

Руководство системного администратора UNIX

2-е издание



Настройка производительности UNIX-систем



O'REILLY®

Джан-Паоло Д. Мусумеси и Майк Лукидес

System Performance Tuning

Second Edition

*Gian-Paolo D. Musumeci,
Mike Loukides*

O'REILLY®

Настройка производительности UNIX-систем

Второе издание

*Джан-Паоло Д. Мусумеси,
Майк Лукидес*



*Санкт-Петербург — Москва
2003*

Джан-Паоло Д. Мусумеси, Майк Лукидес

Настройка производительности UNIX-систем, 2-е издание

Перевод Ю.Кунивера

Главный редактор
Зав. редакцией
Научный редактор
Редактор
Корректор
Верстка

*А. Галунов
Н. Макарова
Ф. Торчинский
Ю. Кунивер
С. Беляева
Н. Гриценко*

Мусумеси Д.-П., Лукидес М.

Настройка производительности UNIX-систем. – Пер. с англ. – СПб: Символ-Плюс, 2003. – 408 с., ил.

ISBN 5-93286-034-0

Книга «Настройка производительности UNIX-систем» отвечает на два важнейших вопроса: как добиться максимального эффекта без покупки дополнительного оборудования и в каких случаях его все же стоит приобрести (больше памяти, более быстрые диски, процессоры и сетевые интерфейсы). Вложение денежных средств – не панацея. Адекватно оценить необходимость обновления и добиться максимальной производительности можно только хорошо представляя работу компьютеров и сетей и понимая распределение нагрузки на системные ресурсы.

Авторы книги оказали неоценимую помощь администраторам, подробно и аргументированно рассказав обо всех тонкостях искусства настройки систем. Полностью обновленное издание ориентировано на Solaris и Linux, но обсуждаемые принципы применимы к любым системам. В книге рассматриваются настройка параметров, управление рабочим процессом, методы измерения производительности, выявление перегруженных и неработоспособных участков сети, добавлен новый материал о дисковых массивах, микропроцессорах и оптимизации программного кода.

ISBN 5-93286-034-0

ISBN 0-596-00284-X (англ)

© Издательство Символ-Плюс, 2003

Authorized translation of the English edition © 2002 O'Reilly & Associates Inc. This translation is published and sold by permission of O'Reilly & Associates Inc., the owner of all rights to publish and sell the same.

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

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

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

Подписано в печать 21.10.2003. Формат 70х100/16. Печать офсетная.

Объем 25,5 печ. л. Тираж 2000 экз. Заказ N

Отпечатано с диапозитивов в Академической типографии «Наука» РАН
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Предисловие	7
1. Введение в настройку производительности	13
Введение в архитектуру компьютера	15
Принципы настройки производительности	20
Настройка статической производительности	25
Заключение	28
2. Управление рабочим процессом	30
Определение параметров рабочего процесса	31
Регулирование рабочей нагрузки	40
Оценка производительности	47
Заключение	58
3. Процессоры	59
Архитектура микропроцессора	61
Кэширование	69
Планирование процессов	76
Многопроцессорная обработка	88
Периферийные соединения	97
Инструменты для контроля производительности процессора	106
Заключение	123
4. Память	124
Реализации физической памяти	125
Архитектура виртуальной памяти	128
Пейджинг и свопинг	139
Потребители памяти	143
Инструменты для измерения производительности памяти	150
Заключение	159

5. Диски	160
Архитектура диска	161
Интерфейсы	170
Общие проблемы производительности	190
Файловые системы	193
Инструменты для анализа	216
Заключение	228
6. Дисковые массивы	229
Терминология	230
Уровни RAID	232
Сравнение программных и аппаратных реализаций RAID	244
Итог по конструкциям дисковых массивов	246
Программные реализации RAID	247
Рецепты RAID	260
Заключение	265
7. Сети	266
Основы сетей	267
Физические носители	271
Сетевые интерфейсы	274
Сетевые протоколы	290
NFS	313
CIFS и UNIX	330
Заключение	331
8. Оптимизация кода	332
Два важнейших принципа	333
Методы анализа кода	340
Примеры оптимизации	356
Взаимодействие с компиляторами	360
Заключение	369
9. Первоочередная настройка	370
Горячая пятерка советов по настройке	371
Рецепты первоочередной настройки	374
Алфавитный указатель	383

Предисловие

Эта книга об искусстве настройки систем, которая необходима для оптимальной производительности прикладных программ. На страницах книги обсуждается выполнение самой настройки и извлечение максимальной пользы из приложений. Здесь рассматриваются базовые алгоритмы, лежащие в основе настройки параметров; их освоение позволит принимать разумные решения в любой операционной среде. Кроме того, затрагивается планирование нагрузки систем – речь пойдет о регулировании системы для решения различных задач.

Для кого написана эта книга

В основном книга предназначена для тех, кто заинтересован в оптимизации производительности своих компьютерных систем. Читатели найдут ясное описание того, как различные компоненты системы взаимодействуют друг с другом и на что следует обращать внимание. Программисты, пишущие или оптимизирующие программы, найдут краткие пояснения к ключам современных компиляторов, а также разбор того, как базовая операционная система управляет запущенными приложениями. Наконец, те читатели, которые просто хотят знать больше о работе компьютеров, найдут на этих страницах интересные разъяснения.

Охват книги

Эта книга в значительной степени ориентирована на операционную среду Solaris (до версии Solaris 8 включительно) и систему Linux. Однако особое значение в книге придается именно Solaris. Для этого есть несколько причин:

- Большинство хранилищ данных работают под управлением Solaris. Такие операционные среды наиболее требовательны к настройке производительности.
- Машины Solaris более других систем сконцентрированы на производительности. Видимо, это объясняется тем, что системы Sun в среднем являются более дорогостоящими, чем их Linux-аналоги.

В результате люди ожидают лучшей производительности, поэтому в этой области в Solaris приложено немало усилий. Если производительность машины Linux недостаточна, то можно купить другую машину и распределить между ними нагрузку – это дешево. Если же Ultra Enterprise 10000 стоимостью несколько миллионов долларов не работает должным образом, а компания каждую минуту теряет из-за этого нетривиальные суммы, то пользователи звонят в Sun Service и требуют ответа.

- Наконец, инструменты для анализа производительности в Solaris *более* серьезные, чем в Linux. Частично это связано с тем, что разработать такие инструменты не так-то просто. А частично с тем, что порой для достижения наилучшей производительности необходимо проводить изменения в аппаратных средствах. Кроме того, Linux – это относительно новая операционная система, разработанная сравнительно небольшим коллективом. В свою очередь, Solaris (и ее предшественница SunOS, давшая уйму обратной связи для процесса разработки Solaris) – это операционная система с богатым прошлым, а Sun – это огромная компания. Возможность тщательного анализа проблемы производительности и принятия грамотных решений по оптимизации в значительной мере основаны на данных, собранных соответствующими инструментами (в противном случае настройка сводится к гаданию на кофейной гуще). Поэтому в конечном итоге настраивать системы Linux чрезвычайно трудно. Такая настройка прямо противоречит одному из основных принципов настройки производительности (см. раздел «Принцип 0: Хорошо понимайте свою операционную среду» главы 1).

Не будем предвзяты в отношении любой операционной системы.¹ Скажем лишь, что серьезная оптимизация производительности в Linux на сегодня является непростой задачей, ибо средств, облегчающих понимание работы Linux, пока не существует.

При запуске Linux совершенно *необходимо* раздобыть и установить пакет *sysstat*, предоставляющий версии *iostat*, *mpstat* и *sar* (в урезанном варианте). Будучи весьма неполными по сравнению с версиями в Solaris, они по крайней мере помогают получить хоть какую-то информацию. На время написания книги эти программы можно было загрузить с веб-сайта их автора Себастьяна Годарда (Sebastien Godard) <http://perso.wanadoo.fr/sebastien.godard/>.

Как читать эту книгу

Эта книга – и справочник и рассказ. Читатели с опытом анализа и настройки производительности, возможно, загорятся желанием немед-

¹ В интересах полноты представления: автор работал в Sun Microsystems около половины того времени, которое заняло написание этой книги.

ленно перескочить к конкретному разделу (наверное, к тому, где затрагиваются слабые, с их точки зрения, места в системе) и станут рассматривать книгу как справочник. Так поступать ни в коем случае не следует.

Эта книга как рассказ

Для наибольшей пользы лучше сначала прочесть эту книгу как рассказ. Навыки, методы размышления и подходы к задачам – вот самое важное, что можно почерпнуть из текста. Такая информация впитывается на протяжении времени, а не приходит путем запоминания пунктов из списка.

Многие из обсуждаемых тем будут казаться довольно простыми и незамысловатыми. На самом деле они напрямую связаны с основами архитектуры современных компьютеров, а именно с тем, как компьютеры вообще работают. Зачастую эти темы весьма запутанны. Не стоит беспокоиться, если разделы книги придется перечитывать, поскольку для усвоения материала необходимо время.

Эта книга как справочник

Для тех, кто прочел рассказ и получил представление о том, как, словно мозаика, складывается из кусочков производительность, эта книга будет полезным справочником. Кто-то может не прочесть рассказ и, рассматривая одну из таблиц (например, описание параметров памяти ядра в разделе «Управление виртуальной памятью в Linux» главы 4), думать, что он усвоил все подробности. На самом деле это не так – он многое упустил. Прочтение этой книги как рассказа позволит в деталях понимать сложные задачи, связанные с настройкой системы.

Структура книги

Эта книга состоит из девяти глав:

Глава 1 «Введение в настройку производительности» закладывает фундамент для всей книги. В ней освещены основные принципы настройки.

Глава 2 «Управление рабочим процессом» описывает управление производительностью на основе осмысления работы системы и ограничения нагрузки. Кроме того, в ней рассказывается о некоторых распространенных методах измерения производительности.

В главе 3 «Процессоры» обсуждаются архитектура современного процессора, кэширование, многопроцессорные системы и то, как операционная система планирует задачи.

Глава 4 «Память» объясняет, как в компьютере организована подсистема памяти и как она взаимодействует с другими компонентами системы.

Глава 5 «Диски» представляет диски в простейшем виде: как функционируют жесткие диски и что можно предпринять в отношении некоторых связанных с ними ограничений.

Глава 6 «Дисковые массивы» в каком-то смысле является продолжением предыдущей главы. В ней обсуждается методика объединения многочисленных дисков в одно логическое устройство.

Глава 7 «Сети» посвящена производительности сети: от физического уровня до протоколов связи, таких как стек TCP/IP, и систем совместного использования файлов Samba и NFS.

Глава 8 «Оптимизация кода» – это лаконичное введение в методику создания быстродействующего программного кода, а также сводка по работе с современными компиляторами.

Глава 9 «Первоочередная настройка» – это краткое изложение некоторых важных пунктов, обсуждаемых на протяжении всей книги. Она представляет собой справочник для быстрого устранения неполадок и конфигурирования системы.

Соглашения по оформлению

В книге приняты следующие соглашения:

Моноширинный шрифт

Используется для примеров исходного кода и фрагментов исходного кода в самом тексте, включая имена переменных и функций. Моноширинный шрифт также применяется для отображения результатов, выданных компьютером, и содержимого файлов.

Моноширинный полужирный

Применяется для команд, набираемых пользователем.

Курсив

Используется для названий команд, имен каталогов и файлов. Также применяется для выделения новых терминов.

Полужирный

Применяется для выделения векторов (в математическом смысле) и ключей команд.

Комментарии и вопросы

Вся информация, приведенная в книге, была по возможности протестирована и проверена. В то же время какие-то детали могли измениться, а опечатки или ошибки быть обнаружены. О таких данных, а так-

же о своих предложениях по поводу будущих изданий можно сообщать по адресу:

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

Кроме того, можно посылать сообщения по электронной почте. Чтобы попасть в список рассылки или заказать каталог, присылайте письмо по адресу

info@oreilly.com

Чтобы задать технические вопросы или прислать комментарии к книге, присылайте письмо по адресу

bookquestions@oreilly.com

Существует сайт, где приведены примеры из книги, список ошибок и планы на будущее. Эта страница доступна по адресу

<http://www.oreilly.com/catalog/spt2/>

За более подробной информацией об этой и других книгах можно обратиться на сайт издательства по адресу

<http://www.oreilly.com>

Благодарности Джан-Паоло Мусумеси

Я в долгу перед многими людьми за их помощь и поддержку во время написания этой книги.

Прежде всего, мне хочется поблагодарить моего редактора, Майка Лукидеса (Mike Loukides). Срок сдачи книги отодвигался много раз, и его терпение, замечательные советы и оптимизм оказались жизненно важными для выхода книги. Также я очень признателен моей организационной команде в Университете штата Иллинойс, особенно Моне Хит (Mona Heath) и Эду Кролу (Ed Krol), и в Sun Microsystems – доктору Кенг-Тай Ко (Keng-Tai Ko) и Мирославу Кливански (Miroslav Klivan-sky). Их поддержка, терпение и понимание были бесценны.

В ходе работы над книгой мои друзья оказывали мне постоянную поддержку. Они давали технические консультации и понимали фразу «Нет, я не приду, я должен работать над книгой», которая неожиданно вошла в мой лексикон на два года. Мне хочется особенно поблагодарить Кейт Вессел (Keith Wessel), Криса Вехнера (Kris Wehner), Майди Аббас (Majdi Abbas), Деб Флигор (Deb Fligor), Джея Крейбича (Jay Kreibich), Адриана Кокрофта (Adrian Cockcroft), Ричарда Макдугала

(Richard McDougall), Элизабет Парсел (Elizabeth Purcell), Тайфун Косаглу (Tayfun Kocaglu), Пола Стронга (Paul Strong), Фила Страччино (Phil Stracchino), Кристофа Харпера (Christof Harper) и Кена МакИнниса (Ken MacInnis). Порядок здесь не важен. Доктор Санья Пател (Sanjay Patel) был очень любезен допустить учащегося на свой курс по архитектуре компьютеров в Университете штата Иллинойс без прохождения предварительных курсов. Его поддержка была неоценима. Роза Платт (Rose Platt), Ненси Вебер (Nancy Weber) и доктор Роберт Кан (Robert Cahn) открыли для меня свои двери. Благодаря им я мог долгое время сидеть и писать в надежном уединении. Кейт Секор (Kate Secor) помогла мне сохранить рассудок, когда я писал последнюю часть этой книги. Она шутит, что была моим «хорошим рабочим тотемом». Когда Кейт была рядом, работа шла значительно быстрее.

Трудно переоценить вклад моей семьи, особенно моих родителей, докторов Диану и Антонио Мусумеси. Присутствие людей, которые вырастили тебя, порою очень обнадеживает. Особенно если они оба написали книги – не так уж это и трудно, правда?¹ Мои братья Доменико и Уолтер не дали мне оторваться от реального мира. Мой дедушка Уолтер напомнил мне, что бывали времена и без компьютеров. Домашний пес Гизмо был постоянным источником забав (очень трудно сосредоточиться, когда 15-футовый бордер-терьер носится по дому со всей скоростью, на какую способен; кроме того, иногда полезно *не* сосредотачиваться ни на чем).

Я рад посвятить эту книгу моей бабушке, Кэтлин Миколайтис.

Благодарности Майка Лукидеса

Прежде всего, я хочу поблагодарить Джан-Паоло Д. Мусумеси (Gian-Paolo D. Musumeci). Я бы никогда не сделал то, что сделал он. Подготовка второго издания книги – это действительно его заслуга. Я горжусь первым изданием, но должен признаться, что в основном это была журналистика. Я общался с группой системных экспертов и записывал их опыт. Это были Дуг Гилмор (Doug Gilmore), Крис Райленд (Chris Ryland), Тен Бронсон (Tan Bronson) и другие эксперты из сообщества Multiflow и не только. (Мало кто знает, что эта книга зарождалась как часть комплекта документации Multiflow TRACE/UNIX, хотя Multiflow прекратил свое существование еще до публикации.) В то время как я – просто журналист, Джан-Паоло является первоклассным специалистом. Он привнес в книгу огромные знания и опыт, которых у меня никогда не было. И как бы я ни гордился первым изданием, второе издание значительно лучше.

¹ Ответ: *очень трудно*.

- *Введение в архитектуру компьютера*
- *Принципы настройки производительности*
- *Настройка статической производительности*
- *Заключение*

1

Введение в настройку производительности

*Тысяча испытаний во владениях зла
ожидает того, кто покусается на основы.*

Henry David Thoreau, 1854

Пожалуй, стремление двигаться быстрее есть сущность человеческой эволюции. Всего триста лет назад наивысшая скорость движения, которую можно было себе представить, составляла несколько десятков миль в час – на борту быстроходного парусного судна со свежим ветром за спиной. Теперь пришло время расширить наши воззрения на максимально достижимую скорость в пределах земной атмосферы – возможно, пятнадцать тысяч миль в час, вдвое больше скорости звука. Таков потолок для штатских лиц, не имеющих доступа к новейшим военным самолетам. Путешествие под парусом из Лондона в Нью-Йорк ранее занимало три недели. Сегодня такое расстояние можно преодолеть всего за два с половиной часа, попивая при этом шампанское.

Врожденное человеческое стремление быть мобильнее выражается во многом: микроволновые печи позволяют быстро приготовить обед, мощные автомобили и мотоциклы несутся на бешеной скорости, электронная почта позволяет общаться со скоростью мысли. Но что происходит, когда почтовый сервер перегружен, если все мы одновременно подключаемся к серверу, чтобы проверить, не пришло ли что-нибудь, пока мы спали? Или когда система снабжения компании, продающей микроволновые печи, способна осилить лишь половину нагрузки, или когда система CAD инженера-механика работает столь медленно, что двигатель для новой модели года, разрабатывающийся с ее помощью, не будет готов в срок?

Все это – задачи, подводящие к необходимости настройки производительности: адаптация скорости компьютерной системы к требованиям по скорости, предъявляемым реальным миром.

Иногда эти задачи менее заметны, иногда более. Необходимо тщательно анализировать изменения в системе, приведшие к тому, что система стала работать недопустимо медленно. Такие воздействия могут как скрываться внутри, например избыточная нагрузка на систему, так и появляться извне. К примеру, незначительное (или полное) изменение алгоритма управления ресурсами в новой редакции операционной системы, то есть нечто чрезвычайно важное для программного обеспечения. Решения порою бывают мгновенными (всего-то выверить одну опцию), а иногда медленными и болезненными (недели анализа, консультаций с производителями и внимательной перепланировки инфраструктуры).

В начале работы над книгой все казалось незатейливым: взять замечательное, но устаревшее первое издание книги Майка Лукидеса (Mike Loukides) и доработать ее с учетом знаний о современных компьютерных системах. Но чем дальше, тем более становилось ясно, что на самом деле эта книга даст намного больше, чем просто описание настройки производительности. В действительности она охватывает две отдельные темы:

Настройка производительности

Искусство повышения производительности для конкретного набора приложений (также известное как «выдавливание крови из камня»).

Планирование нагрузки

Размышления о том, какое оборудование приобрести для решения задач (также известные как «предсказание будущего»).

Эти темы опираются на теорию архитектуры компьютера. Эта книга посвящена не разработке приложений; скорее она фокусирует внимание на операционной системе, базовых аппаратных средствах и их взаимодействии.

Для большинства системных администраторов компьютер представляет собой черный ящик. Это весьма разумно для многих задач: в конце концов, чтобы конфигурировать и обслуживать почтовый сервер, естественно, нет необходимости понимать, как операционная система управляет свободной памятью. Однако при настройке производительности, которая, по существу, зиждется на знании базовых аппаратных средств и того, как они абстрагированы, подлинное понимание поведения системы приходит с обстоятельными сведениями о внутренней работе машины. В этой главе мы кратко обсудим некоторые наиболее важные понятия архитектуры компьютера, а затем обратимся к основным принципам настройки производительности.

Введение в архитектуру компьютера

Полное обсуждение архитектуры компьютера выходит за рамки этой книги. Однако время от времени вопросы архитектуры будут рассматриваться, чтобы при обсуждении системы исходить из первооснов. Интересующимся этим предметом можно посоветовать обратиться к замечательным руководствам по этой теме. Возможно, наиболее читаемыми являются руководства Джона Хеннесси (John Hennessy) «Computer Organization and Design: The Hardware/Software Interface» (Организация и проектирование компьютера: интерфейсы программных и аппаратных средств) и Давида Паттерсона (David Patterson) «Computer Architecture: A Quantitative Approach» (Архитектура компьютера: количественный подход). Обе книги изданы Морганом Кауфманом (Morgan Kaufmann).

Этот раздел будет посвящен двум важнейшим общим концепциям архитектуры: базовым средствам, с помощью которых мы подходим к задаче (уровни представления), и основной модели, на базе которой созданы компьютеры.

Уровни представления

Сведем задачу к обзору того, что присуще работе компьютера: от логических схем, дающих решение важнейшей задачи — как создавать компьютеры общего назначения, до нескольких миллионов бит двоичного кода. По мере прохождения этих этапов задача будет преобразована в более простую (по крайней мере с точки зрения компьютера). Назовем эти шаги *уровнями представления*.

Программное обеспечение: алгоритмы и языки

Перед лицом задачи, предполагающей помощь компьютера, в первую очередь разрабатывается *алгоритм* с целью свести задачу к заданию. Проще говоря, алгоритм — это итеративная последовательность инструкций для выполнения конкретной задачи — например, как должным образом сортировать почту клерку, изучающему и распределяющему входящую корреспонденцию.

Далее программист должен преобразовать этот алгоритм в программу, написанную на *языке*.¹ Как правило, это язык высокого уровня, такой как C или Perl, хотя это может быть и язык низкого уровня, такой как ассемблер. Наличие языка делает жизнь легче: структура и грамматика языков высокого уровня позволяют легко писать сложные программы. Программы, написанные на языке высокого уровня (обычно пере-

¹ Было высказано предположение, что математики — это устройства для преобразования кофе в теоремы. Если это правда, то, видимо, программисты — это устройства для преобразования кофеина и алгоритмов в исходный код.

носимые между различными системами), затем преобразуются компилятором в команды нижнего уровня – в соответствии с архитектурой конкретной системы. Эти команды определяются структурой системы команд.

Структура системы команд

Структура системы команд (Instruction Set Architecture, ISA) – это основной язык микропроцессора: он определяет базовые, неделимые команды, которые могут быть выполнены. ISA служит интерфейсом между программными и аппаратными средствами. Примеры структуры системы команд включают IA-32, используемую в процессорах Intel и AMD, MIPS, реализованную в R-серии микропроцессоров Silicon Graphics/MIPS (пр. R12000), и систему SPARC V9, применяемую в сериях UltraSPARC компании Sun Microsystems.

Аппаратура: микроархитектура, схемы и устройства

На этом уровне происходит погружение в электронику и схемотехнику. Сначала функциональные блоки микроархитектуры и эффективность разработки. Затем, ниже уровня микроархитектуры, реализация функциональных блоков на основе разработки схем: здесь становятся реальными проблемы электрических взаимных наводок. Полное обсуждение уровня аппаратных средств не входит в замысел книги; настройка различных микропроцессоров далеко не всегда возможна.

Модель фон Неймана

Модель фон Неймана (von Neumann) служила базовой моделью при проектировании всех современных компьютерных систем: она дает каркас, на который можно повесить абстракции и наполнить их «плотью», сформированной средствами уровней представления.¹

Модель состоит из четырех ключевых компонентов:

- *Система памяти*, которая хранит как команды, так и данные. Известна как *система с хранимой программой*. Доступ к этой памяти осуществляется с помощью *регистра адреса (memory address register, MAR)*, куда подсистема памяти помещает адрес ячейки памяти, и *регистра данных (memory data register, MDR)*, куда она помещает данные из ячейки с указанным адресом. Более подробно подсистема памяти обсуждается в главе 4.
- По крайней мере один *блок обработки данных*, наиболее известный как *арифметико-логическое устройство (ALU)*. Эти блоки чаще

¹ Отличная книга с более полным описанием модели фон Неймана – Уильям Аспрей (William Aspray) «John von Neumann and the Origins of Modern Computing» (Джон фон Нейман и истоки современной вычислительной техники), издательство MIT Press.

называют *центральными процессорами* (CPU).¹ Этот блок отвечает за выполнение всех команд. Процессор также имеет небольшой объем памяти, называемый *набором регистров*. Обсуждение процессоров будет более подробным.

- *Блок управления*, отвечающий за операции между компонентами модели. Включает в себя *счетчик команд*, содержащий следующую команду для загрузки, и *регистр команд*, в котором находится текущая команда. Особенности модели управления выходят за рамки этой книги.
- Системе необходим энергонезависимый способ хранения данных, а также выдачи их пользователю и принятия входных данных. Это прерогатива подсистемы *ввода-вывода* (I/O). Книга главным образом касается дисковых накопителей как механизмов для ввода-вывода. Они будут обсуждаться в главе 5, а вопросы сетевого ввода-вывода – в главе 7.

Несмотря на существенный прогресс в компьютерной технике за последние шестьдесят лет, устройство компьютеров по-прежнему уместается в этих рамках. Это очень весомое утверждение: несмотря на то, что компьютеры стали мощнее, а их нынешнее применение было невозможно даже представить в конце второй мировой войны, основные идеи, заложенные фон Нейманом и его коллегами, пригодны и сегодня.

Кэши и иерархия памяти

Как будет позднее рассмотрено в разделе «Принципы настройки производительности» этой главы, один из принципов настройки производительности гласит: без компромиссов *не* обойтись. Это было выяснено первопроходцами в этой области, а идеальное решение не найдено и по сию пору. При проектировании системы памяти часто приходится выбирать между ценой, скоростью и емкостью. (Физические параметры, скажем, теплоотвод, также играют свою роль, но в рамках этого обсуждения они скрыты за другими переменными.) Конечно, можно предложить чрезвычайно большие, чрезвычайно быстрые системы, например суперкомпьютер Cray 1S, который использовал очень быстрое статическое ОЗУ (RAM) исключительно для памяти.² Но это решение не всегда можно повторить для других компьютерных устройств.

Итак, проблема состоит в том, что емкость запоминающих устройств обратно пропорциональна производительности, особенно в отношении наивысшего соотношения цена–производительность. Современный

¹ В современных реализациях под термином «CPU» подразумевается как сам центральный процессор, так и блок управления.

² Трудности из-за тепловыделения, касающиеся памяти, были первейшей причиной предусмотреть в системе водоохлаждение. Кроме того, при типичной конфигурации стоимость подсистемы памяти составляет около трех четвертей стоимости машины.

микропроцессор может иметь время цикла, измеряемое в долях наносекунды, в то время как доступ к оперативной памяти вполне может быть в пятьдесят раз медленнее.

Чтобы разрешить эти трудности, возьмем на вооружение так называемую *иерархию памяти*. Она основана на создании пирамиды участков памяти (рис. 1.1). Вверху пирамиды располагаются очень маленькие, чрезвычайно быстрые участки памяти. Ниже представлены более медленные участки, зато соответственно большего размера. В основании пирамиды находится библиотека на лентах: много терабайт, но доступ к запрашиваемой информации может занимать минуты.



Рис. 1.1. Иерархия памяти



С точки зрения микропроцессора оперативная память очень медленная. Все, что связано с обращением к оперативной памяти, неудачно – если только мы не адресуемся к оперативной памяти вместо того, чтобы обратиться к еще более медленному носителю (такому как диск).

Назначение пирамиды – кэшировать наиболее часто используемые данные и команды на более высоких уровнях. Например, если необходимо снова и снова обращаться к одному и тому же файлу на ленте, то было бы неплохо хранить временную копию на следующем по быстродействию уровне хранения (диск). Таким же образом, учитывая существенное преимущество в производительности, можно хранить файл в основной памяти, если к нему идет частое обращение на диске.

Выгоды 64-разрядной архитектуры

Компании, производящие аппаратные и программные средства компьютеров, часто считают необходимым упомянуть размер адресного пространства их систем (обычно 32 или 64 бит). За последние пять лет скачок от 32-разрядных к 64-разрядным микропроцессорам и операционным системам вызвал много крикливой рекламы со стороны отделов сбыта. Истина состоит в том, что хотя в определенных случаях

64-разрядная архитектура работает значительно быстрее 32-разрядной, в основном их производительность сопоставима.

Что понимается под 64 разрядами?

Количество «разрядов» имеет отношение к ширине шины данных. Однако на самом деле все зависит от контекста. Например, можно говорить о 16-разрядной шине данных (скажем, UltraSCSI). Это означает, что такое соединение позволяет передавать 16 бит информации за единицу времени. При условии одинаковости всего остального это означает в два раза более быстрое взаимодействие, нежели в случае с 8-разрядной шиной.

«Битовость» запоминающей системы определяется тем, сколько адресных линий используется при передаче адреса памяти. Например, при наличии 8-разрядной шины и необходимости доступа к 19-му участку памяти применяются соответствующие адресные линии (1, 2 и 5 – исходя из 19 в двоичной системе (00010011); там, где стоит единица, активизируется адресная линия). Заметим, однако, что на 8-битную адресацию налагается ограничение 64 (2^8) адреса в памяти. 32-разрядные системы, следовательно, ограничены 4 294 967 296 (2^{32}) позициями. Так как обычно извлекать информацию из памяти можно однобайтными блоками, это означает, что системе может быть непосредственно доступно не более 4 Гбайт памяти. Переход на 64-разрядные операционные системы и аппаратные средства знаменует, что максимальный объем адресуемой памяти составляет около 16 петабайт (16 777 216 Гбайт), что, вероятно, вполне достаточно для обозримого будущего.

К сожалению, на практике не все так просто. 32-разрядная система SPARC допускает применение более чем 4 Гбайт установленной памяти, но в Solaris ни один отдельный процесс не может использовать более 4 Гбайт. Это вызвано тем, что аппаратура, отвечающая за управление памятью, действительно использует 44-битную адресную схему, но операционная система Solaris отводит одному процессу объем памяти, адресуемый всего лишь 16 бит.

Последствия для производительности

Переход от 32- к 64-разрядной архитектуре к тому же увеличил емкость основной памяти и объема памяти, выделяемой отдельному процессу. Обычный вопрос: что от этого выиграли приложения? Вот несколько видов приложений, выигравших от большего адресного пространства:

- Приложения, ранее не способные применять для решения задач наиболее эффективные с точки зрения времени алгоритмы, поскольку эти алгоритмы были ориентированы на память более 4 Гбайт.
- Приложения, в которых кэширование большого объема данных критически важно, а потому чем больше памяти доступно процессу, тем больше данных может быть кэшировано.

- Приложения в системах, где память – узкое место, вследствие не-померного к ней обращения (много маленьких процессов). Заметим, что в системах SPARC это не было проблемой: каждый процесс мог видеть только 4 Гбайт, но памяти могло быть установлено намного больше.

Вообще говоря, больше всех от 64-разрядных систем выиграли машины для высокопроизводительных вычислений и корпоративных баз данных. Для средних настольных рабочих станций 32 разрядов вполне достаточно.

К сожалению, переход на 64-разрядные системы также означал, что базовые операционные системы и системные вызовы нужно было модифицировать. Порой это приводило к снижению производительности (например, при работе с указателями приходится иметь дело с большим количеством данных). Это означает, что при запуске в 64-разрядном режиме возможно небольшое снижение производительности.

Принципы настройки производительности

В этой книге представлено несколько хорошо зарекомендовавших себя практических правил. Как говорится в давнем техническом бюллетене, выпущенном IBM, такие правила исходят от людей, которые живут за городом и не отягощены производственным опытом.¹

Помня об этом, основы настройки производительности системы можно свести к пяти принципам: хорошо понимайте свою операционную среду, нет ничего по-настоящему бесплатного, пропускную способность и время ожидания не меряют одной меркой (т. к. по сути это разные вещи), ресурсы не должны быть перегружены, а эксперименты следует проводить внимательно.

Принцип 0: следует хорошо понимать свою операционную среду

При недостаточно хорошем понимании своей операционной среды возникающие задачи, скорее всего, не решить. Вот почему концептуальному материалу в этом руководстве придается большое значение. Даже если изменятся отдельные детали реализации алгоритма или же сменится сам алгоритм, теоретические знания о том, что представляет собой задача и как подходить к ее решению, останутся действенными. Понимать задачу – значит обладать гораздо большим потенциалом по сравнению с ситуацией, когда известно решение, но нет понимания того, откуда возникла проблема.

Конечно, в какой-то мере сложные случаи лежат вне компетенции среднего системного администратора: настройка производительности

¹ «MVS Performance Management», GG22-9351-00.

сети на обсуждаемом здесь уровне не требует углубленных знаний того, как реализован стек TCP/IP на уровне модулей и вызовов функций. Заинтересованным в деталях работы различных вариантов операционной системы UNIX следует обратиться к нескольким замечательным руководствам: «Solaris Kernel Internals» (Внутренности ядра Solaris) Ричарда Мак-Дугалла (Richard McDougall) и Джеймса Мауро (James Mauro), «The Design of the UNIX Operating System» (Реализация операционной системы UNIX) Мауриса Баха (Maurice J. Bach) и «Operating Systems, Design and Implementation» (Операционные системы: проектирование и реализация) Эндрю Таненбаума (Andrew Tanenbaum) и Эндрю Вудфулла (Andrew Woodfull). Все книги изданы Prentice Hall.

Конечно, важнейшая ссылка – это сам исходный код. На время написания книги исходный код для всех детально описанных в этой книге операционных систем (Solaris и Linux) является бесплатным и доступным.

Принцип 1: БСНБ!

БСНБ означает, что Бесплатного Сыра Не Бывает.¹ По существу, настройка производительности представляет собой нахождение компромисса между различными характеристиками. Обычно это список из трех желательных свойств, из которых можно выбрать только два.²

Один пример приходит из настройки сетевого уровня TCP, где алгоритм Нагла (Nagle) приносит в жертву время ожидания или время, требуемое для доставки одиночного пакета, в обмен на повышение пропускной способности, или объем данных, которые можно фактически протолкнуть по проводам. (Алгоритм Нагла обсуждается более детально в разделе «Алгоритм Нагла» главы 7.)

Этот принцип часто обязывает делать реальный, значимый и трудный выбор.

Принцип 2: пропускная способность против времени ожидания

Во многих отношениях системные администраторы, оценивающие компьютерные системы, зачастую подобны юнцам, оценивающим автомобили. К сожалению, в обоих случаях существует определенный набор показателей и стремление найти наивысшее значение для наиболее «важного» показателя. Обычно это «величина пропускной способности» для компьютеров и «лошадиная сила» для автомобилей.

¹ Наши извинения Роберту Хайнлайну (Robert A. Heinlein) «The Moon Is A Harsh Mistress» (Луна – суровая хозяйка).

² Мы выполняем ваш заказ быстро, качественно и недорого. Выбирайте любые два пункта. – *Примеч. науч. ред.*

Шаги, которые готовы делать люди, чтобы выжать максимум лошадиных сил из своих четырехколесных транспортных средств, нередко отчасти смехотворны. Незначительное изменение ракурса обнаруживает, что в жемчужине производительности существуют и другие грани. Много усилий тратится на оптимизацию отдельных параметров, что, вполне вероятно, не является настоящим подспорьем. Проиллюстрируем это примитивным сопоставлением (пожалуйста, помните, что речь идет только о сравнении производительности). Транспорт А дает около 250 лошадиных сил, в то время как мощность транспорта В – около 95. Кто-то готов предположить, что транспорт А на практике демонстрирует существенно «лучшую» производительность, нежели транспорт В. Проницательный читатель может спросить: «А какой вес у этих транспортных средств?» Вес транспорта А составляет около 3600 фунтов, в то время как транспорт В весит около 450. Теперь видно, что транспорт В на самом деле значительно быстрее (разгон с нуля до 100 км в час примерно за три с половиной секунды против неторопливого транспорта А с пятью с половиной секундами). Однако если сравнивать настоящую скорость движения по переполненной 101-й магистрали на полуострове Сан-Франциско, транспорт В победит, и даже с большим перевесом, потому что мотоциклам в Калифорнии позволено ездить между полосами движения (между рядами машин).¹

Возможно, в этом мире компромиссов более всего пренебрегают временем ожидания. К примеру, представим вымышленное приложение – почтовый сервер. Назовем его СуперПуперПочта. Рекламные материалы этого приложения обещают, что сервер СуперПуперПочта способен обрабатывать свыше миллиона писем в час. Это может показаться вполне рациональным: такая скорость значительно выше скорости, требуемой большинством компаний. Другими словами, пропускная способность в целом хорошая. А что если взглянуть на производительность этого почтового сервера с другой стороны и спросить, как долго обрабатывается одно письмо? После нескольких наводящих вопросов к отделу сбыта СуперПуперПочты выясняется, что полчаса. Это выглядит противоречиво: как будто программа может обрабатывать самое большее два сообщения в час. Однако оказывается, что изнутри сервер СуперПуперПочта основан на последовательности приемников и перемещает письма к следующему приемнику, когда текущий приемник полностью заполнен. В этом примере, несмотря на приемлемую пропускную способность, время ожидания *ужасно*. Кто захочет послать письмо человеку, сидящему в соседней комнате, если доставка займет полчаса?

¹ Для любознательных. Транспорт А – Audi S4 седан (2,7 литра с двойным турбонаддувом V6). Транспорт В – мотоцикл Honda VFR800FI Interceptor (четырёхклапанный атмосферный двигатель 781 см³). Оба вида транспорта – модели 2000 года. – *Примеч. науч. ред.*

Принцип 3: не перегружайте ресурсы

Каждому, кто управлял автомобилем на загроможденной магистрали, знакома ситуация с ориентированием в незнакомой местности: знак «нижний предел скорости» выглядит злой шуткой! Очевидно, существует много факторов, относящихся к развитию сообщения между штатами, особенно один, тяжело воспринимаемый пригородными участниками движения: пиковая нагрузка существенно отличается от средней, с финансированием дорог всегда проблема и т. д. Более того, добавление еще одной полосы к магистрали (предположим, что место и финансирование это позволяют) обычно приводит к временному закрытию по крайней мере одной активной полосы шоссе. Это неизменно расстраивает «пригородных» еще больше. Стремление достичь «достаточной» пропускной способности всегда присутствует, но из-за препятствий с его реализацией цель достигается не сразу.

Теоретически подход расти «вширь» предпочтителен, когда обвал случился или неотвратим. Обычно это разумный вариант: зачем заботиться о дополнительной пропускной способности, если существующая используется не в полной мере? К сожалению, бывают случаи, когда полная загрузка не оптимальна. Это справедливо для компьютеров, и все же люди часто нагружают свои системы на 100% еще до размышлений о модернизации.

Перегрузка – опасное дело. Общее полезное правило гласит: нагрузка не должна превышать 70% от максимальной в любой момент времени. Это дает запас прочности перед снижением производительности.

Принцип 4: при проведении экспериментов необходима внимательность

Объясним этот принцип на простом примере передачи файла размером 130 Мбайт в сети Gigabit Ethernet по протоколу *ftp*.

```
% pwd
/home/jqpublic
% ls -l bigfile
-rw----- 1 jqpublic staff 134217728 Jul 10 20:18 bigfile
% ftp franklin
Connected to franklin.
220 franklin FTP server (SunOS 5.8) ready.
Name (franklin:jqpublic): jqpublic
331 Password required for jqpublic.
Password: <secret>
230 User jqpublic logged in.
ftp> bin
200 Type set to I.
ftp> prompt
Interactive mode off.
ftp> mput bigfile
```

```
200 PORT command successful.  
150 Binary data connection for bigfile (192.168.254.2,34788).  
226 Transfer complete.  
local: bigfile remote: bigfile  
134217728 bytes sent in 13 seconds (9781.08 Kbytes/s)  
ftp> bye  
221 Goodbye.
```

Читатели, обратившие внимание на производительность, вероятно, будут очень расстроены: ведь можно было ожидать скорость передачи порядка 120 Мбит/с! Между тем, достигнута лишь *десятая* часть этой скорости. Что же, в конце концов, случилось? Существует ли параметр, который не был должным образом установлен? Неисправна сетевая карта? Можно потратить много времени в поисках ответа на эти вопросы. Истина же состоит в том, что в рассмотрение принималась скорость, с которой можно считывать данные из */home*,¹ или скорость, с которой удаленный хост мог бы принимать данные.² Сетевой уровень здесь не является узким местом. Это что-то совсем иное: CPU, диски, операционная система и т. д.



Большое внимание в этой книге уделяется объяснению, *как* и *почему* работают системы, – чтобы в ходе проводимых экспериментов оценивалось именно то, что планировалось оценить.

Существует масса слухов, касающихся анализа производительности. Единственная возможность быть объективным – это понимать суть дела, проводить тесты и накапливать данные.

Вывод из этого примера таков: думать, думать и думать при экспериментах по оценке производительности. При измерениях много чего значимого не лежит на поверхности. А потому следует измерять все, имеющее малейшее отношение к тому, что на самом деле необходимо измерить. Если необходимо протестировать что-либо, важно подобрать соответствующие инструменты, специально предназначенные для диагностики этого компонента. И даже в этом случае нужно быть внимательным: очень легко обжечься.

Настройка статической производительности

В значительной степени эта книга посвящена настройке производительности систем в динамическом режиме. Внимание почти полностью отдано производительности системы, работающей в напряженных

¹ Речь идет об измерении скорости, которое автоматически выполняется программой *ftp* при передаче данных. – *Примеч. науч. ред.*

² Что не обязательно равнозначно «с какой скоростью эти данные могут быть записаны на диск», хотя такое и *может* быть.

условиях. В то же время существует и другой подход к настройке производительности, основанный на *статических* (то есть не зависящих от нагрузки) факторах. Влияние таких факторов способно снизить производительность системы вне зависимости от нагрузки. И не всегда это связано с проблемой соперничества из-за ресурсов.

Безусловно, наиболее причастными к статической производительности можно считать службы имен – средства, с помощью которых извлекается информация об объекте. Примеры служб имен: NIS+, LDAP и DNS.

Симптомы часто бывают расплывчаты: медленный вход в систему, застывшие окна веб-браузера, блокировка нового окна при начальной загрузке или зависший экран регистрации X или CDE. Вот несколько пунктов, расположенных в примерном порядке вероятности, на которые следует обратить внимание:

/etc/nsswitch.conf

Этот файл читается один раз для каждого процесса, использующего службу имен, поэтому для того, чтобы изменения возымели эффект, может понадобиться перезагрузка.

/etc/resolv.conf

Правильно ли указаны сервер имен и домен? Неправильные или чересчур длинные (со многими подкомпонентами) определения доменов могут быть причиной того, что DNS будет выдавать множество запросов. Серверы имен должны быть отсортированы по времени задержки с ответом на запросы.

Запущен ли демон кэширования для службы имен (name service cache daemon, nscd)?

Этот процесс кэширует информацию от службы имен, существенно повышая производительность. Демон *nscd* является штатным для Solaris и доступен для Linux. Исторически *nscd* винили во многих грехах. Со временем от большинства из них избавились, и сейчас этот демон можно запускать. Одно исключение – режим поиска неисправностей, когда *nscd* может скрывать глубинные проблемы службы имен.

Другой возможный источник неприятностей – сетевая файловая система (Network File System, NFS). Важно не допускать вложенного монтирования. Например, пусть рабочая станция монтирует три каталога с трех различных серверов NFS:

```
alpha:/home
bravo:/home/projects
delta:/home/projects/system-performance-tuning-2nd-ed
```

Чтение файла */home/projects/system-performance-tuning-2nd-ed/README* потребует вовлечения всех трех серверов NFS и приведет к чрезмерному трафику. Иметь для всего отдельные точки монтирования было бы намного эффективнее.

Настройка параметров ядра

По ходу книги мы будем предлагать настроить конкретные переменные ядра. Никогда не выполняйте настройку ядра на рабочей системе. Воспроизведите рабочую ситуацию в контролируемых лабораторных условиях, где и проводите эксперименты по настройке. Даже если необходимо произвести изменения на рабочей системе, тщательно и *всесторонне* протестируйте сделанные изменения.

В Solaris параметры ядра можно настраивать «на лету» с помощью *adb* или устанавливать их при начальной загрузке. В качестве примера будем использовать *maxpgio* (более подробную информацию о *maxpgio* можно почерпнуть из раздела «Свободный список» главы 4). Прежде всего, нужно знать, какое ядро будет загружаться (64- или 32-разрядное). Это можно узнать, выполнив *isainfo -v*. Вот пример для 32-разрядной системы:

```
% isainfo -v
32-bit sparc applications
```

И для 64-разрядной системы:

```
% isainfo -v
64-bit sparcv9 applications
32-bit sparc applications
```

Чтобы проверить значение переменной ядра, запускайте *adb* в режиме анализа ядра (*-k*). Чтобы посмотреть значение переменной на 32-разрядной системе, наберите имя переменной, а после нее */D*; на 64-разрядной системе после имени переменной наберите */E*. Если после имени переменной указан неправильный ключ, выходные данные будут бессмысленными. Вот пример для 32-разрядной системы:

```
# adb -k
physmem 3bd6
maxpgio/D
maxpgio:
maxpgio: 40
```

И для 64-разрядной системы:

```
# adb -k
physmem 1f102
maxpgio/E
maxpgio:
maxpgio: 40
```

Если необходимо изменить значение переменной, это можно сделать сразу. Выйдите из *adb*, набрав *<Ctrl>+<D>*, и запустите *adb -kw*. Чтобы изменить переменную, необходимо снова указать соответствующий ключ: */W* для 32-разрядных систем и */Z* для 64-раз-

рядных. В конце строки следует набрать 0t и то десятичное значение, которое должно быть присвоено переменной.

Например, присвоим параметру `maxpgio` значение 100 на 32-разрядной системе:

```
# adb -kw
physmem 3bd6
maxpgio/W 0t100
maxpgio:      0x28      =      0x64
maxpgio/D
maxpgio:
maxpgio:      100
```

На 64-разрядной системе:

```
# adb -kw
physmem 1f102
maxpgio/Z 0t100
maxpgio:      28      =      64
maxpgio/E
maxpgio:
maxpgio:      100
```

Это изменение не сохранится после перезагрузки. Чтобы сохранить эту установку, внесем изменения в `/etc/system`:

```
*
* change kernel variable at boot-time: 08-11-2001 by gdm
set maxpgio=100
```

В Linux, в зависимости от того, была переменная выставлена в дереве `/proc` или нет, есть два пути: либо редактировать исходный код ядра и собирать ядро заново, либо изменять соответствующий файл в `/proc`. Редактировать файлы в `proc` можно напрямую – скажем, изменим в Linux 2.2 значение параметра `min_percent`, отвечающего за минимальный процент памяти системы, доступной для кэширования. Этот параметр можно найти в файле `/proc/sys/vm/buffermem`. Формат этого файла: `min_percent max_percent borrow_percent`. Значение `min_percent` можно изменить непосредственно:

```
# cat /proc/sys/vm/buffermem
2 10 60
# echo "5 10 60" > /proc/sys/vm/buffermem
# cat /proc/sys/vm/buffermem
5 10 60
```

Эти изменения не сохранятся после перезагрузки, но эти строки можно поместить в стартовый сценарий, чтобы выполнять настройку автоматически.

И снова – будьте внимательны и *всегда* тестируйте сделанные изменения!

Проверка всякой всячины

Одинаковые IP-адреса в сети, а также другие ошибки в конфигурировании интерфейсов хоста могут быть источниками неполадок. Необходимо обеспечить жесткий контроль над адресацией IP. Иногда повреждаются кабели и возникают ошибки. В этом случае следует оценить частоту ошибок в работе интерфейса с помощью команды *netstat -i*.¹

Иногда бывают неработоспособны процессоры. Это почти всегда вызывает аварийное завершение работы. Однако в мощных системах Sun (например, E3500-E6500 и E10000) система автоматически перезагрузится, попытается изолировать отказавший элемент и возобновит работу. В результате может показаться, что система перегружилась якобы самовольно, «исключив» несколько процессоров, отказ которых повлек перезагрузку. Хорошее правило – применять команду *psrinfo*, дабы убедиться, что после загрузки работают все процессоры. Причем неважно, по какой причине произошла перезагрузка.

В аппаратном обеспечении возникает немало осложнений, которые могут привести к проблемам статической производительности. Если есть подозрение на неработоспособность, обычно намного легче для психики просто заменить поврежденный элемент. Если это невозможно, готовьтесь к длительному и напряженному процессу поиска неисправностей. Почти всегда полезно держать перед собой белый лист бумаги, чтобы набросать варианты конфигурации и отметить, при каких из них система работает, а при каких нет.

Заключение

Решение любой сложной задачи, как правило, требует определенной подготовки и хорошего понимания базовых принципов. В этой главе обсуждалось, что представляет собой настройка производительности, а также основные идеи в разрезе архитектуры компьютера, важные вопросы 64-разрядной среды, приемы настройки статической производительности и некоторые фундаментальные принципы производительности. Хотя очень трудно говорить об эффективности, скажем, учебных классов в подготовке пилотов, есть надежда, что данная глава дала представление о действующих лицах дальнейшего повествования.

Несколько слов об упражнениях, особенно уместных при намерении прочесть книгу от корки до корки. Вернитесь назад и перечитайте пять принципов настройки производительности (см. раздел «Принципы настройки производительности» ранее в этой главе), отложите эту книгу на час и подумайте, о каких принципах идет речь и что из них

¹ Такие проблемы часто возникают с коаксиальным кабелем (10base2); их очень трудно выявить, и это угнетает. Необходимо приобрести кабель хорошего качества и тщательно его промаркировать.

следует. Это очень широкие, общие постулаты, выходящие за рамки того контекста, в котором они были поданы. Ответственность за их применение ложится на читателя – и это в некотором отношении самый трудный момент, – хотя надеемся, что остальная часть книги будет действенным путеводителем. Приступая к анализу задачи, задайте себе несколько вопросов. Понимаю ли я, что случилось? Если нет, то какие эксперименты необходимо провести, чтобы подтвердить свои догадки? Не ищу ли я или мои клиенты «бесплатный сыр»? На какие компромиссы я готов пойти, чтобы добиться желаемой производительности? Не перегружаю ли я ресурсы? Использую ли я в оценке производительности ту систему мер, которую разработал? Те ли это показатели, которые я имею в виду?

Это сложные вопросы, и неважно, насколько простыми они могут казаться. Это невинно выглядящие лазейки, ведущие во тьму, в запутанные лабиринты. Нет ничего необычного встретить восемь или десять экспертов по производительности, собравшихся обсудить, что же случилось с системой. Они выдвигают теории и проводят эксперименты, чтобы подтвердить свои догадки или опровергнуть. Пять приведенных принципов – это главное, что следует вынести из книги. Их свет не даст заблудиться.

2

- *Определение параметров рабочего процесса*
- *Регулирование рабочей нагрузки*
- *Оценка производительности*
- *Заключение*

Управление рабочим процессом

Чем точнее определена координата, тем менее точно в это мгновение известно количество движения, и наоборот.

Werner von Heisenberg, 1927

Тема «управление рабочим процессом» – скользкая. Ее можно толковать по-разному. В этой главе рассматриваются практические средства, призванные обеспечить нулевой принцип настройки производительности: понимание того, что представляет собой операционная среда. Это сердцевина настройки динамической производительности. Остальная часть книги только улучшает понимание возможных вариантов операционной среды.

Как уже упомянуто, в основном речь пойдет об анализе динамической производительности. Оцениваемая система меняется на наших глазах. В каком-то смысле это подобно наблюдению за прудом. Представим ручей, который вливается в этот пруд. Какое влияние он оказывает на жизнь в пруду? Что произойдет, когда бобры построят на нем свою плотину? Когда ребята обнаружат этот пруд и станут кидать в него камни? Или когда кто-нибудь развеет на его середине пепел давних любовных посланий?

Чем дальше, тем сложнее. Будем руководствоваться непоколебимым принципом физики. Принцип неопределенности Гейзенберга (Heisenberg) гласит: как бы мы ни были аккуратны при проведении измерения, мы всегда возмущаем систему, и часть информации остается вне поля нашего зрения. Однако эти возмущения можно свести к минимуму. По ходу этой главы будет рассмотрено, насколько существенны такие возмущения при оценке производительности.

Такое положение дел может ввести в уныние. Мало того, что эксперименты затрагивают нечто, постоянно изменяющееся под влиянием непонятных факторов, которые не поддаются контролю. Ко всему этому сами измерения, проводимые в системе, возбуждают в ней дополнительные изменения! Тогда стоит ли беспокоиться об измерениях? Ответ прост. Без тщательных, систематических измерений практически невозможно принимать обоснованные аналитические решения. Дело в том, что в этом случае трудно представить, почему существует проблема или что произошло. Всегда лучше что-либо знать о ситуации. Если не стремиться получить о ней четкое представление, то есть все шансы уподобиться администраторам системы культа Карго.¹ И сидеть за столами с деревянными клавиатурами и терминалами², ожидая чародеев, которые прилетят на блестящих серебряных птицах. И за несколько сотен долларов в час будут решать все проблемы. Это не самое лучшее решение для всех, кроме чародеев.

В этой главе управление рабочим процессом разбито на две части: определение параметров рабочего процесса и его регулирование. Такой порядок важен: регулирование, проводимое наобум, вносит настоящую неразбериху. Многие консультанты накопили целые состояния, разбирая завалы после таких действий. Не нужно падать духом, думая о сложности этой темы. Существует немало инструментов, помогающих понять, а затем и ограничить нагрузку, возлагаемую на систему. Многие из них просты в применении. Далее речь пойдет о программах оценки – принятых способах измерения производительности систем с нагрузкой разной величины и характера. Эту главу можно рассматривать как учебник по превращению дерева культа Карго в кремний двадцать первого века.

Определение параметров рабочего процесса

В некотором смысле эксперты по производительности – это светила в высших компьютерного мира.³ Каждый раз, когда какой-нибудь застенчивый проситель приходит к ним со своей бедой, гуру какое-то время

¹ Термин «культ Карго» (cargo cult) возвращает к событиям, происходившим после второй мировой войны. Аборигены с островов Тихого океана строили деревянные взлетно-посадочные полосы и макеты самолетов. Они надеялись, что к ним снова прилетят самолеты с грузами (cargo), которые приземлялись на островах во время войны. – *Примеч. перев.*

² Хотя часто хочется, чтобы пейджер и сотовый телефон были деревянными. Тогда они не смогут звонить.

³ Продолжим эту аналогию. Порою представляется, что эксперты по производительности баз данных обитают в космическом пространстве. Тому есть три серьезных доказательства. Говорить с ними до смешного дорого. Вы или они должны проделать большой путь для аудиенции. Они говорят на странном языке, который трудно понять.

неспешно копается в проблеме, а затем отправляет просителя, требуя большей информации. Определение параметров – это сбор максимально возможного объема данных о системе. Цель этого процесса – выявить модели и тенденции в системе. Такие модели жизненно важны, когда производительность стремительно падает. Можно собрать воедино все данные о нарушениях в работе системы и исходя из них определить, что послужило их причиной. Это весьма похоже на изучение кильватера проплывающего корабля – что это было за судно и каким курсом оно плывет.

Можно провести аналогию между управлением рабочим процессом и финансовыми транзакциями. Первый раз авторы прочли об этом в замечательной книге Адриана Кокрофта (Adrian Cockcroft) «Настройка и производительность Sun» (Sun Performance and Tuning), изданной Prentice Hall. Идея состоит в том, что управление рабочим процессом в компьютерных системах аналогично управлению ведомством с бюджетом, персоналом и прочим, которое выполняет какую-то задачу.

По существу, возможны три исхода:

1. Если не существует плана и эффективного контроля за персоналом, то сотрудники становятся неуправляемыми, захватывая столько бюджета, сколько могут, чтобы обеспечить своим собственным проектам хорошее финансирование. Некоторые сотрудники в этом отнюдь не преуспевают, в то время как другие берут на себя «расследование ситуации» в Мауи.¹ Проект в целом заканчивается полной неразберихой (так называемая «стартовая модель»).
2. Не в меру старательный управленческий персонал создает бюрократические нагромождения: как планировать, оценивать планирование и перепланировать. Эта бюрократическая прослойка съедает весь бюджет. Она мешает тем, кто делает реальную работу, требуя от них ежедневные сводки. Административные издержки практически не позволяют рационально расходовать средства. Работа заканчивается неразберихой, потому что к моменту, когда все должно быть закончено, работа только началась (так называемая «правительственная модель»).
3. В идеальном случае руководители хорошо обдумывают и контролируют финансирование. При этом аппетиты персонала ограничены, но каждый обладает достаточными средствами, чтобы выполнить свою работу. Бюрократические издержки сведены к минимуму, и сводки запрашиваются нечасто. Работа выполнена в срок и в рамках бюджета (так называемая «модель сказочной страны»).

Аналогия с управлением производительностью довольно близкая. Если хотите, роль бюджета играют компьютерные показатели: циклы процессора, скорость дискового ввода-вывода, пропускная способность сети и прочее. Необходимо управлять этими ресурсами, чтобы не ока-

¹ Один из Гавайских островов, фешенебельный курорт. – *Примеч. перев.*

заться в силках стартовой модели, где получение любых ресурсов сводится к их грабежу. Однако если управлять ими с излишним рвением, то это будет похоже на правительственную модель – когда никто ничего не делает, так как слишком много времени уходит на канцелярскую работу. Поэтому следует применять сбалансированный подход: модель сказочной страны. К сожалению, название модели вполне оправдано. Прийти к ней очень трудно, а придерживаться ненамного легче.

Первый шаг в управлении производительностью – наметить ориентиры, которыми будем руководствоваться. Следует выбрать конкретные цели и задать критерии производительности, по которым можно многократно оценивать работу системы. Кроме того, необходимо определить текущее состояние системы. Впоследствии эта информация даст возможность управлять ресурсами и распределять их, улучшая работу пользователей.

Как дисциплина, управление производительностью зародилось десятилетия назад в мире мэйнфреймов, где все потрясюще дорого. Воплощение решений о настройке, основанное на понимании реальной ситуации, обычно окупалось. Фактическое снижение себестоимости часто бывало достаточно большим. Сейчас, когда стоимость вычислительной техники сильно упала, становится сложнее обосновывать время и усилия, затрачиваемые на диагностику и настройку. Приходится опираться на неявные преимущества анализа: проведенная наобум модернизация не оправдывает себя в течение нескольких месяцев, в то время как внимательное изучение системы и ее рациональная модернизация способны существенно улучшить производительность. Прибыль, полученная за эти несколько месяцев благодаря увеличению производительности, может быть намного выше стоимости анализа.

Существуют три важных инструмента, которые можно применить для получения более глубокого представления о поведении системы. Это простые, довольно известные команды по оценке производительности, а также учет процессов и возможности автоматического сбора данных утилиты *sar*. В этой главе кратко коснемся анализа конфигурации сети.

Простые команды

Такие простые утилиты, как *iostat*, *vmstat* и *mpstat*, всем известны. Это базовые инструменты для анализа производительности, которые могут предоставить много полезной информации о том, что происходит в системе. Рассмотрим один из быстрых способов собрать данные о функционировании системы. Он заключается в установке интервала, скажем, в несколько минут, с которым эти утилиты будут запускаться, и перенаправлении их вывода в файл. Возникает затруднение – как отслеживать точное время каждого конкретного сбора данных. Такую заботу может взять на себя следующий сценарий Perl:

```
#!/usr/bin/perl
while (<>) { print localtime() . "": $_"; }
```

Вот результаты этого скрипта, запущенного в системе Linux:

Пример 2.1. *vmstat* в системе Linux

```
# vmstat 5 | chrononome.pl
Sat Jun 30 00:37:28 2001: procs      memory      swap      io      system      cpu
Sat Jun 30 00:37:28 2001: r  b  w  swpd  free  buff  cache  si  so  bi  bo  in  cs  us  sy  id
Sat Jun 30 00:37:28 2001: 1  0  0  5472 26680 8420 177908  0  0  10  37  63  37  5  6  14
Sat Jun 30 00:37:33 2001: 0  0  0  5472 26576 8420 177908  0  0  0  20  163  37  1  0  99
Sat Jun 30 00:37:38 2001: 0  1  0  5472 26540 8420 177916  0  0  0  12  148  41  1  1  98
Sat Jun 30 00:37:43 2001: 0  0  0  5472 26904 8420 177920  0  0  2   5  141  43  1  2  98
Sat Jun 30 00:37:48 2001: 0  0  0  5472 26904 8420 177920  0  0  0   1  129  39  2  1  98
Sat Jun 30 00:37:53 2001: 0  0  0  5472 26904 8420 177920  0  0  0   0  124  36  0  1  99
Sat Jun 30 00:37:58 2001: 0  0  0  5472 26904 8420 177920  0  0  0  10  143  35  1  0  99
```

Некоторые администраторы запускают эти команды с очень коротким интервалом с помощью записи в *crontab*. Это действенный подход, но, с точки зрения авторов, не всегда оправданный. Все зависит от того, что необходимо измерить. Если запустить *vmstat* с интервалом 1800 секунд, то можно ожидать, что каждая строка будет сообщать о средней активности виртуальной памяти за последние полчаса. Однако если запускать *vmstat* с интервалом в две секунды каждые полчаса не из *cron*, то будут получены срезы производительности на получасовых отметках. Полезны и те, и другие данные, но, скорее всего, данные с большим интервалом предпочтительнее, по крайней мере вначале. Если же в данных обнаружатся пробелы, всегда можно запустить еще одну копию этой утилиты с интервалом, позволяющим поймать «горы и долины» активности. Увеличение интервала сбора данных не позволяет зафиксировать долговременные крены в производительности. Как правило, приходится выполнять много сопоставлений на большом объеме данных.

Учет процессов

Учет процессов – это метод, с помощью которого система собирает информацию о каждом запущенном процессе. Эти данные говорят об использовании процессора, активности дискового ввода-вывода, потреблении памяти и других полезных «лакомых кусочках». Вероятно, наибольшая польза от учета процессов заключается в том, что он позволяет выявить наиболее типичную нагрузку, испытываемую системой.

Часть системных администраторов обходят стороной учет процессов из-за боязни высоких накладных расходов. Но сбор отчетных данных по сути не оказывает сильного воздействия на систему: ядро всегда собирает статистические данные, поэтому дополнительные усилия связаны лишь с записью 40 байт в протокол учета. В то же время скрипты, обрабатывающие подобные файлы протоколов, могут работать довольно долго. Поэтому лучше не запускать их в часы пик.

Включение учета процессов

Запуск учета процессов очень прост. В Solaris надо убедиться, что дополнительные пакеты, позволяющие вести учет, установлены. Это па-

кеты `SUNWaccu` и `SUNWaccr`. В Solaris 8 эти пакеты находятся на втором инсталляционном диске.

Первый шаг – создать символические ссылки на скрипты запуска и останова:

```
# ln /etc/init.d/acct /etc/rc0.d/K22acct
# ln /etc/init.d/acct /etc/rc2.d/S22acct
```

Затем можно перезагрузить систему или сразу начать учет, запустив `/etc/init.d/acct start`. Второй шаг – добавить в таблицу `crontab` записи о запуске пользователем `adm` команд для сбора отчетной информации. Пример 2.2 показывает, что нужно добавить.

Пример 2.2. Добавление строк в crontab

```
# min hour day month wkday command
0 * * * * /usr/lib/acct/ckpacct
30 2 * * * /usr/lib/acct/runacct 2> /var/adm/acct/nite/fd2log
30 9 * * 5 /usr/lib/acct/monacct
```

Программа `ckpacct` контролирует размер учетного файла `/usr/adm/racct/`. Команда `runacct` на основе этих данных формирует учетную информацию. Наконец, `monacct` готовит «фискальный» отчет для каждого пользователя. Эти отчеты хранятся в `/var/adm/acct`.

Просмотр учетных данных

Самым полезным инструментом для просмотра учетной информации является `acctcom`. Он показывает текущие учетные данные. Запускать его можно в нескольких режимах: опция `-a` даст усредненные данные для каждого запущенного процесса, `-t` предоставит разбивку «системное/пользовательское время» для каждого процесса, `-u user` покажет все процессы, исполняемые пользователем `user`, и `-C time` отобразит все процессы, потребовавшие более чем `time` секунд процессорного времени. У `acctcom` есть и другие полезные опции. Они могут отличаться от системы к системе, поэтому для более полной информации следует обратиться к своей документации (manual pages).

Системные учетные программы также порождают несколько файлов, которые можно посмотреть на досуге. Они создаются ежедневно в конце каждого учетного периода (указанного при запуске `monacct`).

Автоматизация работы sar

Время от времени в книге будет обсуждаться `sar` – средство для накопления данных о производительности. Кроме того, `sar` можно применять для автоматического сбора данных и их хранения с целью последующего просмотра.

Включение sar

Начать автоматический сбор данных с помощью *sar* достаточно просто. Необходимо раскомментировать соответствующие строки в */etc/init.d/perf* (а именно последние 13) и добавить в системный файл *crontab* поддержку для автоматической записи данных.

Изменения в системных записях *cron*, расположенных в */var/spool/crontabs/sys*, подразумевают принятие некоторых решений: когда необходимо записывать данные (с помощью команды *sa1*) и когда данные следует выдавать (с помощью команды *sa2*). Вот как эти записи выглядят по умолчанию:

```
# min hour day month wkday command
# 0 * * * 0-6 /usr/lib/sa/sa1
# 20,40 8-17 * * 1-5 /usr/lib/sa/sa1
# 5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 1200 -A
```

Первая запись означает, что *sar* будет сохранять данные каждый час семь дней в неделю. Вторая иницирует запись два раза в час, на 20-й и 40-й минутах с начала часа в период повышенной нагрузки (с 8 утра до 5 вечера, с понедельника по пятницу).

Третья запись более сложна. В 18.05 с понедельника по пятницу она иницирует выдачу данных, собранных между 8 часами утра (*-S time*) и одной минутой седьмого (*-e time*) с интервалом 1200 секунд, или 20 минут (*-I seconds*), а также выдает все данные (*-A* для *sar*).

Извлечение данных

Извлечь данные из записей *sar* необыкновенно просто. Нужно лишь запустить *sar*, указав с помощью соответствующих ключей необходимые данные, а также требуемый период времени. Ключи *-S starting-time* и *-e ending-time* применяются для временного интервала, а *-f /var/adm/sa/sadd (dd* — день месяца) для указания выбранного дня.

Виртуальный Адриан

Адриан Кокрофт (Adrian Cockcroft) — знаменитый инженер компании Sun, занимающийся вопросами анализа производительности. Вместе с Риком Петитом (Rich Pettit) он разработал набор программ для сбора и анализа данных в системах Solaris. Эти программы очень подробно описаны в их книге «Производительность и настройка Sun» («Sun Performance and Tuning»), изданной Prentice Hall. Хотя книга слегка устарела, она, безусловно, заслуживает прочтения. Сами программы представляют собой замечательный инструментарий. Этот пакет, называемый SE Toolkit, размещен по адресу <http://www.setoolkit.com/>.

Анализ конфигурации сети

Исторически¹ сети практически не ограничивали производительность. Поэтому значительная часть усилий разработчиков была направлена на изучение и улучшение других аспектов работы системы. Однако вследствие бурного роста приложений Интернета сетевой уровень все чаще и чаще становится ограничивающим фактором. В результате инструменты для анализа производительности сети не настолько совершенны, как средства для анализа других показателей работы. Как же в таком случае анализировать сетевой уровень операционной среды?

Вначале небольшое предупреждение: этот раздел предполагает некоторую осведомленность читателя в концепциях сети.

Начнем с рассмотрения трех основных типов трафика, который присутствует в сетях TCP/IP. Затем кратко обсудим некоторые параметры сетевого трафика, например какова средняя величина пакета. В заключение разберем средства, позволяющие определять типичный трафик в конкретной сети.

Краткие замечания по терминологии

Вот понятия сетевого трафика, которые будут применяться в дальнейшем обсуждении:

- В соединении TCP/IP *клиент* – это сторона, которая *открыла* соединение (инициирующая сторона). *Сервер* – это сторона, которая *приняла* соединение (принимающая сторона). Заметим, что это ровным счетом ничего не говорит о ролях машин вне самого факта соединения!
- *Входящий трафик* – это трафик от клиента к серверу, *исходящий трафик* – трафик в обратном направлении.
- *Пакет без полезной нагрузки* – обычно около 60 байт в пакете (напомним, что существуют различные заголовки – обычно Ethernet, IP и TCP). Эти пакеты имеют прямое отношение к выполнению некоторых действий, например, связанных с протоколом TCP.
- *Маленький пакет* – менее 400 байт в пакете (включая все заголовки).
- *Средний пакет* – пакет размером от 400 до 900 байт.
- *Большой пакет* – пакет размером от 900 до 1500 байт (1500 – для Ethernet).

¹ То есть до эпохи Сети.

Тип 1: запрос-ответ

Первый тип трафика – запрос-ответ. Классические примеры из практики: HTTP, протоколы доставки электронной почты (например, POP3) и исходящие транзакции SMTP (по доставке на удаленный хост). Рисунок 2.1 показывает трафик типа 1.



Рис. 2.1. Трафик типа 1

Трафик запрос-ответ характеризуется высокими скоростями соединения. Однако иногда можно наблюдать и довольно медленные соединения, особенно в современной веб-среде, где широко применяются пакеты сохранения соединения HTTP (keepalives). Входящие пакеты обычно маленькие; исходящие – средней величины, но обычно их немного (меньше двадцати).

Тип 1В: обратный запрос-ответ

Вариация трафика типа 1 – обратный запрос-ответ. Обычно его можно наблюдать во входящих соединениях SMTP. Интересно, что в противоположность трафику типа 1 входящая и исходящая роли здесь поменялись местами: клиент инициирует большую часть передачи данных. Рисунок 2.2 иллюстрирует трафик типа 1В.

Скорость соединения часто умеренная, но иногда носит импульсный характер. На начальном этапе соединения входящие пакеты обычно малы, но затем становятся большими. Исходящие пакеты – это маленькие пакеты или пакеты без полезной нагрузки.



Рис. 2.2. Трафик типа 1В

Тип 2: передача данных

Трафик типа 2 обычно представляет собой передачу больших объемов данных. Наиболее часто это можно встретить в трафике *ftp* и при передаче файлов во время резервного копирования по сети (рис. 2.3).

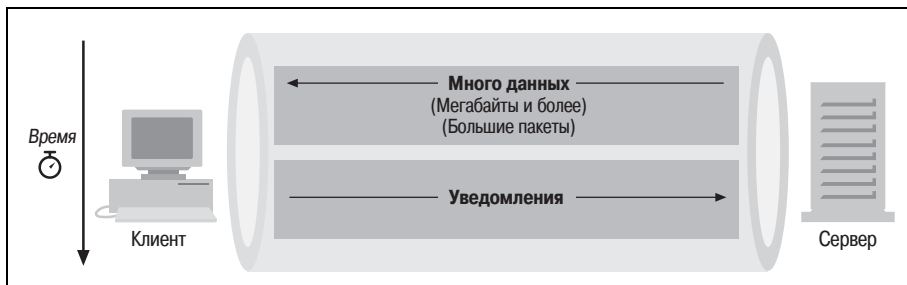


Рис. 2.3. Трафик типа 2

В основном скорость соединения очень низкая. По существу, поток входящих пакетов – это пакеты без полезной нагрузки, однако поток исходящих пакетов почти полностью состоит из полных пакетов.

Тип 3: передача сообщения

Третий и последний тип трафика – передача сообщения. Это наиболее типичный трафик для символьных приложений, таких как *telnet*, *rlogin* и *ssh*. Также его можно часто встретить в таких схемах транзакций баз данных, как *SQLnet*. Некоторые высокопроизводительные программы с параллельной обработкой данных также применяют такие передачи данных. Рисунок 2.4 иллюстрирует трафик типа 3.

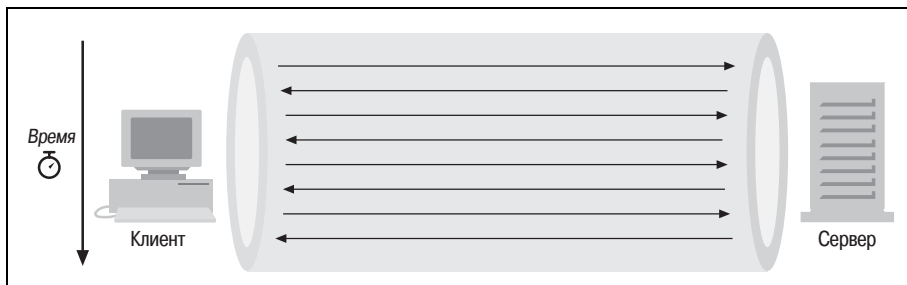


Рис. 2.4. Трафик типа 3

Скорость соединения различна. Обычно она мала, но может быть и достаточно высокой – это в значительной степени зависит от приложений. Для этого типа трафика характерно большое количество разносизмых маленьких пакетов. Для системы подобный трафик представляет большую нагрузку, так как усилия по обработке маленького пакета,

по существу, те же, что и по обработке большого: тот же объем работы, но полезных данных меньше.

Настройка систем с нагрузкой по типу 3 очень трудна. Если при наличии такой нагрузки есть уверенность, что сеть – слабое место в производительности, и производительность нужно повысить, то существуют два пути: либо изменить алгоритм работы приложения, чтобы с трафика типа 3 перейти на трафик типа 1, либо вложить деньги в сетевые аппаратные средства с очень коротким периодом ожидания.¹

Распределение размеров пакетов

Сбор данных о работе сети (например, с помощью *snoop* или *tcpdump*) может дать очень интересную информацию о том, какой трафик есть в сети. Один из возникающих простых вопросов таков: «Если построить график с осями «размер пакета» и «количество пакетов», то что можно будет увидеть?» Проанализировав системы в Интернете, можно заметить, что такой график – трехвершинный (имеет три отчетливых пика). Обычно это веб-системы или почтовые серверы.

- Первый пик в большей степени относится к входящему трафику и почти полностью состоит из пакетов размером около 60 байт. Это уведомления TCP, идущие от браузеров к веб-серверу и подтверждающие прием порции данных.
- Второй пик почти полностью состоит из исходящего трафика (пакеты размером 1540 байт). Это полные пакеты данных, идущие от сервера к браузерам.
- Третий пик замечаешь не сразу: он соответствует пакетам размером около 540 байт и относится к исходящему трафику. Такой трафик встречается в определенных Windows-реализациях TCP/IP, которые для соединения TCP устанавливают «размер максимального сегмента» 536 байт. В результате другая сторона не может передать клиенту любой пакет размером более 536 байт.

Если существует возможность влиять на трафик в сети, то такие данные могут послужить поводом к серьезным изменениям. Правильно настроенные алгоритмы приложений и стеки TCP – путь к устранению «средних пиков» на отметке 540 байт и соответствующему повышению эффективности.

Регулирование рабочей нагрузки

Следующий этап настройки системы – это внедрение регулирования рабочей нагрузки. В техническом плане этот этап часто прямолинеен, но в

¹ Myrinet имеет репутацию быстрого сетевого оборудования с коротким периодом ожидания. Такое оборудование не нашло широкого распространения вне специализированных рынков сбыта.