Optimizing Oracle Performance

Cary Millsap and Jeff Holt

O'REILLY®

Oracle

Оптимизация производительности

Кэри Миллсап и Джефф Хольт

Санкт-Петербург — Москва
2006

Кэри Миллсап, Джефф Хольт

Oracle. Оптимизация производительности

Перевод П. Шера

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*О. Летаев*
Редактор	*В. Овчинников*
Корректор	*Н. Аникеева*
Верстка	*О. Макарова*

Миллсап К., Хольт 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 Хольт, 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 Хольта 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». 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 70x100 1/16 . 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

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 *tkprof*) 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. Несо-

[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 бóльшую 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, имеющихся 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 прочитать 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 математи-

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 A «Глоссарий» содержит определения технических терминов, употребляемых в книге.
- Приложение B «Греческий алфавит» представляет собой таблицу греческих букв и соответствующих английских и русских эквивалентов и призвано помочь в чтении главы 9.
- Приложение C «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 Oracle9*i*, 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 попробуете 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 E 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 Миллсап

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 Миллсап

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 мгновенье 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)

Мика Адлер
(Micah Adler)

Филипп Олмес
(Philip Almes)

Энди Бэйли
(Andy Bailey)

Карла Бэйси
(Karla Baisey)

Владимир Баррьер
(Vladimir Barriere)

Кен Баумгарднер
(Ken Baumgardner)

Кертис Беннетт
(Curtis Bennett)

Дарен Бок
(Darren Bock)

Кеннет Брэди
(Kenneth Brady)

Филипп Бригз
(Phillip Briggs)

Майкл Браун
(Michael Brown)

Тим Бунс
(Tim Bunce)

Др. Бёрт Бёрнс
(Dr. Burt Burns)

Лассе Кристенсен
(Lasse Christensen)

Кэрол Колрейн
(Carol Colrain)

Руди Корси
(Rudy Corsi)

Кэрол Дако
(Carol Dacko)

Доминик Дельмолино
(Dominic Delmolino)

Дэвид Дэмпси
(David Dempsey)

Кёрти Дешпанде
(Kirti Deshpande)

Йоханнес Джернæс
(Johannes Djernæs)

Грег Догерти
(Greg Doherty)

Элен Дудар
(Ellen Dudar)

Нэнси Душкин
(Nancy Dushkin)

Джулиан Дайк
(Julian Dyke)

Мортен Иган
(Morten Egan)

Джин Эмерсон
(Jean Emerson)

Бьерн Энгсис
(Bjørn Engsig)

Дэйв Энсор
(Dave Ensor)

Барри Эпштейн
(Barry Epstein)

Генри Фахей
(Henry Fahey)

Марк Фарнхем
(Mark Farnham)

Роберт Фейнер
(Robert Feighner)

Зак Фрис
(Zach Friese)

Питер Грэм
(Peter Gram)

Дональд Гросс
(Donald Gross)

Кайл Хэйли
(Kyle Hailey)

Стефан Хэйсли
(Stephan Haisley)

Тереза Хэйсли
(Theresa Haisley)

Рей Хэмлит
(Ray Hamlett)

Ахмер Хасан
(Ahmer Hasan)

Джим Хендон
(Jim Herndon)

Дэйв Херрингтон
(Dave Herrington)

Кэрол Хипп
(Carol Hipp)

Др. Майрон Глинка
(Dr. Myron Hlynka)

Торбен Хольм
(Torben Holm)

Марк Хостман
(Mark Horstman)

Мемдух Ибрагим
(Mamdouh Ibrahim)

Джонатан Интнер
(Jonathan Intner)

Линн Изабелла
(Lynn Isabella)

Кен Якобс
(Ken Jacobs)

Нил Йенсен
(Neil Jensen)

Фил Джоэл
(Phil Joel)

Гумундур Джисепсон
(Gumundur Jysepsson)

Дэрри Кэбснелл
(Derry Kabcenell)

Джордж Кадифа
(George Kadifa)

Майк Кол
(Mike Kaul)

Брайан Куш
(Brian Kush)

Арман Садат Киаэ
(Armand Sadat Kyaee)

Том Кайт
(Tom Kyte)

Рей Лэйн
(Ray Lane)

Санг Чул Ли
(Sang Chul Lee)

Джонатан Льюис
(Jonathan Lewis)

Маргарет Льюис
(Margaret Lewis)

Джим Литтлфилд
(Jim Littlefield)

Хуан Лоайза
(Juan Loaiza)

Андреа Лопес
(Andrea Lopez)

Скотт Лавинфосс
(Scott Lovingfoss)

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

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, пытаясь 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$TRACE, по крайней мере, в Oracle9*i* Release 2, представляется нецелесообразным, т. к. это представление недокументировано, ненадежно и не поддерживается. Представление V$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 Oracle9*i* 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);
$sth = $dbh->prepare(q(select * from v$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$, как правило, нет атрибута принадлежности к сессии. Чтобы понять, к чему это приводит, представьте, что профиль ресурсов показывает, что время отклика расходуется в основном на ожидание освобождения защелок. Представление V$LATCH указывает на активное использование двух различных защелок в период выполнения исследуемой пользовательской операции. Какая из защелок 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 защелки только в том случае, когда запрос на получение защелки прошел спин-фазу[1], но был отвергнут, вследствие чего процесс ядра совершает системный вызов, освобождающий процессор для другого, произвольно выбранного процесса. Если попытка получения защелки 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 инструментарий Тома 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 защелку, 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 защелки. 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 применяются 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-

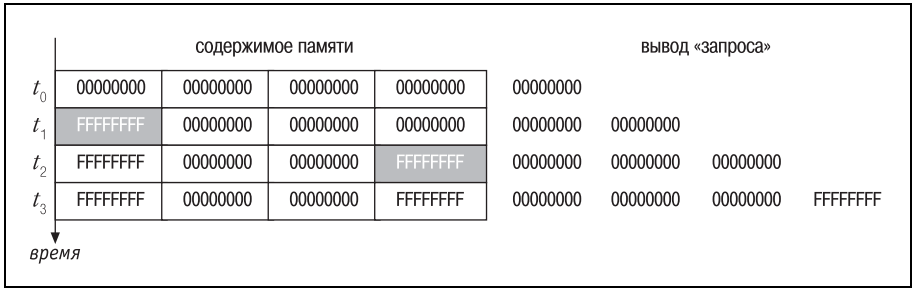

Рис. 8.1. *Проблема, вызванная несогласованностью чтения: в выходном потоке может быть представлено состояние памяти, никогда в действительности не существовавшее*

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 cu необработанных трассировочных данных.

CONSISTENT_GETS

Соответствует статистике cr 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 Oracle9*i*, 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 Oracle8*i* 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 Oracle9*i* 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 напомню, 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_TIMEOUTS                                     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 Oracle8*i*:

```
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 Oracle9*i*, 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 Oracle9*i*. 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 защелок 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, привлекшими 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≥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 Oracle8*i* дело обстояло именно так. Однако в Oracle9*i* эти два предиката не равнозначны.

Ядро Oracle9*i* 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 Oracle9*i* 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, защелки 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$SESSION_WAIT
select s.inst_id,s.indx,s.ksussseq,e.kslednam, e.ksledp1,s.ksussp1,s.ksussp
1r,e.ksledp2, s.ksussp2,s.ksussp2r,e.ksledp3,s.ksussp3,s.ksussp3r, round(s.
ksusstim / 10000), s.ksusewtm, decode(s.ksusstim, 0, 'WAITING', -2, 'WAITED
 UNKNOWN TIME',  -1, 'WAITED SHORT TIME', 'WAITED KNOWN TIME')  from x$ksus
ecst s, x$ksled e where bitand(s.ksspaflg,1)!=0 and bitand(s.ksuseflg,1)!=0
 and s.ksussseq!=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^k единиц, причем $10^k/10^4 = 10^2$. 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 Oracle9*i*, 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 LIO, 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:

                                                    LIOs                    LIOs
   Stmt Id    PIO blks         LIOs Rows       per Row       Runs    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_ID`[1]), 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 Oracle8*i* 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 = %s.%06d\n", "t0", time2str("%T", $sec0), $msec0;
printf "%24s = %s.%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 промежутке 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 состоянии 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 соотносятся 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 помошью 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 простою. 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 $1 + 2 + 3 + ... + 100 = 5050$ секунд, а продолжительность ожиданий чтения сокета будет равна $99 + 98 + 97 + ... + 0 = 4950$ 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 (DBW0)  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 DBW0, 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. сообщающие 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 побеспокоиться.

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?