задержки в очереди к перегруженному устройству.
4. Проверьте, не создает ли излишнюю нагрузку конфигурация самого устройства. Вот примеры:
 - Мне встречались системы с двумя и более зеркалами, сконфигурованными так, что узкое место возникало при чтении и записи через единственный контроллер.
 - Системы RAID уровня 5 обычно имеют недостаточную производительность ввода/вывода. Применение массива RAID5 не обязательно является ошибкой. Однако не все отдают себе отчет в том, что для достижения соответствующей производительности ввода/вывода массив RAID уровня 5, как правило, требует в два или в четыре раза больше дисков, чем может показаться на первый взгляд [Millsap (2000a)].
 - Иногда удастся переместить нагрузку с перегруженного устройства на более свободное на проблемном интервале времени. Системные администраторы называют такую операцию *балансировкой ввода/вывода*. В начале 1990-х годов я встречал много систем Oracle, в которых проблема задержек ввода/вывода была вызвана крайне неудачным размещением файлов (когда, например, все файлы базы данных Oracle находились на одном диске). Ду-

маю, теперь такие ошибки встречаются реже. Однако если уж такая безумная конфигурация встретится на вашем пути, то она наверняка будет сопровождаться непомерными задержками ввода/вывода, независимо от того, генерирует приложение избыточные вызовы или нет.

- Разумеется, причиной низкой производительности могут быть и сбои оборудования. Неисправность дискового контроллера, вызывающая перезапросы и тайм-ауты, может намного ухудшить время отклика. Если устройства ввода/вывода работают аномально медленно, проверьте в системном журнале, не тратит ли операционная система время на безуспешные попытки заставить оборудование работать.

При достаточном опыте и творческом подходе шаги 3 и 4 могут дать превосходный результат.

Как выявить конкурирующую нагрузку

Идентификация программ, конкурирующих с пользовательской операцией, чем-то похожа на обычную настройку производительности, во всяком случае, для этого применяются те же инструменты.



Существует множество инструментов для детального анализа различных ресурсов. Для начала отлично подойдут фиксированные представления Oracle, описанные в главе 8.

Детальное изучение подробностей функционирования вызывающего задержки устройства может напомнить старый способ настройки методом проб и ошибок (метод С из главы 1), однако здесь есть важное отличие. Оно связано с уникальной характеристикой метода R – детерминированным *выбором цели*. Не необходимости просеивать бесчисленное множество метрик, гадая, какая из них может оказать заметное влияние на производительность, а какая – нет. Точно известно, какой именно ресурс требует улучшения. И известно это из профиля ресурсов.

Самая сложная часть задачи поиска конкурентов пользовательской операции связана с проблемой сбора диагностических данных в правильной области. Это как раз тот случай, когда очень кстати подробная история всех событий в системе за время существования исследуемой проблемы производительности, аналогичная имеющейся в X\$TRACE. Без такой подробной истории диагностических данных могут возникнуть трудности с определением программ, конкурирующих с пользовательской операцией, даже если с момента ее завершения прошло всего несколько минут. Есть несколько путей для достижения успеха, в частности:

Журналы выполнения пакетных заданий

Практически всегда наибольшую конкурирующую нагрузку в системе создают пакетные задания. В большинстве хороших программ управления пакетными заданиями предусмотрена возможность за-

писи в журнал времени работы каждого из них. Отталкиваясь от этой информации, часто удается быстро определить, какие из программ конкурируют за определенный ресурс. Собрав при следующем плановом запуске этих программ необходимые диагностические данные, можно превратить предположения в точное знание.

Аудит уровня соединения

Экземпляр Oracle может быть легко сконфигурирован так, чтобы записывать в журнал статистику использования ресурсов сеансом. Эта статистика поможет определить, какие сеансы больше всего загружают систему в определенные периоды времени. Обладая такой информацией, можно, поговорив с пользователями, сделать достаточно достоверные предположения о том, какие программы создают конкуренцию за данное устройство. Как и в предыдущем случае, переход от предположений к точному знанию требует сбора диагностики в правильно определенной области для подозреваемых программ при следующем их запуске. Для начала изучите сведения о представлении `DBA_AUDIT_SESSION` в документации Oracle.

Учет использования ресурсов в операционной системе

Некоторые операционные системы предоставляют возможность сбора и хранения статистики производительности для отдельных программ. Это может оказаться очень важным, т. к. конкуренция за некоторый ресурс не обязательно может быть вызвана другим процессом Oracle.

Собственные измерительные средства

Для аналитика по производительности нет ничего лучше, чем код приложения, способный сам рассказать, чем он занят в каждый момент времени. Если можно встроить измерительные средства в медленно работающую программу (допустим, вы сами ее написали), *сделайте* это. Как это сделать, подробно описано в главе 7.

Отыскав программу, конкурирующую с пользовательской операцией за дефицитный ресурс, обратитесь к рекомендациям из предыдущего раздела «Исключение лишних вызовов». Теперь работа сводится к уже рассмотренным действиям, позволяющим выяснить, насколько в действительности оправданны требования к высокой загрузке ресурса.

Модернизация оборудования

Модернизация оборудования – последняя из возможностей, которую следует рассматривать при повышении производительности. Последнее место она занимает по вполне очевидным причинам:

- Дорогостоящая модернизация оборудования редко способна дать тот же эффект, что и недорогие мероприятия по исключению бесполезной нагрузки.

- Плохо спланированная модернизация оборудования может в действительности *ухудшить* производительность пользовательской операции, которую вы пытаетесь ускорить.

Любая модернизация оборудования связана с риском. С первого взгляда понятно, что отдача от инвестиций в более производительное оборудование может оказаться значительно ниже ожидаемой. Многие руководители считают модернизацию беспроигрышным вложением, считая, что «Разве может быть слишком много процессоров (памяти, дисков и т. д.)?». Расхожее мнение гласит, что даже если модернизация напрямую не решит имеющуюся проблему производительности, то как она сможет навредить? Ведь дополнительные мощности так или иначе будут использованы, разве нет? Увы, не всегда. Далеко не все осознают риск, связанный с принятием такого решения. Одну из возможных ситуаций я уже приводил в разделе «Почему правильный выбор так важен». Случай, описанный в разделе «Пример 1: Недостатки общесистемных данных» главы 12 – еще одна иллюстрация этой проблемы:

Модернизация оборудования направлена на повышение производительности *некоторой* части системы, но вопрос в том, поможет ли эта модернизация привести систему в соответствие с *бизнес-целями* ее владельца.

Первое формальное объяснение этого тезиса, противоречащего интуиции, я встретил в книге Нейла Гюнтера (Neil Gunther) «The Practical Performance Analyst» (Аналитик-практик производительности) [Gunther (1998) 117–122]. Когда я продемонстрировал пример Гюнтера участникам международной конференции Oracle, многие стали подходить к трибуне и рассказывать, что этот пример, *наконец*, помог им понять причину странных результатов, погубивших некоторые проекты. Я был польщен, но вместе с тем немного удивлен тем, что так много людей столкнулось с ухудшением производительности в результате модернизации оборудования. Более близкое знакомство с законом Амдала помогло мне понять, что *любая* модернизация может привести к снижению производительности *какой-либо* пользовательской операции вследствие возникновения конкуренции за ресурс, который не был модернизирован. Вопрос только в том, *заметит* ли кто-нибудь эти ухудшения.

Если модернизация оборудования не привела к росту производительности, то худший сценарий для проекта трудно себе вообразить. Вот как это происходит. Компания достаточно долго терпит недостаточную производительность, и, в конце концов, коллективные мучения достигают некоторого порога, за которым следует выделение денег на модернизацию. Размер ожиданий прямо пропорционален выделяемой сумме. В пятницу, перед Большой Модернизацией, вся компания нервничает в ожидании понедельника, ведь «Мы потратили на решение этой проблемы такие деньги, что все должно просто *летать*». И вот, в понедельник, все не только не летает, но и становится сильно *хуже*. Во вторник руководство обсуждает, стоит ли сотруднику, предложившему эту модернизацию, приходить на работу в среду.

Сотрудники, ответственные за принятие решений о модернизации, сталкиваются с интересными противоречиями:

- Модернизация часто рассматривается как недорогая альтернатива дорогим аналитикам, хотя потенциальный выигрыш от нее не сравним с возможным выигрышем от уменьшения нагрузки.
- Модернизация часто считается совершенно безопасной, хотя она связана со значительными рисками. Эти риски требуют проведения серьезного, тщательного и, возможно, дорогого предварительного анализа.

Успешная модернизация обычно не оправдывает возлагавшихся на нее надежд. Если модернизация *не удалась*, опасности подвергается карьера модернизатора. Провалы часто бывают столь оглушительными, что восстановить доверие спонсоров уже не удастся.

Нужно ли вообще модернизировать оборудование? Разумеется, во многих случаях это необходимо. Но я заклинаю вас не рассматривать модернизацию как первое средство от проблем производительности. Вам действительно не хватает мощности системы? Вполне возможно, что она загружена бесполезной работой. Так что, пожалуйста, исключите лишние обращения к ресурсам, прежде чем модернизировать их. А прежде чем приступить к модернизации, обязательно обдумайте следующие вопросы:

Не начинайте модернизацию до тех пор, пока не убедитесь, что модернизируемый ресурс (а) улучшит выполнение важных пользовательских операций и (б) повредит только второстепенным операциям.

И, разумеется, не упускайте из виду тот факт, что второстепенная пользовательская операция завтра может стать очень важной, если ее выполнение замедлится.

Как предсказать результат

Одно из наиболее привлекательных свойств профиля ресурсов состоит в том, что его формат позволяет легко предсказать результат предполагаемых действий. На рис. 10.2 показана простая книга Microsoft Excel, которую можно использовать для этой цели.

В приведенной здесь книге Excel я указал в блоке «Baseline», что компонент под названием «thing to be improved» (улучшаемый параметр) сейчас вызывается 200 раз и это занимает 300 секунд. Другой компонент времени отклика, названный «all other» (все остальное) имеет длительность 100 секунд и вызывается один раз.



Количество вызовов, указанное для компонента «all other» не имеет значения, т. к. мы не планируем оценивать результаты его изменения.

В столбцах с G по K представлены исходные данные в полном формате профиля ресурсов.

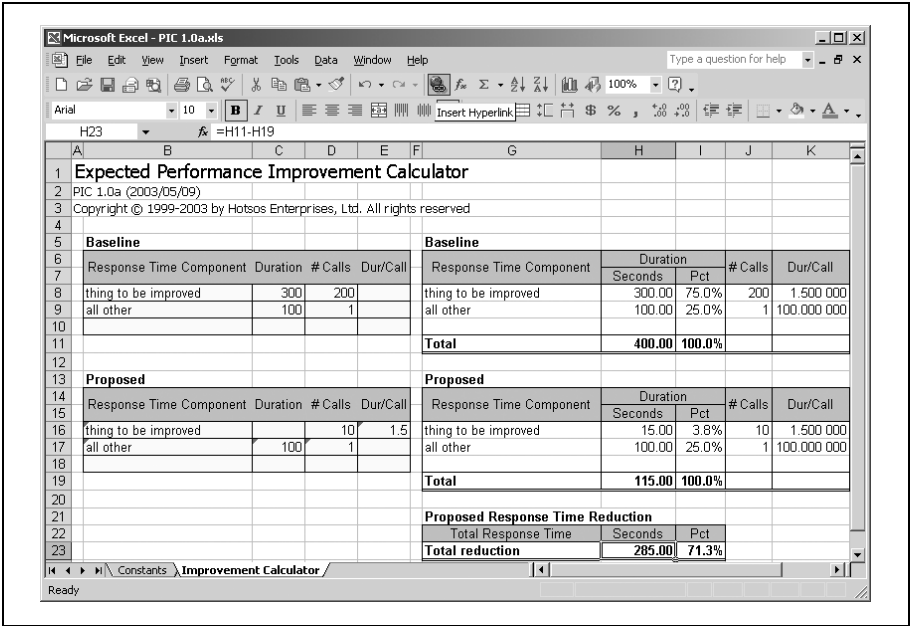


Рис. 10.2. Простой калькулятор повышения производительности, показывающий предполагаемое улучшение времени отклика при уменьшении количества вызовов «улучшаемого параметра» с 200 до 10

В блоке «Proposed» мы указали, что намереваемся сократить количество вызовов компонента «thing to be improved» до 10. В ячейке E16 содержится формула =K8, означающая, что длительность отдельного вызова предположительно не изменится. В столбцах с G по K блока «Proposed» представлены исходные данные и расчетное итоговое значение уменьшенного времени отклика. Если действительно удастся уменьшить количество вызовов «thing to be improved» с 200 до 10, то можно ожидать уменьшения времени отклика на 285 секунд, что на 71% меньше его первоначального значения.

Excel хорошо подходит для такого типа расчетов, т. к. позволяет сразу увидеть, что получится, если попытаться определенным способом повысить производительность. Например, что лучше – сократить количество вызовов «thing to be improved» с 200 до 10 или уменьшить задержку при выполнении отдельного вызова с 1,5 до 0,5 секунды? Из рис. 10.3 видно, что в данном случае лучше всего уменьшить количество вызовов. Разумеется, этот инструмент позволяет оценить влияние на производительность двух этих действий одновременно.



Эта простая модель не учитывает сопутствующей выгоды от уменьшения количества вызовов, которая связана с уменьшением составляющей задержки в очереди в длительности выполнения отдельного вызова.

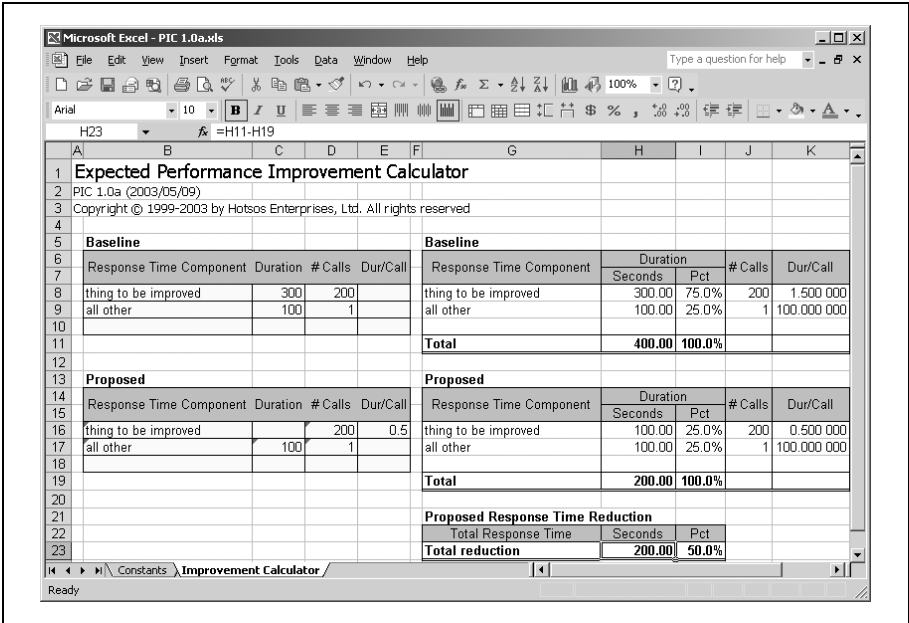


Рис. 10.3. Калькулятор повышения производительности ясно показывает, что улучшение времени отклика за счет уменьшения длительности отдельного вызова с 1,5 до 0,5 секунды для компонента «thing to be improved» меньше показанного на рис. 10.2

Как узнать, что работа завершена

Один из самых серьезных недостатков традиционного метода настройки Oracle заключается в том, что отсутствует критерий завершения. В результате основанные на этом методе проекты могут длиться до тех пор, пока не иссякнет терпение или деньги (по причине возникновения маниакально-настроечного психоза (СТД), упоминавшегося в главе 3). В методе R, наоборот, условие завершения определено:

Если даже наиболее действенные мероприятия не дают достаточного улучшения, отложите оптимизацию производительности до лучших времен.

Как определить, что самые эффективные мероприятия не дают достаточного улучшения? Если можно точно предсказать прибыль, получаемую от проекта повышения производительности, то это элементарная задача финансового анализа. Компания должна вкладывать средства в деятельность, способствующую одновременному повышению прибыли, возврата инвестиций и притока денежных средств. Если финансовая эффективность проекта превышает эффективность всех остальных путей вложения денег компанией, то проект должен быть реализован. В действительности большинство проектов повышения производительности вполне могут обходиться без формального финансового ана-

лиза. В штате многих компаний имеется персонал, решающий задачи повышения производительности там, где в этом возникает необходимость. Во многих случаях штатные аналитики еженедельно несколько часов посвящают вопросам производительности. Расходы, вызванные наличием у аналитика даже острой формы синдрома СТД третьей степени настолько незначительны, что остаются незамеченными в большинстве компаний.

Даже в тех ситуациях, когда аналитики по производительности располагают свободным временем, неудобства доставляет отсутствие возможности определить, действительно ли пользовательская операция *оптимизирована*, т. е. выполняется так быстро, как это возможно. Это практически нельзя вычислить, измеряя производительность по общесистем-ной статистике, как это практикуется в обычных методах настройки. В то же время время отклика, положенное в основу метода R, позволяет узнать, есть ли еще у пользовательской операции резервы повышения производительности. По профилю ресурсов и данным расширенной трассировки, по которым он построен, очень просто определить, есть ли возможность дальнейшего улучшения времени отклика операции.

В примере 10.7 показано, как выглядит профиль ресурсов, когда пользовательская операция *оптимизирована*. Обратите внимание на следующие характерные моменты:

- 1. Общее время отклика невелико. Это обязательное условие. Если общее время отклика недостаточно мало, значит, работа не закончена и не имеет значения, что остальные параметры в норме. «Достаточно малое» значение определяется бизнесом, об этом рассказывалось в главе 4.
- 2. В общем времени отклика преобладает процессорное время (обычно более 80% общего времени отклика), но приложение не расходует его понапрасну.
- 3. Операции чтения и записи файлов базы данных отнимают времени больше, чем любой другой компонент времени отклика, за исключением процессорного времени. Количество вызовов чтения и записи мало.
- 4. Операции, не связанные с использованием процессора и доступом к файлам, занимают очень незначительное время.

Если профиль ресурсов похож на приведенный в примере 10.7 и общее время отклика (❶) достаточно мало, то работа закончена. Если общее время отклика недостаточно мало, а остальные показатели (❷, ❸ и ❹) в норме, то заметно повысить производительность можно только путем замены процессора на более быстрый.

Пример 10.7. Образцовый профиль ресурсов

Response Time Component	Duration	# Calls	Dur/Call

CPU service	❷ [small]s ≥80.0%		

[database reads or writes]	③ [small]s <20.0%	[few]
[everything else]	④ [small]s <10.0%	[few]

Total	① [small]s 100.0%	

Почему в оптимальном профиле ресурсов работа процессора и физический ввод/вывод занимают верхние строчки? Это естественно. Конечно, *что-то* должно быть наверху, если общее время отклика отлично от нуля. Что вы хотели бы там увидеть? В действительности база данных выполняет очень простую работу: управляет данными, находящимися в долговременном хранилище. База данных читает, обрабатывает и записывает данные. В оптимальном профиле ресурсов процессорная обработка опережает физический ввод/вывод по той причине, что проблемы производительности ввода/вывода решаются проще и дешевле (обычно путем оптимизации SQL в приложении). Если решение проблемы стоит недорого, она, скорее всего, будет решена, что уменьшит затрачиваемое из-за нее время, и в верхних строках профиля окажется другое событие. Таким образом, по мере оптимизации производительности системы, процессорное время перемещается в начало профиля ресурсов. Даже если длительность процессорной обработки мала, она обычно занимает верхнюю строчку, т. к. большинство приложений требуют от Oracle значительно больше времени на обработку данных, чем на их чтение и запись.

Вся прочая активность базы данных обычно «необходима, но нежелательна». Например, событие `latch free` сигнализирует об использовании ресурса, призванного предотвратить повреждение ряда внутренних структур данных Oracle в условиях высокой конкуренции [Millsap (2001c)]. Нам необходима эта возможность «сделать паузу, пока защелка недоступна», но наши приложения будут выполняться быстрее, если мы сможем добиться того, чтобы они ожидали события `latch free` как можно реже. Подавляющее большинство так называемых событий ожидания Oracle относятся к категории «необходимая возможность, которой следует избегать».



Основным ресурсом, потребляемым хорошо отлаженным запросом, будет, скорее всего, процессорное время. Однако из этого не следует, что если основной потребляемый запросом ресурс — процессорное время, то он хорошо отлажен. Вполне возможно, что запрос мог бы расходовать гораздо меньше процессорного времени и возвращать при этом верный результат. Наиболее подходящий способ для этого... (барабанная дробь) ...исключение лишних вызовов ЛЮ.

Работа заканчивается, когда расходы на сокращение количества и продолжительности вызовов начинают превышать выигрыш от прироста производительности.

11

Лечение согласно диагнозу

Повышение производительности Oracle невозможно без понимания технологий, стоящих за каждым компонентом, вносящим заметный вклад в общее время отклика пользовательской операции. Их изучение лучше всего начать с руководства Oracle «Database Concepts» на сайте <http://technet.oracle.com/>. Присутствующие в профиле ресурсов компоненты времени отклика прямо соотносятся с измеряемыми операциями ядра Oracle, описанными в упомянутом документе. Например, в нем описано, как процесс LGWR копирует содержимое журнального буфера в файл оперативного журнала. Для учета времени, затраченного на ожидание выполнения данной операции процессом LGWR, ядро Oracle использует событие ожидания `log file sync`.

Существует множество подобных событий ожидания. Их количество в ядре Oracle растет от версии к версии (табл. 11.1). К счастью, вам не обязательно разбираться в деталях каждого события ожидания Oracle. Обычно не требуется одновременно держать в памяти сведения о более чем двух событиях ожидания – тех, которые доминируют в пользовательской операции, с которой вы в данный момент работаете. И это хорошо, т. к. изучение некоторых событий требует времени. Думаю, чем изучать и держать в памяти многочисленные тонкости десятков событий, важнее сосредоточиться на следующем:

- Понимать, как правильно *выбрать* события, наиболее важные в данный момент. Об этом рассказывается в первой части книги.
- Знать в общих чертах смысл часто встречающихся в вашей системе компонентов времени отклика, включая:
 - CPU service
 - unaccounted-for
 - SQL*Net message from client
 - Различные события чтения read

- Дополнительно одно или два события, характерные для вашей системы

Эти знания можно почерпнуть из данной книги, руководства Oracle «Database Concepts», а также изучая на практике время отклика исследуемых пользовательских операций своей системы.

- Знать, где при необходимости можно найти подробную информацию о компонентах времени отклика. Я предпочитаю получать эту информацию из следующих источников:
 - Документация по продуктам Oracle на сайте <http://technet.oracle.com/>
 - Бюллетени технической поддержки Oracle MetaLink и сообщения об ошибках на сайте <http://metalink.oracle.com/>
 - Статья «YAPP» Аньо Колка (Anjo Kolk) и Шари Ямагучи (Shari Yamaguchi) [Kolk and Yamaguchi (1999)]
 - «Книга со шмелем» Стива Адамса (Steve Adams) [Adams (1999)] и его сайт <http://www.ixora.com.au>¹
 - Поисковый сайт Google <http://www.google.com>, помогший мне в поисках информации о событиях ожидания в Интернете

Таблица 11.1. Количество событий ожидания возрастает в каждой новой версии Oracle (источник: `select count(*) from v$event_name`)

Версия Oracle	Количество событий ожидания
7.3.4	106
8.1.7	215
9.0.1	287
9.2.0	361
10.0.1	500 (предположительно)

За пределами профиля ресурсов

Профиль ресурсов – превосходное средство, с которого можно начать поиск компонента времени отклика пользовательской операции, требующего дальнейшего анализа. Но что делать после того, как основной компонент времени отклика найден? Ответ прост:

Найдите исходный текст (SQL или PL/SQL) курсора, вносящего наибольший вклад в продолжительность этого компонента.

Как рассказывалось в разделе «Опережающее атрибутирование» главы 5, это легко сделать с помощью данных расширенной трассировки

¹ Автор, по-видимому, считает, что «книга со шмелем» Стива Адамса известна каждому уважающему себя специалисту по производительности Oracle. На всякий случай напоминаю, что эта книга называется «Oracle8i Internal Services». – *Примеч. науч. ред.*

SQL. Отыскав код SQL курсора, вносящего наибольший вклад в основной компонент времени отклика, можно приступить к созданию лекарства. Следующим шагом должно стать выяснение причины, по которой найденный код вызывает такие затраты времени в компоненте. Как это сделать, рассказывается в следующем разделе.

Компоненты времени отклика

Начиная работу над этой главой, я предполагал подробно описать около двадцати событий ожидания Oracle, которые с наибольшей вероятностью могут встретиться в вашей работе. Однако за последние несколько лет корпорация Oracle так хорошо потрудились над совершенствованием документации по событиям ожидания, что повторение здесь этой информации стало бы пустой тратой времени. Документация по настройке производительности Oracle9i release 2 [Oracle (2002)] написана достаточно хорошо. Вместо того чтобы пересказывать то, что вы можете свободно найти в других источниках, я в этой книге сосредоточил усилия на трех направлениях, не освещенных в стандартной документации Oracle:

- Рассмотрение двух псевдособытий («событий», в действительности таковыми не являющихся), длительность которых может быть определена по данным файла расширенной трассировки SQL: CPU service и unaccounted-for.
- Более полное описание нескольких так называемых *событий простоя*, которое другие авторы опускают, не считая их заслуживающими внимания. Эти события приобретают исключительную важность в свете дополнительных возможностей диагностики, которые дают правильный выбор области сбора диагностических данных.
- Акцент на *исключении бесполезной нагрузки* при возникновении различных событий ожидания.

Данный раздел посвящен псевдособытиям и так называемым событиям простоя. Ниже в этой главе будет рассказано об исключении нагрузки.

Псевдособытия Oracle

Вы, наверное, уже заметили, что там, где можно было бы ожидать термина «событие ожидания» Oracle, употребляется термин «компонент времени отклика». Причина проста. То, что называется компонентами времени отклика, состоит из двух разных вещей: настоящих событий ожидания Oracle, описанных в V\$EVENT_NAME, и двух других важных компонентов, отсутствующих в этом представлении:

```
CPU service  
unaccounted-for
```

Несмотря на то, что ни один из этих компонентов формально не относится к «событиям ожидания Oracle», оба они представляют собой *из-*

меримые (и часто существенные) составляющие времени отклика любой пользовательской операции. Они рассматриваются здесь наравне с событиями ожидания, потому что каждая микросекунда, затраченная ядром Oracle на «работу», добавляет ко времени отклика пользовательской операции ровно столько же, сколько и микросекунда, потраченная на «ожидание». Последние версии *Statspack* также включают процессорное время в число пяти основных «событий ожидания».

CPU service

Практически каждый профиль ресурсов Oracle будет содержать компонент *CPU service*, показывающий затраты процессорного времени. Часто он вносит основной вклад в значение времени отклика как для эффективно реализованных пользовательских операций, так и для совершенно неэффективных. Вопрос в том, насколько оправдана такая потребность в процессорном времени (см. главу 5).

При диагностике причин чрезмерных затрат процессорного времени прежде всего надо установить, какие вызовы базы данных ответственны за это в наибольшей степени. Для этого необязательно применять опережающее атрибутирование, т. к. статистики *c*, характеризующие затраты процессорного времени, находятся непосредственно в тех строках файла трассировки, которые соответствуют вызовам базы данных. Отыскав вызовы, потребляющие основную часть процессорного времени, нетрудно найти и соответствующие секции *PARSING IN CURSOR*, содержащие исходный код (на SQL или PL/SQL), породивший эти вызовы.

В первую очередь следует обратить внимание на вызовы базы данных, имеющие наибольшие значения *c* без учета своих потомков. Если большие значения *c* для вызова базы данных возникают вследствие суммирования затрат времени их рекурсивным потомством, то затраты процессорного времени без учета потомства определяются способом, описанным в разделе «Двойной учет рекурсивного SQL» главы 5. Вспомните, статистики потребления процессорного времени и операций ЛЮ для вызова базы данных (*c*, *cr* и *cu*) относятся к тем, которые включают в себя затраты, сделанные их потомством.

Если вызовы, в наибольшей степени ответственные за продолжительность компонента *CPU service* вашей пользовательской операции, найдены, то дальнейшие действия (в зависимости от типа вызова базы данных) таковы:

Большое значение «с» для вызова FETCH, EXEC, UNMAP или SORT UNMAP

Если много небольших значений *c* распределено среди значительного количества вызовов *FETCH* или *EXEC*, исключите лишние вызовы базы данных везде, где это возможно, и объедините оставшиеся вызовы в наименьшее возможное количество вызовов (например, обрабатывайте строки массивами, а не по одной).

Если большое значение *c* принадлежит отдельному вызову базы данных `FETCH` или `EXEC`, то сначала выясните, не связана ли загрузка процессора с операциями логического ввода/вывода (LIO):

Большое количество операций LIO (ориентировочно больше 10 вызовов LIO на каждую неагрегированную строку каждой таблицы, указанной во фразе FROM)

При большом количестве операций LIO в вызове оптимизируйте SQL, как описано ниже в разделе «Оптимизация логического ввода/вывода» этой главы.

Малое количество операций LIO

Если количество операций LIO невелико, то возможно, что серверный процесс Oracle тратит много процессорного времени на операции сортировки или хеширования. События 10032 и 10033 предоставляют подробную информацию об операциях сортировки, а событие 10104 – о хеш-соединениях.

Возможно также, что большие затраты времени вызваны операциями приведения типов. Например, просмотр таблицы со сравнением дат в каждой строке может очень сильно загрузить процессор.

Наконец, время может расходоваться на выполнение излишне усложненного кода PL/SQL. Понятно, что выполнение инструкций PL/SQL требует работы процессора, даже если они не содержат обращений к базе данных. Большие значения *c* в строках файла трассировки `EXEC` для PL/SQL-блоков при малом количестве LIO (без учета потомков) часто означают, что время расходуется на выполнение ветвления, присваивания и других инструкций языка. Для детальной диагностики выполнения кода PL/SQL можно воспользоваться имеющимся в Oracle пакетом `DBMS_PROFILER` [Kyte (2001)].

Большое значение «с» для вызова PARSE

Если много небольших значений *c* распределено среди большого количества вызовов `PARSE`, исключите как можно больше таких вызовов, следуя методике, изложенной ниже в разделе «Оптимизация разбора» этой главы.

Если большое значение *c* принадлежит отдельному вызову `PARSE`, то выясните, нельзя ли упростить команду SQL. Рассмотрите также возможность уменьшения значения параметра `OPTIMIZER_MAX_PERMUTATIONS` (дополнительная информация доступна по адресу <http://www.ixora.com.au/q+a/0010/19140702.htm>).

unaccounted-for

Как рассказывалось в главе 7, существуют пять источников неучтенного времени в файле трассировки, соответствующего ненулевому значению Δ в уравнении $e = c + \sum e_i + \Delta$:

- Эффект влияния измерителя
- Двойной учет использования процессора
- Ошибка квантования
- Время, в течение которого процесс не выполняется
- Неизмеряемое время

Наличие такого количества неопределенностей в одном уравнении может вызвать сомнения в возможности определения вклада каждой из составляющих в Δ . На самом деле работать с неучтенным временем `unaccounted-for` достаточно просто. Три из пяти его составляющих имеют ограничения, позволяющие считать их влияние на время отклика пренебрежимо малым. Если неучтенное время дает наибольший вклад в профиль ресурсов пользовательской операции и данные получены в *корректно определенной области*, то оно практически всегда вызвано наличием *неизмеряемого времени* или *времени «невыполнения»*. Если источником неучтенного времени является отсутствие средств измерения в коде ядра Oracle, то, возможно, установка очередного обновления сократит количество реальных источников неопределенности до одного.



На практике большие значения неучтенного времени, как правило, вызваны ошибками сбора данных, подобными тем, что описаны в главах 3 и 6. При аккуратном и *точном* выборе данных в процессе сбора диагностической информации можно полностью использовать те преимущества, которые дает обладание сведениями о продолжительности компонента `unaccounted-for`.

Подробно об этих пяти составляющих компонента времени отклика `unaccounted-for` рассказывалось в главе 7. Резюмируем сказанное там:

Эффект влияния измерителя

Эффект влияния измерителя незначителен (порядка нескольких микросекунд на вызов `gettimeofday` или `getrusage`), поэтому в целом им обычно можно пренебречь. Если влияние измерителя вас беспокоит, то можно измерить его при помощи методики, изложенной в главе 7. Этот эффект вызывает небольшое увеличение Δ .

Двойной учет использования процессора в статистиках с и ela

Эффект двойного учета в большинстве событий невелик. Наибольшее влияние этого эффекта, виденное мной, проявлялось в событиях `db file scattered read`, перемещавших большие объемы данных (порядка 100 Кбайт и больше). В таких ситуациях наблюдался двойной учет примерно десяти миллисекунд на одно событие чтения. Двойной учет процессорного времени немного уменьшает значение Δ .

Ошибка квантования

Совокупный эффект, вызванный ошибкой квантования, также мал. В силу того, что вероятности возникновения положительных и отрицательных ошибок квантования одинаковы, эти ошибки ком-

пенсируют друг друга, в результате чего их совокупное влияние на Δ близко к нулю.

Время, в течение которого процесс не выполняется

Если в профиле ресурсов длительность компонента `unaccounted-for` велика, то почти наверняка вы столкнулись либо со временем «невыполнения», либо с неизмеряемым временем. В табл. 11.2 приведено хорошее эмпирическое правило. Время, в течение которого процесс не выполняется, всегда увеличивает Δ .

Неизмеряемое время

Если приложение выполняет значительный объем кода ядра Oracle, в котором отсутствуют средства измерения, то эффект неотличим от того, который вызван временем невыполнения процесса. В главе 7 рассказывалось, как обнаружить наличие неизмеряемых участков кода. Если даже в системе, не обремененной многочисленными переключениями контекста или подкачкой страниц, во времени отклика преобладает неучтенное время, то следует обновить ядро Oracle, чтобы добавить измерительные средства в участки кода, выполняемые пользовательской операцией. Неизмеряемый код всегда вызывает изменение Δ в сторону увеличения.

Таблица 11.2. Эмпирическое правило работы с неучтенной длительностью Δ ; здесь R – общее время отклика анализируемой пользовательской операции

Условие	Рекомендация
Величина Δ отрицательна и составляет не более 10% полного времени отклика. То есть $\Delta < -0,1R$	Очень редкий (но теоретически возможный) случай, когда ошибка, вызванная двойным учетом, преобладает в статистике файла трассировки. (Обычно этот эффект можно обнаружить, проанализировав данные расширенной трассировки для единственного вызова базы данных.)
Величина Δ находится в пределах от -10% до +10% полного времени отклика, то есть $-0,1R \leq \Delta \leq +0,1R$	Компонент <code>unaccounted-for</code> можно игнорировать. Его вклад в полное время отклика настолько мал, что нет необходимости выяснять причину его возникновения.
Величина Δ положительна и превосходит 10% полного времени отклика. То есть $+0,1R \leq \Delta$	Если длительность <code>unaccounted-for</code> не преобладает в полном времени отклика, ее можно игнорировать. В противном случае, если пользовательская операция небольшую часть времени проводит в состоянии готовности к выполнению (глава 7), обратитесь к статьям Oracle MetaLink за информацией о неизмеряемом коде ядра Oracle. Если пользовательская операция проводит в состоянии готовности к выполнению слишком много времени, то, вполне вероятно, что производительность процессора или объем памяти недостаточны для имеющейся нагрузки.

Не бывает «неважных» событий

Авторы большинства публикаций тщательно различают события *простая* и события «*непростая*». При этом такие авторы обычно описывают события «непростая» как важные, а события простая – как не важные. Но я лично призываю вас применять лишь одну классификацию для событий ожидания Oracle: различать их по признаку *внутри/между*, о котором мы говорили в главе 5. Осознание данного отличия необходимо для использования фундаментального отношения между статистиками файла трассировки: $e \in c + \Sigma ela$, не допуская как двойного учета времени, так и его пропуска. Я призываю отказаться от других классификаций событий ожидания Oracle, потому что *любое* событие может внести важный вклад в общее время отклика.

Существует единственный законный критерий определения важности составляющей времени отклика:

Если составляющая вносит значительный вклад во время отклика корректно выбранной пользовательской операции, то она важна; в противном случае *t.* можно пренебречь.

Поэтому я не согласен с заявлениями о том, что какие-то события важны, а какие-то другие – нет. *Любое* событие может быть важным. Не имеет значения, что это за событие, и сколько авторов высказались в пользу того, что оно не важно. Если событие вносит значительный вклад во время отклика корректно выбранной пользовательской операции, то оно является важным для нас.

В каком случае события простая могут оказаться важными? Если время, потраченное на их исполнение, вносит значительный вклад в формирование времени отклика. Практически в любом профиле ресурсов встречаются два события ожидания: SQL*Net message to client и SQL*Net message from client. Событие *from client* бесспорно относится к так называемым событиям простая. Ядро Oracle использует события *to client* и *from client* для измерения производительности межпроцессного взаимодействия, которое реализуется через SQL*Net. В примере 11.1 показано, как эти события работают внутри ядра Oracle.

Пример 11.1. Типичный фрагмент кода, выполняемого ядром Oracle по завершении вызова базы данных

```
# здесь завершается вызов базы данных

# результат вызова БД передается клиенту через SQL*Net
ela0 = gettimeofday;
write(SQLNET, ...);
ela1 = gettimeofday;
nam = 'SQL*Net message to client';
ela = ela1 - ela0;
printf(TRCFILE, "WAIT %d: nam='%s' ela= %d' ...", cursor, nam, ela, ...);
printf(TRCFILE, "\n");

# прослушиваем SQL*Net на предмет дальнейших инструкций от клиента
```

```
ela0 = gettimeofday;
read(SQLNET, ...);
ela1 = gettimeofday;
nam = 'SQL*Net message from client';
ela = ela1 - ela0;
printf(TRCFILE, "WAIT #%d: nam='%s' ela= %d' ...", cursor, nam, ela, ...);
printf(TRCFILE, "\n");
```

здесь начинается обработка следующего вызова БД

О простое

О событиях простоя упоминается в любой хорошей книге об интерфейсе ожидания Oracle. Авторы специально выделяют события простоя в связи с тем, что они представляют собой фрагмент кода ядра Oracle, в котором процесс ядра ожидает инструкции на выполнение некоторого действия. Средства системного мониторинга преднамеренно игнорируют статистики для событий простоя. Например, утилита Oracle *Statpack* включает в себя таблицу STAT\$IDLE_EVENT, содержащую названия событий, которые администраторы баз данных обычно опускают в отчетах *Statpack* относительно V\$SYSTEM_EVENT. К таким событиям относятся:

```
dispatcher timer
lock manager wait for remote message
pipe get
pmon timer
PX Idle Wait
PX Deq Credit: need buffer
PX Deq Credit: send blkd
rdbms ipc message
smon timer
SQL*Net message from client
virtual circuit status
```

В разделе «Проблема „событий простоя“» главы 8 сказано, что ряд авторов выделяет различные события ожидания в разряд «простоя» для решения проблемы, вызванной некорректным сбором диагностических данных. Некорректный выбор диагностических данных приводит к тому, что аналитики не учитывают события простоя, тогда как они содержат жизненно важную информацию. Если же диагностические данные собраны корректно, то лучше уже не отвлекаться на размышления о том, является ли какое-то событие «событием простоя». Вместо того, чтобы заниматься классификацией событий по признаку «простой»/«не простой», я разделяю события ожидания на происходящие *внутри* вызовов базы данных и *между* такими вызовами.

Неверной будет и попытка уравнивать то, что другие авторы называют событиями простоя, моим событиям *между* вызовами. События *между* вызовами и события простоя – это не одно и то же. Например, событие PX Deq Credit: need buffer выполняется внутри вызова базы данных. Другие события, также часто относимые к событиям простоя: SQL*Net more data from client и SQL*Net more data from dblink, также исполняются в контексте вызовов базы данных, а не между ними.

События SQL*Net message to client обычно занимают всего несколько микросекунд. Но события SQL*Net message from client могут выполняться гораздо дольше, например:

Время раздумий пользователя

Если вы подключаетесь к серверу Oracle в 08:00 и до 10:00 не выполняете ни одного вызова базы данных, то Oracle запишет 7200 секунд для SQL*Net message from client в V\$SYSTEM_EVENT.TIME_WAITED в 10:00.

Время выполнения клиентской программы

Отчет *Financial Statement Generator* в Oracle Applications (пакетная программа, работающая как клиентский процесс Oracle) обычно выполняет несколько вызовов базы данных, а затем достаточно долгое время занимается обработкой полученных данных в клиентской программе на С. С точки зрения серверного процесса Oracle это время, потраченное клиентом, воспринимается просто как время, связанное с вызовом чтения, показанным в примере 11.1, поэтому оно записывается на счет SQL*Net message from client.

Ожидание между вызовами

Даже если приложение быстро выполняет два последовательных вызова базы данных, между ними часто имеет место событие SQL*Net message from client, вызывающее задержку в сотни микросекунд.

Необходимо обратить внимание на пару моментов. Во-первых, просто посмотрев на диагностические данные, мы не можем сказать, какое «время простоя базы данных» в действительности является временем отклика какой бы то ни было программы, а какое «время простоя базы данных» просто соответствует тому времени, которое пользователь провел, не глядя на монитор. Чтобы узнать это, необходимо наблюдать за пользователем. Во-вторых, несмотря на то, что ожидание между последовательными быстро следующими друг за другом вызовами БД составляет всего несколько микросекунд, в итоге все это время суммируется. Например, даже если средняя продолжительность события SQL*Net message from client составляет мизерные 500 микросекунд, 1 000 000 вызовов базы данных приведут к формированию 500-секундного (а это 8,3 минуты) времени отклика.

Работа с событиями SQL*Net, заметно влияющими на время отклика

Сделанные наблюдения помогают понять, как следует работать с профилем ресурсов, в котором преобладают длительности событий между вызовами базы данных. Если профиль ресурсов состоит в основном из событий, происходящих между вызовами базы данных, необходимо сделать следующее:

1. Убедиться в том, что продолжительность события между вызовами действительно является составляющей какого-то времени отклика.

Если это не так, то следует исправить ошибку сбора данных и построить новый профиль ресурсов.

2. Если вклад события между вызовами в значение времени отклика велик из-за наличия нескольких больших задержек между вызовами, то следует проверить, почему приложение провело так много времени между вызовами.
3. Если же вклад события между вызовами в значение времени отклика велик из-за большого количества вызовов события, то следует проверить, зачем приложение выполняет так много различных вызовов базы данных.
4. Если не удастся уменьшить количество различных вызовов базы данных, то следует проверить, можно ли уменьшить среднюю продолжительность отдельного события ожидания. Например, избавиться от ненужных вызовов базы данных других процессов, которые могут приводить к образованию очередей к сетевым ресурсам.



Заметьте, что эта последовательность шагов полностью совпадает с процедурой, описанной в главе 5. Сначала избавляемся от ненужных вызовов, а затем от ненужной конкуренции между процессами.

Вероятнее всего, в профиле ресурсов будут преобладать длительности таких событий SQL*Net:

*SQL*Net message from client*

Средство от этого недуга заключается в удалении максимального количества ненужных вызовов базы данных. Например, удаление излишних вызовов разбора (подробная информация приведена в разделе «Оптимизация разбора»). Обратитесь к возможностям обработки массивов Oracle, позволяющим в одном вызове работать с множеством строк, а не с одной.¹

*SQL*Net more data from client*

Если вы в своих приложениях передаете в вызовы разбора огромные текстовые строки SQL, прекратите это делать. Вызывайте из приложения хранимые процедуры. Однако если большие массивы значений отправляются для связывания в заполнители («переменные связывания»), то, возможно, не удастся сократить время ожидания данного события без создания новых, более серьезных трудностей.

*SQL*Net message from dblink*

Подумайте о возможности локальной репликации данных вместо удаленной работы по каналу базы данных. Репликация может быть

¹ Этот же совет может быть полезен и для избавления от событий single-task message. Событие single-task message не является событием SQL*Net, но оно используется в однозадачной конфигурации аналогично событию SQL*Net message from client в двузадачных конфигурациях.

весьма эффективной, особенно для таблиц, которые изменяются очень редко или для которых использование слегка устаревших данных создает лишь незначительное неудобство. Если переработать план выполнения команды SQL так, чтобы между экземплярами передавалось меньше данных, то можно сократить потребность в выполнении подобных событий.



Для оценки количества пересылок по сети, генерируемых приложением, можно подсчитать количество выполнений событий, имя которых включает в себя строку SQL*Net. Исследование значений `ela` для таких событий позволит ответить на вопрос о том, значителен ли вклад сетевых задержек в общее время отклика.

Работа с другими событиями, заметно влияющими на время отклика

Иногда вам приходится сталкиваться с чрезмерно большим вкладом во время отклика со стороны события, о существовании которого вы раньше даже не подозревали. Существует множество событий ожидания, на которых я не буду здесь подробно останавливаться. Однако в любом случае метод решения задачи не меняется, даже если главная проблема производительности вызвана чем-то, о чем вы ничего не знаете. Последовательность шагов вам уже знакома:

1. Определяем, какое событие потребляет большую часть времени отклика пользовательской операции.
2. Определяем, какая операция приводит к столь частому выполнению события.
3. Делаем так, чтобы приложение реже выполняло данную операцию.

Выполнение первого шага подробно описано в этой книге. Шаги 2 и 3 не представляют особых трудностей. Значение событий обычно отражается в их названиях. Например, события PX генерируются фрагментами кода, отвечающими за параллельное выполнение Oracle (parallel execution). Если вы знаете, как реализуется параллельное выполнение, вам будет понятен смысл событий PX: главный сеанс поручает работу подчиненным сеансам и ожидает результатов. Большую часть работы выполняют подчиненные. Рассмотрим еще один пример: событие запроса к глобальному кэшу `global cache cr request` – это то, что выдает процесс ядра Oracle, поддерживающий технологию RAC (Real Application Clusters), когда ему необходим доступ к буферу базы данных, удерживаемому другим экземпляром RAC. Как освободиться от выполнения запросов к глобальному кэшу? Требовать меньшего количества буферов от другого экземпляра или меньше обращаться к буферам вообще (см. далее раздел «Оптимизация логического ввода/вывода»).

Даже если вы никогда не слышали о подобном событии, и его имя ни о чем вам не говорит, метод R помогает найти решение. В начале главы я уже ссылался на ряд бесплатных источников информации в Ин-

тернете. И даже в худшем из случаев, который только можно себе представить, все не так плохо. Если вся глобальная сеть не в силах вам помочь, звоните в службу поддержки корпорации Oracle:

Звонок в службу поддержки (метод R не применяется): Моя система работает медленно. Мы в отчаянии и не понимаем, почему все так плохо. Как нам решить проблему?

Ответ, конечно, зависит от того, кто из аналитиков вам ответит, но будьте готовы к тому, что ничего конкретного не узнаете. Если же обратиться к методу R, то вопрос в службу поддержки можно сформулировать гораздо точнее:

Звонок в службу поддержки (с применением метода R): Время отклика самой важной пользовательской операции в моей системе составляет 75,32 секунды. Больше 73 секунд из этого времени тратятся на выполнение события под названием `resmgr:waiting in check2`. Не подскажите ли, во-первых, какая операция моего приложения приводит к выполнению данного фрагмента кода ядра Oracle? И, во-вторых, что можно сделать для того, чтобы выполнять его реже?

Это надо выучить наизусть: выяснить, почему событие было выполнено; выяснить, как избежать этого в следующий раз. Такой подход действительно эффективен.

Исключение ненужной работы

Документация корпорации Oracle, посвященная событиям ожидания, очень изменилась с тех времен, когда руководство по настройке сервера Oracle7 предлагало до смешного плохие советы пользователям нового представления `V$SESSION_WAIT` (см. главу 8). Однако, как мне кажется, существует одна область, которой корпорации Oracle следовало бы уделить больше внимания – *снижение рабочей нагрузки*. Я думаю, что одна из причин, по которой Oracle не говорит о снижении нагрузки применительно к событиям ожидания, заключается в том, что события ожидания упоминаются в этом руководстве в главе под названием «Instance Tuning» (Настройка экземпляра). Вероятно, такое название заставило автора ограничиться описанием действий по «настройке», не требующих изменения приложения.

Однако нас с вами ничто не ограничивает. Даже если вы имеете дело со сторонним программным обеспечением, которое нельзя менять без участия поставщика, не забывайте о возможности повышения производительности за счет снижения рабочей нагрузки. Конечно, в некоторых случаях избавление от ненужной нагрузки может потребовать изменения приложения. Не падайте духом. Часто оказывается, что убедить поставщика программного обеспечения усовершенствовать производительность поставляемого приложения совсем несложно.



Наилучшая возможность убедить поставщика прикладного программного обеспечения улучшить производительность приобретенного у него приложения состоит в том, чтобы предоставить неоспоримые количественные аргументы в пользу того, что повышение производительности приложения делает вас (и других клиентов вашего поставщика) счастливее.

Главное преимущество метода R заключается в том, что чем бы ни было вызвано ухудшение производительности пользовательской операции, эта причина будет определена. Огорчительно ли известие о том, что поставщик программы, проектируя ее, сделал ужасную ошибку, и что счастья пользователям не видать, пока она не будет исправлена? Возможно. Но если и правда единственный, кто может способствовать повышению производительности приложения – это его поставщик, то узнать об этом надо как можно раньше, чтобы не тратить время и деньги на заведомо безрезультатные действия.

Последующие разделы дополняют информацию о событиях ожидания, которую можно найти на сайтах компании Oracle и в других интернет-ресурсах. В каждом из разделов описано несколько способов избавления от ненужной нагрузки, а также показано, как такая нагрузка проявляется в составе времени отклика пользовательской операции.

Оптимизация логического ввода/вывода

Многие операции с данными в Oracle выполняются в области *кэша буферов базы данных*, расположенного в наборе сегментов разделяемой памяти ядра Oracle, который называется *системной глобальной областью*. Поэтому все аналитики производительности уделяют внимание тому, что происходит в кэше буферов базы данных. Сообщается о том, что доступ к буферам кэша осуществляется из сотен различных фрагментов кода ядра Oracle [Lewis (2003)]. Из всех таких обращений к буферам наибольшие затраты связаны с *операциями логического ввода/вывода Oracle (LIO)*. Количество LIO для вызова базы данных равно сумме значений его статистик `sg` и `su` из данных трассировки SQL.

Практически в каждой из Oracle-систем, которые я встречал, более 50% общего процессорного времени, отданного приложениям Oracle, занимали ненужные вызовы LIO. Во многих случаях можно избавиться от более чем 90% нагрузки на систему без потери какой бы то ни было функциональности для бизнеса.



Избыточные обращения к буферам для базы подобны патологическому ожирению для человека. Как лишние двадцать фунтов жира вредят работе почти любой подсистемы организма (кровеносной, почечной, костно-мышечной, зрительной...), так и избыточные вызовы LIO ухудшают производительность практически любой подсистемы приложения Oracle.

Обращение к слишком большому количеству буферов вызывает дополнительную нагрузку на процессор и приводит к росту времени, проведенному вне исполнения, проявляющемуся в больших неучтенных длительностях. Избыточные операции LIO приводят к возникновению ожиданий освобождения зацелок в цепочках буферов кэша (cache buffer chains) и служат причиной избыточных системных вызовов чтения, проявляющихся в виде событий db file sequential read или db file scattered read.

Многие события ожидания Oracle могут служить иллюстрацией вредных побочных эффектов ненужных обращений к буферам. Например, *любая* производительность ухудшается по мере увеличения очереди к процессору. Событие ожидания Oracle log file sync – один из первых признаков возрастания задержек, возникающих в очереди к процессору. Негативные последствия избыточных обращений к буферам могут проявиться в абсолютно неожиданных местах. Например, если избыточные обращения к буферам приводят к усилению конкуренции за дисковый ввод/вывод, процессы записи DBWR могут стоять в очереди позади вызовов чтения. Если процесс DBWR не успевает за изменениями буферов, то в приложениях появляются события ожидания free buffer waits, write complete waits и даже log file switch (checkpoint incomplete). Зачастую оказывается, что причина проблем с ожиданием освобождения буфера buffer busy waits кроется в избыточном количестве операций LIO. Ошибки, приводящие к ненужным операциям LIO, могут вызвать даже лишние выполнения событий SQL*Net message from client.

Причины частых проблем с LIO

Существует ряд причин, по которым столь многие системы страдают от избыточных вызовов LIO. Одна из них заключается в том, что всех нас учили, что обращение к памяти выполняется гораздо быстрее, чем к диску, из чего следовало, что в бесконечных обращениях к памяти нет ничего страшного [Millsap (2001c)]. Более глубокая причина того, что огромное количество приложений Oracle страдает от избыточности операций LIO, состоит в том, что пользователи могут создать проблему множеством способов. Рассмотрим небольшой пример:

Пользователи приложений

Пользователи приложений могут заставить их выполнять ненужные обращения к буферам разными способами. Можно выполнять запросы, не пользуясь фильтрами, например, поиск продавца с названием «Хегох» можно осуществлять посредством запроса «вслепую» вместо того, чтобы наложить на имя ограничение вида X%. Можно запрашивать отчеты, не указывая аргументов, например можно запустить формирование отчета по продажам всей компании начиная с ее основания вместо того, чтобы сформировать нужный отчет по своему отделу за определенный месяц. Кроме того, в те моменты, когда система начинает работать медленнее из-за из-

бытка вызовов ЛЮ, пользователи могут отправлять на исполнение одну и ту же операцию несколько раз, в результате чего одни и те же вызовы ЛЮ выполняются по несколько раз.

Администраторы приложений

Ошибки, приводящие к ненужным обращениям к буферам, могут совершать и администраторы приложений. В конфигурируемых приложениях, например в Oracle e-Business Suite, способы организации плана бухгалтерских счетов и настройки приложения могут серьезно повлиять на количество вызовов ЛЮ, совершаемых типовыми бизнес-функциями. Некоторые приложения, подобные продукту Oracle General Ledger, имеют собственные встроенные средства оптимизации запросов. Применяя подобные средства и пренебрегая при этом внимательным изучением их влияния на производительность, можно породить много ненужных вызовов ЛЮ. Ошибки администраторов приложений в деле архивирования и очистки данных могут привести в приложение миллионы ненужных вызовов ЛЮ.

Администраторы экземпляров

К ошибкам администратора экземпляра, которые могут привести к избыточным вызовам ЛЮ, относится неудачный выбор десятков параметров, подобных `HASH_AREA_SIZE` и `DB_FILE_MULTIBLOCK_READ_COUNT`, влияющих на работу оптимизатора запросов Oracle по стоимости.

Администраторы данных

Администраторы данных могут инициировать избыточные вызовы ЛЮ огромным количеством способов. Вероятно, самый распространенный из них – предоставление некорректной информации о таблицах и индексах оптимизатору Oracle по стоимости (Cost-Based Query Optimizer – СВО) из-за того, что статистика собрана неправильно. Таблицы, страдающие миграцией или сцеплением строк в тяжелой форме, порождают больше вызовов ЛЮ. То же самое происходит, если неудачно выбраны значения `PCTFREE` и `PCTUSED`. Отсутствие в соответствующих ситуациях индексированных таблиц, кластеров и разделов может привести к обработке излишних ЛЮ. Отсутствие ограничений (например, не назначены столбцы, выступающие в качестве первичных или внешних ключей, не определено, какие столбцы допускают использование `NULL`, а какие – нет) может помешать оптимизатору запросов Oracle использовать планы выполнения, ограничивающие количество вызовов ЛЮ. Конечно же, недостаточное количество индексов (или просто *неправильные* индексы) может вызвать ненужные вызовы ЛЮ для запросов, а избыточное количество индексов может привести к выполнению лишних вызовов ЛЮ в командах `INSERT`, `UPDATE` и `DELETE`.

Разработчики приложений

Естественно, что решения разработчиков приложений оказывают огромное влияние на количество ЛЮ. Некоторые виды конструкций

SQL делают невозможным применение эффективного плана выполнения запроса ядром Oracle. Например, наличие в инструкции WHERE предиката `TRUNC(START_DATE) = TO_DATE(:b1, 'mm/dd/rr')` может помешать серверному процессу Oracle использовать индекс для `START_DATE`, который мог бы значительно уменьшить количество LIO. Бывает и так, что в коде приложения качество SQL очень высоко, но вызовов LIO все равно слишком много. Так, приложение с построчной выборкой из курсора Oracle может выполнять в сто раз больше вызовов LIO, чем такое же приложение, в котором реализован механизм *выборки массивом (array fetch)* для извлечения 100 строк в одном вызове LIO. Игнорирование возможности выборки массивом приводит к дополнительной нагрузке не только на базу данных, но и на сеть. Дополнительные вызовы базы данных, необходимые для обработки многочисленных маленьких наборов строк приводят к тому, что возрастает количество выполнений события SQL*Net message from client, а это сразу же отражается на времени отклика пользовательской операции.

Проектировщики приложений

Проектировщики приложений также могут сделать невозможным создание эффективного приложения с небольшим количеством LIO. Одно приложение отслеживания материальных ценностей, над которым несколько лет назад работал Джефф, не позволяло определить местонахождение предмета хранения без воссоздания полной истории его перемещений. Вместо быстрого индексированного просмотра «где находится нечто», приложение требовало применения сложного и долго выполняющегося запроса `CONNECT BY`.

Учитывая, что масса разных специалистов одновременно должна выполнить свою работу безошибочно, чтобы исключить проблемы с LIO, не приходится удивляться наличию избыточных вызовов LIO в большинстве систем.

Оптимизация SQL

Оптимизация неэффективного SQL, несомненно, важнейшее средство повышения производительности в арсенале аналитика по производительности Oracle. Если вызов базы данных порождает более десятка вызовов LIO для каждой строки, возвращаемой из каждой таблицы фразы `FROM` соответствующей команды SQL, то, скорее всего, производительность такой команды можно повысить. Например, операция соединения трех таблиц, возвращающая две строки, вероятно, должна требовать не более 60 вызовов LIO.



Любая относительная оценка при определенных условиях может оказаться неверной. Одним из таких условий является формирование результата запроса как итога агрегирования. Например, запрос, возвращающий сумму (одну строку) для таблицы из миллиона строк, вполне законно может требовать выполнения более десяти вызовов LIO.

Приложения, выполняющие код SQL, порождающий множество вызовов LIO, создает серьезные препятствия для масштабируемости систем с большим количеством пользователей. Лишние вызовы LIO не только отнимают мощность процессора, но и могут вызвать большое количество событий `latch free` для защепок `cache buffers chains` [Millsap (2001с)]. Получение и освобождение защепок само по себе может привести к избыточному потреблению мощности процессора, особенно в конфигурациях, где аналитики увеличили значение, заданное для `_SPIN_COUNT` по умолчанию (обычно *не стоит* этого делать).

В настоящее время есть ряд полезных ресурсов по оптимизации SQL: [Ensor and Stevenson (1997a, 1997b); Harrison (2000); Lewis (2001b, 2002); Kyte (2001); Adams (2003); Lawson (2003); Holt et al. (2003)].¹ Участники разнообразных списков рассылки, например Oracle-L (<http://www.cybcon.com/~jkstill>), также делают замечательное дело, помогая друг другу писать эффективный код. Все эти ресурсы содержат хорошие советы о том, как писать эффективный SQL, применяя методы, часть из которых перечислена ниже:

- Диагностика выполнения команд SQL посредством таких инструментов, как `tkprof`, `EXPLAIN PLAN`, и отладочных событий 10032, 10033, 10046, 10079, 10104 и 10241.
- Диагностика поведения оптимизатора запросов Oracle при помощи отладочного события, подобного 10053.
- Работа с текстом SQL, направленная на использование более эффективных планов выполнения.
- Выбор эффективной стратегии индексирования с тем, чтобы обеспечить сокращение объема данных для запросов без создания дополнительной нагрузки в операциях `INSERT`, `UPDATE`, `MERGE` и `DELETE`.
- Применение хранимых планов выполнения с целью заставить оптимизатор запросов Oracle использовать выбранный вами план.
- Создание приемлемой статистики для таблиц, индексов и базы данных с тем, чтобы наилучшим образом информировать оптимизатор запросов Oracle о ваших данных.
- Разработка таких физических моделей данных, которые облегчают хранение и выборку в контексте приложения.
- Разработка логических моделей данных, облегчающих хранение и выборку в контексте приложения.

Оптимизация разбора

Избыточный разбор – это верный путь к невозможности обеспечить масштабируемость приложения для работы с большим количеством

¹ Кроме того, я с нетерпением жду выхода новой книги по оптимизации SQL, которую сейчас пишет Дэн Тэй (Dan Tay) (<http://www.singingsql.com>).

пользователей [Holt and Millsap (2000)]. Студенты обычно приходят к нам, считая, что полные разборы (hard parses) значительно замедляют обработку транзакций, а вот в частичном разборе (soft parse) нет ничего страшного. Более того, иногда люди считают, что полного разбора можно избежать, а частичного – нет. Оба мнения верны лишь наполовину. Полные разборы *действительно* так ужасны, как о них думают, и их можно избежать, используя в коде SQL переменные связывания вместо литералов. Однако частичный разбор столь же ужасен, и часто без него тоже можно обойтись.

Многие авторы употребляют словосочетание «частичный разбор» (soft parse) как синоним для «вызова разбора» (parse call). Мне больше нравится термин «вызов разбора», т. к. он обращает наше внимание на приложение, внутри которого в действительности может быть предпринято спасительное действие. Если же говорить о «частичном разборе», то внимание акцентируется на базе данных, которая *не* является местом решения нашей проблемы. И вот почему. Каждый раз, когда серверный процесс Oracle получает вызов разбора от приложения, этому процессу необходимо использовать процессор сервера базы данных. Если будет обнаружен подходящий для этого запроса разделяемый курсор в кэше курсоров сеанса или библиотечном кэше Oracle, то вызов разбора никогда не приведет к *полному* разбору, и окажется, что затраты на разбор совсем не так велики, как могли бы быть. Однако *отсутствие разбора* обходится еще дешевле, чем частичный разбор. Наилучшую масштабируемость для большого количества пользователей имеют приложения, в которых разбор происходит как можно реже. Следует по возможности избавиться от всех ненужных вызовов разбора.



Для обеспечения масштабируемости лучше всего, чтобы приложения осуществляли минимально возможное количество *вызовов базы данных*. Именно в этом направлении развивается Oracle Call Interface (OCI). Например, в версии 8 OCI предприняты меры для снижения количества пересылок между клиентом и сервером (http://otn.oracle.com/tech/oci/htdocs/Developing_apps.html). Версия 9.2 OCI идет еще дальше, делая так, что многие вызовы базы данных приложения вообще не достигают базы данных (http://otn.oracle.com/tech/oci/htdocs/oci9ir2_new_features).

В системах с высокой конкурентностью и неоправданно многочисленными вызовами разбора большое количество событий CPU service зачастую коррелирует с большим количеством событий ожидания latch free для библиотечного кэша, разделяемого пула и других защелок. Само по себе получение и освобождение защелок может привести к избыточному потреблению мощности процессора, особенно в конфигурациях, где аналитики увеличили значение, заданное для `_SPIN_COUNT` по умолчанию (опять-таки, обычно *не стоит* этого делать). Более того, избыточные вызовы разбора могут привести к ненужным задержкам SQL*Net message from client, способным добавить до нескольких секунд лишнего време-

ни отклика на каждую секунду реальной работы, выполняемой в базе данных. Наконец, вызовы разбора для длинных фрагментов SQL создают ненужные задержки SQL*Net more data from client, которые также могут внести существенный вклад в увеличение времени отклика.

Если ухудшение производительности вызвано большим количеством вызовов разбора, то можно воспользоваться следующими способами снижения нагрузки:

- Старайтесь обходиться без строковых литералов в инструкциях WHERE. Заменяйте их переменными связывания (заполнителями), особенно когда строковый литерал имеет высокую кардинальность (т. е. строка может иметь множество значений). Использование строковых литералов вместо переменных связывания приводит к затратам процессорного времени (CPU service), а также вызывает в системах с высокой конкурентностью ненужные события ожидания освобождения защелок для библиотечного кэша, разделяемого пула и объектов кэша строк.
- Старайтесь выносить вызовы разбора за циклы, с тем чтобы приложение имело возможность несколько раз повторно выполнить курсор, подготовленный одним вызовом разбора. Псевдокод примера 11.2 показывает, как это сделать.

Пример 11.2. Разбор внутри цикла серьезно препятствует масштабируемости

```
# ПЛОХОЙ, немасштабируемый код приложения
for each v in (897248, 897249, ...) {
    c = parse("select ... where orderid = ".v);
    execute(c);
    data = fetch(c);
    close_cursor(c);
}

# ХОРОШИЙ, масштабируемый код приложения
c = parse("select ... where orderid = :v1");
for each v in (897248, 897249, ...) {
    execute(c, v);
    data = fetch(c);
}
close_cursor(c);
```

- Отключите те функции интерфейсного драйвера, которые приводят к увеличению количества вызовов разбора по отношению к их числу в исходном коде приложения. Например, Perl DBI содержит атрибут уровня prepare под названием ora_check_sql. Его значение по умолчанию равно 1, что означает два вызова разбора для каждого вызова функции Perl prepare. Первый вызов разбора выполняется с тем, чтобы помочь SQL-разработчику приложения быстрее отладить свой исходный код за счет предоставления более подробной диагностической информации для неудачных вызовов разбора. Од-

нако в промышленных системах такую функцию следует отключить, т. к. она приводит к излишним вызовам разбора.

- Используйте для приложений многозвенную архитектуру, в которой каждая служба приложения выполняет разбор всех своих команд SQL ровно один раз, а затем повторно выполняет курсоры в течение всего времени работы.
- Не передавайте в вызовы разбора длинные текстовые строки SQL. Применяйте вместо этого вызовы хранимых процедур. Отправка длинных текстовых строк SQL в вызовах разбора приводит к перерасходу процессорной мощности на сервере (даже в случае применения переменных связывания). Даже если SQL-текст является полностью разделяемым, ядро Oracle должно проверять объектные привилегии при каждом получении текстовой строки SQL от пользователя с новым идентификатором (если ядро получает вызов хранимой процедуры, то такая процедура работает в контексте своего владельца, поэтому привилегии на объекты внутри пакета проверяются однократно, если разработчик приложения не укажет, что следует использовать привилегии вызывающего) [Adams (2003) 371–372]. Передача длинных текстовых строк SQL также может привести к ненужной нагрузке на сеть, что будет проявляться в задержке SQL*Net more data from client для процесса, осуществляющего разбор, и в столь же долгих задержках SQL*Net message from client для остальных процессов.
- Сведите к минимуму применение публичных синонимов в приложении, если количество ссылок на объекты очень велико [Adams (2003) 373–375]. Дополнительную информацию можно найти с помощью *google.com* на сайте *www.ixora.com.au* (поиск по словам «public synonym»).

Оптимизация записи

Ядро Oracle спроектировано так, что задачи записи сконцентрированы в небольшом количестве специализированных фоновых процессов. Большую часть операций записи Oracle осуществляют процессы DBWR, LGWR и ARCH. В большинстве баз данных в основном выполняется чтение, а не запись. Однако многие системы подчиняются серьезным соглашениям об уровне обслуживания бизнес-функций, которые требуют обеспечения высокой эффективности записи. И даже в системах, в которых запись играет второстепенную роль по отношению к чтению, медленные операции записи могут косвенно подпортить производительность чтения. Например, низкая производительность процесса DBWR может проявиться во времени отклика запроса появлением событий *free buffer waits*. Избыточные операции записи могут оказаться в очереди к устройствам хранения впереди законных операций чтения, что приведет к ухудшению производительности *db file sequential read*, *db file scattered read* или *direct path read*.

Существует несколько способов повышения производительности записи DBWR, LGWR и ARCH за счет оптимизации рабочей нагрузки. Чаще всего необходимая оптимизация заключается в избавлении от ненужных операций LIO (см. раздел «Оптимизация логического ввода/вывода»). Лишние операции LIO могут привести к появлению ненужных вызовов чтения операционной системы, которые могут занять в очереди место перед вызовами записи DBWR, что может привести к более длительным, чем ожидалось, задержкам для операций `db file single write` и `db file parallel write`, исполняемых процессом DBWR.

Следует убедиться, что все операции записи, выполняемые приложением, на самом деле необходимы. Существует множество коварных причин, по которым приложение может генерировать больше вызовов записи, чем нужно, например:

- Ядро Oracle формирует данные отката/восстановления для каждого индексного блока, изменяемого командой `INSERT`, `UPDATE` или `DELETE`, поэтому наличие лишних индексов может привести к формированию множества ненужных данных отката в системах обработки транзакций. Например, вставка в таблицу с тремя индексами создаст приблизительно в десять раз большую нагрузку, чем вставка в неиндексированную таблицу [Ensor and Stevenson (1997a), 147].
- Некоторые приложения формируют ненужные данные отката, устанавливая значения столбцов в их текущие значения. Например, убедитесь, что в команде `SQL`, которая изменяет значение статуса с `N` на `Y` на основании проверки некоторых условий, инструкция `WHERE` содержит предикат `AND STATUS='N'`. Автоматические генераторы приложений часто заменяют значения столбцов на уже имеющиеся. Это происходит при формировании команды `UPDATE`, которая обновляет каждый столбец, имеющий значение, приведенное на текущем экране. Значениями с экрана следует обновлять значения лишь тех столбцов, которые были *изменены* пользователем, а не *всех* столбцов.
- Пользователи приложений и администраторы базы данных могут выполнять операции для таблиц и индексов, по умолчанию помеченные как `LOGGING`, хотя могли бы работать так же хорошо, будучи помечены как `NOLOGGING`. (Ключевые слова `LOGGING` и `NOLOGGING` заменяют устаревшие `RECOVERABLE` и `UNRECOVERABLE`.)



Если действительно необходимо, чтобы операцию можно было восстановить, то не надо задавать значение `NOLOGGING`. Допустим, вы не хотите использовать операции, помеченные как `NOLOGGING`, для базы данных, включенной в конфигурацию постоянного резервирования (*hot standby architecture*). Oracle9i предлагает режим `FORCE LOGGING` как замену параметра `NOLOGGING`.

Выбор конфигурации системы также влияет на степень ее загруженности. Например, дисковые конфигурации RAID уровня 5 особенно

уязвимы для ожиданий, вызванных операциями записи. Каждая запись, выполняемая процессом Oracle DBWR – это побочная запись, которую массив RAID уровня 5 обрабатывает чрезвычайно неэффективно, если только он не был сконфигурирован с достаточным объемом кэша. Когда при интенсивной непрерывной записи оказывается, что объем кэша недостаточен, производительность дискового массива RAID уровня 5 ухудшается приблизительно в четыре раза по сравнению с ожидаемой. Лучшее решение состоит в том, чтобы избавиться от такого объема лишней нагрузки, снизив скорость постоянного ввода/вывода до приемлемого уровня. Если это невозможно, то можно выбрать один из предлагаемых ниже путей:

- Увеличить размер кэша (на самом деле это лишь отложит возникновение проблемы, но не исключено, что удастся отложить ее как раз на столько, чтобы преодолеть время пиковой нагрузки ввода/вывода приложения).
- Увеличить количество дисковых групп RAID уровня 5, предназначенных для обслуживания запросов ввода и вывода приложения.
- Выбрать другой RAID-массив, обеспечивающий более высокую производительность ввода/вывода без приобретения дополнительных дисков или памяти. Можно, например, применить чередование данных (striping) и зеркалирование (mirroring).

Признаки масштабируемости приложения

Создание масштабируемых приложений – это тяжелая работа. Она гораздо сложнее, чем все то, что можно решить при помощи советов, изложенных на паре страниц. Но вы, наверное, уже обратили внимание на то, что существуют два утверждения о производительности Oracle, которые я считаю непреложными истинами:

1. Работа большинства систем обеспечивается большим количеством оборудования; они работают медленно из-за *ненужных трат* ресурсов.
2. Экономически гораздо эффективнее *избавиться* от бессмысленных трат, чем пытаться уйти от проблемы, наращивая мощность оборудования.

Я утверждаю, что при создании быстрого масштабируемого приложения следует придерживаться Золотого Правила Разработки Приложений:

Не заставляйте приложение выполнять *никакие* операции, кроме абсолютно необходимых.

Так, конечно, может рассуждать только лентяй, и не такому отношению к труду нас учили наши папы и мамы. Но приложение становится медленным, немасштабируемым (что не одно и то же [Millsap (2001a)]) и в конце концов экономически неэффективным именно потому, что

делает то, в чем нет *необходимости*. Приведу несколько советов, призванных немного конкретизировать наше Золотое Правило:

3. Не формируйте отчеты, которые никто не читает.
4. Не формируйте больше выходных данных, чем необходимо.
5. Выполняйте бизнес-операции не чаще, чем этого требует бизнес.
6. Избегайте конструкций SQL, которые обращаются к большому количеству блоков кэша буферов базы данных, чем это необходимо.
7. Не изменяйте значение столбца на значение, совпадающее с текущим.
8. Передавайте данные приложению по мере их готовности и не заставляйте его периодически спрашивать, нет ли для него какой-нибудь работы.
9. Не формируйте данные отката и восстановления, если не планируете воспользоваться обеспечиваемыми ими возможностями.
10. Не разбирайте команды SQL, если возможен предварительный разбор с последующим их совместным использованием.
11. Не обрабатывайте DML построчно, применяйте выборки массивом, пакетные вставки и т. д.
12. Не блокируйте данные чаще и дольше, чем это абсолютно необходимо.

Не претендуя на полноту списка, я все же верю, что он поможет вам осознать, как должно выглядеть не страдающее избыточным весом приложение.

12

Учебные примеры

Тот, кто добрался до этой страницы, теоретически уже готов заниматься проектами повышения производительности, применяя абсолютно новый способ. Однако теоретическая готовность и практическая работа – это совершенно разные вещи. Новый Метод R нарушает традиции, поэтому заставить себя его применять может оказаться достаточно сложно. Практически все книги, статьи, веб-страницы, программы, консультанты, коллеги и друзья, к которым вы могли обращаться до того, как познакомились с этой книгой, давали вам советы по «настройке», которые полностью противоречат тому, о чем в ней рассказывается. Для того чтобы убедить вас опробовать Метод R, я приведу в этой главе несколько примеров его практического применения. Надеюсь, что посмотрев, как мы с коллегами решаем некоторые задачи, весьма часто встречающиеся в процессе оптимизации, вы будете лучше представлять, что может дать Метод R.

Обучение Методу R очень похоже на обучение родному языку. Сначала человек наблюдает, как другие делают то, чему он хочет научиться. Ребенок учится говорить *не* при помощи синтаксических схем и таблиц падежей. Если эти схемы и таблицы и встретятся ему вообще, то уже *после* того, как он довольно много узнает о языке. Наверное, в школе вам приходилось изучать подобные вещи. Делалось это для того, чтобы вы могли проанализировать с формальной точки зрения, как люди говорят и пишут, и чтобы помочь вам в будущем общаться более эффективно.

Я помогу вам изучить Метод R тем же способом. В последующих разделах рассмотрено несколько типичных проектов повышения производительности Oracle, прослеженных от начала и до конца. Изучив эти примеры, вы быстро обратите внимание на наличие в них ряда типовых действий, которые можно будет повторить в ситуациях, отличающихся от рассмотренных. Если же вы решите серьезно заняться повышени-

ем производительности Oracle, выбрав это своей специальностью, то вам понадобятся синтаксические правила, представленные в части II.

У примеров этой главы разные источники:

- Клиенты Hotsos Profiler, предложившие компании *hotsos.com* файлы трассировки для анализа.
- Студенты Hotsos Clinic, которые приносили файлы трассировки на занятия для их анализа непосредственно в аудитории.
- Выезды специалистов по производительности Hotsos к клиентам.
- Вопросы и ответы, появляющиеся в общедоступных группах новостей.

Все эти примеры реальны. Я нигде не привожу конкретный источник и не говорю о конкретных людях, участвовавших в проекте, но каждый из описанных случаев был реально поставленной перед нами задачей, которая не поддавалась решению до тех пор, пока не был применен Метод Р. Познакомившись с этим методом поближе, вы увидите, что его корректное применение неизбежно приводит к одному из двух результатов:

- отыскивается основная причина ухудшения производительности и появляется возможность определить, какого повышения производительности следует ожидать, либо
- появляются доказательства экономической необоснованности улучшения производительности для исследуемой пользовательской операции.

Пример 1: обманчивые общесистемные данные

После нескольких месяцев попыток договориться с клиентом из Далласа о визите, в понедельник в отделе продаж *hotsos.com* раздался телефонный звонок. Люди, с которыми мы пытались договориться о встрече, зашли в тупик, пытаясь найти причину ухудшения производительности, и собирались дать нам шанс все исправить. В среду мы этим шансом воспользовались.

Что-то похожее мы уже видели. Компания несколько месяцев боролась с ухудшением производительности, связанным со слишком большим временем отклика одной программы. Все эти месяцы штатные специалисты компании старались найти решение, но ни одна из их попыток не была успешной. В конце концов они опустили руки, и руководство решило выделить серьезную сумму на решение проблемы. Поэтому в выходные, предшествующие полученному нами телефонному звонку, компания заменила в системе процессоры на более производительные. Модернизация прошла успешно, и, конечно же, все надеялись (хоть и нервничали немного), что в понедельник все будет работать гораздо лучше.

К их ужасу производительность медленной программы после чрезвычайно дорогостоящего обновления оборудования стала *еще хуже*. Причем *значительно* хуже. Поэтому в понедельник они нам позвонили: «Приходите и покажите нам, что можно сделать». Через два дня мы сели в машину и поехали. Вот что мы обнаружили:

- Компания работала с продуктом Oracle Payroll. Он был сконфигурирован традиционным способом: пакетные задания выполнялись на сервере базы данных, а десятки пользователей, работавших через броузеры, были распределены по зданию и связаны локальной сетью (LAN).
- Производительность программы PYUGEN мешала нормальной работе компании уже несколько месяцев. Когда мы появились, программа PYUGEN могла обрабатывать около 27 заданий в минуту. Производительность надо было увеличить вдвое.
- Мониторинг производительности выполнялся при помощи хорошо известного полнофункционального средства, которое запрашивало данные фиксированных представлений V\$SYSTEM_EVENT и V\$LATCH. Это средство показывало, что источником неприятностей в системе были события ожидания освобождения защепок latch free. Подавляющее большинство ожиданий latch free относилось к защелкам цепочек буферов кэша.
- Специалисты компании правильно поняли, что конкуренция за защелки цепочек буферов кэша является вероятным признаком неэффективного SQL (т. е. содержащего большое количество LIO). Однако разработчики приложений заказчика проанализировали программу PYUGEN и не смогли уменьшить количество LIO в SQL.
- Только что произведенное повышение частоты всех двенадцати процессоров системы с 700 МГц до 1 ГГц значительно *ухудшило* производительность программы PYUGEN. Отсутствие повышения производительности при замене процессоров стало последней каплей, заставившей клиента пригласить к себе команду *hotsos.com*.

Определение цели

К моменту нашего прибытия клиент уже завершил этап выбора пользовательской операции, определив, что основную проблему для производительности системы представляет собой PYUGEN. Поэтому нашим первым шагом по прибытии был сбор корректно выбранных диагностических данных. Для этого клиента сбор данных расширенной трассировки SQL был прост, т. к. время отклика пользовательской операции формировалось исключительно выполнением программы PYUGEN. Для включения и отключения расширенной трассировки SQL мы применили собственную свободно распространяемую программу Sparky (<http://www.hotsos.com>).

Программа, которую мы собирались исследовать, выполнялась более получаса, в результате чего было сформировано около 70 Мбайт данных расширенной трассировки. Данные трассировки обрабатывались программой Hotsos Profiler, которая сформировала профиль ресурсов, приведенный в примере 12.1.

Пример 12.1. Профиль ресурсов для программы Oracle Payroll

Response Time Component	Duration		# Calls	Dur/Call
SQL*Net message from client	984.0s	49.6%	95,161	0.010340s
SQL*Net more data from client	418.8s	21.1%	3,345	0.125208s
db file sequential read	279.3s	14.1%	45,084	0.006196s
CPU service	248.7s	12.5%	222,760	0.001116s
unaccounted-for	27.9s	1.4%		
latch free	23.7s	1.2%	34,695	0.000683s
log file sync	1.1s	0.1%	506	0.002154s
SQL*Net more data to client	0.8s	0.0%	15,982	0.000052s
log file switch completion	0.3s	0.0%	3	0.093333s
enqueue	0.3s	0.0%	106	0.002358s
buffer busy waits	0.2s	0.0%	67	0.003284s
SQL*Net message to client	0.2s	0.0%	95,161	0.000003s
db file scattered read	0.0s	0.0%	2	0.005000s
SQL*Net break/reset to client	0.0s	0.0%	2	0.000000s
Total	1,985.3s	100.0%		

Данные профиля ресурсов удивили всех, кто работал над проектом последние несколько месяцев. Уже по одному только профилю ресурсов с абсолютной уверенностью можно утверждать, что вклад ожидания освобождения защепок в общее время отклика PYUGEN был чрезвычайно мал. Если бы компании удалось вообще исключить событие ожидания latch free из системы, время выполнения программы изменилось бы на 1%.



Такое в нашей работе бывает часто: многие проблемы производительности пользовательских операций невозможно определить на основе исследования данных в масштабе всей системы. Данные из представления V\$SYSTEM_EVENT верны; просто они не имеют отношения к решаемой проблеме. Помните – по агрегированным данным нельзя восстановить составляющие.

На самом деле представление V\$SYSTEM_EVENT ясно указывало на то, что основным событием ожидания является SQL*Net message from client, но ведь каждый уважающий себя аналитик производительности Oracle знает, что надо исключить из рассмотрения все события SQL*Net, т. к. они являются событиями «простоя».

Около 50% общего времени отклика PYUGEN было занято выполнением системных вызовов чтения SQL*Net. Расположение событий SQL*Net message from client во главе профиля ресурсов заставило нас быстро перепроверить собранные данные с тем, чтобы убедиться, что такое гла-

венствующее положение событий, происходящих между вызовами базы данных, не является результатом ошибки сбора данных. Ошибки не было. События SQL*Net message from client и их продолжительности были равномерно распределены по файлу трассировки и являлись результатом тысяч вызовов базы данных. Если принять в рассмотрение еще одно событие SQL*Net, SQL*Net more data from client, то окажется, что мы выявили более 70% общего времени отклика PYUGEN.

Диагностика и лечение

Естественно, нельзя игнорировать 70% времени отклика программы, даже если соответствующие события называют событиями «простоя». Будь то событие простоя или нет, в любом случае это время вошло в пользовательское время отклика, значит, с ним нужно разобраться. Если бы статистика собиралась не так аккуратно (у нас корректно выбрана программа и временная область), то события SQL*Net message from client, вероятно, занимали бы гораздо больше времени в полученных данных. При такой ошибке сбора *необходимо* исключить из рассмотрения так называемые события простоя.

Мы начали исследование с первой строки профиля ресурсов. Поскольку речь шла о поставляемом приложении, мы считали, что вряд ли удастся манипулировать количеством вызовов базы данных, и поэтому сконцентрировались на столбце продолжительности вызова. И значение данного столбца показалось нам подозрительным – его порядок отражал скорее работу в локальной сети (сотые доли секунды, о чем мы говорили в главе 10), чем межпроцессное взаимодействие (IPC) (тысячные доли секунды или меньше). Поэтому мы перепроверили, действительно ли пакетная программа PYUGEN выполняется на сервере базы данных (там же, где и соответствующий PYUGEN серверный процесс Oracle), для чего обратились к данным V\$SESSION, автоматически собранным программой Sparky при включении сбора (см. пример 12.2).

Пример 12.2. Данные, полученные Sparky от V\$SESSION подтвердили, что процесс PYUGEN действительно работает на том же хосте, что и его серверный процесс Oracle

```
Oracle instance = prod (8.1.6.3.0)
      host = dalunix150.xyz.com (OSF1 V5.1)
      program = PYUGEN@dalunix150.xyz.com (TNS V1-V2) (session 611)
      trace file = /prod/u001/app/oracle/admin/prod/udump/ora_922341.trc
      line count = 1,760,351 (0 ignored, 0 oracle error)
      t0 = Wed Sep 12 2001 14:10:27 (388941433)
      t1 = Wed Sep 12 2001 14:43:32 (389139973)
interval duration = 1,985.40s
      transactions = 672 (672 commits, 0 rollbacks)
```

Все правильно, имя хоста, выведенное справа от символа @ в V\$SESSION, точно совпадает с именем узла (Node name) из преамбулы файла трассировки SQL. PYUGEN точно работает на том же самом хосте, что и сервер

базы данных. Почему же тогда при работе программы PYUGEN наблюдаются такие задержки SQL*Net message from client? В поисках ответа мы обратились к файлу *tnsnames.ora* системы. Оказалось, что системный инженер, стремясь облегчить жизнь системных администраторов, установил для всей системы единый псевдоним TNS. Пакетные задания направлялись на тот же адаптер протокола TCP/IP, что и броузеры клиентов системы.

Разработать стратегию, отлично вписавшуюся в рамки благоприятствования системному администрированию, было несложно. Можно было добавить второй псевдоним в файл *tnsnames.ora*. Второй псевдоним был бы идентичен первому, только имя у него было бы другое, а синтаксис (PROTOCOL=BEQ) занял бы место (PROTOCOL=TCP). Заказчику надо было бы остановить и заново запустить Oracle Applications Concurrent Manager, указав новый псевдоним, использующий адаптер протокола *bequeath*. Новый файл *tnsnames.ora* можно было бы без побочных эффектов распространить по всей системе. Все, кроме запустивших Concurrent Manager, продолжали бы использовать тот же TNS-псевдоним, что и ранее.

Прежде чем претворять это изменение в жизнь, заказчик провел простой эксперимент. Была выполнена команда SELECT, которая потребовала бы нескольких тысяч вызовов базы данных из сеанса SQL*Plus, работающего на сервере базы данных, и для нее были сняты временные характеристики. Сначала команду выполнили в сеансе, установленном при помощи старого псевдонима, использующего адаптер протокола TCP/IP, а затем в сеансе, установленном при помощи нового псевдонима, использующего адаптер протокола *bequeath*. Тест показал, что использование адаптера протокола *bequeath* снизило задержки SQL*Net message from client до менее чем 0,001 секунды. Выполнив лишь одно это изменение, мы могли ожидать сокращения общего времени отклика программы как минимум на 40% (рис. 12.1).

На самом деле у нас есть причины ожидать улучшения более чем на 40%. Второй по значимости вклад во время отклика PYUGEN вносит еще одно событие SQL*Net – SQL*Net more data from client. Это событие вызвано последовательностью вызовов разбора, передававших слишком длинные текстовые строки SQL по SQL*Net от клиента к серверу (вместо того чтобы достигать той же цели посредством вызовов хранимых процедур). Длинные текстовые строки SQL не помещались в один пакет SQL*Net, поэтому ядро Oracle довольно долго ожидало второго и, возможно, последующих пакетов SQL*Net в ходе вызовов разбора. Конечно, учитывая, что Oracle Payroll поставляется без исходных текстов, наши надежды на быстрое уменьшение количества выполнений данного события были достаточно призрачны. Однако можно было надеяться, что задержка SQL*Net more data from client частично связана с передачей по сети, и что смена адаптера протокола ускорит исполнение этого события.

Microsoft Excel - PIC 1.0a.xls

File Edit View Insert Format Tools Data Window Help

Type a question for help

Arial 10 B U

E16 0.001

1	Expected Performance Improvement Calculator									
2	PIC 1.0a (2003/05/09)									
3	Copyright © 1999-2003 by Hotsos Enterprises, Ltd. All rights reserved									
4										
5	Baseline					Baseline				
6	Response Time Component	Duration	# Calls	Dur/Call	Response Time Component	Duration		# Calls	Dur/Call	
7						Seconds	Pct			
8	thing to be improved	984	95161		thing to be improved	984.00	49.6%	95,161	0.010 340	
9	all other	1001.3	1		all other	1,001.30	50.4%	1	1,001.300 000	
10										
11					Total	1,985.30	100.0%			
12										
13	Proposed					Proposed				
14	Response Time Component	Duration	# Calls	Dur/Call	Response Time Component	Duration		# Calls	Dur/Call	
15						Seconds	Pct			
16	thing to be improved		95,161	0.001	thing to be improved	95.16	4.8%	95,161	0.001 000	
17	all other	1001.3	1		all other	1,001.30	50.4%	1	1,001.300 000	
18										
19					Total	1,096.46	100.0%			
20										
21						Proposed Response Time Reduction				
22						Total Response Time	Seconds	Pct		
23						Total reduction	888.84	44.8%		

Constant: Improvement Calculator

Ready

Рис. 12.1. Можно ожидать, что уменьшение задержки событий SQL*Net message from client events для одного вызова с 0,010 секунды до 0,001 секунды уменьшит время отклика PYUGEN больше, чем на 40%

Результаты

Результаты оказались просто замечательными. Производительность программы Payroll возросла с 27 заданий в минуту до 61. Проверка предложенного изменения *tnsnames.ora* заняла 15 минут и еще около недели ушло на процедуру контроля за изменениями. В целом мы провели у заказчика менее четырех часов. Из них два заняла инсталляция Sparky (потребовалось обновить Perl на хосте сервера базы данных), и чуть менее получаса потребовалось для прогона программы PYUGEN с включенной трассировкой SQL уровня 8. Оставшиеся полтора часа были посвящены встречам, приветствиям, анализу, тестированию и подведению итогов.

Да, кстати... почему программа Payroll стала работать *медленнее* после модернизации процессоров? Ей требовалось немного процессорного времени, поэтому такая модернизация имела чрезвычайно малый положительный эффект на PYUGEN. Большая часть времени уходила на простаивание в очереди для сетевого обслуживания. При этом наряду с заданием Payroll исполнялись и другие программы. Обновление процессора сделало *их* более быстрыми, в результате чего *они* стали интенсивнее генерировать сетевые вызовы (число которых осталось неиз-

менным после модернизации процессоров) в течение более короткого промежутка времени. В результате конкуренция за сетевое обслуживание в процессе работы Payroll усилилась. Следовательно, каждый сетевой вызов ввода/вывода, совершаемый программой Payroll, стал чуть более медленным, чем до модернизации. Ухудшение сетевого времени отклика перечеркнуло небольшое улучшение времени обслуживания процессором, что привело к итоговому снижению производительности Payroll. А это было нехорошо... ведь программа Payroll имела самый высокий бизнес-приоритет в системе.

Мораль

Рассмотренный пример иллюстрирует несколько важных моментов:

- Нельзя опираться на данные V\$, определяя причину низкой производительности системы. Главную роль в постановке задачи должен играть бизнес. Реальная проблема производительности системы – это то, что вызывает недопустимо большое время отклика для самой важной, с точки зрения бизнеса, пользовательской операции.
- По агрегированным данным нельзя восстановить составляющие. Анализ общесистемных статистик для экземпляра не позволяет гарантированно определить, в чем причина плохой производительности одной конкретной программы.
- Модернизация оборудования – более сомнительная, в смысле повышения производительности, операция, чем можно себе представить. Можно не только зря потратить много денег, но и снизить производительность «усовершенствуемой» программы.
- Выявить и устранить причину плохой производительности посредством старого метода проб и ошибок практически невозможно. Плохая производительность может быть вызвана огромным количеством разнообразных факторов. Не надо проверять все, что могло бы ее вызвать, достаточно исследовать пользовательскую операцию и найти причину.

Пример 2: большие затраты процессорного времени

Одним из первых клиентов *hotsos.com* была компания, в которой работали с Oracle Financials и Oracle Manufacturing. Наблюдалось неудовлетворительное время отклика в нескольких программах, как собственных, так и приобретенных на стороне. Клиент пригласил нас не для того, чтобы исправить сложившееся положение, а для того, чтобы мы обучили его сотрудников тому, как это сделать. Для начала следовало научить нового аналитика по производительности, как собирать и анализировать правильные диагностические данные. Вторым пунктом было облечение нашей пре-бета версии программ Sparky и Hotsos Profiler в некую подобающую форму, с тем чтобы клиент мог пользо-

ваться ими и после нашего отъезда. Для нас эта работа была очень хорошим опытом. Новый аналитик по производительности был администратором приложений и никогда ранее особо не занимался задачами повышения производительности.

Через пару месяцев наше общение с новым аналитиком от ежедневных телефонных звонков перешло к еженедельному обмену электронными письмами. Однажды он позвонил, чтобы просто поздороваться и рассказать об одном из своих достижений. Эту историю мы вам и расскажем.

Определение цели

За несколько предшествующих недель он проделал замечательную работу, исследовав список медленных пользовательских операций, важных для его компании. Он признался, что решил заняться этим, когда подозрительное отсутствие жалоб на производительность неожиданно обеспечило ему некоторое количество свободного времени. Он решил проверить, почему одно пакетное задание работает так долго (если вы помните, прежде чем заняться усовершенствованием производительности, новый аналитик работал непосредственно с приложениями и поэтому хорошо знал, как долго оно работает). Он выполнил трассировку программы. Профиль ресурсов для данного файла трассировки приведен в примере 12.3.

Пример 12.3. Профиль ресурсов для программы Oracle Purchasing

Response Time Component	Duration		# Calls	Dur/Call
CPU service	1,527.5s	60.8%	158,257	0.009652s
db file sequential read	432.0s	17.2%	62,495	0.006913s
unaccounted-for	209.6s	8.3%		
global cache lock s to x	99.9s	4.0%	3,434	0.029083s
global cache lock open s	85.9s	3.4%	3,507	0.024502s
global cache lock open x	57.9s	2.3%	1,930	0.029990s
latch free	26.8s	1.1%	1,010	0.026505s
SQL*Net message from client	19.1s	0.8%	6,714	0.002846s
write complete waits	11.1s	0.4%	155	0.071806s
enqueue	11.1s	0.4%	330	0.033606s
row cache lock	11.1s	0.4%	485	0.022887s
log file switch completion	7.3s	0.3%	15	0.487333s
log file sync	3.3s	0.1%	39	0.084872s
wait for DLM latch	3.0s	0.1%	91	0.032418s
global cache lock busy	1.5s	0.1%	11	0.139091s
DFS lock handle	1.4s	0.1%	43	0.032558s
global cache lock null to x	0.9s	0.0%	8	0.112500s
rdcms ipc reply	0.6s	0.0%	7	0.081429s
global cache lock null to s	0.4s	0.0%	7	0.060000s
library cache pin	0.1s	0.0%	7	0.015714s
SQL*Net message to client	0.0s	0.0%	6,714	0.000003s
file open	0.0s	0.0%	13	0.000000s

SQL*Net more data from client	0.0s	0.0%	2	0.000000s

Total	2,510.5s	100.0%		

Как видите, в профиле доминируют обслуживание процессором и чтение файлов базы данных, причем вместе они образуют почти 80% общего времени отклика. Около 10% времени отклика занимают операции блокировки глобального кэша, необходимые Oracle Parallel Server, а оставшиеся 10% распределены между событием `unaccounted-for` и множеством несущественных событий.

На основе одного лишь профиля ресурсов невозможно определить, чрезмерно ли загружен процессор. Но очевидно, что для заметного уменьшения 42-минутного значения времени отклика потребуется уменьшить и составляющую `CPU service`. В подобном случае первым должен возникать вопрос: «Какая из команд SQL несет ответственность за такую загрузку процессора?». Благодаря Hotsos Profiler получить ответ на этот вопрос чрезвычайно просто, предоставляя в своем выводе специальную секцию, которая перечисляет пять команд SQL, внесших наибольший вклад в каждую из составляющих времени отклика (см. пример 12.4).

Пример 12.4. Hotsos Profiler определяет вклад команд SQL в расходование времени процессора

SQL Statement Id	Duration	
704365403	1,066.4s	69.8%
3277176312	371.9s	24.3%
1107640601	8.5s	0.6%
3705838826	6.5s	0.4%
529440951	6.0s	0.4%
111 others	68.7s	4.5%
Total	1,527.5s	100.0%

Основную долю процессорного времени потребляют всего две команды SQL. Выходные данные Hotsos Profiler содержат идентификаторы команд, которые являются гиперссылками, приводящими вас к данным, показанным в примере 12.5. Подобные сведения предоставляет утилита Oracle *tkprof*, которой надо указать порядок сортировки `sort=prscpu, ехесру, fchсру`. При таком упорядочивании интересующие вас команды SQL будут находиться в верхней части списка.

Пример 12.5. SQL-текст и статистики производительности для команды 704365403, потребляющей основное время процессора в данном сеансе

```
Statement Text
update po_requisitions_interface set requisition_header_id=:b0
where (req_number_segment1=:b1 and request_id=:b2)

Statement Cumulative Database Call Statistics
```

Action	Action Count	Rows	----- Elapsed	Response Time CPU	----- Other	LIO Blocks	PIO Blocks
Parse	0	0	0.0	0.0	0.0	0	0
Execute	1,166	0	1,455.0	1,066.4	388.6	8,216,887	3,547
Fetch	0	0	0.0	0.0	0.0	0	0
Total	1,166	0	1,455.0	1,066.4	388.6	8,216,887	3,547
Per Exe	1	0	1.3	0.9	0.3	7,047	3
Per Row	1,166	1	1,455.0	1,066.4	388.6	8,216,887	3,547

Чрезвычайно полезная информация, позволяющая сделать ряд интересных наблюдений:

- Команда, потребляющая основное время процессора, – это очень простая команда UPDATE, которая выполнялась 1166 раз.
- В ходе 1166 выполнений данной команды UPDATE ни разу не была обработана ни одна строка.
- Для каждого выполнения потребовалось в среднем 7047 операций LIO (8 216 887 операций LIO, поделенных на 1166 исполнений), позволивших установить, что ни одна строка не соответствует простому условию фразы WHERE.
- Коэффициент попаданий в кэш буферов базы данных для данной команды весьма «хорош», он составляет:

$$\begin{aligned} CHR &= \frac{LIO - PIO}{LIO} \\ &= \frac{8,216,887 - 3,547}{8,216,887} \\ &= 0.999568 \end{aligned}$$

Ирония в том, что одной из причин, по которой данная команда никогда не привлекала серьезного внимания аналитиков по производительности системы, мог быть как раз хороший коэффициент попаданий – средства мониторинга производительности воспринимали эту команду как образцовую.

Диагностика и лечение

В главе 11 приводилось простое эмпирическое правило, относящееся к количеству вызовов LIO. Оно гласит, что если команде SQL требуется более десяти операций LIO для каждой возвращаемой строки каждой таблицы из фразы FROM, то команда, скорее всего, выполняет слишком много операций LIO. Конечно, наша команда UPDATE – это не запрос с фразой FROM, но она исполняет практически такой же фрагмент кода ядра Oracle, как следующий запрос:

```
select requisition_header_id=:b0
```

```
from po_requisitions_interface
where (req_number_segment1=:b1 and request_id=:b2)
```

Какое количество ЛЮ необходимо для того, чтобы определить, что запрос не возвращает строк? Мне кажется, что меньше десяти, и вот почему: если бы по двум столбцам REQ_NUMBER_SEGMENT1 и REQUEST_ID существовал составной индекс, то ядро Oracle могло бы определить, что запрос не возвращает строк, просто пройдя по индексу от корня к листу. Количество операций ЛЮ, необходимых для выполнения такого прохода, совпадает с высотой индекса. Высота индекса – это его значение BLEVEL (например, из DBA_INDEXES, для исследуемых сегментов индекса) плюс один. Высота самых гигантских индексов, о которых я когда-либо слышал, не превышала семи. Следовательно, будем ожидать, что при наличии составного индекса по столбцам REQ_NUMBER_SEGMENT1 и REQUEST_ID количество операций ЛЮ в расчете на одно выполнение не будет больше семи.

Вы, наверное, помните, что потребляемое вызовом базы данных процессорное время обычно пропорционально количеству выполняемых им операций ЛЮ. И если удастся уменьшить количество операций ЛЮ для вызова с 7047 до просто 7, то можно ожидать, что уменьшится общее потребление процессорного времени вызовами базы данных тоже в 1000 раз. Следовательно, можно ожидать, что уменьшение количества ЛЮ приведет к уменьшению суммарного потребления процессорного времени данной командой UPDATE с 1066,4 секунды до приблизительно 1 секунды. Ожидаемое улучшение приблизительно на 1000 секунд – это достаточно значительное улучшение времени отклика, которое стоит того, чтобы проверить такое предположение. Рекомендуемая деятельность по повышению производительности состоит в создании составного индекса по столбцам REQ_NUMBER_SEGMENT1 и REQUEST_ID.

Результаты

Общее время отклика программы действительно уменьшилось намного больше, чем на предполагавшиеся 1000 секунд. Такое достижение обусловлено сопутствующими факторами, в числе которых следующие:

- Второй по значимости вклад в загрузку процессора в рамках сеанса вносит команда 3277176312, в которой есть точно такая же фраза WHERE, как и в команде 704365403. Поэтому создание индекса оказывает потрясающий эффект на оба самых значимых компонента общего времени отклика.
- Уменьшение количества ЛЮ снижает общую рабочую нагрузку сеанса не только в смысле процессорного времени, но и в других. А именно, устранение многих обращений к буферам базы данных в сеансе обычно приводит к сокращению количества операций дискового чтения. Избавление от операций ЛЮ почти всегда сопровождается уменьшением количества вызовов РЮ. Кроме того, чем

меньше вызовов LIO, тем короче может стать ожидание событий `global cache lock...`, `latch free` и других.

Однако создание нового индекса также вызывает риск принесения сопутствующего *ущерба*. В данном случае вероятность нанесения запросу ущерба минимальна, т. к. базовая таблица является интерфейсной, и к ней обращаются лишь несколько SQL-команд приложения. Для подстраховки следует при создании нового индекса (или удалении старого) перепроверить все планы выполнения приложения и убедиться, что любые их изменения, вызванные изменением схемы, не принесут вреда (это можно сделать, например, при помощи средства Project Laredo, представленного на сайте <http://www.hotsos.com>).

Мораль

Этот случай обращает наше внимание на следующие важные факты:

- Оптимизировать SQL часто оказывается проще, чем можно было ожидать. Главное – знать, *какую* команду SQL следует оптимизировать.
- Сопутствующая выгода от уменьшения количества вызовов LIO может быть весьма значительной.
- Создание или удаление индекса может иметь как положительные, так и отрицательные последствия. Для уменьшения риска необходимо проанализировать *все* потенциальные изменения планов выполнения, к которым может привести манипулирование индексами.
- Коэффициент попаданий в кэш буферов базы данных для команды SQL не может выступать в качестве надежного критерия ее эффективности.

Пример 3: длительные события SQL*Net

Этот случай имел место на заключительном этапе занятий Hotsos Clinic. После двух с половиной дней лекций, которые я читал вместе с одним коллегой, мы предложили слушателям принести в класс файлы трассировки и попробовать открыто продиагностировать их совместными усилиями. Идея была в том, что мы мужественно брались за диагностику файла любого, кто не побоялся бы рассказать на публике о медленной работе своих приложений. Было очень весело. Студенты имели возможность увидеть, действительно ли работает на практике то, чему мы их учили, и могли поделиться с аудиторией своими новыми идеями. Студент, представивший свою программу на рассмотрение, обычно получал готовое решение серьезной проблемы. А мы могли познакомиться с множеством чрезвычайно интересных проблем производительности приложений.

И вот в последний день занятий очаровательная молодая девушка, сидевшая в последнем ряду, принесла нам компакт-диск. Она объяснила,

что файл трассировки, записанный на этом диске, относится к купленному приложению, созданному на PowerBuilder, которое всегда работало очень медленно. Более того, компания, офис которой находился на противоположной стороне улицы, применяла то же самое приложение и также имела серьезные претензии к его производительности. История продолжалась бесконечно – на каждой встрече пользователи интересовались друг у друга, не удалось ли кому-то заставить его работать быстрее. Никто не понимал, почему приложение работает так медленно.

Все происходит естественным путем.

Определение цели

Представлялась чудесная возможность демонстрации мощи Метода R – замечательно, что мы можем решить проблему производительности, с которой безуспешно пытались справиться годами! Ситуация полностью отражала описанное в первых абзацах главы 1.

Когда я увидел профиль ресурсов, построенный по файлу трассировки, мое сердце опустилось. Там присутствовало множество событий SQL*Net message from client, и практически ничего больше (пример 12.6). Заговорив, я в первую очередь высказал сожаление – эффективность нашей помощи, наверное, не могла быть такой, как нам бы этого хотелось. Очевидно было, что профиль состоял в основном из времени бездействия пользователя или из времени выполнения неизмеряемого кода на сервере приложений или еще из чего-то подобного.

Пример 12.6. Профиль ресурсов приложения, написанного на PowerBuilder

Response Time Component	Duration		# Calls	Dur/Call
SQL*Net message from client	166.6s	91.8%	6,094	0.027338s
CPU service	9.7s	5.3%	18,750	0.000515s
unaccounted-for	1.9s	1.1%		
db file sequential read	1.6s	0.9%	1,740	0.000914s
log file sync	1.1s	0.6%	681	0.001645s
SQL*Net more data from client	0.3s	0.1%	71	0.003521s
SQL*Net more data to client	0.1s	0.1%	108	0.001019s
free buffer waits	0.1s	0.0%	4	0.022500s
db file scattered read	0.0s	0.0%	34	0.001176s
SQL*Net message to client	0.0s	0.0%	6,094	0.000007s
log file switch completion	0.0s	0.0%	1	0.030000s
latch free	0.0s	0.0%	1	0.010000s
log buffer space	0.0s	0.0%	2	0.005000s
direct path read	0.0s	0.0%	5	0.000000s
direct path write	0.0s	0.0%	2	0.000000s
Total	181.5s	100.0%		

Но слушательница уверяла нас, что не спала на лекциях и что хорошо знакома с ошибками сбора данных и временем бездействия пользова-

теля, и что ничего подобного в ее файле трассировки нет. Она начала сбор данных расширенной трассировки SQL непосредственно перед тем, как пользователь нажал кнопку «ОК» для запуска операции, и завершила сбор данных сразу же, как только заметила, что система вернула результат (пользователь отключился от приложения). Пользователь действительно ждал отключения около трех минут после нажатия кнопки. Приложение было двухзвенным, промежуточного прикладного звена не существовало, следовательно, не было и неизмеряемого прикладного кода. Более того, для сбора данных она применяла наше средство – Sparky.

Н-да...

Если честно, то одну улику я пропустил. Выходные данные текущей версии Hotsos Profiler содержат не только информацию о средней длительности вызова, но и минимальное и максимальное значения времени ожидания для каждого события (я не привожу все эти сведения в тексте примера 12.6, т. к. ширина страницы слишком мала). Максимальная продолжительность события SQL*Net message from client составила около нескольких секунд. И если бы я обратил более пристальное внимание на количество вызовов (# Calls в примере 12.6), то понял бы, что речь не идет об ошибке сбора данных или задумчивом пользователе.

В то время у меня еще не было ясного представления о том, как именно бороться с такой проблемой, и мы, под руководством Джеффа, принялись изучать вывод Hotsos Profiler. Руководствуясь принципом *опережающего атрибутирования* (см. главу 11), мы искали вызовы базы данных, которые следовали за событиями SQL*Net message from client. С тех пор компания Hotsos доработала программу Hotsos Profiler, и теперь подобные проблемы легко решаются всего за пару минут. Здесь описан процесс диагностики, реализованный в современной усовершенствованной версии.

Диагностика и лечение

Для того чтобы ничего не упустить, мы чуть внимательнее посмотрели на продолжительности событий SQL*Net message from client. В примере 12.6 этого не видно, т. к. страницы книги недостаточно широки, но на самом деле вывод Hotsos Profiler показывает, что максимальная продолжительность выполнения SQL*Net message from client составила 17,43 секунды. Быстрый поиск строки `ela= 1743` (обратите внимание на пробел, который ядро Oracle вставляет между символами = и 1) в файле трассировки показал, что в хвосте файла действительно *присутствует* небольшая ошибка сбора данных. Между двумя строками `XCTEND` притаилось событие SQL*Net message from client со значением `ela`, равным 17,43 секунды. Первая фиксация соответствовала концу пользовательской операции. Вторая фиксация имела место, когда пользователь отключился от приложения. Девушке потребовалось несколько секунд на то, чтобы заметить, что операция завершилась. После исправления

этой небольшой ошибки сбора данных профиль ресурсов пользовательской операции стал выглядеть так, как показано в примере 12.7.

Пример 12.7. Профиль ресурсов из примера 12.6 после исправления ошибки сбора данных, составляющей 17,43 секунды

Response Time Component	Duration		# Calls	Dur/Call
SQL*Net message from client	149.2s	91.0%	6,093	0.024482s
CPU service	9.7s	5.9%	18,750	0.000515s
unaccounted-for	1.9s	1.2%		
db file sequential read	1.6s	1.0%	1,740	0.000914s
log file sync	1.1s	0.7%	681	0.001645s
SQL*Net more data from client	0.3s	0.2%	71	0.003521s
SQL*Net more data to client	0.1s	0.1%	108	0.001019s
free buffer waits	0.1s	0.1%	4	0.022500s
db file scattered read	0.0s	0.0%	34	0.001176s
SQL*Net message to client	0.0s	0.0%	6,094	0.000007s
log file switch completion	0.0s	0.0%	1	0.030000s
latch free	0.0s	0.0%	1	0.010000s
log buffer space	0.0s	0.0%	2	0.005000s
direct path read	0.0s	0.0%	5	0.000000s
direct path write	0.0s	0.0%	2	0.000000s
Total	164.0s	100.0%		

Следующий вопрос, требующий ответа, звучит так: «Какая команда SQL несет ответственность за оставшуюся длительность SQL*Net message from client?». Выходные данные Hotsos Profiler автоматически предоставляют ответ на этот вопрос (см. пример 12.8).¹

*Пример 12.8. Hotsos Profiler определяет вклад каждой команды SQL в длительность события SQL*Net message from client*

SQL Statement Id	Duration	
1525010069	23.1s	15.5%
2038166283	18.7s	12.5%
1966856986	17.6s	11.8%
1547563725	13.9s	9.3%
3230460720	10.8s	7.2%
77 others	65.1s	43.6%
Total	149.2s	100.0%

Процентное соотношение, приведенное в примере 12.8, показывает, что не существует какой-то одной команды SQL, отвечающей за непро-

¹ К сожалению, *tkprof* и Trace File Analyzer не в состоянии помочь в решении подобной задачи. Однако, опираясь на принцип опережающего атрибутирования, можно по необработанным данным трассировки определить, какие команды вносят основной вклад.

порционально большой вклад в длительность события SQL*Net message from client. Вот почему нельзя заметно сократить длительность этого события, ограничившись рассмотрением только одной команды SQL. Но начнем все же с первой, текст и статистики производительности которой приведены в примере 12.9.

*Пример 12.9. SQL-текст и статистики производительности для команды 1525010069, вносящей основной вклад в длительность события SQL*Net message from client в сеансе*

```
Statement Text
INSERT INTO STAGING_AREA (
  DOC_OBJ_ID, TRADE_NAME_ID, LANGUAGE_CODE, OBJECT_RESULT, GRAPHIC_FLAG,
  USER_LAST_UPDT, TMSP_LAST_UPDT
) VALUES (
  1000346, 54213, 'ENGLISH', '<BLANK>', 'N', 'sa',
  TO_DATE('11/05/2001 16:40:54', 'MM/DD/YYYY HH24:MI:SS')
)
```


Statement Cumulative Database Call Statistics							
Cursor Action			----- Response Time -----			LIO	PIO
Action	Count	Rows	Elapsed	CPU	Other	Blocks	Blocks
Parse	696	0	0.9	0.8	0.1	0	0
Execute	348	348	1.7	1.6	0.0	5,251	351
Fetch	0	0	0.0	0.0	0.0	0	0
Total	1,044	348	2.6	2.4	0.1	5,251	351
Per Exe	1	1	0.0	0.0	0.0	15	1
Per Row	1	1	0.0	0.0	0.0	15	1

Я не включил это в показанный здесь вывод, но по данным Hotsos Profiler в файле трассировки имелось 347 «похожих» команд. Profiler считает две команды SQL *похожими* в том и только в том случае, если они идентичны во всем (с точки зрения ядра Oracle), кроме значений литералов. Утилита Oracle tkprof не производит подобного суммирования неразделяемых команд SQL; вместо этого она вывела бы 348 различных команд SQL, указав, что каждая из них потребляет очень мало ресурсов. Конечно, это несколько усложнило бы анализ, но вся информация, необходимая для того, чтобы определить, что команды должны быть разделяемыми, присутствует в файле трассировки SQL.

Данные примера 12.9 поясняют, почему различных команд SQL так много. Команды место заполнителей (переменных связывания) занимают литералы:

```
1000346
54213
'ENGLISH'
'<BLANK>'
'N'
```

```
'sa'
'11/05/2001 16:40:54'
```

Значение, которое сразу же бросается в глаза как абсолютно препятствующее повторному использованию – это значение даты в последней позиции. Оно соответствует столбцу `TMS_LAST_UPDT` – временной метке последнего обновления. Сколько раз приложение могло бы повторно использовать идентичную команду SQL, содержащую жестко запрограммированное значение времени с секундным разрешением?

Конечно, вы наверняка думаете, что правильный ответ – ноль, но на самом деле тут есть небольшая хитрость. Данное конкретное приложение повторно использует команду ровно один раз (т. е. данное приложение в целом делает это ровно два раза). Обратите внимание на количество разборов и выполнений данной команды (и ей подобных): для 348 команд в целом насчитывается 696 вызовов разбора и 348 выполнений. Именно так, количество разборов в два раза больше количества выполнений! В разделе «Оптимизация разбора» главы 11 я советовал выносить вызовы разбора из циклов, чтобы приложение могло повторно использовать курсор, подготовленный одним вызовом разбора, несколько раз. Вывод Hotsos Profiler показывает (хотя этого и не видно в примере 12.9), что 696 вызовам разбора соответствует 348 непопадания в библиотечный кэш (из статистики `mis` необработанных данных трассировки). Невероятно, но данное приложение действительно разбирает каждую из этих команд SQL *дважды* для каждого вызова выполнения, как показано в примере 12.10.

Пример 12.10. Приложение выполняет два разбора для каждого выполнения... это нигде не годится

```
# ОЧЕНЬ ПЛОХОЙ, немасштабируемый код приложения
for each v in (897248, 897249, ...) {
    c = parse("select ... where orderid = ".v); # результат просто игнорируется
    c = parse("select ... where orderid = ".v); # второй раз повторяется
                                           # тот же разбор

    execute(c);
    data = fetch(c);
    close_cursor(c);
}
```

Первый вызов разбора для каждой команды приводит к непопаданию в библиотечный кэш (полный разбор), а второй – к попаданию в библиотечный кэш (частичный разбор).

Теперь давайте вспомним о том, в чем собственно состоит наша цель. События SQL*Net message from client занимают основную часть пользовательского времени отклика. Пример 12.7 показывает, что данная пользовательская операция выполняет в целом 6093 таких события, общий вклад которых в значение времени отклика составляет 149,2 секунды (каждое потребляет приблизительно 0,024482 секунды за вызов). SQL*Net message from client – это событие ожидания, которое ядро Oracle

выполняет между вызовами базы данных. Поэтому избавление от вызовов базы данных приведет к ликвидации некоторой части времени отклика, относящейся к событиям SQL*Net message from client. К счастью, мы только что обнаружили возможность избавления от 348 вызовов разбора. Надо просто найти способ прекратить двойной разбор для каждого вызова выполнения. Ожидаемая экономия при этом составит около 8,5 секунды за сеанс (около 5% общего времени отклика сеанса).

Не очень впечатляющее начало, но мы еще не закончили. В главе 11 я говорил, что при удалении вызова разбора из цикла удастся устранить *все вызовы разбора, кроме одного*. Если мы используем переменные связывания и сделаем код похожим на масштабируемый код приложения, приведенный в примере 11.2, то при этом мы избавимся от 695 ненужных вызовов разбора. Ожидаемая экономия пользовательского времени при этом составит около 17 секунд (приблизительно 10% общего времени отклика сеанса).

Нас заинтересовало, сколько еще команд SQL может страдать от тех же проблем, что и первая исследованная нами. Обратившись к подробной статистике для всех команд в выводе Hotsos Profiler, мы обнаружили, что кандидатами на устранение являются более 3000 вызовов базы данных. Ожидаемое улучшение времени отклика должно было составить около 73 секунд, т. е. порядка 45% общего времени отклика сеанса.

Но и это еще не все. Если помните, пользователь нажимает клавишу «ОК» всего один раз, а затем более 180 секунд ждет результата, при этом никакие другие входные данные для приложения не предполагаются. Но что же мы видим в примере 12.9? Команда INSERT, выполняющая вставку одной строки, выполняется 348 раз, обрабатывая в целом 348 строк. Зачем приложению выполнять 348 вызовов базы данных для вставки 348 строк? В Oracle имеется функция вставки массивом, которая может уменьшить количество команд INSERT с 348 до 4 (если приложение может работать с массивами из 100 строк). Если удастся реализовать подобное сокращение количества вызовов базы данных, то общее число вызовов в сеансе уменьшится более чем на 650 вызовов. Дополнительная экономия составит около 16 секунд, т. е. еще 10% общего времени отклика сеанса.

Если удастся осуществить все предложенные исключения вызовов базы данных, то будет удалено около 3650 вызовов, в результате чего время отклика сократится приблизительно на 89 секунд или на 54%. После избавления от этих вызовов базы данных пользователь может рассчитывать на улучшение времени отклика со 164 секунд до примерно 75 секунд. Возможно, что с приложением одновременно работает множество пользователей, тогда ненужные вызовы базы данных поглощают значительную часть пропускной способности сети. Если дело обстоит именно так, то избавление от ненужных вызовов базы данных

может снизить и задержки в очереди к сети, так что 0,024-секундная задержка событий SQL*Net message from client может уменьшиться до приблизительно 0,015 секунды. В этом случае оставшиеся после оптимизации примерно 2400 событий SQL*Net message from client будут выполнены всего за 37 секунд, тогда общее уменьшение пользовательского времени отклика составит около 110 секунд.

Результаты

При анализе файлов трассировки на занятиях у нас не всегда была возможность увидеть конечный результат наших рекомендаций. Это был как раз такой случай – рассматриваемая пользовательская операция осуществлялась в приложении стороннего производителя, поэтому три предложенные нами усовершенствования требовали его вмешательства. Студентка передала наши предложения поставщику программного обеспечения:

- Не выполнять каждый разбор дважды, сократить количество вызовов разбора для многих курсоров вдвое.
- Использовать переменные связывания и вынести вызовы разбора из циклов для дальнейшего снижения количества вызовов разбора до одного на курсор.
- Уменьшить общее количество вызовов базы данных за счет применения обработки массивами вместо построчной обработки.

Когда я пишу эту главу, мы и наша слушательница все еще ждем реакции поставщика. Ее компания планирует переход на новую версию программы вскоре после выхода книги. Мы будем держать вас в курсе происходящего на нашем сайте в Интернете.

Мораль

Эта история учит нас следующему:

- Нельзя просто игнорировать события ожидания SQL*Net message from client. Если они вносят значительный вклад во время отклика корректно выбранной пользовательской операции, то необходимо обратить на них внимание.
- Даже если с кодом SQL все в порядке, слишком большое количество вызовов базы данных может привести к ухудшению производительности. В этом случае может обнаружиться, что в командах SQL на самом деле есть изъяны. Но решение этой проблемы до устранения избыточных вызовов базы данных приведет к незначительным и практически незаметным результатам.

Пример 4: длительные события чтения

Мы регулярно получаем замечательные письма от людей, которые хотят рассказать нам о том, как им помогли наш метод и наши инстру-

менты. Данный пример появился в результате одного такого письма от нашего исландского знакомого, которому удалось уменьшить время отклика запроса с 6,5 часа до 10,9 секунды и исправить не найденную ранее функциональную ошибку – и все это за счет добавления в код SQL четырех байтов. Здесь рассказывается, как Метод R помог ему найти вызывавшую проблемы команду SQL. В итоге время отклика важного пакетного задания уменьшилось с восьми часов до одного.

Определение цели

Выбор приложения осуществлялся стандартным образом. Система принадлежала банку. Одно из пакетных заданий приложения выполнялось так долго, что не укладывалось в отведенное для него время. Оно запускалось каждый вечер в 23:00, а заканчивалось зачастую не раньше следующего полудня. Банку приходилось ограничивать количество обновляемых счетов, чтобы пакетное задание успевало завершить работу к открытию учреждения. Именно для этого задания наш знакомый и собрал данные расширенной трассировки SQL. В примере 12.11 приведен профиль ресурсов для восьмичасового выполнения программы.

Пример 12.11. Профиль ресурсов пакетного задания, которое выполняется почти восемь часов

Response Time Component	Duration		# Calls	Dur/Call
db file scattered read	19,051.1s	68.4%	1,828,249	0.010420s
CPU service	6,889.3s	24.7%	959,148	0.007183s
db file sequential read	1,892.7s	6.8%	406,417	0.004657s
latch free	29.0s	0.1%	1,071	0.027106s
log file switch completion	1.6s	0.0%	14	0.112143s
SQL*Net message from client	0.3s	0.0%	10	0.034000s
log buffer space	0.1s	0.0%	1	0.100000s
log file sync	0.1s	0.0%	4	0.022500s
file open	0.1s	0.0%	54	0.001296s
buffer busy waits	0.0s	0.0%	14	0.002143s
undo segment extension	0.0s	0.0%	2,111	0.000014s
SQL*Net message to client	0.0s	0.0%	10	0.000000s
Total	27,864.4s	100.0%		

Верхние строки ясно говорят о неэффективности SQL: множество событий файлового чтения и использования процессора. Остается выявить команды SQL, потребляющие столько ресурсов. В данном случае выполнение *tkprof* с параметром `sort=prsdsk,exedsk,fchdsk` приводит к тому, что во главе списка находится команда SQL, содержащая наибольшее количество блоков PIO.



Однако имейте в виду, что сортировка в *tkprof* по количеству блоков PIO – это не то же самое, что сортировка по общей продолжительности вызовов PIO. На самом деле нам нужна сортировка команд по общей продолжительности ввода/вывода,

но *tkprof* предоставляет такую информацию только в версии 9i. Поэтому при работе с *tkprof* необходимо визуально исследовать ее выходные данные с тем, чтобы убедиться, что выявлена именно интересующая вас команда SQL.

Наш аналитик знал, какая команда SQL внесла наибольший вклад в ухудшение производительности db file scattered read, т. к. он опирался на данные Hotsos Profiler, приведенные в примере 12.12.

Пример 12.12. Hotsos Profiler определяет вклад каждой команды SQL в длительность события db file scattered read

SQL Statement Id	Duration	
-----	-----	-----
1163242303	19,028.9s	99.9%
1626975503	6.7s	0.0%
808413641	5.2s	0.0%
3187134541	3.7s	0.0%
1594054818	2.4s	0.0%
8 others	4.3s	0.0%
-----	-----	-----
Total	19,051.1s	100.0%

Диагностика и лечение

Теперь необходимо проанализировать производительность SQL-команды 1163242303 (это значение hv для команды из секции PARSING IN CURSOR необработанных данных трассировки). Текст и статистики производительности данной команды приведены в примере 12.13.



Имейте в виду, что значение hv лишь почти уникально (т. е. hv не уникально). Две разные команды SQL могут иметь одно и то же значение hv. Такое встречается нечасто, но все же встречается.

Пример 12.13. Код SQL и статистики производительности для команды 1163242303, вносящей основной вклад в длительность события db file scattered read в исследуемом сеансе

```
Statement Text
SELECT EIGANDI, INNLENT_ERLENT, VERDTRYGGING, SKULDFLOKKUN, VBRTEGUND, FLOKKUR
FROM V_SKULDABREF_AVOXTUN
WHERE EIGANDI = :b1
AND INNLENT_ERLENT = :b2
AND ((RAFVAETT = :b3 )
OR ((RAFVAETT = :b4 )
AND (INNLAUSN IS NULL
OR INNLAUSN > :b5 )
AND (VIDMIDDAGS <= :b6 )))
GROUP BY EIGANDI, INNLENT_ ERLENT, VERDTRYGGING, SKULDFLOKKUN, VBRTEGUND, FLOKKUR
ORDER BY EIGANDI, INNLENT_ ERLENT, VERDTRYGGING, SKULDFLOKKUN, VBRTEGUND, FLOKKUR

Statement Cumulative Database Call Statistics
Cursor Action          ----- Response Time -----          LIO          PIO
```

Action	Count	Rows	Elapsed	CPU	Other	Blocks	Blocks
Parse	3,739	0	1.9	0.7	1.2	147	17
Execute	3,739	0	1.7	1.6	0.2	0	0
Fetch	4,212	473	23,466.4	4,135.6	19,330.8	36,566,201	36,550,345
Total	11,690	473	23,470.0	4,137.9	19,332.1	36,566,348	36,550,452
Per Exe	1	0	6.3	1.1	5.2	9,780	9,092
Per Row	8	1	49.6	8.8	40.9	77,307	71,868

Код SQL этой команды вполне понятен. Если отвлечься от исландских названий объектов, то команда – это просто запрос из одного объекта. Очевидно, что здесь нет даже соединения. Однако план выполнения запроса, полученный из строк STAT файла трассировки и приведенный в примере 12.14, показывает, что все обстоит немного по-другому.

Там происходит нечто более сложное, чем простой запрос к одной таблице. В действительности V_SKULDABREF_AVOXTUN – это представление. Неотработанный файл трассировки это подтверждает. Вызов разбора для SELECT, приведенный в примере 12.13, потребовал рекурсивных вызовов разбора, выполнения и выборки из VIEW\$, аналогично запросу из DBA_OBJECTS, описанному в главе 5, поэтому следующим объектом нашего внимания становится определение представления V_SKULDABREF_AVOXTUN. Файл расширенной трассировки SQL в Oracle не включает в себя определения всех представлений, к которым обращаются команды SQL, упомянутые в этом файле. Однако поскольку из файла трассировки ясно видно, что V_SKULDABREF_AVOXTUN является представлением, остается лишь обратиться за его определением к DBA_VIEWS. Определение интересующего нас представления приведено в примере 12.15.

Пример 12.14. План выполнения трудоемкой команды SELECT перед оптимизацией

Rows	Row	Source	Operation (Object Id)
473		SORT ORDER BY	
473		SORT GROUP BY	
974		VIEW V_SKULDABREF_AVOXTUN	
974		SORT UNIQUE	
974		UNION-ALL	
686		HASH JOIN	
886		TABLE ACCESS FULL SKULDABREF_AVOXTUN(21435)	
103,247		TABLE ACCESS FULL VBR_FLOKKAR(19409)	
288		FILTER	
288		HASH JOIN	
940		TABLE ACCESS BY INDEX ROWID BREF(19460)	
2,649		INDEX RANGE SCAN(20593)	
10,744		TABLE ACCESS FULL VBR_FLOKKAR(19409)	

Пример 12.15. Определение представления, лежащего в основе проблемы производительности

```

CREATE OR REPLACE VIEW v_skuldabref_avoxtn (
    eigandi,
    innlent_erlent,
    verdtrygging,
    skuldflokkun,
    vbrtegund,
    flokkur,
    rafvaett,
    ostadladur,
    brefnumer,
    vidmiddags,
    innlausn,
    nafnverd,
    kaupkrafa,
    ees,
    vextir )
AS
select
    s.eigandi,
    s.innlent_erlent,
    s.verdtrygging,
    s.skuldflokkun,
    s.vbrtegund,
    s.flokkur,
    'N' rafvaett,
    nvl(f.ostadlad, 'N') ostadladur,
    s.brefnumer,
    s.vidmiddags,
    s.innlausn,
    s.nafnverd,
    s.kaupkrafa,
    s.ees,
    null vextir
from f.jastofn.vbr_flokkar f,
     fja_pfm.skuldabref_avoxtn s
where f.audkenni=s.flokkur and
       f.rafvaett is null

union
select
    s.eig eigandi,
    'I',
    Fja_pfm.Ymis_Foll.TegundTryggingar(f.visitale) verdtrygging,
    f.skuldaranumer skuldflokkun,
    f.vbrtegund vbrtegund,
    s.aud flokkur,
    'J' rafvaett,
    'N' ostadladur,
    'x' brefnumer,

```

```
to_date(null) vidmiddags,
to_date(null) innlausr,
s.nav nafnverd,
0 kaupkrafa,
'+' ees,
null vextir
from fjastofn.vbr_flokkar f,
fja_pfm.bref s
where f.audkenni=s.aud and
f.rafvaett is not null
/
```

Аналитик обсудил данное определение представления с разработчиком, который представлял себе его значение для бизнес-процессов. По результатам обсуждения они пришли к выводу, что определение представления было некорректным. Использование в определении представления объединения UNION двух команд SELECT вместо UNION ALL приводило к возникновению *двух* проблем:

- Выполнялась чрезвычайно дорогостоящая, но ненужная операция над источником строк SORT UNIQUE (новые данные, передаваемые ядром Oracle release 9.2 в строках STAT, ярче отражают чудовищность этих затрат).
- Возникла и просто ошибка, т. к. операция UNION ошибочно исключала строки, которые были необходимы пользователям приложения.

В примере 12.16 представлен план выполнения команды из примера 12.13 после того, как было исправлено определение представления (в определение были добавлены байты ALL).

Пример 12.16. План выполнения трудоемкой команды SELECT после оптимизации представления

Rows	Row	Source	Operation (Object Id)

	473	SORT	ORDER BY
	473	SORT	GROUP BY
	974	VIEW	V_SKULDABREF_AVOXTUN
	974	UNION	-ALL
	686	HASH	JOIN
	886	INDEX	RANGE SCAN(21436)
103,314		TABLE	ACCESS FULL VBR_FLOKKAR(19409)
	288	FILTER	
	288	HASH	JOIN
	940	INDEX	RANGE SCAN(29887)
10,744		TABLE	ACCESS FULL VBR_FLOKKAR(19409)

Пример 12.17 содержит более волнующие новости. Запрос, который раньше потреблял 23470,0 секунд времени отклика, теперь занимает всего 10,9 секунды. Он выводит тот же (а на самом деле – лучший) результат, причем тратит на это приблизительно на 6,5 часа меньше.

Пример 12.17. Код SQL и статистики производительности для команды 1163242303 после изменения представления. Обратите внимание, что текст SQL полностью повторяет приведенный в примере 12.13, изменилось только определение представления. В целом время отклика для команды уменьшилось с более чем 23 000 секунд до примерно 10

```
Statement Text
SELECT EIGANDI, INNLENT_ERLENT, VERDTRYGGING, SKULDFLOKKUN, VBRTÉGUND, FLOKKUR
FROM V_SKULDABREF_AVOXTUN
WHERE EIGANDI = :b1
AND INNLENT_ERLENT = :b2
AND ((RAFVAETT = :b3 )
OR ((RAFVAETT = :b4 )
AND (INNLAUSN IS NULL
OR INNLAUSN > :b5 )
AND (VIDMIDDAGS <= :b6 )))
GROUP BY EIGANDI, INNLENT_ ERLENT, VERDTRYGGING, SKULDFLOKKUN, VBRTÉGUND, FLOKKUR
ORDER BY EIGANDI, INNLENT_ ERLENT, VERDTRYGGING, SKULDFLOKKUN, VBRTÉGUND, FLOKKUR
```

Statement Cumulative Database Call Statistics							
Cursor Action			----- Response Time -----			LIO	PIO
Action	Count	Rows	Elapsed	CPU	Other	Blocks	Blocks
-----	-----	-----	-----	-----	-----	-----	-----
Parse	3,722	0	2.0	0.6	1.4	44	1
Execute	3,722	0	1.3	1.4	-0.1	14	0
Fetch	4,195	473	7.6	2.8	4.8	44,764	792
-----	-----	-----	-----	-----	-----	-----	-----
Total	11,639	473	10.9	4.8	6.2	44,822	793
Per Exe	1	0	0.0	0.0	0.0	12	0
Per Row	8	1	0.0	0.0	0.0	95	2

Результаты

Результаты были ошеломляющими. В примере 12.18 приведен профиль ресурсов задания после оптимизации. Общее время отклика для восьмичасового пакетного задания сократилось до чуть более одного часа.

Пример 12.18. Профиль ресурсов для того же задания, выполнение которого занимало почти восемь часов (сравните с примером 12.11). После оптимизации задание выполняется чуть больше часа

Response Time Component		Duration		# Calls	Dur/Call
-----		-----		-----	-----
CPU service	2,684.7s	73.9%		953,452	0.002816s
db file sequential read	847.6s	23.3%		77,944	0.010874s
unaccounted-for	93.2s	2.6%			
db file scattered read	5.8s	0.2%		295	0.019627s
log file switch completion	1.6s	0.0%		7	0.234286s
latch free	1.0s	0.0%		362	0.002873s
file open	0.1s	0.0%		49	0.002041s
log file sync	0.1s	0.0%		7	0.011429s
buffer busy waits	0.0s	0.0%		1	0.010000s

SQL*Net message from client	0.0s	0.0%	10	0.001000s
SQL*Net message to client	0.0s	0.0%	10	0.000000s

Total	3,634.1s	100.0%		

Перед исправлением представления V_SKULDABREF_AVOXTUN банк ограничивал количество счетов, для которых запускалось задание. В противном случае выполнение задания на несколько часов опаздывало бы к открытию банка. После внесения исправлений они могли передавать в запрос практически любые параметры, все равно его выполнение не очень замедлялось. После оптимизации длительность пакетного задания никогда даже не приближалась к восьмичасовому значению, которое соответствовало прежнему заданию.

Мораль

- Этот случай обращает наше внимание на следующие важные факты:
- Профиль ресурсов, отражающий большие длительности событий db file... и CPU service, обычно указывает, что где-то в пользовательской операции исполняется неэффективный код SQL. Для исправления ситуации необходимо найти этот код.
 - Файл расширенной трассировки SQL содержит всю информацию, необходимую аналитику для оптимизации процесса выбора цели (поиска основной причины проблемы производительности пакетного задания). Даже несмотря на то, что данные трассировки не включают в себя определение представления, они содержат сведения, которые обратили наше внимание на такой источник ухудшения производительности, как определение представления.
 - Иногда внимательный анализ производительности приводит к выявлению функциональных ошибок. В данном случае аналитик по производительности смог определить, что запрос выполнял не только больше работы, чем следовало бы, но и в некоторых обстоятельствах просто возвращал некорректный результат.

Закключение

Покидая в 1999 г. корпорацию Oracle, чтобы основать собственное дело *hotsos.com*, я не так уж хорошо умел оптимизировать производительность Oracle. Мне казалось, что когда-то это *было* так, но теперь выясняется, что даже это не совсем так. Однако за четыре года, прошедших с момента создания новой компании, я думаю, что значительно усовершенствовался. Мне была доступна роскошь обучения на двух самых мощных (с моей точки зрения) пособиях, из тех, с которыми человек когда либо имел дело:

Погружение

Я смог погрузиться в сферу оптимизации производительности Oracle. В течение четырех лет моя профессиональная жизнь была сосредоточена на изучении оптимизации производительности Oracle, на выполнении оптимизации и на обучении такой оптимизации.

Копирование хороших примеров

Мне представилась возможность учиться на опыте Джеффа Хольта и еще многих людей, перечисленных мною в предисловии.

Решение о погружении, конечно, за вами. Но я искренне надеюсь, что эта новая книга станет для вас полезным источником хороших идей и примеров, которые вы сможете использовать в своей повседневной профессиональной деятельности для более быстрого и полного избавления от неприятностей, связанных с ухудшением производительности.

Спасибо за то, что читаете эту книгу. Надеюсь, что она окажется для вас полезной.

Часть IV. Приложения



Глоссарий

Mathematica

Программный пакет, позволяющий быстро и точно выполнить математические вычисления аналитически, численно и графически.

UTC

См. *Всеобщее скоординированное время (Coordinated Universal Time)*.

Аналитическое решение (Closed form solution)

Математический результат, который может быть выражен в символической форме. Противоположность математического результата, который может быть выражен только приближенно в числовой форме.

Время между поступлениями τ (Interarrival time)

Промежуток времени между последовательными поступлениями запросов в систему массового обслуживания. Время между поступлениями – величина, обратная скорости поступления.

Время обслуживания S (Service time)

Время использования ресурса системы массового обслуживания, затраченное на обслуживание поступившего запроса.

Время обслуживания – величина, обратная скорости обслуживания.

Время отклика R , задержка (Response time (R), latency)

Время, затрачиваемое системой или функциональной единицей на то, чтобы отреагировать на переданные ей входные данные. В системах массового обслуживания время отклика складывается из времени обслуживания и задержки в очереди.

Время размышлений (Think time)

Время, потраченное на том уровне, который находится на один выше, чем исследуемый уровень технологического стека.

Всеобщее скоординированное время (Coordinated Universal Time, UTC)

Международный стандарт времени. Сейчас этот термин используется вместо устаревшего «среднее время по Гринвичу» (Greenwich Meridian Time, GMT). Ноль часов по UTC соответствует полуночи в английском городе Гринвиче, расположенном на нулевом меридиане. Сутки в универсальном времени содержат 24 часа, поэтому, например, 4 часа пополудни обозначаются в нем как 16:00 UTC («шестнадцать часов, ноль минут»). (Источник – <http://www.ghcc.msfc.nasa.gov/utc.html>.)

Выборка (Sampling)

См. *опрос*.

Вызов базы данных (Database call)

Подпрограмма ядра Oracle.

Глоссарий (Glossary)

Приложение к книге, в котором автор может в значительной степени избежать вмешательства редактора и дать такие определения терминов, которые, на его взгляд, хотя бы немного помогут читателю.

Деление интервала пополам (Interval bisection)

Численный метод приближенного решения уравнений. Когда невозможно получить результат в символьном виде, следует применять численные методы решения, к которым относится и метод деления пополам. Приведенный псевдокод демонстрирует применение данного метода:

```
function solve(function f, real s, real delta, real a, real b) {
  # Требуется найти такое значение x, где f(x) == s.
  # Возвращаемое значение – интервал, содержащий x,
    длина интервала < delta.
  # Известно, что решение находится в интервале [a,b] (т.е. a < x < b).
  # Функция f должна быть непрерывной и монотонной для всех x
    в интервале [a,b].
  # Вычисление заканчивается, когда x попадает в delta-окрестность
    истинного решения.
  f = f - s;                                # f == s при f - s == 0
  if (not (f(a) < f(b))) f = -f;             # удостовериться, что f возрастает
  while (not (b - a < delta))
    if (f((a+b)/2) < 0) a = (a+b)/2;         # решение справа от (a+b)/2
    else b = (a+b)/2;                       # решение слева от (a+b)/2
  return [a,b];
}
```

Диаграмма последовательности (Sequence diagram)

Графический способ изображения времени отклика, в соответствии с которым ресурсы технологического стека представляются на нескольких параллельных осях времени. Потребление ресурса отображается отрезком на оси данного ресурса. Движение запросов

и услуг между уровнями технологического стека изображается направленными линиями, соединяющими оси времени.

Динамическое представление производительности (Dynamic performance view)

См. *фиксированное представление*.

Дисциплина обслуживания FCFS (First-come, first-served)

Дисциплина массового обслуживания, при которой очередная порция услуг предоставляется первому запросу в очереди, независимо от его присвоенного ему класса обслуживания. Другое название этой дисциплины – FIFO (*first-in, first-out* – первым вошел, первым вышел).

Дисциплина очереди (Queue discipline)

Правила, определяющие порядок обслуживания конкурирующих просителей. В качестве примеров дисциплин очереди приведем обслуживание по принципу «первым пришел – первым обслужен» (*first-come, first-served* – FCFS); обслуживание с приоритетами; первоочередное обслуживание самых настырных личностей.

Задержка (Latency)

Синоним *времени отклика*.

Задержка в очереди (Queueing delay) (W)

Время, которое запрос, поступающий в систему массового обслуживания, тратит на ожидание обслуживания от ресурса, занятого обслуживанием другого запроса. Задержка в очереди – это *не* то же самое, что значение `ela` из строк `WAIT` данных файла трассировки, передаваемых ядром Oracle.

Закон Амдала (Amdahl's law)

Формализованное Джином Амдалом [Amdahl (1967)] интуитивное правило, позволяющее аналитику производительности рассчитать значимость различных предполагаемых действий по повышению производительности:

Повышение производительности, достигаемое за счет некоторого улучшения, ограничено той долей времени выполнения, которая приходится на улучшаемый компонент.

Защелка (Latch)

Структура данных, созданная для предотвращения одновременного выполнения определенного участка кода ядра Oracle двумя процессами. Разработчики ядра Oracle применяют простой протокол получения защелок, позволяющий их программам избегать повреждения объектов, расположенных в разделяемой памяти. Протокол применения защелок в Oracle выглядит приблизительно так [Millsap (2001c)]:

```

пока защелка для планируемой операции недоступна {
    ожидание
}
получить защелку
выполнить требуемую операцию
освободить защелку

```

Избыточное ограничение (Over-constrained)

Элемент спецификации, определяющий противоречивые условия. Например, следующие требования накладывает избыточное ограничение: «Значение x должно быть меньше 0,001.... Значение x должно быть больше 0,009». Обычно такие противоречивые условия входят в состав более сложных требований, выявление противоречий в которых требует гораздо более сложного анализа и, как правило, значительных затрат.

Излом ρ^* (Knee)

Значение коэффициента использования ρ , при котором отношение времени отклика R к ρ (R/ρ) минимально. Соответствующий излому коэффициент ρ часто считается оптимальным для системы массового обслуживания, поскольку при этом одновременно достигаются небольшое время отклика и высокая степень использования ресурсов системы.

Измерительные средства (Instrumentation)

Строки кода, добавленные в исходный текст программы с целью измерения ее производительности.

Интегральная функция распределения (Cumulative distribution function, CDF)

Вероятность того, что случайная переменная X имеет значение, меньшее или равное заданному значению x , обозначаемая как $P(X \leq x)$. Интегральная функция распределения времени отклика особенно полезна при составлении соглашения об уровне обслуживания, т. к. делает возможными формулировки вида «Время отклика пользовательской функции f составит не менее r секунд как минимум в p процентах выполнений f ».

Интенсивность поступления λ (Arrival rate)

Количество запросов, поступающих в систему массового обслуживания в единицу времени на заданном временном интервале.

Интенсивность трафика ρ (Traffic intensity)

Средний коэффициент использования параллельного канала обслуживания в системе массового обслуживания.

Интервальный таймер (Interval timer)

Цифровое устройство для измерения времени, «тикающее» через равные промежутки времени.

Канал обслуживания, параллельный канал обслуживания m , s или s (Service channel, parallel service channel)

Ресурс системы массового обслуживания, который принимает поступающие в систему запросы. Количество параллельных каналов обслуживания внутри системы в разных источниках может обозначаться буквами m (как в нашей книге, где мы пишем $M/M/m$), s ($M/M/c$) или s .

Коэффициент использования (Utilization)

Отношение объема использованного ресурса к общему имеющемуся его объему за определенный период времени.

Коэффициент попаданий в кэш буферов базы данных (Database buffer cache hit ratio)

Отношение $(L - P)/L$, где L – количество вызовов *логического ввода/вывода* Oracle (LIO), а P – количество вызовов *физического ввода/вывода* Oracle (PIO). См. *обманчивость относительных величин*.

Критерий согласия хи-квадрат (Chi-square goodness-of-fit test)

Статистический критерий, позволяющий определить, насколько хорошо набор значений подчиняется определенному закону распределения.

Логический ввод/вывод (LIO), логическое чтение (Logical I/O, Oracle logical I/O, Oracle logical read)

Операция, в которой ядро Oracle получает и обрабатывает содержимое блока из кэша буферов базы данных. Код операции LIO в Oracle включает инструкции для проверки наличия запрошенного блока в кэше буферов, изменения внутренних структур данных (такие, как хеш-цепочки буферов кэша и цепочки LRU), захвата блока и разбора и фильтрации его содержимого. Операции LIO в Oracle выполняются в двух режимах: *согласованном* (*consistent*) и *текущем* (*current*). В согласованном режиме блок может быть скопирован (или *клонирован*), а изменениям подвергается копия, что позволяет восстановить состояние блока на заданный момент времени. В текущем режиме блок просто извлекается из кэша «как есть».

Максимальная рабочая пропускная способность λ_{\max} (Maximum effective throughput)

Наибольшая пропускная способность, которая может быть достигнута в системе массового обслуживания при том условии, что среднее время отклика не превысит заданного предельного значения.

Маниакально-настроечный психоз (Compulsive tuning disorder (CTD))

Термин, который Гаджа Вайдианатха (Gaja Vaidyanatha) и Кирти Дешпанде (Kirti Deshpande) ввели для обозначения последствий применения метода повышения производительности, не имеющего условия завершения:

У многих администраторов баз данных вошло в привычку заниматься настройкой до тех пор, пока они могут найти хоть что-то, поддающееся настройке. Это не только сводит их (и пользователей) с ума из-за простоев системы, но и не дает заметного улучшения производительности. Нет сомнений, что распространение маниакально-настроечного психоза (Compulsive Tuning Disorder – CTD) среди администраторов приобретает характер эпидемии [Vaidyanatha and Deshpande (2001) 8]

Масштабируемость (Scalability)

Степень изменения времени отклика в зависимости от некоторого выбранного параметра. Например, можно говорить о масштабируемости запроса применительно к количеству возвращаемых им строк, о масштабируемости системы применительно к числу работающих процессоров и т. д.

Математическое ожидание (Expected value) $E[X]$

Среднее значение случайной переменной.

Метод (Method)

Детерминированная последовательность шагов. Качество метода определяется его действенностью, эффективностью, измеримостью, прогнозируемостью, достоверностью, определенностью, конечностью и практичностью.

Методология (Methodology)

Учение о методах и средствах деятельности.

Однако в последние годы слово «методология» часто употребляют как претенциозную замену слова «метод» в техническом и научном контексте... Злоупотребление словом «методология» размывает важное концептуальное различие между инструментами научного исследования (собственно «методами») и принципами, определяющими пути их применения и интерпретации – различие, которое если не широкая публика, то хотя бы научное и образовательное сообщества, должны были бы поддерживать. (Источник: «American Heritage Dictionary of the English Language» (Словарь американского (культурного) наследия).)

Модель массового обслуживания $M/M/m$ (или $M/M/c$) (queueing model)

Набор математических формул, способных предсказать производительность систем массового обслуживания, удовлетворяющих пяти определенным критериям:

- Значения интервалов времени между поступлениями запросов распределяются экспоненциально.
- Значения времени обслуживания распределяются экспоненциально.
- Имеется m параллельных каналов обслуживания с одинаковыми функциональностью и производительностью, каждый из которых в равной мере способен обслужить любой прибывающий запрос.

- Отсутствует ограничение на длину очереди. Ни один попавший в очередь запрос не покидает ее, не получив требуемого обслуживания.
- Очередь придерживается дисциплины обслуживания FCFS (первым пришел, первым обслужен). Система обслуживает запросы в порядке их получения.

Надежность (Reliable)

Способность метода обеспечивать одинаковую степень правильности при *каждом* исполнении. Например, метод, дающий неверный ответ при каждом применении, надежен. Метод, который дает верный ответ при каждом применении, также является надежным. Метод ненадежен, если он иногда выдает правильный ответ, а иногда неправильный.

Напрасные траты (Waste)

Все, от чего можно избавиться, не утратив при этом ничего полезного. В контексте рабочей нагрузки компьютерной системы напрасной является любая нагрузка, которую можно устранить без потери функциональности для бизнеса.

Настраивать (Tune)

Увеличивать производительность работы некоторого объекта. См. *оптимизировать*.

Обманчивость относительных величин (Ratio fallacy, ratio games)

Недостаток, присущий *любой* относительной величине, состоящий в том, что производительность измеряемой системы ухудшается при очевидном улучшении ее характеристик, выраженных в относительных величинах. Величины, представляющие собой отношения, обманчивы потому, что их значение можно изменить, изменив *как* числитель, *так и* знаменатель. Когда улучшают отношение, ухудшая характеристики измеряемой системы, это называется *играми* с системой.

Например, консультант, вознаграждение которого пропорционально его коэффициенту занятости, может играть со схемой выплат, выторговывая для себя меньшую норму производительности. Торговый представитель может играть с коэффициентом удачных сделок, реже посещая тех потенциальных клиентов, вероятность совершения сделок с которыми ниже. Администратор базы данных может играть с коэффициентом попаданий в кэш буферов, увеличивая количество вызовов LIO к блокам, расположенным в памяти. Играть можно с *любым* относительным показателем.

Оператор DDL (Data definition language (DDL) statement)

Оператор SQL, выполняющий создание, изменение, обслуживание или удаление объекта схемы, или же изменяющий привилегии пользователя.

Оператор DML (Data manipulation language (DML) statement)

Оператор SQL, выполняющий выборку, вставку, изменение, удаление или блокировку данных.

Опережающее атрибутирование (Forward attribution)

Метод, при котором длительность, приведенная в строке `WAIT #n` файла трассировки, сопоставляется первому вызову базы данных для курсора `#n`, *следующему* за строкой `WAIT` файла трассировки. Такое атрибутирование длительностей событий ожидания Oracle позволяет точно идентифицировать исходный код приложения, отвечающий за данное время «ожидания».

Опрос (Polling)

Метод измерения процессом некоторого явления, состоящий в фиксации состояния системы через определенные, обычно постоянные, интервалы времени. Другое название – *выборка (sampling)*.

Оптимизатор по правилам (Rule-based optimizer (RBO), Oracle rule-based query optimizer)

Компонент ядра Oracle, который рассчитывает план выполнения запроса, руководствуясь статическим списком приоритетов операций над источниками строк. На результат выбора влияют только конфигурация схемы базы данных, SQL-текст и встроенный в исходный текст ядра Oracle список приоритетов операторов.

Оптимизатор по стоимости (Cost-based optimizer (CBO), Oracle cost-based query optimizer)

Компонент ядра Oracle, который рассчитывает план выполнения запроса, выбирая план с наименьшей ожидаемой стоимостью. На результат выбора влияют параметры сеанса и экземпляра Oracle, статистика таблиц и индексов, статистика использования процессора и ввода/вывода Oracle, определения схемы, хранимые планы выполнения, текст SQL и модель стоимости запросов, встроенная в код ядра Oracle.

Оптимизация SQL (SQL optimization)

Процесс удаления фрагментов кода из операций ядра Oracle, возникающих при выполнении инструкций, написанных на SQL.

Оптимизировать (Optimize)

Довести до максимума экономическую ценность некоторого объекта. Ср. с *настраивать*.

Ошибка квантования (Quantization error)

Разница между действительной продолжительностью события и его же продолжительностью, измеренной с помощью дискретных часов.

Ошибка переполнения (Overflow error)

Ошибка, возникающая, когда результат операции сложения или умножения не помещается в отведенное ему в памяти место. Например, переменная j целого типа разрядностью n может принимать значения от 0 до 2^{n-1} . Если к j , равной 2^{n-1} , прибавить единицу, то j будет присвоено значение 0.

Параллелизм (Concurrency)

Характеризует количество пользователей и пакетных заданий, требующих одновременного обслуживания.

Параллельный канал обслуживания (Parallel service channel)

См. *канал обслуживания*.

Планировщик (Scheduler)

Подпрограмма операционной системы, распределяющая циклы процессора между конкурирующими процессами операционной системы.

Подкачка страниц (Paging)

Процесс переноса страниц из оперативной памяти на диск, вызываемый требованием выделить память в объеме, превышающем установленный в системе.

Пользовательская операция (User action)

Единица работы, выходные данные и производительность которой являются значимыми для бизнеса, например заполнение поля или формы или выполнение одной или нескольких программ.

Прерывание (Interrupt)

Сигнал, посылаемый оборудованием ядру операционной системы. Из книги [Bach (1986) 16, 22]:

Система UNIX позволяет таким устройствам, как внешние устройства ввода/вывода и системные часы, асинхронно *прерывать* работу центрального процессора. Получив сигнал прерывания, ядро операционной системы сохраняет свой текущий *контекст* (застывший образ выполняемого процесса), устанавливает причину прерывания и обрабатывает его. После того, как прерывание будет обработано ядром, прерванный контекст восстановится и работа продолжится так, как будто ничего не случилось... В системах Unix прерывания обрабатываются специальными функциями ядра операционной системы, которые вызываются в контексте текущего процесса.

Прерывание по таймеру (Clock interrupt)

Прерывание, извещающее ядро операционной системы о завершении одного или более интервалов времени [Bovet and Cesati (2001) 140].

Приведенная стоимость (ПС) (Present value (PV))

Приведенная стоимость (ПС) определяется по следующей формуле:

$$PV = \frac{C_1}{1 + r}$$

где C_1 – стоимость в будущем, r – норма прибыли, закладываемая инвесторами при приеме отложенных платежей [Brealey and Myers (1988), 12–13]. Формула ПС позволяет сравнивать стоимости, существующие в различные моменты времени в будущем. Например, если ожидаемая годовая норма прибыли составляет $r = 0,07$, то приведенная к текущему моменту величина ожидаемой через год прибыли от проекта \$10 000 составит на текущий момент лишь \$9345,79. Если бы вы сегодня инвестировали \$9345,79 под 7% годовых, то через год ваши вложения увеличились бы до \$10000.

Пробуксовка (Thrashing)

Потребление избыточной мощности исключительно для обеспечения собственных накладных расходов системы. Например, планировщик центрального процессора операционной системы обычно потребляет менее $x\%$ общей мощности процессора системы (сегодня во многих системах $x = 10$). Когда нагрузка системы увеличивается настолько, что планировщик процессора начинает потреблять более $x\%$ мощности исключительно для выделения процессора, говорят о пробуксовке планировщика.

Программа (Program)

Последовательность компьютерных команд, выполняющая некоторую бизнес-функцию.

Пропускная способность (Throughput) (X)

Скорость выполнения запросов в системе массового обслуживания (их количество в единицу времени)

Профиль ресурсов (Resource profile)

Таблица, предлагающая удобное разложение времени отклика на составляющие. Обычно профиль ресурсов отображает как минимум: 1) компонент времени отклика, 2) общую продолжительность действий определенной категории, 3) количество выполненных операций данной категории. В профиле ресурсов события часто расположены в порядке убывания их продолжительности.

Пул соединений (Connection pooling)

Технология, благодаря которой множество пользовательских сеансов могут совместно работать с меньшим количеством сеансов Oracle. Разработана с целью уменьшить количество операций открытия и закрытия *соединений* в базе данных, что позволяет повысить производительность систем с очень большим количеством пользователей.

Разрешение таймера (Resolution, clock resolution)

Фактическая продолжительность между последовательными тиками цифрового таймера. Разрешение таймера – величина, обратная тактовой частоте.

Распределение Пуассона (Poisson distribution)

Распределение, для которого функция плотности вероятности имеет вид:

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!}, \quad x = 0, 1, 2, \dots,$$

где λ – среднее значение распределения. В 1909 г. Агнер Эрланг (Agner Erlang) показал, что скорость поступления вызовов в телефонной сети подчиняется распределению Пуассона. В компьютерных системах многие процессы поступления запросов и процессы обслуживания также подчиняются этому распределению.

Рекурсивный SQL (Recursive SQL)

Любая команда SQL, для которой в данных трассировки Oracle присутствует отличное от нуля значение глубины курсора `dep`.

Риск (Risk)

Неопределенность в будущих прибылях или затратах. Количественно оценивается при помощи распределения вероятности [Bodie, et al. (1989) 112].

Сеанс (Session)

См. *сеанс Oracle*.

Сеанс Oracle (Oracle session)

Из руководства «Oracle Database Concepts»:

Сеанс – это определенное соединение пользователя с экземпляром Oracle посредством пользовательского процесса.

В Oracle различают *соединение* (путь взаимодействия) и сеанс. Можно быть соединенным с Oracle и не иметь сеанса. С другой стороны, можно открыть одновременно несколько сеансов, использующих единственное имеющееся соединение.

Система (System)

Для поставщика информации *система* обычно представляет собой набор процессов, файлов и сегментов разделяемой памяти, которые составляют приложение. Для потребителя информации *система* – это сущность, предоставляющая обслуживание в ответ на пользовательские действия. Несовпадение этих двух образов часто приводит к такой «оптимизации» со стороны поставщиков информации, которая или никак не сказывается на производительности важных пользовательских операций, или даже ухудшает ее.

Систематическая ошибка (Systematic error)

Результат некоторой экспериментальной «ошибки», приводящей к постоянному отклонению в измерениях. Систематические ошибки обычно постоянны для всех измерений или же медленно изменяются с течением времени [Lilja (2000)].

Системный вызов (System call, sys call)

Подпрограмма ядра операционной системы [Stevens (1992) 20].

Скорость выполнения X (Completion rate)

Пропускная способность системы массового обслуживания.

Скорость обслуживания μ (Service rate)

Количество запросов на обслуживание, которое один канал системы массового обслуживания может обработать за выбранную единицу времени. Скорость обслуживания – величина, обратная времени обслуживания.

Случайная величина (Random variable)

Функция, значением которой является случайное число. Случайная величина характеризуется математическим ожиданием, распределением и, возможно, другими параметрами, такими как среднее квадратическое отклонение.

Событие простоя (Idle event)

Событие ожидания Oracle, возникающее между вызовами базы данных. Слово «простой» здесь означает, что ядро Oracle закончило обработку вызова базы данных и ожидает следующего. Многие аналитики утверждают, что события простоя, встретившиеся в диагностических данных, следует игнорировать. Но если область сбора данных выбрана правильно, то события простоя представляют для диагностики такую же ценность, как и любые другие.

Событийно обусловленные измерения (Event-based measurement)

Измерение процессом некоторого явления, сопровождающееся записью времени каждого изменения в состоянии системы. Противоположность *опросу (polling)*.

Соглашение об уровне обслуживания (Service level agreement – SLA)

Соглашение между поставщиком и потребителем информации, которое определяет ожидаемую производительность приложения и уровни доступности.

Соединение (Connection)

См. *Соединение Oracle*.

Соединение Oracle (Oracle connection)

Из руководства «Oracle Database Concepts»:

Соединение – это путь взаимодействия между процессом пользователя и экземпляром Oracle. Путь взаимодействия устанавливается с помощью доступных механизмов межпроцессного взаимодействия (если пользовательский процесс и сервер Oracle работают на одном компьютере) или средствами сетевого ПО (если приложение и сервер Oracle работают на разных компьютерах и общаются по сети).

Сопутствующая выгода (Collateral benefit)

Незапланированный положительный побочный эффект некоторого действия. Выгода, полученная благодаря удачному стечению обстоятельств, от действий, направленных на что-то иное.

Сопутствующий ущерб (Collateral damage)

Незапланированный отрицательный побочный эффект некоторого действия.

Спецификация (Specification)

Формальное письменное изложение желаемого результата проекта.

Теория массового обслуживания (Queueing theory)

Раздел математики, позволяющий прогнозировать различные аспекты производительности, такие как время отклика и задержка в очереди.

Технологический стек (Technology stack)

Модель, представляющая компоненты системы, такие как оборудование, операционную систему, ядро базы данных, прикладное программное обеспечение, бизнес-правила и пользователей, в виде отдельных уровней многоуровневой архитектуры.

Устойчивая система массового обслуживания (Stable queueing system)

Система массового обслуживания, коэффициент использования которой в пересчете на один канал находится в диапазоне $0 \leq \rho < 1$. В устойчивой системе массового обслуживания при рассмотрении длительного периода времени количество завершенных исполнений равно количеству поступивших запросов.

Учет микросостояний (Microstate accounting)

Так в Sun Microsystems называли механизм, с помощью которого операционная система измеряет потребление ресурса центрального процессора. При этом применяются событийно обусловленные измерения, а не опрос. Это позволяет заметно уменьшить *ошибку квантования*.

Физический ввод/вывод, физическая запись (Physical I/O (PIO), Oracle physical I/O, Oracle physical write)

Операция, в которой ядро Oracle обращается к одному или нескольким блокам базы данных посредством системного вызова чтения. В большинстве случаев вызов PIO порождается вызовом LIO, но не все вызовы PIO обслуживаются в кэше буферов Oracle. Имейте в ви-

ду, что вызов PIO не всегда бывает действительно «физическим», т. к. данные могут быть получены из кэша операционной системы, дискового массива или самого диска.

Фиксированное представление (Fixed view)

Псевдотаблица Oracle, имя которой начинается с префикса V\$ или GV\$, предоставляющая посредством SQL доступ к данным экземпляра, хранимым в разделяемой памяти. Другое название – *динамическое представление производительности (dynamic performance view)*.

Фрагмент кода (Code path)

Последовательность инструкций, которая должна быть выполнена для получения некоторого результата. *Сокращение фрагмента кода* – это процесс повышения производительности путем удаления ряда инструкций без снижения функциональности программы.

Функция плотности вероятности (Probability density function, pdf)

Вероятность того, что случайная переменная X примет определенное значение x , обозначаемая как $f(x) = P(X = x)$. Например, функция плотности вероятности для случайной переменной, описывающей результат подбрасывания симметричной монеты, имеет такой вид:

$$f(x) = \begin{cases} 0.5, & \text{if } x \text{ is heads;} \\ 0.5, & \text{if } x \text{ is tails.} \end{cases}$$

Циклическое обслуживание (Round robin -RR)

Дисциплина массового обслуживания, в которой процессы выбираются один за другим, так что все элементы множества получают возможность исполнения до повторного исполнения какого-либо элемента множества [Comer (1984) 56].

Частота (тактовая частота) (Frequency (clock frequency))

Количество тактов, отсчитываемое дискретными часами в единицу времени. Тактовая частота – величина, обратная разрешению таймера.

Экономический эффект (Net payoff)

Разность приведенной стоимости прибыли от проекта и приведенной стоимости затрат на его выполнение.

Экспоненциальное распределение (Exponential distribution)

Распределение, для которого функция плотности вероятности имеет вид:

$$f(x) = \frac{1}{\theta} e^{-x/\theta}, \quad 0 \leq x < \infty,$$

где μ – среднее значение распределения. Экспоненциальное распределение имеет большое значение в теории массового обслуживания в силу того, что реальные значения интервалов между запросами и времени обслуживания часто распределены именно по этому закону. (Это равносильно утверждению о том, что процесс поступления запросов и процесс обслуживания часто подчиняются распределению Пуассона.)

Эрланг, Агнер Краруп (Erlang, Agner Krarup) (1878–1929)

Датский математик, который первым начал изучать проблемы телефонных сетей. Известен как основоположник теории массового обслуживания.

Эффект влияния измерителя (Measurement intrusion effect)

Систематическая погрешность, обусловленная тем, что длительности выполнения подпрограммы с измерительными средствами и без них неодинаковы.

В

Греческий алфавит

Здесь вашему вниманию предложен перечень букв греческого алфавита с их эквивалентами на русском и английском языках. Английское произношение приводится в неочевидных случаях (например, греческая буква «альфа» произносится, как «а» в слове «father», а греческая буква «эта», – как «е» в слове «hey»).

Греческая буква		Греческое название	Английский эквивалент	Английское произношение
Α	α	альфа (alpha)	a	«father»
Β	β	бета (beta)	b	
Γ	γ	гамма (gamma)	g	
Δ	δ	дельта (delta)	d	
Ε	ε	эпсилон (epsilon)	e	«end»
Ζ	ζ	дзета (zêta)	z	
Η	η	эта (êta)	ê	«hey»
Θ	θ	тета (thêta)	th	
Ι	ι	иота (iota)	i	«it»
Κ	κ	каппа (kappa)	k	
Λ	λ	лямбда (lambda)	l	«box»
Μ	μ	мю (mu)	m	
Ν	ν	ню (nu)	n	
Ξ	ξ	кси (xi)	ks	
Ο	ο	омикрон (omikron)	o	«off»
Π	π	пи (pi)	p	

Греческая буква		Греческое название	Английский эквивалент	Английское произношение
Ρ	ρ	ро (rho)	r	
Σ	σ, ς	сигма (sigma)	s	«say»
Τ	τ	тау (tau)	t	
Υ	υ	ипсилон (upsilon)	u	«put»
Φ	φ	фи (phi)	f	
Χ	χ	хи (chi)	ch	«Bach»
Ψ	ψ	пси (psi)	ps	
Ω	ω	омега (omega)	ô	«grow»

Источник: <http://www.ibiblio.org/koine/greek/lessons/alphabet.html>.

С

Оптимизация коэффициента попаданий в кэш буферов базы данных

После того как я в 1989 г. пришел в Oracle, мои наставники вскоре научили меня, что коэффициент попаданий в кэш буферов базы данных информативен только в том смысле, что его по-настоящему большое значение обычно свидетельствует о каких-то неполадках [Millsap (2001b)]. За несколько лет, прошедших после моего первого знакомства с этой темой, разгорелась настоящая битва между защитниками коэффициента попаданий в кэш буферов базы данных, считающими его основным показателем производительности, и их противниками, полагающими, что коэффициент слишком ненадежен для того, чтобы использовать его для подобных характеристик. На самом деле это была не совсем битва. Существует огромное количество доказательств тому, что коэффициенты попаданий ненадежны; обманчивость аналогичных относительных величин, используемых в других областях, описана неоднократно (например, [Jain (1991)] и [Goldratt (1992)]).

Одним из неоспоримых доказательств (и самым забавным) того, что коэффициенты попаданий не заслуживают доверия, является PL/SQL-процедура `choose_a_hit_ratio`, написанная Коннором МакДональдом. Процедура Коннора позволяет увеличить коэффициент попаданий в кэш буферов базы данных до любого значения в диапазоне от его текущего значения до 99,9999999%. Как этого достичь? Добавив в систему ненужную рабочую нагрузку. Именно так. Надо задать желаемое значение коэффициента попаданий в кэш буферов, а `choose_a_hit_ratio` добавляет в систему соответствующую излишнюю нагрузку. Это бесспорно доказывает, что высокий коэффициент попаданий в кэш буферов базы данных не гарантирует эффективности системы. В своей статье на сайте <http://www.oracledba.co.uk> Коннор благодарит Джонатана Льюиса (Jonathan Lewis) за некоторые позаимствованные у него прие-

мы. А я в свою очередь хотел бы поблагодарить Коннора за то, что он разрешил мне использовать его работу в этой книге.

Оригинал PL/SQL-процедуры Коннора доступен по адресу <http://www.oracledba.co.uk>. В примере C.1 та же идея выражена на Perl, что позволяет мне сделать чуть больше, например, выводить приглашения на ввод и печатать хронометрические данные при формировании ЛЮ. Приведенный ниже код можно скачать (как и другие примеры из книги) с сайта O'Reilly <http://www.oreilly.com/catalog/optoracle/>.

Пример C.1. Программа на Perl, позволяющая увеличить коэффициент попаданий в кэш буферов базы данных до практически любого значения

```
#!/usr/bin/perl

# $Header: /home/cvs/cvm-book1/set_hit_ratio/set-bchr.pl,v 1.3 2003/05/08
# 06:37:50 cvm Exp $
# Cary Millsap (cary.millsap@hotsos.com)
# based upon the innovative work of Connor McDonald and Jonathan Lewis
# Copyright (c) 2003 by Hotsos Enterprises, Ltd. All rights reserved.

use strict;
use warnings;
use Getopt::Long;
use Time::HiRes qw(gettimeofday);
use DBI;

# получаем параметры командной строки
my %opt = (
    service      => "",
    username     => "/",
    password     => "",
    debug        => 0,
);

GetOptions(
    "service=s" => \$opt{service},
    "username=s" => \$opt{username},
    "password=s" => \$opt{password},
    "debug"     => \$opt{debug},
);

sub fnum($;$) {
    # возвращаем строковое представление числового
    # значения в формате %.${precision}f с указанными
    # разделителями
    my ($text, $precision, $separator) = @_;
    $precision = 0 unless defined $precision;
    $separator = "," unless defined $separator;
    $text = reverse sprintf "%.${precision}f", $text;
    $text =~ s/(\d\d\d)(?=\d)(?! \d*\. )/$1$separator/g;
    return scalar reverse $text;
}

sub stats($) {
```

```

# получаем статистики LIO и PIO из любого
# указанного дескриптора БД
my ($dbh) = @_;
my $sth = $dbh->prepare(<<'END OF SQL', {ora_check_sql => 0});
select name, value from v$sysstat
where name in ('physical reads', 'db block gets', 'consistent gets')
END OF SQL
$sth->execute();
my $r = $sth->fetchall_hashref("NAME");
my $pio = $r->{'physical reads'}->{VALUE};
my $lio = $r->{'consistent gets'}->{VALUE} + $r->{'db block gets'}->{VALUE};
if ($opt{debug}) {
    print "key='$_', val=$r->{$_}->{VALUE}\n" for (keys %$r);
    print "pio=$pio, lio=$lio\n";
}
return ($lio, $pio);
}

sub status($$$) {
    # выводим раздел статусов
    my ($description, $lio, $pio) = @_;
    print "$description\n";
    printf "%15s LIO calls\n", fnum($lio);
    printf "%15s PIO calls\n", fnum($pio);
    printf "%15.9f buffer cache hit ratio\n", ($lio - $pio) / $lio;
    print "\n";
}

# получаем из командной строки искомое значение коэффициента попаданий
my $usage = "Usage: $0 [options] target\n\t";
my $target = shift or die $usage;
my $max_target = 0.999_999_999;
unless ($target =~ /\d*\.\d+/ and 0 <= $target and $target <= $max_target) {
    die "target must be a number between 0 and $max_target\n";
}

# подключаемся к Oracle
my %attr = (RaiseError => 0, PrintError => 0, AutoCommit => 0);
my $dbh = DBI->connect(
    "dbi:Oracle:$opt{service}", $opt{username}, $opt{password}, \%attr
);
END {
    # выполняется при выходе из программы
    $dbh->disconnect if defined $dbh;
}

# вычисляем и выводим исходные статистики
my ($lio0, $pio0) = stats $dbh;
status("Current state", $lio0, $pio0);

# вычисляем и выводим объем дополнительной нагрузки,
# необходимый для «улучшения» коэффициента попаданий до

```

```

# запрошенного значения
my $waste;
if ($target < ($lio0 - $pio0)/$lio0) {
    die "Your database buffer cache hit ratio already exceeds $target.\n";
} elsif ($target > $max_target) {
    die "Setting your hit ratio to $target will take too long.\n";
} else {
    # формула предоставлена Коннором МакДональдом
    $waste = sprintf "%.0f", $pio0/(1 - $target) - $lio0;
}
my ($lio1, $pio1) = ($lio0 + $waste, $pio0);
status("Increasing LIO count by ".fnum($waste)." will yield", $lio1, $pio1);

# запрос подтверждения изменения коэффициента
print <<"EOF";
*****
                        WARNING
Responding affirmatively to the following prompt will create the
following effects:
1) It will degrade the performance of your database while it runs.
2) It might run a very long time.
3) It will "improve" your system's buffer cache hit ratio.
4) It will prove that a high database buffer cache hit ratio is
   an unreliable indicator of Oracle system performance.
                        ВНИМАНИЕ!
Подтверждение данного изменения приведет к таким последствиям:
1) Производительность вашей базы данных на время такого изменения ухудшится.
2) Изменение может выполняться очень долго.
3) Коэффициент попаданий в кэш буферов вашей системы действительно
   «улучшится».
4) Будет доказано, что высокий коэффициент попаданий в кэш буферов базы
   данных не является достоверным показателем высокой производительности
   системы Oracle.
*****

EOF
print qq(Enter 'y' to "improve" your hit ratio to $target: );
my $response = <>;
exit unless $response =~ /^[Yy]/;
print "\n";

# создаем таблицу DUMMY
my $sth;
$sth = $dbh->prepare(<<'END OF SQL', {ora_check_sql => 0});
drop table dummy
END OF SQL
$sth->execute if $sth; # игнорировать ошибки
$sth = $dbh->prepare(<<'END OF SQL', {ora_check_sql => 0});
create table dummy (n primary key) organization index as
select rownum n from all_objects where rownum <= 200
END OF SQL
$sth->execute;

```

```

# отключаем connect-by возможности версии 9i для того, чтобы обеспечить
# множество вызовов LIO
# идея предложена Коннором Мак-Дональдом
$sth = $dbh->prepare(<<'END OF SQL', {ora_check_sql => 0});
alter session set _old_connect_by_enabled = true;
END OF SQL
$sth->execute if $sth; # ignore errors

# выполняем необходимое количество вызовов LIO
# авторство следующего запроса принадлежит Джонатану Льюису
$sth = $dbh->prepare(<<'END OF SQL', {ora_check_sql => 0});
select count(*)
from (select n from dummy connect by n > prior n start with n = 1)
where rownum < ?
END OF SQL
my $e0 = gettimeofday;
$sth->execute($waste);
my $e1 = gettimeofday;
my $e = $e1 - $e0;
$sth->finish;
printf "Performed %s LIO calls in %.6f seconds (%s LIO/sec)\n\n",
      fnum($waste), $e, fnum($waste/$e);

# вычисляем и выводим итоговые статистики
my ($lio2, $pio2) = stats($dbh);
status("Final state", $lio2, $pio2);

exit;

__END__

=head1 NAME

set-bchr - установить коэффициент попаданий в кэш буферов вашей базы данных
в более высокое значение

=head1 SYNOPSIS

set-bchr
  [--service=I<h>]
  [--username=I<u>]
  [--password=I<p>]
  [--debug=I<d>]
  I<target>

=head1 DESCRIPTION

В <set-bchr> вычисляет текущий коэффициент попаданий в кэш буферов
(используя общепринятые формулы), определяет, сколько дополнительной
нагрузки необходимо добавить в систему с тем, чтобы довести значение
коэффициента до I<target>, затем обеспечивает реальное выполнение
дополнительной работы, которая приведет к повышению коэффициента до
нужного значения I<target>. Значение I<target> должно быть десятичным
числом в интервале от 0 до 0.999999999.

```


Использование `B<set-bchr>` может увеличить значение коэффициента попаданий в кэш буферов базы данных вашей системы, но оно приведет к УХУДШЕНИЮ ПРОИЗВОДИТЕЛЬНОСТИ СИСТЕМЫ НА ВРЕМЯ СВОЕЙ РАБОТЫ. Цель `B<set-bchr>` в том, чтобы шуточно, но недвусмысленно показать, что коэффициент попаданий в кэш буферов базы данных не является надежной характеристикой производительности системы. Если вы планируете использовать программу с тем, чтобы обманом заставить клиентов или руководителей поверить в то, что вы работаете лучше, чем это есть на самом деле, что ж – попробуйте.

```
=head2 Options
```

```
=over 4
```

```
=item B<--service=>I<h>
```

Имя службы Oracle, к которой будет подключаться `B<vprof>`. Значение по умолчанию – "" (пустая строка), в этом случае `B<vprof>` будет подключаться, используя, например, псевдоним Oracle TNS по умолчанию.

```
=item B<--username=>I<u>
```

Имя схемы Oracle, к которой будет подключаться `B<vprof>`. Значение по умолчанию – "/".

```
=item B<--password=>I<p>
```

Пароль Oracle, который `B<vprof>` будет использовать при подключении. Значение по умолчанию – "" (пустая строка).

```
=item B<--debug=>I<d>
```

При установке в 1 `B<vprof>` выводит внутренние структуры данных в дополнение к своему обычному выводу. Значение по умолчанию – 0.

```
=back
```

```
=head1 EXAMPLES
```

Использование `B<set-bchr>` будет аналогично приведенному далее примеру – я «повышаю» коэффициент попаданий в кэш буферов до приблизительно 0.92:

```
$ set-bchr --username=system --password=manager .92
```

```
Current state
```

```
    37,257,059 LI0 calls
```

```
    3,001,414 PIO calls
```

```
0.919440394 buffer cache hit ratio
```

```
Increasing LI0 count by 260,616 will yield
```

```
    37,517,675 LI0 calls
```

```
    3,001,414 PIO calls
```

```
0.920000000 buffer cache hit ratio
```

```
*****
```

ВНИМАНИЕ!

Подтверждение данного изменения приведет к таким последствиям:

- 1) Производительность вашей базы данных на время такого изменения ухудшится.
- 2) Изменение может выполняться очень долго.

- 3) Коэффициент попаданий в кэш буферов вашей системы действительно «улучшится».
- 4) Будет доказано, что высокий коэффициент попаданий в кэш буферов базы данных не является достоверным показателем высокой производительности системы Oracle.

Enter 'y' to "improve" your hit ratio to .92: y

Performed 260,616 LIO calls in 46.592340 seconds (5,594 LIO/sec)

Final state

37,259,288 LIO calls

3,001,414 PIO calls

0.919445213 buffer cache hit ratio

=head1 AUTHOR

Кэри Миллсап (cary.millsap@hotsos.com), значительная часть информации извлечена из оригинальной работы Коннора МакДональда.

=head1 BUGS

`B<set-bchr>` не обязательно увеличивает коэффициент попаданий в кэш буферов базы данных в точности до значения `I<target>`, но подходит к нему очень близко.

`B<set-bchr>` вычисляет коэффициент попаданий в кэш буферов базы данных Oracle, используя традиционную формулу $R = (LIO - PIO) / LIO$, где LIO – это сумма статистик согласованного чтения и чтения блоков базы данных, а PIO – это значение статистики физического чтения. Подобное вычисление LIO некорректно. Подробности можно найти в [Lewis (2003)].

=head1 COPYRIGHT

Copyright (c) 2003 by Hotsos Enterprises, Ltd. All rights reserved.

D

Формулы теории массового обслуживания М/М/т

В табл. D.1 собраны для удобства формулы теории массового обслуживания М/М/т, рассмотренной в главе 9.

Таблица D.1. Формулы теории массового обслуживания М/М/т

Определение	Формула/обозначение
Среднее количество поступающих запросов	A
Среднее количество выполненных запросов	C
Период измерения	T
Среднее время занятости	B
Количество параллельных каналов обслуживания	m
Средняя интенсивность поступления запросов	$\lambda = \frac{A}{T}$
Средний интервал поступления запросов	$\tau = \frac{1}{\lambda}$
Средняя пропускная способность системы	$X = \frac{C}{T}$
Среднее время обслуживания	$S = \frac{B}{C}$
Средняя скорость обслуживания	$\mu = \frac{1}{S}$
Средний коэффициент использования системы	$U = \frac{B}{T}$
Средний коэффициент использования или загрузки одного сервера (если их всего m)	$\rho = \frac{U}{m}$

Определение	Формула/обозначение
Вероятность постановки поступающего запроса в очередь (Эрланг)	$C(m, \rho) = P(\geq m \text{ jobs}) =$ $= \frac{\frac{(m\rho)^m}{m!}}{(1-\rho) \sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \frac{(m\rho)^m}{m!}}$
Средняя задержка в очереди	$W = \frac{C(m, \rho)}{m\mu(1-\rho)}$
Среднее время отклика	$R = S + W$
Интегральная функция распределения времени отклика	$P(R \leq r) = F(r) = \frac{m(1-\rho) - W_q(0)}{m(1-\rho) - 1} (1 - e^{-\mu r}) -$ $- \frac{1 - W_q(0)}{m(1-\rho) - 1} (1 - e^{-(m\mu - \lambda)r})$ <p>где $W_q(0) = 1 - \frac{(m\rho)^m p_0}{m!(1-\rho)}$</p> $\text{и } p_0 = \left(\sum_{n=0}^{m-1} \frac{(m\rho)^n}{n!} + \frac{(m\rho)^m}{m!(1-\rho)} \right)^{-1}, \rho < 1$



Ссылки

[Adams (1999)]

Adams, S. 1999. *Oracle8i Internal Services for Waits, Latches, Locks, and Memory*. Sebastopol CA: O'Reilly & Associates.

[Adams (2003)]

Adams, S. 2003. *Oracle Internals and Advanced Performance Tuning*. Copenhagen: Course presented at Miracle Master Class 2003, 13–15 Jan. 2003.

[Allen (1994)]

Allen, A. O. 1994. *Computer Performance Analysis with Mathematica*. Cambridge MA: AP Professional.

[Amdahl (1967)]

Amdahl, A. 1967. «Validity of the single processor approach to achieving large scale computing capabilities» in *AFIPS Conf. Proc.*, vol. 30.

[Ault and Brinson (2000)]

Ault, M. R.; Brinson, J. M. 2000. *Oracle8 DBA: Performance Tuning Exam Cram*. Scottsdale AZ: Coriolis.

[Bach (1986)]

Bach, M. J. 1986. *The Design of the UNIX Operating System*. Englewood Cliffs NJ: Prentice Hall.

[Bentley (1988)]

Bentley, J. 1988. *More Programming Pearls: Confessions of a Coder*. Reading MA: Addison-Wesley.

[Bodie, et al. (1989)]

Bodie, Z.; Kane, A.; Marcus, A. J. 1989 *Investments*. Homewood IL: Irwin.

[Bovet and Cesati (2001)]

Bovet, D. P.; Cesati, M. 2001. *Understanding the Linux Kernel*. Sebastopol CA: O'Reilly.

[Brealey and Myers (1988)]

Brealey, R. A.; Myers, S. C. 1988. *Principles of Corporate Finance* (3ed). New York: McGraw-Hill.

[Breitling (2002)]

Breitling, W. 2002. «A look under the hood of CBO: the 10053 event». <http://www.hotsos.com>: Hotsos.

[Chiesa (1996)]

Chiesa, D. P. 1996. *Unix Performance Measurement*. <http://www.transarc.ibm.com/Library/whitepapers/tg/node14.html>.

[Cockroft (1998)]

Cockroft, A. 1998. «Prying into processes and workloads», in *Unix Insider*, 1 Apr. 98. <http://www.sun.com/sun-on-net/itworld/UIR980401perf.html>: Sun.

[Comer (1984)]

Comer, D. 1984. *Operating System Design, the XINU Approach*. Englewood Cliffs NJ: Prentice-Hall.

[CRC (1991)]

Standard Mathematical Tables and Formulae (29ed). Boca Raton FL: CRC Press.

[Dowd (1993)]

Dowd, K. 1993. *High Performance Computing*. Sebastopol CA: O'Reilly.

[Engsig (2001)]

Engsig, B. 2001. «Efficient use of bind variables, cursor_sharing and related cursor parameters». <http://otn.oracle.com/deploy/performance>: Oracle Corp.

[Ensor and Stevenson (1997a)]

Ensor, D.; Stevenson, I. 1997. *Oracle Design*. Sebastopol CA: O'Reilly.

[Ensor and Stevenson (1997b)]

Ensor, D.; Stevenson, I. 1997. *Oracle8 Design Tips*. Sebastopol CA: O'Reilly.

[Erlang (1909)]

Erlang, A. K. 1909. «The Theory of Probabilities and Telephone Conversations», *Nyt Tidsskrift for Matematik B*, vol 20, 1909.

[Erlang (1917)]

Erlang, A. K. 1917. «On the rational determination of the number of circuits», in *The Life and Works of A. K. Erlang*, 1948. Brockmeyer, E.;

Halstrom, H.; Jensen, A. (eds.). *Trans. Danish Academy of Tech. Sci.*
См. также <http://plus.maths.org/issue2/erlang/#allref>, где можно познакомиться с биографией Агнера Эрланга.

[Feuerstein (1998)]

Feuerstein, S.; Beresiewicz, J.; Dawes, C. 1998. *Oracle PL/SQL Built-ins Pocket Reference*. Sebastopol CA: O'Reilly.

[Feynman (1999)]

Feynman, R. P. 1999. *The Pleasure of Finding Things Out*. Cambridge MA: Perseus.

[Frisch (2002)]

Frisch, А. 2002. *Essential System Administration* (3ed). Sebastopol CA: O'Reilly.

[Frisch (1998)]

Frisch, А. 1998. *Essential Windows NT System Administration*. Sebastopol CA: O'Reilly.

[Goldratt (1992)]

Goldratt, E. M. *The Goal: a Process of Ongoing Improvement* (2ed). Great Barrington MA: North River Press.

[Gray and Neuhoff (1998)]

Gray, R. M.; Neuhoff, D. L. 1998. «Quantization» in *IEEE Transactions on Information Theory*, Vol. 44, No. 6, October 1998.

[Gross and Harris (1998)]

Gross, D.; Harris, C. M. 1998. *Fundamentals of Queueing Theory* (3ed). New York: Wiley.

[Gunther (1998)]

Gunther, N. J. 1998. *The Practical Performance Analyst: Performance-by-Design Techniques for Distributed Systems*. New York: McGraw-Hill.

[Gurry and Corrigan (1996)]

Gurry, M.; Corrigan, P. 1996. *Oracle Performance Tuning* (2ed). Sebastopol CA: O'Reilly.

[Hailey (2002)]

Hailey, K. 2002. «Direct Oracle SGA Memory Access». <http://oraperf.sourceforge.net>: SourceForge.

[Harrison (2000)]

Harrison, G. 2000. *Oracle SQL High-Performance Tuning* (2ed). Upper Saddle River NJ: Prentice Hall PTR.

[Hawking (1988)]

Hawking, S. W. 1988. *A Brief History of Time*. New York: Bantam.¹

[Hogg and Tanis (1977)]

Hogg, R. V.; Tanis, E. A. 1983. *Probability and Statistical Inference*. New York: Macmillan.

[Holt (2000a)]

Holt, J. 2000. «Predicting multi-block read call sizes». <http://www.hotsos.com>: Hotsos.

[Holt (2000b)]

Holt, J. 2000. «Why are Oracle's read events 'named backwards'?». <http://www.hotsos.com>: Hotsos.

[Holt and Millsap (2000)]

Holt, J.; Millsap, C. 2000. «Scaling applications to massive concurrent user counts». <http://www.hotsos.com>: Hotsos.

[Holt et al. (2003)]

Holt, J.; Millsap, C.; Minutella, R.; Goodman, G. 2003. *Hotsos Clinic OP101: Optimizing Oracle SQL*. <http://www.hotsos.com>: Hotsos.

[Jagerman (1974)]

Jagerman, D. L. 1974. «Some Properties of the Erlang Loss Function» in *Bell Sys. Tech. J.* 55: 525.

[Jain (1991)]

Jain, R. 1991. *The Art of Computer Systems Performance Analysis*. New York: Wiley.

[Kachigan (1986)]

Kachigan, S. K. 1986. *Statistical Analysis: an Interdisciplinary Introduction to Univariate and Multivariate Methods*. New York: Radius Press.

[Kennedy and Everest (1994)]

Kennedy, J.; Everest, A. 1994. *Effective Interviewing! An Advanced Seminar for Achieving Superior Results*. San Rafael CA: Management Team Consultants, Inc.

[Kleinrock (1975)]

Kleinrock, L. 1975. *Queueing Systems, Vol. 1: Theory*. New York: Wiley.

¹ Хокинг С. «Краткая история времени: от большого взрыва до черных дыр», СПб.: Амфора, 2005 г.

[Knuth (1971)]

Knuth, D. E. 1971. «Empirical Study of FORTRAN Programs» in *Software—Practice and Experience*, April/June 1971, Vol. 1, No. 2, pp. 105–133.

[Knuth (1981)]

Knuth, D. E. 1981. *The Art of Computer Programming (2ed)*, Vol. 2 *Seminumerical Algorithms*. Reading MA: Addison-Wesley.¹

[Kolk (1996)]

Kolk, A. 1996. *Description of Oracle7 Wait Events and Enqueues*. Redwood Shores CA: Oracle Corp. internal document.

[Kolk and Yamaguchi (1999)]

Kolk, A.; Yamaguchi, S. 1999. *Yet Another Performance Profiling Method (or YAPP-Method)*. <http://www.oraperf.com>: OraPerf.

[Kolk (2001)]

Kolk, A. 2001. *New Oracle9i Timing Features*. <http://www.oraperf.com>: Precise Software Solutions.

[Kyte (2001)]

Kyte, T. 2001. *Expert One-on-One Oracle*. Birmingham UK: Wrox.

[Kyte (2002)]

Kyte, T. 2002. «Reducing LIOs» in *AskTom*. http://asktom.oracle.com/pls/ask/f?p=4950:8:1683948::NO::F4950_P8_DISPLAYID,F4950_P8_CRITERIA:6749454952894: Oracle.

[Laplace (1812)]

de Laplace, P. S. 1812. *Théorie Analytique des Probabilités*.

[Lawson (2003)]

Lawson, C. 2003. *The Art and Science of Oracle Performance Tuning*. Birmingham UK: Curlingstone.

[Lewis & Papadimitriou (1981)]

Lewis, H. R.; Papadimitriou, C. H. 1981. *Elements of the Theory of Computation*. Englewood Cliffs NJ: Prentice Hall.

[Lewis (2001a)]

Lewis, J. 2001. «Folk Lore and Fairy Tales», <http://www.jlcomp.demon.co.uk/myths.html>: JL Computer Consultancy.

[Lewis (2001b)]

Lewis, J. 2001. *Practical Oracle8i: Building Efficient Databases*. Upper Saddle River NJ: Addison-Wesley.

¹ Кнут Д. «Искусство программирования, том 2. Получисленные алгоритмы», 3-е издание, М.: Вильямс, 2000 г.

[Lewis (2002)]

Lewis, J. 2002. *Optimising Oracle*. Course presented at Miracle Master Class 2003, 23–25 Jan. 2002.

[Lewis (2003)]

Lewis, J. 2003. «The database gets better but the metrics look worse», in *IOUG Live 2003 Proceedings*. <http://www.ioug.org>.

[Lilja (2000)]

Lilja, D. J. 2000 *Measuring Computer Performance: a Practitioner's Guide*. Cambridge UK: Cambridge Press.

[Maloney et al. (1992)]

Malony, A. D.; Reed, D. A.; Wijshoff, H. A. G. 1992. «Performance Measurement Intrusion and Perturbation Analysis» in *IEEE Transactions on Parallel and Distributed Systems*, July 1992, Vol. 3, No. 4, pp. 433–450.

[McDonald (2000)]

McDonald, C. 2000. Various hints, tips, and observations, <http://www.oracledba.co.uk>.

[Millsap (1999)]

Millsap, C. V. 1999. «Performance Management: Myths & Facts», <http://www.hotsos.com>: Oracle.

[Millsap (2000a)]

Millsap, C. V. 2000. «Is RAID 5 Really a Bargain?» <http://www.hotsos.com>: Hotsos.

[Millsap (2000b)]

Millsap, C. V. 2000. «Batch Queue Management and the Magic of ‘2’», <http://www.hotsos.com>: Hotsos.

[Millsap (2001a)]

Millsap, C. V. 2001. «Scalability is a Rate of Change», <http://www.hotsos.com>: Hotsos.

[Millsap (2001b)]

Millsap, C. V. 2001. «Why a 99% + Database Buffer Cache Hit Ratio is Not Ok», <http://www.hotsos.com>: Hotsos.

[Millsap (2001c)]

Millsap, C. V. 2002. «Why You Should Focus on LIOs Instead of PIOs», <http://www.hotsos.com>: Hotsos.

[Millsap (2002)]

Millsap, C. V. 2002. «When to Use an Index», <http://www.hotsos.com>: Hotsos.

[Millsap and Holt (2002)]

Millsap, C. V.; Holt, J. L. 2002. «Useful Constants for the Oracle Performance Analyst», <http://www.hotsos.com>: Hotsos.

[Morle (1999)]

Morle, J. 1999. *Scaling Oracle8i: Building Highly Scalable OLTP System Architectures*. Upper Saddle River NJ: Addison-Wesley.

[Musumeci and Loukides (2002)]

Musumeci, G. D.; Loukides, M. 2002. *System Performance Tuning* (2ed). Sebastopol CA: O'Reilly.¹

[Nemeth et al. (2000)]

Nemeth, E.; Snyder, G.; Seebass, S.; Hein, T. R. 2000. *Unix System Administration Handbook* (3ed). Englewood Cliffs NJ: Prentice-Hall PTR.²

[Olkin et al. (1994)]

Olkin, I.; Gleser, L. J.; Derman, C. 1994. *Probability Models and Applications* (2ed). New York: Macmillan.

[Oracle (1996)]

Oracle Corp. 1996. *Oracle7 Server Tuning*. Redwood Shores CA: Oracle Corp.

[Oracle OCI (1999)]

Oracle Corp. 1999. *Oracle Call Interface Programmer's Guide*. Redwood Shores CA: Oracle Corp.

[Oracle (2002)]

Oracle Corp. 2002. *Oracle9i Database Performance Tuning Guide and Reference Release 2 (9.2)*. Redwood Shores CA: Oracle Corp.

[Pelz (2000)]

Pelz, D. 2000. *Dave Pelz's Putting Bible: the Complete Guide to Mastering the Green*. New York: Doubleday.

[Rivenes (2003)]

Rivenes, A. 2003. *Oracle 9.2 Event 10046 Segment-Level Statistics*. <http://www.appsdba.com>: AppsDBA Consulting.

[Schrag (2002)]

Schrag, R. 2002. *Interpreting Wait Events to Boost System Performance*. http://www.dbspecialists.com/presentations/wait_events.html: Database Specialists.

¹ Мусумеси Д.-П., Лукидес М. «Настройка производительности UNIX-систем», 2-е издание, СПб.: Символ-Плюс, 2003 г.

² Немет Э., Снайдер Г., Сибасс С., Хейн Т. Р. «UNIX: руководство системного администратора. Для профессионалов», СПб.: Питер, 2003 г.

[Stanford (2001)]

Stanford University. 2001. *Human Subjects Manual: a comprehensive reference guide for Stanford researchers, administrators, students, and staff involved in human subjects research*, <http://humansubjects.stanford.edu/manual>: Stanford University.

[Stevens (1992)]

Stevens, W. R. 1992. *Advanced Programming in the Unix Environment*. Reading MA: Addison-Wesley.

[Vaidyanatha et al. (2001)]

Vaidyanatha, G. K.; Deshpande, K.; Kostelac, J. A. Jr. 2001. *Oracle Performance Tuning 101*. New York: Osborne/McGraw-Hill.

[Vernon (2001)]

Vernon, M. K. 2001. *CS 547: Computer System Modeling Fundamentals*. <http://www.cs.wisc.edu/~vernon/cs547/01/assignments/s5.pdf>: University of Wisconsin Madison.

[Wall et al. (2000)]

Wall, L.; Christiansen, T.; Orwant, J. 2000. *Programming Perl*. Sebastopol CA: O'Reilly & Associates.¹

[Wolfram (1999)]

Wolfram, S. 1999. *Mathematica*. Champaign IL: Wolfram.

[Wood (2003)]

Private conversation with Graham Wood of Oracle Corporation's Server Technologies group.

¹ Уолл Л., Кристиансен Т., Орвант Д. «Программирование на Perl», 3-е издание, СПб.: Символ-Плюс, 2002 г.

Алфавитный указатель

A

ALTER SESSION SET EVENTS,
команда, 168

B

BACKGROUND_DUMP_DEST,
параметр, 159
MTS и, 163
каталог, 222

C

CDF (Cumulative Distribution Function),
интегральная функция распределе-
ния, 284, 422
CPU service, компонент, 368

D

DBMS_SUPPORT, пакет, 158
DBMS_SYSTEM, пакет, 158

E

E-Business Suite, новые версии, 81

G

getrusage, операция, 207
gettimeofday, операция, 206

H

Hotsos Profiler, программа, 17

L

Linux, модель M/M/m и, 323
LIO (Logical Input Output), логический
ввод/вывод, 423

M

M/M/m, модель массового
обслуживания, 295, 424
пример с решением, 321
анализ чувствительности, 331
вычисление количества
процессоров, 324
подбор параметра в Microsoft
Excel, 331
подбор параметров, 328
польза оптимистической модели,
327
проверка на применимость, 323
поведение, 307
излом кривой производительно-
сти, 310
колебания времени отклика, 315
масштабируемость многоканаль-
ной системы, 307
чувствительность к параметрам,
317
системы массового обслуживания
M/M/m, 296
системы массового обслуживания
не-M/M/m, 297
экспоненциальное распределение,
297
проверка на соответствие, 299
программа для проверки на, 301
распределение Пуассона и, 298

O

OCI (Oracle Call Interface), интерфейс
уровня вызовов, 383
Oracle
MetaLink, бюллетени технической
поддержки и сообщения об
ошибках, 366

Oracle

- время ожидания, составляющие, 292
- документация по базам данных, 366
- интерфейс ожидания, 15
 - данные фиксированных представлений Oracle и, 267
- источники данных о времени выполнения, 88
- псевдособытия, 367
 - unaccounted for, 369
- ядро
 - измерение времени, 184
 - неучтенное время, 193
 - влияние измерителя, 194
 - время, в течение которого процесс не выполняется, 216
 - двойной учет занятости процессора, 198
 - код ядра Oracle без измерительных средств, 219
 - ошибки квантования, 200
 - получение диагностических данных, 240
- oraus.msg, файл, 152

P

- PDF (Probability Density Function), функция плотности вероятности, 289, 432
- Perl DBI ora_check_sql, атрибут, 384
- PX (Parallel eXecution), параллельное выполнение, 162
 - подчиненный процесс, 162

S

- sctrace, программа, 189
- SMP (Symmetrical MultiProcessing), симметричная многопроцессорность, 279
- Sparky, программа, 17
- SQL (Structured Query Language), язык структурированных запросов
 - WHERE, инструкция, избавление от литералов в, 384
 - временные статистики трассировки, 122
 - вызовы разбора и текстовые строки, 385

- команды, ранжирование по эффективности, 236
- оптимизация, 381
- рекурсивный, 126, 429
- SQL-запросы к фиксированным представлениям Oracle, 244
- исследование всех событий ожидания системы, 259
- поиск неэффективного SQL, 246
- поиск определения фиксированного представления, 245
- поиск причин зависания сеанса, 249
- поиск причин зависания системы, 251
- проекты повышения производительности, 87
- создание приблизительного профиля ресурсов сеанса, 252
- средства тестирования, 244
- START_TRACE_IN_SESSION, пакет, 158
- Statspack, утилита, события простоя, отсутствующие в отчетах, 373
- strace, программа, 189

T

- tkprof, утилита
 - анализ команд SQL, 410
 - ошибочные результаты, 120
- TRACEFILE_IDENTIFIER, атрибут, 159
- truss, программа, 189
- tusc, программа, 189

U

- USER_DUMP_DEST, параметр, 159
 - MTS и, 163
 - каталог, 222

V

- V\$-данные
 - см. также данные фиксированных представлений Oracle, 225
- V\$EVENT_NAME, фиксированное представление, 267
- V\$FIXED_VIEW_DEFINITION, фиксированное представление, 245
- V\$SESS_IO, фиксированное представление, 237

V\$SESSION_EVENT, фиксированное представление, 240, 267
V\$SESSION_WAIT, фиксированное представление, 241, 244, 267
 первая документация для, 242
V\$SESSTAT, фиксированное представление, 238
V\$SQL, фиксированное представление, 235
V\$SYSSTAT, фиксированное представление, 238
V\$SYSTEM_EVENT, фиксированное представление, 239, 267

W

Windows, orausr.msg и, 152

A

анализ времени отклика, 42
 диаграммы последовательности, 42
 профиль ресурсов, 44
архитектура, многозвенные приложения, 385

B

ввод/вывод, измерение, 237
включение расширенной трассировки SQL, 152
влияние измерителя, измерение времени ядром Oracle, 194
временная область, погрешность времени включения трассировки, 168
временные метки, файлы данных трассировки SQL, 111
временные статистики
 данные трассировки SQL, 122
время отклика, 419
 вклад других событий, 376
 излом кривой производительности, 422
информация, извлечение из необработанных данных трассировки SQL, 135
колебания в модели массового обслуживания M/M/m, 315
концентрация на, 39
метод настройки производительности на основе, 49, 335
 возражения, 54

 оценка эффективности, 62
определение цели
 длительное обслуживание процессором, 397
 обманчивые общесистемные данные, 391
составляющие, 365, 367
 просмотр в порядке убывания, 41
 псевдособытия Oracle, 367
 события простоя, 372
средства анализа, 42
 диаграммы последовательности, 42
 профиль ресурсов, 44
теория массового обслуживания и, 278
учет, 122
эволюция модели, 131
выбор
 диагностические данные, 77
 метод улучшения производительности на основе времени отклика, 346
выбор пользовательских операций, 64
выбор пути повышения производительности, 91
 экономически оптимального пути, 93
вызовы базы данных, 420
 время внутри, 125
 пропуск, 220
 время между, 125
 пропуск, 220
длительность, отсутствие данных о фиксированных представлениях Oracle, 231
исключение лишних при работе с профилем ресурсов
 запросы и услуги в технологическом стеке, 349
опережающее атрибутирование для событий
 внутри вызовов, 140
 между вызовами, 141
отношения родитель-потомок, 126
продолжительность, 123
рекурсивные статистики, 129
файлы данных трассировки SQL, 114
 рекурсивный SQL, 126
 учет времени отклика, 122

вытеснение, 187

выявление ошибки квантования, 211

Г

греческий алфавит, 434

Д

данные диагностические

см. диагностические данные

данные расширенной трассировки SQL

для ядра версии не старше Oracle8i,
отсчет времени, 136

извлечение информации о времени
отклика из, 135

интерпретация, 107

начиная с ядра версии Oracle9i,
отсчет времени, 138

неполные данные рекурсивного SQL,
180

ошибки сбора, устранение, 167

погрешность времени включения
трассировки, 168

пропуск времени при отключении
трассировки, 178

сбор, 148

понимание архитектуры
приложения, 148

файлы

временные метки, 111

время между вызовами базы
данных, 125

вызовы базы данных, 114

запись в, 221

знакомство, 107

идентификация курсора, 112

идентификация приложения,
112

идентификация сеанса, 111
имена, 159

конфиденциальность и, 148

маркеры конца транзакции, 120

номера курсоров, 111

операции над источником строк,
118

определения элементов, 110

переменные связывания, 117

поиск, 159

продолжительность вызова базы
данных, 123

рекурсивные отношения между
вызовами базы данных, 126
события ожидания, 116
учет времени отклика, 122

данные фиксированных представлений
Oracle, 224

SQL-запросы, 244

исследование всех событий
ожидания системы, 259

поиск неэффективного SQL, 246

поиск определения фиксирован-
ного представления, 245

поиск причин зависания сеанса,
249

поиск причин зависания
системы, 251

создание приблизительного
профиля ресурсов сеанса, 252

средства тестирования, 244

интерфейс ожидания Oracle, 267

недостатки, 225

влияние измерителя при опросе,
226

другие ошибки, 231

избыток источников данных, 225

отсутствие данных о длительно-
сти вызовов БД, 231

отсутствие деталей, 226

отсутствие согласованности
чтения, 232

сложность выбора временной
области, 230

сложность выбора операций, 230

чувствительность к переполне-
нию, 231

детерминизм, 80

диагноз, лечение согласно, 365

составляющие времени отклика,
367

диагностические данные

см. также данные расширенной
трассировки SQL, 77, 107

анализ, 94

выбор, 77

сбор, 77

агрегирование, 86

для долго выполняющихся поль-
зовательских операций, 86

область данных, 81

ошибки определения области
данных, 81

сбор данных о времени выполнения операций в Oracle, 88
диаграммы последовательности, 42
теория массового обслуживания и, 276

З

задержка, 421
закон Амдала, 40
защелки, конкуренция, 46

И

идентификация
курсора, файлы данных трассировки SQL, 112
приложения, файлы данных трассировки SQL, 112
сеанса, файлы данных трассировки SQL, 111
имена файлов расширенной трассировки SQL, 159
интегральная функция распределения времени отклика *см. CDF*
интенсивность трафика, 279
интерпретация данных расширенной трассировки SQL
см. также данные расширенной трассировки SQL, 107
интерфейс ожидания
данные фиксированных представлений Oracle и, 267
исключение лишних при работе с профилем ресурсов, 348
использование процессора
сложности измерения времени, 206
getrusage, операция, 207
gettimeofday, операция, 206
исследование всех событий ожидания системы, 259
бесконечный ресурс ожидания, 264
важность выбора корректной области, 267
проблема нормирования, 261
исходный код, трассировка собственного, 153
чужого, 155

К

квантование, ошибки, 200, 427
выявление, 211
диапазон значений, 213
определение, 203
разрешающая способность измерительной системы, 200
сложности измерения процессорного времени, 206
коммерческий подход к решению проблемы времени отклика, 345
конкуренция между процессами, устранение при работе с профилем ресурсов, 354
решение проблемы с задержками, 355
решение проблемы с конкурирующими нагрузками, 357
концентрация на времени отклика системы, 39
концентрация на пользовательских операциях, 37
коэффициент использования системы, 279
кэш буферов, 378
коэффициент попаданий, 47, 423
ненадежность, 436
оптимизация, 436

Л

логический ввод/вывод, 47

М

маркеры конца транзакции
файлы данных трассировки SQL, 120
масштабируемость, 424
Метод R, настройка производительности, 49
методы настройки производительности, 34
детерминизм, 80
метод проб и ошибок, 34
на основе времени отклика, 49
возражения, 54
оценка эффективности, 62
требования к хорошему методу, 35

Н

ненужная работа, исключение, 377

оптимизация

LIO, 378–379

SQL, 381

операции записи, 385

разбор, 382

непривилегированный режим, 185

номера курсоров, файлы данных

трассировки SQL, 111

О

оборудование

модернизация и ухудшение

производительности, 32

обслуживание клиентов, новый

стандарт, 91

ограничение стоимости проектов

повышения производительности, 101

операции над источником строк

файлы данных трассировки SQL,

118

опережающее атрибутирование, 140,

426

определение цели

время отклика

длительное обслуживание

процессором, 397

обманчивые общесистемные

данные, 391

оптимизатор по стоимости, 47

оптимизация, 426

LIO, 378

SQL, 381

достижения технологии, 37

закон Амдала, 40

концентрация на времени

отклика системы, 39

концентрация на пользователь-

ских операциях, 37

коэффициент попаданий в кэш

буферов, 436

операции записи, 385

разбор, 382

освобождение защелки

события ожидания, 38

отсчет времени, 135

формулы, 139

оценка эффективности, 62

ошибки

квантования, 200, 427

выявление, 211

диапазон значений, 213

определение, 203

сложности измерения

процессорного времени, 206

точность измерений, 200

неполные данные рекурсивного SQL,

180

переполнения, 427

данные фиксированных

представлений Oracle,

чувствительность к, 231

пропуск времени при отключении

трассировки, 178

сбор данных расширенной

трассировки SQL, 167

П

память

кэш буферов базы данных, 378

системная глобальная область, 378

устранение излишних обращений,
48

параллельное выполнение

см. также PX, 162

переменные связывания

использование во избежание

полного разбора, 383

файлы данных трассировки SQL,

117

переход по прерыванию, 186

переход по системному вызову, 185

погрешность времени включения

трассировки, 168

подчиненный процесс параллельного

выполнения, 162

поиск

неэффективный SQL, 246

определение фиксированного

представления, 245

причины зависания сеанса, 249

причины зависания системы, 251

файлы расширенной трассировки

SQL, 159

для клиент-серверных

приложений, 160

для приложений с пулами

соединений, 163

- многопоточный сервер (MTS) и, 162
 - пользовательские операции, 427
 - выбор, 64
 - диаграммы последовательности, 42
 - долго выполняющиеся, сбор диагностических данных производительности для, 86
 - концентрация на, 37
 - создание спецификации для проектов повышения производительности, 70
 - приоритеты пользователей, 74
 - пользовательский режим, 185
 - привилегированный режим, 185
 - приложения
 - признаки масштабируемости, 387
 - с пулами соединений, поиск файлов трассировки для, 163
 - принцип неопределенности Гейзенберга, 201
 - прогнозирование повышения производительности, 360
 - программное обеспечение, самоизмерение, 189
 - загрузка процессора, 192
 - фактическая продолжительность работы, 190
 - проекты повышения производительности, 31
 - возможные выгоды при небольших улучшениях, 347
 - ограничение стоимости, 101
 - прогнозирование затрат, 100
 - прибыли, 97
 - рисков, 101
 - экономическая эффективность, 96
 - спецификации, 64
 - избыточные ограничения, 75
 - надежность, 64
 - создание, 70
 - учебные примеры, 389
 - длительное обслуживание процессором, 396
 - длительные события SQL*Net, 408
 - длительные события чтения, 415
 - обманчивые общесистемные данные, 390
 - производительность
 - анализ и принцип неопределенности Гейзенберга, 201
 - лучший специалист, 52
 - модели, 271
 - повышение производительности, *см. также* проекты повышения производительности прогнозирования, 360
 - выбор пути, 91
 - выбор экономически оптимального пути, 93
 - профиль ресурсов, 44, 428
 - как работать с, 343–344
 - в порядке убывания времени отклика, 345
 - исключение лишних вызовов, 348
 - как узнать, что работа завершена, 362
 - модернизация оборудования, 358
 - устранение конкуренции между процессами, 354
 - поиск нужной составляющей времени отклика, 366
 - преобладание событий SQL*Net, 374–375
 - процессор, занятость
 - двойной учет, 198
 - процессы операционной системы, управление, 184
 - переход по прерыванию, 186
 - переход по системному вызову, 185
- Р**
- разбор, оптимизация, 382
 - расширенная трассировка SQL
 - включение, 152
 - причины недостаточного использования, 16
 - собственного исходного кода, 153
 - чужого исходного кода, 155
 - рекурсивный SQL, 126, 429
 - ресурсы
 - оптимизация SQL, 382
 - получение статистик потребления, 239
 - список рассылки Oracle-L, 16

С

- сбор
 - данные расширенной трассировки SQL, 148
 - ошибки сбора, 167
 - понимание архитектуры приложения, 148
- диагностические данные, 77
- агрегирование, 86
- для долго выполняющихся пользовательских операций, 86
- область данных, 81
- ошибки определения области данных, 81
- сеансы, 429
 - данные фиксированных представлений о текущем состоянии, 241
 - создание приблизительного профиля ресурсов, 252
- симметричная многопроцессорность
 - см. также SMP*, 279
- системный вызов, переход по, 185
- случайные величины, 288, 430
- события ожидания Oracle, 87, 141, 159, 170, 188, 230, 231, 242, 243, 251, 257, 258, 260, 263–266, 275, 364, 365, 367, 372, 374, 375, 379, 384, 385, 391, 394, 403, 426
 - CPU service, 368
 - файлы данных трассировки SQL, 116
- события простоя, 260, 430
 - в фоновых сеансах, 265
 - их важность, 372
 - проблема, 260
- создание спецификации для проектов повышения производительности, 70
- приоритеты пользовательских операций, 74
- списки рассылки, оптимизация SQL, 382
- список рассылки Oracle-L, 16
- средний коэффициент использования канала, 279
- средства для анализа времени отклика, 42
 - диаграммы последовательности, 42
 - профиль ресурсов, 44
- счетчики событий, 39

Т

- теория массового обслуживания, 270, 431
 - время обслуживания, 276
 - входные и выходные параметры модели, 277
 - время и скорость обслуживания, 280
 - задержка в очереди и время отклика, 280
 - интегральная функция распределения времени, 284
 - каналы обслуживания, 278
 - коэффициент использования, 278
 - максимальная рабочая пропускная способность, 283
 - поступающие и выполненные запросы, 278
 - устойчивость, 278
- для специалиста по Oracle, 270
- задержка в очереди, 276
- интерфейс ожидания и, 291
 - время ожидания Oracle, 292
 - различные обозначения, 294
- использование греческого алфавита в формулах, 280
- случайные величины, 288
 - математическое ожидание, 288
 - распределение вероятностей, 290
 - функция плотности вероятности, 289
- технологический стек, 42, 431
 - запросы и услуги, 349
- трассировка исходного кода собственного, 153
- чужого, 155

У

- управление процессами операционной системы, 184
 - переход по прерыванию, 186
 - переход по системному вызову, 185
- учебные примеры, повышение производительности, проекты, 389
- длительное обслуживание процессором, 396
 - диагностика и лечение, 399
 - мораль, 401
 - определение цели, 397

- результаты, 400
- длительные события SQL*Net, 408
 - диагностика и лечение, 403
 - мораль, 408
 - определение цели, 402
 - результаты, 408
- длительные события чтения, 415
 - диагностика и лечение, 410
 - мораль, 415
 - определение цели, 409
 - результаты, 414
- обманчивые общесистемные данные, 390
 - диагностика и лечение, 393
 - мораль, 396
 - определение цели, 391
 - результаты, 395

Ф

- файлы расширенной трассировки SQL
 - временные метки, 111
 - время между вызовами базы данных, 125
 - вызовы базы данных, 114
 - запись в, 221
 - знакомство, 107
 - идентификация курсора, 112
 - идентификация приложения, 112
 - идентификация сеанса, 111
 - имена файлов, 159
 - конфиденциальность и, 148
 - маркеры конца транзакции, 120
 - начиная с ядра версии Oracle9i, 137
 - номера курсоров, 111
 - операции над источником, 118
 - определения элементов, 110
 - переменные связывания, 117
 - поиск, 159
 - продолжительность вызова базы данных, 123
 - рекурсивные отношения между вызовами базы данных, 126
 - события ожидания, 116
 - учет времени отклика, 122
 - ядро версии не старше Oracle8i, 136
- физический ввод/вывод, 47
- фиксированное представление Oracle, 224
- формула Эрланга, 281

- формулы
 - отсчет времени, 139
 - теория массового обслуживания, использование греческого алфавита, 280
- функция плотности вероятности, 289

Ц

- циклы, извлечение вызовов разбора из, 384

Э

- экономика очередей, 273
- Энсор, Дэйв, 345
- Эрланг, Агнер, 281, 289, 298, 433
- эффект влияния измерителя, 433
 - запрос данных фиксированных представлений Oracle, 226

Я

- ядро Oracle
 - измерение времени, 184
 - неучтенное время, 193
 - влияние измерителя, 194
 - время, в течение которого процесс не выполняется, 216
 - двойной учет занятости процессора, 198
 - код ядра Oracle без измерительных средств, 219
 - ошибки квантования, 200
 - получение диагностических данных, 240

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-078-2, название «Oracle. Оптимизация производительности» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.