Red Hat Enterprise Linux 6

Оптимизация производительности

Оптимизация пропускной способности в Red Hat Enterprise Linux 6

Редакция 4.0

Last Updated: 2017-10-13

Red Hat Enterprise Linux 6 Оптимизация производительности

Оптимизация пропускной способности в Red Hat Enterprise Linux 6
Редакция 4.0

Red Hat. Отдел документации

Под редакцией

Don Domingo

Laura Bailey

Юридическое уведомление

Аннотация

В этом документе рассматриваются методы оптимизации производительности систем Red Hat Enterprise Linux 6. Несмотря на то что большинство приведенных здесь процедур было тщательно протестировано, Red Hat рекомендует провести проверку проектируемых конфигураций, прежде чем приступить к их внедрению в рабочее окружение. Дополнительно следует создать резервные копии исходных настроек и данных.

Содержание

ГЛАВА 1. ОБЗОР

В руководстве по оптимизации производительности рассматривается конфигурация и оптимизация Red Hat Enterprise Linux. Несмотря на то, что документ содержит информацию о Red Hat Enterprise Linux 5, в основном, инструкции рассчитаны на Red Hat Enterprise Linux 6.

Документ разбит на разделы, в которых обсуждаются отдельные подсистемы Red Hat Enterprise Linux. Для каждой подсистемы рассматривается следующее:

Функции

Описание подсистем начинается с обзора их функций в Red Hat Enterprise Linux 6, а также обновлений Red Hat Enterprise Linux 6, которые способствовали повышению их производительности.

Анализ

В секциях анализа обсуждаются основные характеристики производительности отдельных подсистем, что поможет оценить их значимость в реальных окружениях.

Дополнительно будут рассмотрены способы сбора статистики подсистем (что известно как профилирование). Инструменты профилирования могут подробно рассматриваться в других документах.

Конфигурация

Возможно, наиболее важная информация приведена в секциях, посвященных изменению конфигурации с целью повышения производительности подсистем Red Hat Enterprise Linux 6.

Оптимизация одной подсистемы может отрицательно сказаться на работе другой подсистемы. Стандартная конфигурация Red Hat Enterprise Linux 6 является оптимальной для *большинства* служб с *умеренной нагрузкой*.

Приведенные в этом документе процедуры были тщательно протестированы инженерами Red Hat в экспериментальных и рабочих окружениях. Тем не менее, рекомендуется провести тщательную проверку проектируемых конфигураций, прежде чем приступить к их развертыванию в рабочем окружении. Также следует создать резервные копии исходных настроек и данных.

1.1. ЦЕЛЕВАЯ АУДИТОРИЯ

Этот документ рассчитан на следующих специалистов:

Системные и бизнес-аналитики

Документ предоставляет общую информацию о производительности Red Hat Enterprise Linux 6 и отдельных подсистем до и после оптимизации. Такого уровня детализации достаточно, чтобы помочь потенциальным клиентам оценить пригодность платформы для работы в условиях интенсивной нагрузки.

Руководство содержит ссылки на специализированные документы, после ознакомления с которыми у читателя будет достаточно знаний для формирования высокоуровневой стратегии развертывания и оптимизации Red Hat Enterprise Linux 6. Это позволит не только разрабатывать новые схемы инфраструктур, но и выполнять анализ предлагаемых проектов.

Такая структура документа рассчитана на читателей с общими знаниями подсистем Linux и сетей уровня предприятия.

Системные администраторы

Материал рассчитан на специалистов уровня RHCE[1] и системных администраторов с аналогичным опытом работы с Linux (3-5 лет). Результаты каждой схемы конфигурации и их влияние на производительность будут рассматриваться более подробно.

Задача администратора заключается в подборе стратегии оптимизации в соответствии с требованиями нагрузки. Это означает необходимость понимания рисков и компромиссов, которые нужно будет учесть при выборе конфигурации, призванной повысить производительность той или иной подсистемы.

1.2. ГОРИЗОНТАЛЬНОЕ МАСШТАБИРОВАНИЕ

Основным критерием при улучшении производительности является возможность дальнейшего *масштабирования*. Функции производительности оцениваются с точки зрения их эффективности для разных объемов нагрузки — от уровня простого веб-сервера до крупной серверной фермы.

Масштабирование позволяет быстро адаптироваться в зависимости от степени нагрузки и корректировать инфраструктуру растущей организации.

Увеличение производительности Red Hat Enterprise Linux осуществляется за счет *горизонтального* и *вертикального масштабирования*. Горизонтальное масштабирование более распространено и заключается в распределении нагрузки между несколькими стандартными компьютерами.

Обычно компьютеры, входящие в состав серверной фермы, представляют собой компактные blade-серверы или серверы высотой 1U. Некоторые фермы допускают использование больших систем с множеством сокетов, а крупные сети уровня предприятия могут включать комбинации разных типов — большие системы выполняют роль высокопроизводительных серверов (серверов баз данных), а небольшие выступают в качестве выделенных программных серверов (почтовых и веб-серверов).

Горизонтальное масштабирование значительно упрощает управление растущей IT-инфраструктурой. Так, например, организации среднего размера изначально может быть достаточно двух серверов. С ростом бизнеса, объемов продаж и увеличением штата требования будут усложняться, но горизонтальное масштабирование позволит наращивать инфраструктуру посредством добавления компьютеров с аналогичной конфигурацией.

Таким образом, горизонтальное масштабирование добавляет дополнительный уровень абстракции, значительно облегчая управление оборудованием и наращивание инфраструктуры.

1.2.1. Параллельные вычисления

Горизонтальное масштабирование не только упрощает администрирование аппаратных ресурсов, но и является предпочтительной стратегией развития с учетом постоянных изменений на рынке оборудования.

Комплексные приложения уровня предприятия одновременно обрабатывают тысячи задач. Раньше это выполнялось компьютерами с одним ядром, в то время как на сегодняшний день практически все процессоры являются многоядерными. Поддержка нескольких ядер в одном сокете позволяет достичь многопроцессорности даже в обычных настольных компьютерах и ноутбуках.

Начиная с 2010 года стандартные процессоры Intel и AMD стали включать от 2 до 16 ядер. Эти процессоры популярны в плоских 1U и blade-серверах, которые теперь могут содержать до 40

ядер. Такие решения получили широкое распространение в силу экономичности и высокой производительности.

Оптимальная производительность достигается при максимальной занятости всех ядер. Это означает, что 32 ядра сервера должны обрабатывать 32 задачи. Если серверная стойка содержит 10 серверов, вся группа может одновременно обрабатывать минимум 320 задач. Если задачи являются составляющими одного большого задания, их обработку надо будет координировать.

Red Hat Enterprise Linux легко адаптируется к изменяющимся требованиям и обновлению оборудования. Раздел 1.3, «Распределенные системы» рассматривает технологии горизонтального масштабирования.

1.3. РАСПРЕДЕЛЕННЫЕ СИСТЕМЫ

Горизонтальное масштабирование Red Hat Enterprise Linux реализуется посредством *распределенных вычислений*. Технологии распределенных вычислений строятся на трех уровнях:

Взаимодействие

Горизонтальное масштабирование достигается за счет параллельного выполнения задач. Для их координации необходимо обеспечить возможность взаимодействия между процессами и системами.

Хранение данных

Размещения данных на локальных дисках может быть недостаточно — для эффективного масштабирования надо добавить дополнительный уровень абстракции для распределенного пространства хранения, что позволит его наращивать, добавляя необходимое оборудование.

Управление

В распределенных схемах уровень управления является наиболее важным, так как именно на этом уровне осуществляется координация программных и аппаратных компонентов, их взаимодействия, размещения данных и использования общих ресурсов.

Далее каждый уровень будет рассмотрен подробно.

1.3.1. Взаимодействие

Этот уровень контролирует передачу данных и охватывает:

- оборудование,
- программное обеспечение.

Самый простой способ установки связи между системами состоит в использовании *общей памяти*. Ключевыми характеристиками общей памяти являются скорость полосы пропускания и низкая задержка при обработке запросов чтения и записи.

Ethernet

Ethernet является наиболее распространенным способом передачи данных между компьютерами. На сегодняшний день по умолчанию используется *Gigabit Ethernet* (GbE), и серверы обычно имеют 2-4 порта Gigabit Ethernet. Именно GbE чаще всего используется для организации

распределенных схем. Даже если Ethernet-карты поддерживают более высокую скорость передачи, GbE может по-прежнему использоваться для выделенного управляющего интерфейса.

10GbE

Стандарт 10GbE (10 Gigabit Ethernet) становится все более популярным при проектировании мощных серверов, так как его скорость в 10 раз превышает скорость GbE. Его основным преимуществом является обеспечение баланса между скоростью передачи и вычислений для многоядерных процессоров. 10GbE позволяет поддерживать оптимальный уровень производительности системы без ограничения скорости связи.

К сожалению, технология 10GbE является довольно дорогой. В то время как цена карт 10GbE падает, стоимость линий связи (особенно оптоволоконных) и сетевых коммутаторов остается высокой. Возможно, в будущем их стоимость снизится, но на сегодняшний день этот стандарт используется лишь для критических приложений и серверных структур.

Infiniband

Скорость Infiniband значительно выше по сравнению с 10GbE. Помимо соединений TCP/IP и UDP, используемых с Ethernet, Infiniband поддерживает взаимодействие с общей памятью, что позволяет использовать этот стандарт протоколами удаленного прямого доступа к памяти (RDMA, Remote Direct Memory Access).

RDMA позволяет передавать данные между системами напрямую без обмена TCP/IP и участия сокетов, тем самым исключая излишнюю задержку, что может оказаться критическим фактором для некоторых приложений.

Infiniband обычно используется для высокопроизводительных вычислений (HPTC, High Performance Technical Computing), где скорость работы и отсутствие задержек имеют критическое значение. Для достижения необходимого уровня производительности иногда имеет смысл инвестировать в Infiniband, а не в процессоры и память.

RoCCE

RoCCE (RDMA over Ethernet) объединяет соединения Infiniband с прямым доступом к памяти и инфраструктуру 10GbE. Рынок продукции 10GbE постоянно расширяется, поэтому вполне можно ожидать, что со временем RDMA и RoCCE будут использоваться более широко.

Все перечисленные стандарты полностью поддерживаются в Red Hat Enterprise Linux 6.

1.3.2. Хранение данных

Схемы распределенных вычислений используют разные экземпляры общего хранилища. Это возможно за счет следующего:

- Разные системы хранят данные в одном месте.
- Пространство хранения состоит из разных устройств хранения.

Типичным примером хранилища служит локальный диск, что идеально подходит, если все операции выполняются на одном узле. Но с ростом инфраструктуры и увеличением числа компьютеров управление дисками значительно усложняется.

Распределенное хранение данных упрощает и автоматизирует администрирование накопителей при росте инфраструктуры. Совместное использование такого пространства хранения множеством систем также существенно облегчает работу администратора.

Объединение наборов накопителей в один том выгодно и администраторам, и пользователям. Это добавляет новый уровень абстракции — пользователи видят единственный ресурс, размер которого администратор может с легкостью нарастить, добавив дополнительное оборудование. Другие достоинства включают отказоустойчивость и многопутевые возможности.

NFS

NFS (Network File System) позволяет серверам и пользователям обращаться к файловым системам через TCP или UDP. Эта файловая система используется для хранения данных, к которым обращаются разные приложения, и подходит для хранения больших объемов информации.

SAN

Сети хранения данных (SAN, *Storage Area Network*) используют протоколы Fibre Channel и iSCSI для удаленного доступа к данным. Инфраструктура Fibre Channel объединяет в себе адаптеры шин, коммутаторы, массивы данных и характеризуется высокой скоростью работы и пропускной способностью. Пространство данных SAN отделено от сети, что обеспечивает определенный уровень гибкости при дизайне системной структуры.

Главным достоинством SAN является возможность создания окружения для администрирования пространства хранения данных:

- управление доступом к пространству хранения данных;
- управление большими объемами данных;
- подготовка систем;
- резервное копирование и репликация данных;
- создание снимков;
- восстановление систем;
- обеспечение целостности данных;
- миграция данных.

GFS2

Файловая система GFS2 (*Global File System 2*) допускает одновременное выполнение чтения и записи данных, расположенных на разных узлах в кластере. Таким образом, узлы смогут обращаться к одним и тем же данным, как будто они расположены на одном диске.

За поддержку целостности данных при одновременном выполнении операций чтения и записи отвечает механизм блокирования DLM (Distributed Lock Manager), разрешающий только одной системе выполнять запись в заданное место в определенный момент.

GFS2 идеально подходит для отработки отказа приложений, для которых высокая готовность накопителей является критически важным фактором.

За дальнейшей информацией обратитесь к *руководству по администрированию кластера* и *GFS2* по адресу http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

1.3.3. Конвергенция сетей

Наиболее распространенным способом взаимодействия компьютеров по сети является Ethernet,

в то время как для передачи данных используется выделенное окружение Fibre Channel SAN. Довольно часто управление системами также осуществляется из выделенных сетей или при помощи *пульсирующих сигналов*[2].

Создание множества каналов подключения к каждому серверу является дорогим и громоздким решением, именно поэтому и возникла необходимость объединения соединений, что реализуется при помощи протоколов *FCoE* и *Internet SCSI*.

FCoE

Протокол FCoE (Fibre Channel over Ethernet) использует физическую инфраструктуру 10GbE для передачи пакетов данных через единственную конвергентную карту (CNA, *converged network card*). Стандартный трафик TCP/IP и операции передачи данных тоже могут обрабатываться подобным образом. Конвергенция сетей позволяет использовать одну сетевую карту для обработки множества сетевых подключений.

Достоинства FCoE:

Сокращение числа соединений

FCoE сокращает число сетевых подключений в два раза. При необходимости можно использовать больше каналов связи, однако одного такого подключения обычно достаточно для обработки сетевого трафика и данных. Обычно такая схема подходит для компактных серверов в силу экономии места.

Экономичность

Сокращение числа подключений ведет к снижению количества используемых сетевых адаптеров, кабелей и сетевых коммутаторов, тем самым значительно снижая себестоимость сетевой инфраструктуры. Так как со временем цена оборудования тоже уменьшается, экономия растет в геометрической прогрессии.

С учетом того, что себестоимость технологии 10GbE также постепенно снижается, а конвергентные сетевые адаптеры интегрируются в одну микросхему, их доступность приводит к все более широкому использованию, что со временем приведет к снижению цен.

iSCSI

Протокол iSCSI (*Internet SCSI*) тоже используется для конвергенции сетей и предоставляет альтернативу FCoE. Несмотря на то что iSCSI не включает полный диапазон функций управления, его достоинствами являются гибкость и экономичность.

---

[1] Red Hat Certified Engineer (см. http://www.redhat.com/training/certifications/rhce/).

[2] *Пульс* — обмен сообщениями между системами с целью проверки их состояния. Если система не отвечает, это может служить знаком ее сбоя, после чего она может быть отключена, а ее функции переданы другой системе.

ГЛАВА 2. ОСОБЕННОСТИ ПРОИЗВОДИТЕЛЬНОСТИ RED HAT ENTERPRISE LINUX 6

2.1. 64-РАЗРЯДНЫЕ ПРОЦЕССОРЫ

Red Hat Enterprise Linux 6 поддерживает 64-разрядные процессоры, которые теоретически могут использовать до 16 *эксабайт* памяти. В Red Hat Enterprise Linux 6 также официально поддерживается до 8 терабайт физической памяти.

В будущих выпусках Red Hat Enterprise Linux 6 размер поддерживаемой памяти будет увеличиваться, так как Red Hat будет продолжать совершенствовать функции, позволяющие использовать блоки памяти больших размеров. Примеры такой оптимизации в Red Hat Enterprise Linux 6 включают:

- Прозрачные и очень большие страницы
- Улучшения неравномерного доступа к памяти

Далее они будут рассмотрены более подробно.

Прозрачные и очень большие страницы

Реализация *очень больших страниц* в Red Hat Enterprise Linux 6 увеличивает эффективность управления памятью независимо от уровня нагрузки. Размер больших страниц составляет 2 МБ (размер стандартной страницы равен 4 КБ), что облегчает масштабирование приложений, позволяя работать с гигабайтами и даже терабайтами памяти.

Управлять большими страницами вручную довольно сложно, поэтому Red Hat Enterprise 6 представляет концепцию *прозрачных страниц* для автоматизации задач управления.

Раздел 5.2, «Увеличение размера страниц» содержит подробную информацию.

Оптимизация NUMA

Современные системы поддерживают неравномерный доступ к памяти (NUMA, *Non-Uniform Memory Access*). NUMA значительно упрощает проектирование схемы оборудования для крупных систем, но в то же время усложняет разработку приложений. Представим ситуацию, где NUMA работает и с локальной, и с удаленной памятью. Так как обращение к удаленной памяти занимает намного дольше по сравнению с локальным доступом, это негативно скажется на производительности операционной системы, приложений, и поэтому конфигурацию системы потребуется откорректировать.

Red Hat Enterprise Linux 6 включает оптимизированные функции управления пользователями и приложениями в системах NUMA, включая привязку процессоров, объединение их в наборы cpuset, утилиту numactl, а также контрольные группы, позволяющие сопоставить приложения процессорам.

Раздел 4.1.1, «Топология NUMA» содержит дальнейшую информацию о NUMA в Red Hat Enterprise Linux 6.

2.2. СПИН-БЛОКИРОВКА

При проектировании систем следует помнить, что процессы не должны изменять память, используемую другими процессами. Такие изменения могут привести к искажению данных и сбою системы. Чтобы этого не случилось, операционная система блокирует участок памяти, выполняет операцию и снова его освобождает.

Распространенным решением является *спин-блокирование* — блокирование с ожиданием, то есть процесс периодически проверяет наличие доступа к разделяемому ресурсу и блокирует его, как только ресурс освобождается. Если к ресурсу обращается несколько процессов, доступ получит тот процесс, который первым запросил его состояние после освобождения.

Недостаток этого подхода состоит в том, что в системе NUMA не все процессы обладают равноправным доступом. Процессы, выполняемые на том же узле NUMA, где расположен разделяемый ресурс, имеют преимущество, в то время как удаленные процессы не смогут часто его использовать, и их производительность пострадает.

Red Hat Enterprise Linux решает эту задачу при помощи спин-блокировок по мере поступления запросов, добавляя процессы в очередь, тем самым обеспечивая равномерность обработки.

Несмотря на то что такой тип блокировки характеризуется более высокими издержками, он значительно облегчает масштабирование и увеличивает производительность в системах NUMA.

2.3. ДИНАМИЧЕСКАЯ СТРУКТУРА СПИСКОВ

Операционная система требует наличия определенного набора данных для каждого процессора. В Red Hat Enterprise Linux 5 для этого в памяти выделяется массив фиксированного размера. Обращение к массиву осуществляется посредством индексации, что идеально подходит для систем с небольшим числом процессоров в силу простоты и скорости.

Однако при увеличении числа процессоров доступ к фиксированному массиву будет усложняться, так как все больше процессоров будут к нему обращаться.

В Red Hat Enterprise Linux 6 используется *динамическая структура списков*, что позволяет динамически изменять размер массива. Так, для восьми процессоров список будет содержать 8 записей, а для 2048 процессоров — 2048.

Динамический размер списков обеспечивает более тонкий контроль блокирования. Например, в многопроцессорной схеме можно будет параллельно обновить данные для процессоров 6, 72, 183, 657, 931 и 1546.

2.4. БЕЗТАКТОВОЕ ЯДРО

Раньше ядро Linux периодически генерировало прерывание для опроса процессора. Полученные результаты использовались для распределения нагрузки.

Подобные прерывания (также известны как *такты процессора*) могут генерироваться сотни и тысячи раз в секунду независимо от нагрузки, поэтому даже бездействующий процессор отвечает на множество таких запросов. Таким образом, процессоры не могут эффективно использовать режим энергосбережения.

Максимальный уровень энергосбережения достигается при переходе системы в спящий режим. Чтобы предотвратить ненужные попытки пробуждения, ядро Red Hat Enterprise Linux 6 вместо периодических прерываний генерирует их при необходимости.

Безтактовое ядро позволяет поддерживать систему в спящем режиме и пробуждать ее только при получении заданий.

За дальнейшей информацией обратитесь к *руководству по управлению энергопотреблением* по адресу http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

2.5. КОНТРОЛЬНЫЕ ГРУППЫ

Red Hat Enterprise Linux предоставляет множество инструментов для оптимизации производительности растущих систем и адаптации к росту числа процессоров. Однако это требует наличия необходимых знаний и опыта. Раньше крупные системы были редкостью в силу их высокой стоимости, и это оправдывало индивидуальный подход. Постепенно они становились все более доступны, поэтому для их оптимизации стали требоваться более эффективные средства.

Задачу усложняет тот факт, что все чаще для консолидации служб используются мощные системы. Один сервер способен справляться с объемом нагрузки, для обработки которого раньше требовалось от 4 до 8 серверов. На сегодняшний день многие системы в среднем диапазоне намного мощнее ранних высокопроизводительных машин.

Большинство современных приложений изначально разрабатывается с учетом параллелизма. Однако немногие могут эффективно использовать больше восьми потоков, и их обычно устанавливают в 32-процессорных системах.

Производительность стандартных систем уже достигла уровня производительности вчерашних дорогих машин. Высокая производительность по низкой себестоимости позволяет осуществлять обработку большего числа служб на меньшем числе машин.

Некоторые типы общих ресурсов (такие как каналы ввода-вывода) не всегда быстро адаптируются к увеличению числа процессоров, что может привести к длительным периодам ожидания освобождения ресурсов и негативно скажется на производительности приложений.

Red Hat Enterprise Linux 6 решает эту проблему с помощью *контрольных групп*. Контрольные группы (cgroups) позволяют предоставлять ресурсы по мере необходимости. Так, например, администратор может выделить 80% ресурсов четырех процессоров, 40% дискового ввода-вывода и 60 ГБ памяти приложению базы данных. Веб-приложению, которое выполняется на том же узле, может быть выделено два процессора, 50% сетевого трафика и 2 ГБ памяти.

Быстродействие поддерживается за счет ограничения потребления ресурсов процессами. Дополнительное преимущество контрольных групп состоит в их способности автоматической коррекции в зависимости от нагрузки.

Компоненты контрольных групп:

- список задач,
- ресурсы.

Назначенные группе задачи будут выполняться в ее пределах. Их дочерние процессы также будут выполняться в пределах группы, что позволяет ими управлять как единым целым. Администратор может распределять следующие ресурсы:

- наборы процессоров;
- память;
- ввод-вывод;
- сетевые ресурсы.

Администратор может настроить число процессоров, сопоставить задачи процессорным узлам[3] и выделить процессорное время набору задач. Правильная настройка cpuset критична для достижения высокого уровня производительности и позволяет равномерно распределять вычислительные ресурсы между задачами.

Управление ресурсами ввода-вывода и передачи данных по сети осуществляется другими контроллерами.

Как уже говорилось, контрольные группы выделяют ресурсы приложениям, после чего система сможет автоматически распределять нагрузку между приложениями, поддерживая оптимальную работоспособность.

За дальнейшей информацией обратитесь к *руководству по управлению ресурсами* по адресу http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

2.6. ОПТИМИЗАЦИЯ ФАЙЛОВОЙ СИСТЕМЫ

Red Hat Enterprise Linux 6 включает улучшенные возможности хранения данных, включая поддержку ext4 и XFS (см. Глава 7, *Файловые системы*).

Ext4

Файловая система ext4 в Red Hat Enterprise Linux 6 используется по умолчанию. Ее основной особенностью является увеличение максимального размера раздела до 1 эксабайта и файла до 16 ТБ. В свою очередь, Red Hat Enterprise Linux 6 поддерживает файловые системы с максимальным размером 16 ТБ и файлы размером до 16 ТБ. Другие достоинства ext4:

- уменьшение размера метаданных за счет использования экстентов;
- отложенное выделение блоков;
- контрольные суммы журналов.

Раздел 7.3.1, «Ext4» содержит более подробную информацию о файловой системе ext4.

XFS

64-разрядная файловая система XFS включает функции журналирования и оптимально подходит для организации больших файлов и файловых систем на одном компьютере. XFS изначально была рассчитана для работы на больших серверах. Особенности XFS:

- отложенное выделение блоков;
- динамическое выделение узлов;
- индексация B-tree;
- онлайн-дефрагментация и увеличение размера файловой системы;
- комплексные алгоритмы предварительного чтения метаданных.

Несмотря на то что размер XFS может исчисляться эксабайтами, Red Hat поддерживает максимум 100 ТБ (см. Раздел 7.3.2, «XFS»).

Размер загрузочных дисков

Обычно BIOS поддерживает диски размером до 2.2 ТБ. Системы Red Hat Enterprise Linux 6, использующие BIOS, могут поддерживать диски большего размера за счет использования *глобальной таблицы разделов*. Исключение составляют загрузочные диски — их размер не может превышать 2.2 ТБ. Дело в том, что BIOS изначально создавалась для IBM и, несмотря на то что она постепенно развивалась, UEFI (*Unified Extensible Firmware Interface*) лучше приспособлен для поддержки нового оборудования.

Red Hat Enterprise Linux 6 поддерживает UEFI, который может использоваться вместо BIOS. Такая схема допускает работу с разделами размером больше 2.2 ТБ (включая загрузочные).

ВАЖНО

Red Hat Enterprise Linux 6 не поддерживает UEFI в 32-битных системах x86.

ВАЖНО

Конфигурация загрузки UEFI и BIOS может значительно отличаться, поэтому при загрузке установленной системы должны использоваться те же микропрограммы что и при установке. То есть нельзя выполнить установку в системе с BIOS и загрузить эту установку в системе с UEFI.

Red Hat Enterprise Linux 6 поддерживает UEFI 2.2. Если оборудование поддерживает UEFI 2.3, проблем с загрузкой Red Hat Enterprise Linux 6 не будет, но возможности новой версии будут недоступны. http://www.uefi.org/specs/agreement/ содержит подробную информацию о спецификациях UEFI.

[3] Узел — набор процессоров или ядер в сокете.

ГЛАВА 3. АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ

В этой главе рассматриваются утилиты мониторинга и анализа производительности системы и приложений, а также примеры их использования. Полученная с их помощью статистика поможет обнаружить слабые места в системе, снижающие производительность.

3.1. /PROC

`proc` предоставляет собой каталог, отражающий иерархическую структуру файлов состояния ядра Linux. Пользователи и программы могут обращаться к этому каталогу для получения информации о системе.

`proc` также содержит информацию об оборудовании и текущих процессах. Большинство файлов в этом каталоге доступно только для чтения, но некоторые (в **`/proc/sys`**) могут быть изменены.

За дальнейшей информацией обратитесь к *руководству по развертыванию* по адресу http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

3.2. СИСТЕМНЫЕ МОНИТОРЫ GNOME И KDE

GNOME и KDE предоставляют собственные программы для мониторинга производительности систем.

GNOME

Системный монитор GNOME возвращает информацию о системе и использовании ресурсов. Чтобы его открыть, выполните команду **`gnome-system-monitor`** или в главном меню системы выберите **Приложения** > **Системные** > **Системный монитор**.

Вкладки программы:

Система

Информация об операционной системе и оборудовании.

Процессы

Список активных процессов и их статус. Выбранные процессы могут быть остановлены, запущены, завершены, или можно изменить их приоритет.

Ресурсы

Диаграммы использования процессора, памяти, пространства подкачки и сети.

Файловые системы

Список файловых систем, включая их тип, каталог подключения, размер свободного и занятого пространства.

Подробную информацию о **системном мониторе GNOME** можно найти в его разделе справки или в *руководстве по развертыванию* по адресу http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

KDE

Системный монитор KDE позволяет наблюдать за нагрузкой системы и состоянием процессов. Чтобы его открыть, выполните команду **ksysguard** или в главном меню выберите **Приложения** > **Система** > **Системный монитор**.

Вкладки системного монитора KDE:

Таблица процессов

Список процессов в алфавитном порядке. Можно их отсортировать по другим критериям: по занятости процессора, памяти, владельцу или приоритету. Полученные результаты можно дополнительно отфильтровать.

Загрузка системы

Диаграмма использования ресурсов процессора, памяти, пространства подкачки и сети. Наведите курсор на интересующий график для получения подробной информации.

За более подробной информацией обратитесь к справке программы.

3.3. КОМАНДЫ МОНИТОРИНГА

Red Hat Enterprise Linux также включает текстовые утилиты мониторинга производительности. Их преимущество состоит в том, что они могут работать за пределами уровня выполнения 5.

top

top показывает изменения статистики системы и активности процессов в реальном времени. Формат результатов можно настроить с сохранением настроек после перезагрузки.

По умолчанию процессы упорядочены в зависимости от их нагрузки на процессор, что позволяет сразу увидеть, какие процессы потребляют больше всего ресурсов.

Подробную информацию о **top** можно найти на справочной странице **man top**.

ps

ps возвращает снимок состояния работающих процессов.

ps возвращает более подробную статистику чем **top**, но не обновляет ее динамически.

Подробную информацию можно найти на справочной странице **man ps**.

vmstat

vmstat позволяет получить подробную информацию об активности процессов, памяти, процессора, ввода-вывода устройств и пр.

Получаемые с помощью **vmstat** отчеты тоже изменяются динамически, но не так часто как **top**.

Подробную информацию можно найти на справочной странице **man vmstat**.

sar

sar возвращает информацию об активности системы на текущий день. Стандартный формат включает статистику использования процессора через каждые 10 минут.

```
12:00:01 AM     CPU     %user     %nice   %system   %iowait    %steal
%idle
```

```
12:10:01 AM     all      0.10      0.00      0.15      2.96      0.00
96.79
12:20:01 AM     all      0.09      0.00      0.13      3.16      0.00
96.61
12:30:01 AM     all      0.09      0.00      0.14      2.11      0.00
97.66
...
```

sar подходит для генерации отчетов с заданным интервалом.

Подробную информацию можно найти на справочной странице **man sar**.

3.4. TUNED И KTUNE

tuned отслеживает занятость системных компонентов и динамически изменяет настройки системы исходя из полученной информации. Так, система сможет своевременно реагировать на изменение нагрузки, увеличивая производительность активных устройств и снижая энергопотребление неактивных.

ktune в комбинации с **tuned-adm** предоставляет целый набор профилей для поддержки производительности и уменьшения энергопотребления. Их можно использовать готовыми или адаптировать исходя из требований окружения.

Профили **tuned-adm**:

default

Стандартный профиль энергосбережения. Включает модули для процессора и дисков.

latency-performance

Профиль для корректирования задержки ответа сервера. Отключает механизмы **tuned** и **ktune**, изменяет режим **cpuspeed** на **performance**. Для контроля дискового ввода-вывода используется планировщик **deadline**. В качестве обязательного значения **cpu_dma_latency** регистрируется **0**.

throughput-performance

Профиль для коррекции производительности обработки данных. Рекомендуется при отсутствии доступа к большому хранилищу данных. Эквивалентно **latency-performance** за исключением:

- Минимальный интервал ***kernel.sched_min_granularity_ns*** равен **10** миллисекундам.
- Значение ***kernel.sched_wakeup_granularity_ns*** (предварительная активация) равно **15** миллисекундам.
- Значение ***vm.dirty_ratio*** равно 40%.
- Включены возможности прозрачного использования страниц большого размера.

enterprise-storage

Этот профиль рекомендуется для конфигурации пространства данных для крупных серверов уровня предприятия, что включает защиту кэша за счет переключения на питание от батареи и управление кэшем на диске. Аналогичен профилю **throughput-performance** за

исключением того, что повторное монтирование систем осуществляется с параметром **barrier=0**.

virtual-guest

Этот профиль рекомендуется для конфигурации пространства данных для крупных серверов уровня предприятия, что включает защиту кэша за счет переключения на питание от батареи и управление кэшем на диске. Аналогичен профилю **throughput-performance** за исключением следующих факторов:

- значение ***readahead*** равно **4x**,
- файловые системы, которые не являются корневыми и загрузочными, монтируются с параметром **barrier=0**.

virtual-host

virtual-host основан на профиле ***enterprise-storage***. Он уменьшает объем подкачки виртуальной памяти и допускает более агрессивный подход к записи «грязных» страниц. Этот профиль доступен в Red Hat Enterprise Linux 6.3 и рекомендуется для хостов виртуализации, включая KVM и Red Hat Enterprise Virtualization.

За дальнейшей информацией обратитесь к *руководству по управлению энергопотреблением* по адресу http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

3.5. ПРОФИЛИРОВАНИЕ ПРИЛОЖЕНИЙ

Профилирование — сбор информации о выполнении программы с целью определения способов оптимизации ее быстродействия, уменьшения использования памяти и т.п. Средства профилирования приложений упрощают этот процесс.

Red Hat Enterprise Linux 6 предоставляет три инструмента профилирования: **SystemTap**, **OProfile**, **Valgrind**. Их детальное изучение выходит за рамки этого руководства, поэтому далее приведена лишь обзорная информация.

3.5.1. SystemTap

SystemTap предназначен для мониторинга операционной системы и ядра. Полученная статистика аналогична выводу **netstat**, **ps**, **top**, **iostat**, но SystemTap дополнительно предоставляет функции для анализа и фильтрации полученных данных.

SystemTap позволяет точно определить слабые места в работе системы и приложений.

На основе SystemTap построен дополнительный модуль Function Callgraph для Eclipse, отвечающий за сбор статистики о состоянии программ, включая сведения о вызовах функций, их результатах, времени выполнения, переменных пространства пользователя.

За дальнейшей информацией обратитесь к документу под названием *Введение в SystemTap* по адресу http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

3.5.2. OProfile

oprofile представляет собой утилиту мониторинга производительности систем Linux и использует собственные функции процессора для сбора статистики ядра и системы, что включает обращение к памяти, число запросов к кэшу L2 и полученных прерываний. OProfile

может определить степень занятости процессора и идентифицировать процессы, которые чаще всего к нему обращаются.

OProfile может использоваться в Eclipse в виде дополнительного модуля Eclipse OProfile, что позволяет с легкостью определить наиболее ресурсоемкие фрагменты кода и выполнять функции OProfile из командной строки.

Ограничения OProfile включают:

- Точность статистики не является стопроцентной, так анализ мог быть начат не с инструкции, вызвавшей прерывание, а с соседней.
- OProfile допускает, что процессы будут запускаться и останавливаться множество раз, и накапливает полученные результаты. При необходимости их можно очистить.
- OProfile идентифицирует процессы, которым не хватает процессорных ресурсов, но не определяет бездействующие процессы, ожидающие своей очереди.

За дальнейшей информацией обратитесь к *руководству развертыванию* на сайте http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/ и к документации **oprofile** в **`/usr/share/doc/oprofile-<версия>`**.

3.5.3. Valgrind

Valgrind предоставляет инструменты профилирования и коррекции производительности приложений. С их помощью Valgrind идентифицирует ошибки памяти и потоков, а также позволяет своевременно обнаружить переполнение стека и архивов. Дополнительно можно создать профиль кэша и памяти кучи с целью определения факторов, которые помогут улучшить скорость работы приложения и уменьшить использование памяти.

Valgrind анализирует работу приложения, выполняя его не на основном процессоре, а на его симуляторе, точно определяет задействованные процессы и отправляет собранную статистику в заданный файл или сетевой сокет. Время работы Valgrind зависит от используемых инструментов, но обычно занимает в 4-50 раз дольше по сравнению с выполнением кода напрямую.

Valgrind может выполняться и без перекомпиляции приложения, но при компиляции можно включить отладочную информацию, что значительно облегчит поиск проблем в коде.

В Red Hat Enterprise Linux 6.4 для повышения эффективности отладки в Valgrind был добавлен отладчик gdb.

За дальнейшей информацией о Valgrind обратитесь к *руководству по развертыванию* по адресу http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/ или к справочной странице **`man valgrind`**. Дополнительная документация:

- **`/usr/share/doc/valgrind-<версия>/valgrind_manual.pdf`**
- **`/usr/share/doc/valgrind-<версия>/html/index.html`**

Раздел 5.3, «Профилирование памяти при помощи Valgrind» содержит дальнейшую информацию.

3.5.4. Perf

perf предоставляет набор счетчиков, с помощью которых пользователь может оценить результаты выполнения команд в системе.

`perf stat`

Сбор стандартной статистики системных событий, включая сведения об инструкциях и временных циклах. Флаги помогут получить информацию о дополнительных событиях. В Red Hat Enterprise Linux 6.4 команда **`perf stat`** стала работать с контрольными группами. Подробную информацию можно найти на справочной странице **`man perf-stat`**.

`perf record`

Осуществляет сбор данных и сохраняет их в файл, анализ которого может быть выполнен с помощью **`perf report`**. Подробную информацию можно найти на справочной странице **`man perf-record`**.

`perf report`

Получает сведения о производительности из файла и формирует отчет. Подробную информацию можно найти на справочной странице **`man perf-report`**.

`perf list`

Возвращает список доступных событий. События могут отличаться в зависимости от оборудования мониторинга и конфигурации системы. Подробную информацию можно найти на справочной странице **`man perf-list`**.

`perf top`

Осуществляет сбор статистики в реальном времени аналогично **top**. Подробную информацию можно найти на справочной странице **`man perf-top`**.

За дальнейшей информацией обратитесь к *руководству по управлению ресурсами* по адресу http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

3.6. RED HAT ENTERPRISE MRG

Red Hat Enterprise MRG Realtime включает инструмент **Tuna** для изменения настраиваемых параметров системы. **Tuna** изначально разрабатывался для работы в окружении Realtime, но может использоваться и для настройки стандартных систем Red Hat Enterprise Linux.

Tuna управляет следующими характеристиками:

- настройки питания, прерываний и обнаружения ошибок в BIOS;
- сетевые параметры, включая использование TCP и объединение прерываний;
- ведение журналов в файловых системах с журналированием;
- системное журналирование;
- определяет процессоры, которые будут обрабатывать прерывания и пользовательские процессы;
- использование пространства подкачки;
- обработка исключений нехватки памяти.

За дальнейшей информацией обратитесь к *руководству пользователя Tuna* и *руководству по настройке Realtime* по адресу http://access.redhat.com/site/documentation/Red_Hat_Enterprise_MRG/.

ГЛАВА 4. ПРОЦЕССОР

Современные компьютеры могут быть оборудованы несколькими *процессорами*, которые представляют собой интегральные схемы, вставляемые в специальный разъем (*сокет*) материнской платы. *Сокет* соединяется с другими сокетами, контроллерами памяти и другими периферийными устройствами. С точки зрения операционной системы сокет логически объединяет процессоры и их ресурсы.

Red Hat Enterprise Linux включает множество инструментов для отслеживания активности процессора. Полученная информация помогает разработать стратегию оптимизации производительности (см. Раздел 4.1.2, «Коррекция производительности процессора»).

ТОПОЛОГИЯ

Раньше компьютеры были оборудованы меньшим числом процессоров, что допускало использование симметричной многопроцессорной схемы доступа к памяти (SMP, *Symmetric Multi-Processor*). Со временем число процессоров на сокет увеличивалось, и стоимость такой схемы значительно возросла. На сегодняшний день многопроцессорные системы используют неравномерный доступ к памяти (NUMA, *Non-Uniform Memory Access*).

Процессоры AMD уже раньше имели такую инфраструктуру за счет технологии HT (*Hyper Transport*), в то время как Intel начал внедрять возможности NUMA в схемы QPI (*Quick Path Interconnect*). Конфигурация SMP и NUMA отличается, так как при выделении ресурсов должна учитываться топология системы.

ПОТОКИ

В терминологии Linux поток — абстракция для обозначения передачи данных. Потоку выделяется сегмент исполняемого кода, который он может выполнять на процессоре. Планирование обработки потоков на свободных процессорах осуществляется на уровне операционной системы.

Операционная система распределяет нагрузку между процессорными ядрами, тем самым стараясь максимизировать занятость процессора, но это не означает, что производительность приложений также улучшится. Например, перенос потока на процессор в другой сокет вместо того, чтобы дождаться освобождения текущего процессора, только замедлит работу. Это следует учесть при проектировании высокопроизводительных приложений (см. Раздел 4.2, «Планирование занятости процессоров»).

ПРЕРЫВАНИЯ

Прерывания также могут оказывать влияние на производительность. Запросы прерываний обрабатываются операционной системой и обычно сообщают о наступлении событий, будь то получение данных или завершение операции записи в сеть.

В этой главе будет рассказано, как повысить эффективность обработки прерываний.

4.1. ТОПОЛОГИЯ ПРОЦЕССОРОВ

4.1.1. Топология NUMA

Изначально компьютеры были оборудованы одним процессором. Иллюзия параллельных вычислений создавалась на уровне операционной системы за счет переключения потоков. Однако увеличение скорости обработки растущего числа заданий не могло продолжаться бесконечно, поэтому впоследствии стали добавляться дополнительные процессоры, что позволило на самом деле реализовать возможности параллелизма.

Ранние многопроцессорные системы использовали параллельную шину для соединения процессора с памятью, выделяя одинаковые сегменты времени для доступа к памяти. Это носит название симметричной многопроцессорности и подходит для небольшого числа процессоров. Но при увеличении числа процессоров до 8-16 требования к полосе пропускания существенно возрастают, что оставляет меньше места для периферийных компонентов.

Увеличение числа процессоров возможно за счет объединения следующих компонентов:

1. Последовательные шины.

2. Топологии NUMA.

Последовательные шины характеризуются высокой скоростью передачи пакетов данных. Изначально они служили для организации высокоскоростного сообщения между процессорами, между процессорами и контроллерами памяти, и другими периферийными компонентами. Теперь одна шина может объединять 32-64 соединений, что существенно сокращает место на плате.

Со временем вместо подключения отдельных процессоров к плате разработчики стали объединять ядра процессоров в один модуль, а вместо предоставления равноправного доступа к памяти была разработана стратегия неравномерного доступа (NUMA, Non-Uniform Memory Access). В этой схеме реализации компьютерной памяти каждому сокету выделяется отдельный сегмент памяти для высокоскоростного доступа. Дополнительно сокеты соединены друг с другом, что позволяет им обращаться к памяти других сокетов.

В качестве простого примера рассмотрим плату с двумя сокетами, в каждый из которых подключен 4-ядерный процессорный модуль. Таким образом, общее число процессоров составляет 8. Каждый сокет соединен с модулем памяти размером 4 ГБ, то есть общий объем ОЗУ составляет 8 ГБ. Представим, что процессоры 0-3 размещены в сокете 0, а 4-7 в сокете 1. Дополнительно каждый сокет сопоставлен отдельному узлу NUMA.

Для доступа процессора с номером 0 к памяти из банка 0 требуется 3 такта: один такт для передачи адреса контроллеру памяти, второй для установки доступа к памяти, третий — для выполнения операции записи или чтения. В то же время для доступа процессора 4 потребуется 6 тактов, так как он расположен в другом сокете, что добавляет две операции — обращение к контроллеру локальной памяти в сокете 1 и обращение к контроллеру удаленной памяти в сокете 0. Если в это время к памяти обращается другой процессор, контроллеры должны будут установить порядок доступа, что, естественно, замедлит работу. Синхронизация кэша процессора с памятью еще больше усложняет задачу.

Последние модели процессоров Intel (Xeon) и AMD (Opteron) используют функции NUMA. Роль межпроцессорной шины для процессоров AMD выполняет HyperTransport (HT), а для Intel — QuickPath Interconnect (QPI).

Так как системные архитектуры могут значительно отличаться, точно предсказать снижение производительности вследствие обращения к памяти в других сокетах довольно сложно. Каждое межпроцессорное соединение увеличивает задержку. Так, если для доступа к памяти надо осуществить два перехода по шине, формула расчета времени доступа будет выглядеть так: *2N + время доступа к памяти* (где N — время задержки при подключении к шине).

Принимая это во внимание, высокопроизводительные приложения должны избегать доступа к удаленной памяти в схемах с топологией NUMA, предпочитая локальную память.

Для этого при проектировании приложений надо ответить на следующие вопросы:

1. Какова *топология* системы?

2. Где выполняется приложение?

3. Где размещен ближайший банк памяти?

4.1.2. Коррекция производительности процессора

В этой секции рассматриваются способы коррекции производительности процессора.

Изначально схема NUMA предназначалась для организации доступа одного процессора к разным банкам памяти. Позднее появились многоядерные процессоры, где ядра получают равные отрезки времени для доступа к локальной памяти и могут совместно использовать кэш. Недостаток такого подхода состоит в том, что каждое подключение к памяти, ядру и кэшу на другом сокете добавляет небольшую задержку.

Рисунок 4.1, «Локальный и удаленный доступ к памяти» демонстрирует два узла NUMA. Каждый узел включает 4 процессора, контроллер и банк памяти. Последовательность действий при обращении процессоров с узла 1 к памяти:

1. Процессор (ядро 0-3) передает адрес памяти локальному контроллеру.

2. Контроллер устанавливает доступ к памяти.

3. Процессор выполняет операции чтения и записи в область памяти с заданным адресом.

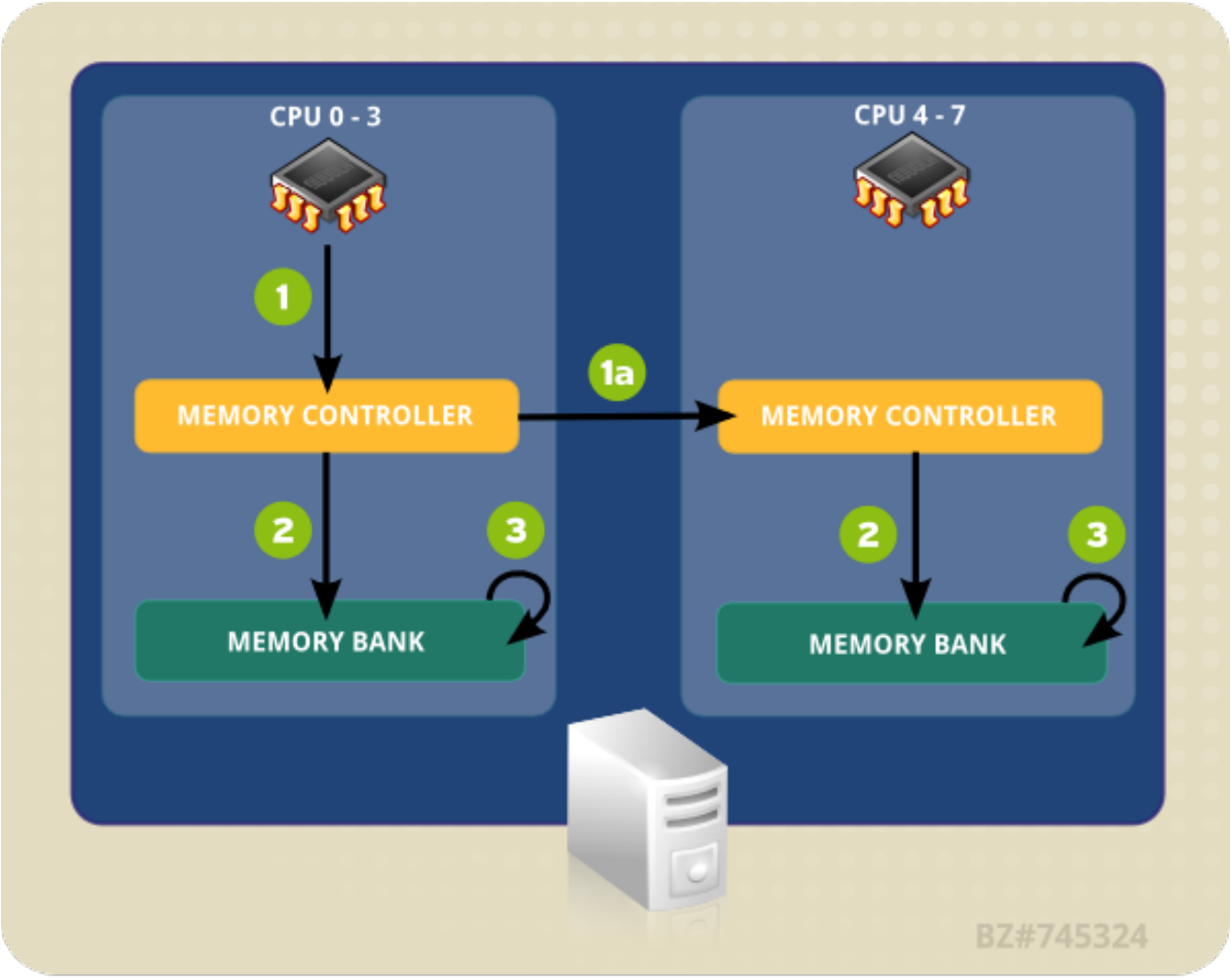


Рисунок 4.1. Локальный и удаленный доступ к памяти

Если процессор на одном узле обращается к банку памяти на другом узле, цикл выполнения будет таким:

1. Процессор (ядро 0-3) передает адрес памяти удаленному контроллеру.
    1. Удаленный контроллер памяти получает запрос доступа.
2. Контроллер устанавливает доступ к локальному банку памяти.
3. Процессор выполняет операции чтения и записи область памяти с заданным адресом.

Таким образом, обращение к удаленной памяти может занять почти в два раза больше времени. При проектировании высокопроизводительных приложений такие соединения следует минимизировать.

При этом следует учитывать:

- топологию системы (схему связи компонентов);
- процессорное ядро, на котором запускается программа;
- расположение банка памяти.

Red Hat Enterprise Linux 6 предоставляет инструменты для коррекции производительности систем NUMA. Они будут рассмотрены ниже.

4.1.2.1. taskset

taskset позволяет привязать процесс к определенному процессору. Недостаток состоит в том, что **taskset** не гарантирует выделение локальной памяти. Эту функцию успешно выполняет **numactl** (см. Раздел 4.1.2.2, «**numactl**»).

Процессоры определяются с помощью маски. Минимальное значение соответствует первому процессору, максимальное — последнему. Например, **`0x00000001`** может обозначать процессор с номером 0, а **`0x00000003`** — процессоры 0 и 1.

Ниже приведена команда привязки работающего процесса к процессору, заданному с помощью маски.

```
# taskset -p маска PID
```

Следующая команда привяжет заданную программу к процессору. Значение *программа* может содержать ее имя и аргументы.

```
# taskset маска -- программа
```

Аргумент **`-c`** позволяет определить список или диапазон процессоров:

```
# taskset -c 0,5,7-9 -- myprogram
```

Подробную информацию можно найти на справочной странице **`man taskset`**.

4.1.2.2. numactl

numactl выполняет процессы в соответствии с политикой распределения памяти. Политика определяет правила для заданного процесса и его дочерних процессов. **numactl** получает топологию системы из **/sys**.

Файловая система **/sys** содержит схему соединения процессоров, памяти и периферийных устройств. Так, например, **/sys/devices/system/cpu** содержит информацию о процессорах, а **/sys/devices/system/node** — об узлах NUMA и их взаимном расположении.

В системах NUMA расстояние между процессором и памятью имеет огромное значение — чем дальше они расположены, тем медленнее доступ. Поэтому при проектировании требовательных к производительности приложений следует выбирать память из наиболее близкого банка.

Высокопроизводительные приложения — особенно многопоточные — должны выполняться на нескольких процессорных ядрах. Так как размер кэша первого уровня небольшой, то при выполнении нескольких потоков на одном ядре не исключена вероятность того, что один поток вытеснит из кэша данные, используемые предыдущим потоком. Таким образом, часть времени будет затрачена на поочередную запись в кэш. Чтобы этого не случилось, рекомендуется привязать приложение не к одному процессорному ядру, а к узлу, что позволит потокам совместно использовать кэш на разных уровнях. Исключение составляют приложения, использующие одни и те же данные в кэше, — их производительность при выполнении на одном процессорном ядре не пострадает.

numactl позволяет привязать приложение к одному процессорному ядру или узлу NUMA и выделить локальную память. Доступные параметры включают:

--show

Возвращает настройки правил NUMA для текущего процесса. Пример: **numactl --show**.

--hardware

Возвращает список доступных узлов.

--membind

Выделяет память только на заданных узлах. Если памяти недостаточно, команда вернет ошибку. Формат: **numactl --membind=*список_узлов программа***. Список узлов может содержать разделенные запятой номера узлов и диапазоны. Подробную информацию можно найти на справочной странице **man numactl**.

--cpunodebind

Выполнение программы и дочерних процессов на процессорах, принадлежащих указанным узлам. Формат: **numactl --cpunodebind=*список_узлов программа***. Список узлов содержит разделенные запятой номера узлов и диапазоны. Подробную информацию можно найти на справочной странице **man numactl**.

--physcpubind

Выполнение программы и дочерних процессов на указанных процессорах. Формат: **numactl --physcpubind=*список_процессоров программа***. Список содержит разделенные запятой номера процессоров (номера можно получить из **/proc/cpuinfo**.) Подробную информацию можно найти на справочной странице **man numactl**.

--localalloc

Память должна выделяться только на текущем узле.

--preferred

Если возможно, память будет выделяться на заданном узле. В случае неудачи будут выбираться другие узлы. Формат: **numactl --preferred=*номер_узла***. Подробную информацию можно найти на справочной странице **man numactl**.

Пакет numactl включает в свой состав библиотеку **libnuma**, которая предоставляет интерфейс для создания собственной политики NUMA. За подробной информацией обратитесь к справочной странице **man numa(3)**.

4.1.3. numastat

ВАЖНО

numastat был разработан Энди Клином и изначально представлял собой сценарий Perl. Впоследствии он был существенно доработан и добавлен в Red Hat Enterprise Linux 6.4.

Несмотря на то что утилита **numastat** полностью совместима со своей ранней версией, их параметры и результаты работы могут значительно отличаться.

numastat выводит статистику распределения памяти (включая попадание и промахи) для процессов и операционной системы. По умолчанию **numastat** покажет число занятых страниц памяти и список событий для каждого узла NUMA.

Оптимальная производительность характеризуется низкими значениями **numa_miss** и **numa_foreign**.

numastat также возвращает информацию о распределении памяти между узлами NUMA.

Можно сопоставить результаты **numastat** и **top**, чтобы убедиться, что потоки выполняются на тех же узлах, где была выделена память.

Стандартная статистика

numa_hit

Число успешно выделенных страниц на узле.

numa_miss

Число страниц, которые должны были быть выделены на другом узле, но из-за нехватки памяти были выделены на текущем узле. Каждому событию **numa_miss** соответствует событие **numa_foreign** на другом узле.

numa_foreign

Число страниц, выделенных на другом узле, которые изначально были предназначены для текущего узла. Каждому событию **numa_foreign** соответствует событие **numa_miss** на другом узле.

interleave_hit

Число успешно выделенных страниц с использованием чередования.

local_node

Число успешно выделенных страниц памяти для локального процесса.

other_node

Число страниц, выделенных на этом узле процессу, выполняемому на другом узле.

Далее перечислены параметры, которые в качестве единиц измерения используют мегабайты.

-c

Компактное представление таблицы данных. Обычно используется при наличии большого числа узлов NUMA, однако ширину столбцов и расстояние между ними предсказать невозможно. Размер памяти будет округляться до ближайшего мегабайта.

-m

Возвращает статистику памяти для каждого узла. Формат аналогичен **/proc/meminfo**.

-n

Возвращает ту же информацию что и исходная версия **numastat** (numa_hit, numa_miss, numa_foreign, interleave_hit, local_node, other_node) в обновленном формате с использованием мегабайт в качестве единиц измерения.

-p ***шаблон***

Возвращает информацию о распределении памяти в соответствии с заданным шаблоном. Если шаблон содержит цифры, **numastat** интерпретирует их как идентификатор процесса. В противном случае **numastat** будет искать совпадение в строках команд.

За параметром **-p** следуют дополнительные фильтры.

-s

Сортировка результатов по убыванию, то есть процессы, потребляющие больше всего ресурсов в соответствии с содержимым столбца **total**, будут приведены в начале списка.

Если дополнительно указать *узел*, таблица будет отсортирована по узлам. Пример:

```
numastat -s2
```

Параметр и значение не должны разделяться пробелом.

-v

Подробный вывод.

-V

Возвращает версию **numastat**.

-z

Исключает строки и столбцы с нулевыми значениями. При этом значения, которые округляются до нуля, не будут отфильтрованы.

4.1.4. numad

numad — средство для привязки процессов к процессорам исходя из топологии NUMA. **numad** следит за топологией и динамически подстраивается к изменениям конфигурации, тем самым поддерживая должный уровень производительности.

В некоторых случаях **numad** может улучшить производительность до 50%. **numad** периодически запрашивает информацию из **/proc** и пытается поместить критические процессы на узлы со свободными ресурсами, где производительность будет максимальна. Минимальные требования составляют 50% ресурсов одного процессора и 300 МБ памяти. Необходимый уровень производительности поддерживается за счет переноса процессов между узлами NUMA по мере освобождения их ресурсов.

numad также предоставляет рекомендации, которые могут использоваться другими инструментами управления задачами. За более подробной информацией обратитесь к описанию параметра **-w** на справочной странице **man numad**.

4.1.4.1. Достоинства numad

numad особенно подходит для продолжительных ресурсоемких процессов, особенно если область их выполнения ограничена лишь подмножеством ресурсов.

numad может улучшить производительность приложений, выполняющихся на нескольких узлах NUMA. Но при росте потребляемых ресурсов эффективность **numad** падает.

Для коротких процессов, не потребляющих много ресурсов, **numad** не требуется — так же как и для систем с непоследовательным обращением к памяти.

4.1.4.2. Режимы работы

ПРИМЕЧАНИЕ

При объединении больших объемов памяти службой KSM статистика ядра может оказаться непоследовательной. В будущих выпусках функции работы с NUMA в KSM будет усовершенствованы, но на сегодняшний день в системах в большими объемами свободной памяти рекомендуется отключить KSM.

numad может работать как:

- служба,
- исполняемый модуль.

4.1.4.2.1. numad как служба

Работающая служба **numad** будет динамически корректировать нагрузку.

Запуск службы:

```
# service numad start
```

Чтобы запуск происходил при каждой загрузке системы:

```
# chkconfig numad on
```

4.1.4.2.2. numad как исполняемый модуль

Команда запуска:

```
# numad
```

numad будет работать, пока он не будет остановлен. События будут регистрироваться в **/var/log/numad.log**.

Для управления конкретным процессом выполните:

```
# numad -S 0 -p PID
```

-p PID

Добавляет заданный процесс в список обработки. Добавленный процесс будет обработан, только если он удовлетворяет минимальным требованиям обслуживания.

-S режим

Параметр ***-S*** определяет режим проверки процессов. Так, значение **0** ограничивает управление процессами, включенными в список обработки.

Остановка **numad**:

```
# numad -i 0
```

После остановки **numad** изменения сопоставлений NUMA не будут отменены. При значительном изменении нагрузки **numad** динамически перераспределит ресурсы.

Подробную информацию о **numad** можно найти на справочной странице **man numad**.

4.2. ПЛАНИРОВАНИЕ ЗАНЯТОСТИ ПРОЦЕССОРОВ

Планировщик распределяет нагрузку между процессорами в соответствии с существующими правилами, поддерживая их максимальную занятость. Правила определяют время выполнения потока на одном процессорном ядре.

Политики планирования подразделяются на две категории:

1. Планирование в реальном времени
   - SCHED_FIFO
   - SCHED_RR
2. Стандартное планирование
   - SCHED_OTHER
   - SCHED_BATCH
   - SCHED_IDLE

4.2.1. Планирование в реальном времени

Планирование выполнения потоков реального времени выполняется в первую очередь.

Эти правила применяются к критическим процессам, которые должны завершены как можно быстрее.

SCHED_FIFO

Это правило присваивает потокам фиксированный приоритет от 1 до 99. Планировщик проверяет список потоков SCHED_FIFO и запускает поток с наивысшим приоритетом. Поток будет выполняться до тех пор, пока не завершит работу или не будет вытеснен другим потоком с более высоким приоритетом.

Даже потоки с наименьшим приоритетом будут выполняться до потоков других типов. При наличии единственного потока, работающего в реальном времени, значение **SCHED_FIFO** будет игнорироваться.

SCHED_RR

Это циклический вариант правила **SCHED_FIFO**. Потокам также присваивается приоритет от 1 до 99, и потоки с одинаковым приоритетом будут последовательно выполняться в рамках выделенного интервала. Пользователь не может изменить интервал, но может его узнать при помощи **sched_rr_get_interval(2)**. Обычно это правило применяется при наличии потоков с одинаковым приоритетом.

Подробную информацию о планировании выполнения потоков можно найти в описании стандарта *IEEE 1003.1 POSIX* по адресу http://pubs.opengroup.org/onlinepubs/009695399/functions/xsh_chap02_08.html.

Изначально рекомендуется выбирать низкий приоритет и увеличивать его только в том случае, если была обнаружена заметная задержка при выполнении. Потокам реального времени не выделяются фрагменты времени; потоки **SCHED_FIFO** будут выполняться до тех пор, пока они не завершат работу или не будут вытеснены другими потоками с более высоким приоритетом. Не рекомендуется присваивать значение 99, так как высокий приоритет обычно имеют процессы миграции и мониторинга. Если они будут вытеснены другими процессами, это может привести к блокированию в однопроцессорных системах.

В ядре Linux правило **SCHED_FIFO** также предусматривает механизм ограничения, который предотвращает монопольное использование процессорных ресурсов. Его конфигурацию можно изменить с помощью:

/proc/sys/kernel/sched_rt_period_us

Интервал времени в микросекундах, характеризующий 100% полосы пропускания. По умолчанию равен 1000000 мкс (1 сек.).

/proc/sys/kernel/sched_rt_runtime_us

Интервал выполнения потоков реального времени (в микросекундах). По умолчанию равен 950000 мкс, что эквивалентно 0.95 сек.

4.2.2. Стандартное планирование

Стандартные правила включают **SCHED_OTHER**, **SCHED_BATCH** и **SCHED_IDLE**. При этом **SCHED_BATCH** и **SCHED_IDLE** предназначены для задач с низким приоритетом и подробно рассматриваться не будут.

SCHED_OTHER или SCHED_NORMAL

Используется по умолчанию. Распределение ресурсов осуществляется при помощи планировщика CFS (Completely Fair Scheduler), который приоритизирует потоки в зависимости от значения ***niceness*** (более подробно об этом рассказывается в *руководстве по развертыванию*). Несмотря на то что пользователь в определенной мере может управлять приоритетом, его динамическое изменение возможно только при помощи CFS.

4.2.3. Выбор стратегии

Выбор стратегии распределения ресурсов не является тривиальной задачей. Обычно правила реального времени выбираются для критических задач, которые должны быть обработаны без задержек, в то время как стандартные правила позволяют обслуживать процессы более эффективно, так как они не должны учитывать обработку вытеснения потоков.

При обработке больших объемов потоков пропускная способность (число операций в секунду) имеет огромное значение, поэтому можно выбрать **SCHED_OTHER** и позволить системе управлять процессорными ресурсами самостоятельно.

Если необходимо сократить время ответа, следует выбрать **SCHED_FIFO**. При обработке небольшого числа потоков оптимальной производительности можно достичь, выделив для их обслуживания отдельный узел NUMA.

4.3. ОБРАБОТКА ЗАПРОСОВ ПРЕРЫВАНИЙ

Прерывание — запрос обслуживания на аппаратном уровне. Прерывания могут генерироваться локальным оборудованием или поступать по шине в виде пакета данных (MSI, Message Signaled Interrupt).

Код обработки прерываний ядра получает номер запроса и список зарегистрированных обработчиков прерываний, и по очереди их вызывает. Обработчик получает прерывание, маскирует аналогичные запросы, запрашивает обработку прерывания низкоприоритетным обработчиком и после завершения перестает маскировать запросы.

/proc/interrupts содержит статистику прерываний: номер прерывания, число прерываний этого типа, полученных каждым процессорным ядром, тип прерывания и список драйверов, обрабатывающих это прерывание. Подробную информацию можно найти на справочной странице **man 5 proc**.

Прерываниям соответствует параметр ***smp_affinity***, определяющий ядра, которые будут принимать участие в обслуживании. Его значение можно корректировать с целью улучшения производительности, привязывая прерывания и потоки к одним и тем же ядрам.

Значение ***smp_affinity*** определяется в **/proc/irq/*номер_прерывания*/smp_affinity** в шестнадцатеричном формате. Для его просмотра и изменения необходимы права root.

В качестве примера рассмотрим прерывание драйвера Ethernet на сервере с четырьмя процессорными ядрами. Для начала надо узнать его номер прерывания:

```
# grep eth0 /proc/interrupts
32:   0     140      45       850264      PCI-MSI-edge      eth0
```

Теперь можно просмотреть содержимое файла ***smp_affinity***:

```
# cat /proc/irq/32/smp_affinity
f
```

f означает, что прерывание может обслуживаться на любом процессоре. Ниже этому параметру будет присвоено значение **1**, то есть прерывание будет обслуживаться на процессоре 0.

```
# echo 1 >/proc/irq/32/smp_affinity
# cat /proc/irq/32/smp_affinity
1
```

Можно указать несколько значений, разделив их запятыми. Обычно используется в системах, где число ядер превышает 32. Так, например, ниже обслуживание прерывания 40 разрешается на всех ядрах в 64-ядерной системе:

```
# cat /proc/irq/40/smp_affinity
ffffffff,ffffffff
```

Пример значения ***smp_affinity***, ограничивающий обслуживание прерывания 40 последними 32 ядрами в 64-ядерной системе:

```
# echo 0xffffffff,00000000 > /proc/irq/40/smp_affinity
# cat /proc/irq/40/smp_affinity
ffffffff,00000000
```

ПРИМЕЧАНИЕ

В системах, поддерживающих *управление линией запросов прерывания*, управление может быть передано оборудованию.

4.4. NUMA В RED HAT ENTERPRISE LINUX 6

Red Hat Enterprise Linux 6 включает целый ряд улучшений работы с NUMA, облегчающих адаптацию к изменяющимся требованиям оборудования.

4.4.1. Масштабирование

4.4.1.1. Топология

Ниже перечислены функции, помогающие Red Hat Enterprise Linux идентифицировать оборудование и архитектуру, тем самым облегчая оптимизацию производительности на низком уровне.

Определение топологии во время загрузки

Получение информации об оборудовании, включая сведения о логических процессорах, гиперпотоках, ядрах, сокетах, узлах NUMA и времени сообщения между узлами.

Планировщик CFS

Новый режим планирования обеспечивает равномерное распределение процессорных

ресурсов. В комбинации с определением топологии это позволяет планировать выполнение процессов на ядрах в пределах одного сокета, исключая необходимость обращения к другим сокетам и сохраняя содержимое кэша.

`malloc`

`malloc` теперь оптимизирует выделение памяти, предпочитая выбирать фрагменты, расположенные как можно ближе к ядру, на котором выполняется процесс.

Выделение памяти буферу skbuff

Также оптимизирует выделение памяти, предпочитая выбирать фрагменты, расположенные как можно ближе к ядру, обслуживающему прерывания.

Привязка прерываний устройств

Драйверы устройств регистрируют информацию о процессорах, обрабатывающих прерывания. В дальнейшем можно ограничить их обслуживание процессорами одного сокета, что позволит поддерживать состояние кэша и уменьшая необходимость сообщения между разными сокетами.

4.4.1.2. Синхронизация в многопроцессорном окружении

Координация обработки задач между процессорами требует частых проверок с целью поддержки целостности данных. Red Hat Enterprise Linux включает следующие функции для оптимизации производительности в этой области:

Блокировка RCU (Read-Copy-Update)

Обычно приблизительно 90% блокировок обслуживают операции чтения. Блокировка RCU исключает необходимость получения эксклюзивного доступа для чтения. Этот режим используется при выделении страниц из кэша.

Алгоритмы распределения памяти процессоров и сокетов

Многие алгоритмы были обновлены и позволяют точно координировать работу процессорных ядер в одном сокете. Вместо спин-блокировок теперь используются методы индивидуального блокирования сокетов, а обновленные карты выделения памяти и списки страниц позволяют более эффективно использовать структуры карт данных.

4.4.2. Виртуализация

Так как KVM использует функции ядра, производительность виртуальных машин KVM напрямую зависит от его оптимизации и может достигать уровня производительности физических систем. Это возможно за счет оптимизации сетевого доступа и ввода-вывода. Ниже перечислены улучшения, имеющие прямое отношение к NUMA.

Привязка процессоров

С целью оптимизации использования локального кэша и сокращения числа обращений к удаленной памяти виртуальные системы могут быть привязаны к конкретному сокету.

Большие страницы THP

Использование больших страниц (Hugepages) позволяет сократить конкуренцию блокировок и уменьшить число операций управления памятью в буфере трансляции адресов (TLB, Translation Lookaside Buffer), что может повысить производительность виртуальных машин на

20%.

Ввод-вывод на уровне ядра

Подсистема ввода-вывода для виртуальных систем теперь реализована на уровне ядра, что значительно сокращает взаимодействие между узлами и время на переключение контекста и синхронизацию.

ГЛАВА 5. ПАМЯТЬ

Эта глава содержит обзор функций управления памятью в Red Hat Enterprise Linux и способов оптимизации использования памяти.

5.1. HUGETLB

Преобразование физических адресов памяти в виртуальные осуществляется на основе соответствий в таблице страниц. Буфер TLB (Translation Lookaside Buffer) представляет собой кэш, в котором хранятся последние полученные адреса, что улучшает быстродействие за счет уменьшения числа обращений к таблице.

Размер кэша ограничен, поэтому при получении адреса, не содержащегося в кэше, он будет запрошен из таблицы страниц. Такая ситуация известна как «промах TLB». Приложения с высокими требованиями к памяти чаще сталкиваются с промахами в силу взаимосвязи между их требованиями к памяти и размером страниц в TLB. Так как каждый промах означает необходимость обращения к таблице страниц, число таких обращений необходимо минимизировать.

Буфер HugeTLB (Huge Translation Lookaside Buffer) использует страницы большого размера, что позволяет кэшировать больше адресов за раз, тем самым уменьшая вероятность промахов и увеличивая производительность приложений.

Подробную информацию о HugeTLB можно найти в **/usr/share/doc/kernel-doc-*версия*/Documentation/vm/hugetlbpage.txt**

5.2. УВЕЛИЧЕНИЕ РАЗМЕРА СТРАНИЦ

Стандартный размер страницы памяти — 4096 байт. Таким образом 1 мегабайт содержит 256 страниц, 1 гигабайт — 256 тысяч страниц и т.п. Процессоры включают встроенный *модуль управления памятью*, который содержит полный список страниц, каждой из которых соответствует *запись* в таблице.

Существует два основных способа управления большими объемами памяти:

- увеличение числа записей в модуле управления памятью;
- увеличение размера страниц.

Первый метод является довольно дорогим, так как аппаратные модули управления памятью поддерживают всего лишь тысячи записей. Алгоритмы, справляющиеся с обработкой тысяч страниц (что исчисляется мегабайтами), теряют эффективность при работе с миллионами и даже миллиардами страниц. Если приложение требует больше страниц памяти, чем поддерживается аппаратно, то управление страницами передается программным механизмам, что существенно замедляет работу системы.

Red Hat Enterprise Linux 6 использует второй метод.

Так называемые большие страницы (hugepages) представляют собой сегменты памяти размером 2 МБ или 1 ГБ. Первый размер подходит для управления гигабайтами памяти, а второй — терабайтами.

Страницы большого размера должны быть назначены во время загрузки. Управлять ими вручную не рекомендуется, но для эффективной работы они могут требовать значительной модификации кода. С этой задачей можно успешно справиться с помощью так называемых

«прозрачных страниц» (THP, transparent huge pages). THP добавляет дополнительный уровень абстракции, облегчающий и автоматизирующий управление большими страницами.

THP скрывает комплексную функциональность страниц от администраторов и разработчиков. Функции THP тщательно тестировались и оптимизировались разработчиками сообщества и самой компании Red Hat для разных приложений, уровней нагрузки, конфигураций и архитектур. Поэтому стандартной конфигурации THP обычно должно быть достаточно.

В настоящее время возможности THP поддерживаются только для анонимной памяти кучи и стека.

5.3. ПРОФИЛИРОВАНИЕ ПАМЯТИ ПРИ ПОМОЩИ VALGRIND

Valgrind предоставляет инструменты для профилирования и анализа производительности программ. В этой главе будут рассмотрены инструменты, которые помогут обнаружить попытки использования неинициализированной памяти и неверного выделения памяти. Для запуска отдельной утилиты выполните:

```
valgrind --tool=утилита программа
```

В этой команде *утилита* — название утилиты. Так, для профилирования памяти можно указать **memcheck**, **massif**, **cachegrind**. Следом за утилитой введите имя анализируемой программы. В режиме диагностики программа выполняется намного медленнее.

Раздел 3.5.3, «Valgrind» содержит описание основных функций. Более подробную информацию, включая описание дополнительных модулей Eclipse, можно найти в *руководстве по развертыванию* по адресу http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/, на справочной странице **man valgrind**, а также в файлах:

- **/usr/share/doc/valgrind-*версия*/valgrind_manual.pdf**
- **/usr/share/doc/valgrind-*версия*/html/index.html**.

5.3.1. Memcheck

Если команда **valgrind** не содержит параметр **--tool**, по умолчанию будет выбрана утилита **memcheck**. Memcheck поможет обнаружить те ошибки, которые другие инструменты не могут диагностировать, включая использование неопределенных и неинициализированных значений, некорректное освобождение памяти кучи, пересечение указателей и утечки памяти. Во время диагностики программы будут работать в 10-30 раз медленнее.

Memcheck возвращает коды обнаруженных ошибок. Их описание можно найти в **/usr/share/doc/valgrind-*version*/valgrind_manual.pdf**.

Memcheck только сообщает об ошибках, но не препятствует их появлению. Так, если обращение программы к памяти приводит к сегментации, Memcheck сообщит об ошибке, но не остановит сегментацию.

Параметры Memcheck включают:

--leak-check

Поиск утечек памяти после завершения программы клиента. По умолчанию имеет значение **summary** и возвращает число обнаруженных утечек. Другие значения включают **yes** и **full**, и возвращают информацию об индивидуальных утечках, а **no** отключает проверку.

--undef-value-errors

yes включает сообщения об ошибках для неинициализированных значений, а **no** их отключает.

--ignore-ranges

Игнорируемые диапазоны. Пример: **--ignore-ranges=0xPP-0xQQ,0xRR-0xSS**.

Полный список параметров можно найти в **/usr/share/doc/valgrind-*версия*/valgrind_manual.pdf**.

5.3.2. Cachegrind

Cachegrind имитирует взаимодействие программы с кэшем и модулем предсказания переходов. Cachegrind следит за выполнением инструкций первого уровня, кэшированием данных и взаимодействием с последним уровнем кэша. Во время диагностики программы будут работать в 20-100 раз медленнее.

Команда запуска Cachegrind:

```
# valgrind --tool=cachegrind программа
```

Cachegrind возвращает статистику по следующим операциям:

- чтение кэша инструкциями первого уровня и промахи чтения;
- чтение кэша и памяти, промахи чтения, промахи чтения кэша последнего уровня;
- запись в кэш и память, промахи записи, промахи записи в кэш последнего уровня;
- выполненные и неверно предсказанные условные ветвления:
- выполненные и неверно предсказанные непрямые переходы.

Cachegrind выводит статистику на консоль и сохраняет более подробную информацию в файл **cachegrind.out.*PID*** (*pid* — идентификатор тестируемого процесса). Для анализа этого файла используется **cg_annotate**:

```
# cg_annotate cachegrind.out.PID
```

ПРИМЕЧАНИЕ

cg_annotate позволяет выводить строки длиной больше 120 знаков. Чтобы облегчить чтение, рекомендуется изменить ширину окна терминала так, чтобы такие строки вмещались полностью.

Созданные профили можно сравнить и определить изменения в производительности программ:

```
# cg_diff первый_профиль второй_профиль
```

Результаты будут сохранены в файл, для чтения которого также можно использовать **cg_annotate**.

Cachegrind содержит следующие параметры:

--I1

Размер, ассоциативность, размер строк кэша инструкций первого уровня. Формат: **--I1=*размер*,*ассоциативность*,*размер_строк***.

--D1

Размер, ассоциативность, размер строк кэша данных первого уровня. Формат: **--I1=*размер*,*ассоциативность*,*размер_строк***.

--LL

Размер, ассоциативность, размер строк кэша последнего уровня. Формат: **--I1=*размер*,*ассоциативность*,*размер_строк***.

--cache-sim

Включает и отключает сбор статистики попадания и промахов кэша. По умолчанию используется значение **yes**.

Отключение обоих параметров **--cache-sim** и **--branch-sim** отменит сбор статистики.

--branch-sim

Включает и отключает сбор статистики об инструкциях ветвления и числе неверных предсказаний. По умолчанию используется значение **no**, так как подобный анализ замедляет Cachegrind примерно на 25%.

Одновременное отключение параметров **--cache-sim** и **--branch-sim** полностью отменит сбор статистики.

Полный список параметров можно найти в **/usr/share/doc/valgrind-*версия*/valgrind_manual.pdf**.

5.3.3. Massif

Massif оценивает размер памяти кучи, используемой программой, что поможет повысить эффективность работы и снизить вероятность нехватки пространства в области подкачки. Massif поможет найти фрагменты программы, которые требуют выделения дополнительной памяти. Подобная диагностика замедлит работу программы примерно в 20 раз.

Формат:

```
# valgrind --tool=massif программа
```

Massif сохраняет статистику в файл **massif.out.*pid***, где *pid* — идентификатор тестируемого процесса.

ms_print генерирует график выделения памяти:

```
# ms_print massif.out.PID
```

Эта команда создаст график выделения памяти на протяжении работы программы.

Параметры Massif включают:

--heap

Включает и отключает сбор статистики памяти кучи. Возможные значения — **yes** (по умолчанию) и **no**.

--heap-admin

Определяет размер административного блока в байтах (по умолчанию **8**).

--stacks

Включает и отключает сбор статистики стека. Возможные значения — **yes** и **no** (по умолчанию). Активация этой возможности значительно замедляет работу Massif. Изначально подразумевается, что стек имеет нулевой размер, что облегчает определение размера секции стека, используемой программой.

--time-unit

Задает единицы времени. Доступные значения: **i** (по выполненным инструкциям, используется по умолчанию), **ms** (в миллисекундах) и **B** (в байтах). Этот параметр определяет формат единиц на графике **ms_print**.

Полный список параметров можно найти в **/usr/share/doc/valgrind-*версия*/valgrind_manual.pdf**.

5.4. ПАРАМЕТРЫ ПРОИЗВОДИТЕЛЬНОСТИ В /PROC

В этой секции рассматриваются параметры, помогающие определить размер памяти и файловой системы, и риски, связанные с их использованием.

Чтобы временно изменить параметр, сохраните его новое значение в proc. Например, чтобы присвоить ***overcommit_memory*** значение **1**, выполните:

```
# echo 1 > /proc/sys/vm/overcommit_memory
```

Путь зависит от изменяемого параметра.

Команда **sysctl** переопределяет параметры с сохранением результатов. За дальнейшей информацией обратитесь к *руководству по развертыванию* по адресу http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

Параметры памяти

Перечисленные параметры определены в **/proc/sys/vm/**.

overcommit_memory

Определяет условия разрешения и отказа запросов больших объемов памяти. Возможные значения:

- **0** (по умолчанию) — ядро использует эвристический алгоритм для расчета перерасхода памяти, принимая во внимание объем доступной памяти и число неверных запросов. Но поскольку выделение памяти осуществляется на основе

эвристического, а не точного алгоритма, это может привести к превышению допустимой нагрузки на память.

- **1** — ядро не обрабатывает перерасход памяти. При этом вероятность превышения нагрузки на память возрастает, но в то же время увеличивается производительность задач, активно использующих память.
- **2** — отказ обработки запросов, запрашивающих память, размер которой превышает суммарный размер памяти пространства подкачки и ОЗУ в соответствии с ***overcommit_ratio***.

ПРИМЕЧАНИЕ

Этот параметр должен использоваться в только в тех системах, где размер пространства подкачки превышает общий размер физической памяти.

overcommit_ratio

Если ***overcommit_memory*** равен **2**, определяет процентную часть физической памяти (см. описание ***overcommit_memory***). По умолчанию равен **50**.

max_map_count

Максимальное число регионов памяти, доступных процессу. По умолчанию равно **65530**.

nr_hugepages

Число страниц большого размера. По умолчанию равно 0. Выделение больших страниц возможно только при наличии достаточного числа следующих друг за другом свободных страниц. Зарезервированные этим параметром страницы не могут использоваться для других целей. За дальнейшей информацией обратитесь к документации в **/usr/share/doc/kernel-doc-*версия_ядра*/Documentation/vm/hugetlbpage.txt**

Параметры ядра

Перечисленные параметры определены в **/proc/sys/kernel/**.

msgmax

Максимальный размер сообщения в очереди (в байтах). По умолчанию равен **65536**. Это значение не может превышает ***msgmnb***.

msgmnb

Максимальный размер очереди сообщений (в байтах). По умолчанию равен **65536**.

msgmni

Максимальное число идентификаторов очередей сообщений (число очередей). Для 64-разрядной архитектуры по умолчанию равно **1985**, а для 32-разрядной — **1736**.

shmall

Размер общей памяти (в байтах), которая может быть использоваться в заданный момент. Для 64-разрядной архитектуры по умолчанию равен **4294967296**, а для 32-разрядной — **268435456**.

shmmax

Максимальный размер сегмента общей памяти (в байтах). Для 64-разрядной архитектуры по умолчанию равен **68719476736**, а для 32-разрядной — **4294967295**.

shmmni

Максимальное число совместно используемых сегментов памяти. По умолчанию равно **4096**.

threads-max

Максимальное число одновременно выполняемых потоков. По умолчанию эквивалентно ***max_threads***. Формула расчета:

```
max_threads = mempages / (8 * РАЗМЕР_ПОТОКА / РАЗМЕР_СТРАНИЦЫ )
```

Минимальное значение ***threads-max*** равно **20**.

Параметры файловой системы

Перечисленные параметры определены в **/proc/sys/fs/**.

aio-max-nr

Максимальное число событий в асинхронных контекстах ввода-вывода. По умолчанию равно **65536**. При изменении этого значения размеры структур данных ядра не изменяются автоматически.

file-max

Возвращает максимальное число дескрипторов файлов в ядре. По умолчанию эквивалентно ***files_stat.max_files***, в качестве значения которого выбирается большее из двух — **(mempages * (РАЗМЕР_СТРАНИЦЫ / 1024)) / 10** или **NR_FILE** (8192 в Red Hat Enterprise Linux). Увеличение этого значения может уменьшить вероятность ошибок, связанных с нехваткой дескрипторов файлов.

Параметры обработки нехватки памяти

Выделение всей доступной памяти, включая пространство подкачки, может привести к панике ядра. Это можно предотвратить, присвоив параметру ***/proc/sys/vm/panic_on_oom*** значение **0**, что вызовет функцию **oom_killer** при нехватке памяти, которая остановит несанкционированные процессы.

Ниже приведен параметр, который может быть задан для отдельных процессов, обеспечивая тем самым тонкий контроль над тем, какие процессы **oom_killer** будет останавливать. Он размещается в **/proc/*PID*/** (*PID* — идентификатор процесса).

oom_adj

Значение в диапазоне от **-16** до **15** определяет ***oom_score*** процесса. Чем больше ***oom_score***, тем больше вероятность того, что **oom_killer** остановит процесс. Значение **-17** отключает **oom_killer** для процесса.

ВАЖНО

Дочерние процессы наследуют ***oom_score*** родительского процесса. Так, например, если **sshd** не использует **oom_killer**, то процессы, запущенные в рамках его сеанса, также не будут использовать. Это может иметь отрицательный эффект, так как **oom_killer** не сможет освободить память таких процессов.

5.5. ВИРТУАЛЬНАЯ ПАМЯТЬ

Виртуальная память используется процессами, кэшем файловой системы и ядром. Степень ее занятости определяется следующими параметрами:

swappiness

Значение от 0 до 100 определяет процент подкачки. Высокие значения означают, что предпочтение будет отдаваться производительности системы, агрессивно подкачивая страницы из физической памяти. Низкие значения избегают подкачки с целью уменьшения задержки обслуживания. По умолчанию равно **60**.

min_free_kbytes

Минимальный размер свободной памяти (в килобайтах). При достижении этого порога начнут выделяться резервные страницы.

ПРЕДУПРЕЖДЕНИЕ

Порог не должен быть слишком высоким или слишком низким.

Слишком низкое значение ***min_free_kbytes*** будет препятствовать освобождению памяти, что завершится зависанием системы и остановкой множества процессов.

В то же время слишком высокий порог (5-10% системной памяти) быстро приведет к нехватке памяти. Linux обычно использует всю доступную оперативную память для кэширования данных файловой системы, а высокий порог означает, что на освобождение памяти будет тратится слишком много времени.

dirty_ratio

Процент «грязных» данных по отношению к общему размеру памяти, при достижении которого начнется их запись (при помощи **pdflush**). По умолчанию равно **20** процентам.

dirty_background_ratio

Процент «грязных» данных по отношению к общему размеру памяти, при достижении которого начнется их запись в фоновом режиме (при помощи **pdflush**). По умолчанию равно **10** процентам.

drop_caches

Очищает кэш, тем самым освобождая память. Допустимые значения:

1

Освобождает память кэша страниц.

2

Освобождает память кэша индексных дескрипторов и записей каталогов.

3

Освобождает память всех вышеперечисленных типов.

Эта операция не является разрушающей. Так как «грязные» объекты не освобождают память, предварительно рекомендуется выполнить **sync**.

ВАЖНО

Не рекомендуется освобождать память с помощью ***drop_caches*** в критических окружениях.

Чтобы временно изменить параметр, сохраните его значение в proc. Например, чтобы присвоить ***swappiness*** значение **50**, выполните:

```
# echo 50 > /proc/sys/vm/swappiness
```

Команда **sysctl** переопределяет параметры с сохранением результатов. За дальнейшей информацией обратитесь к *руководству по развертыванию* по адресу http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

ГЛАВА 6. ВВОД-ВЫВОД

6.1. ХАРАКТЕРИСТИКИ

Red Hat Enterprise Linux 6 включает целый ряд улучшенных функций управления стеком ввода-вывода:

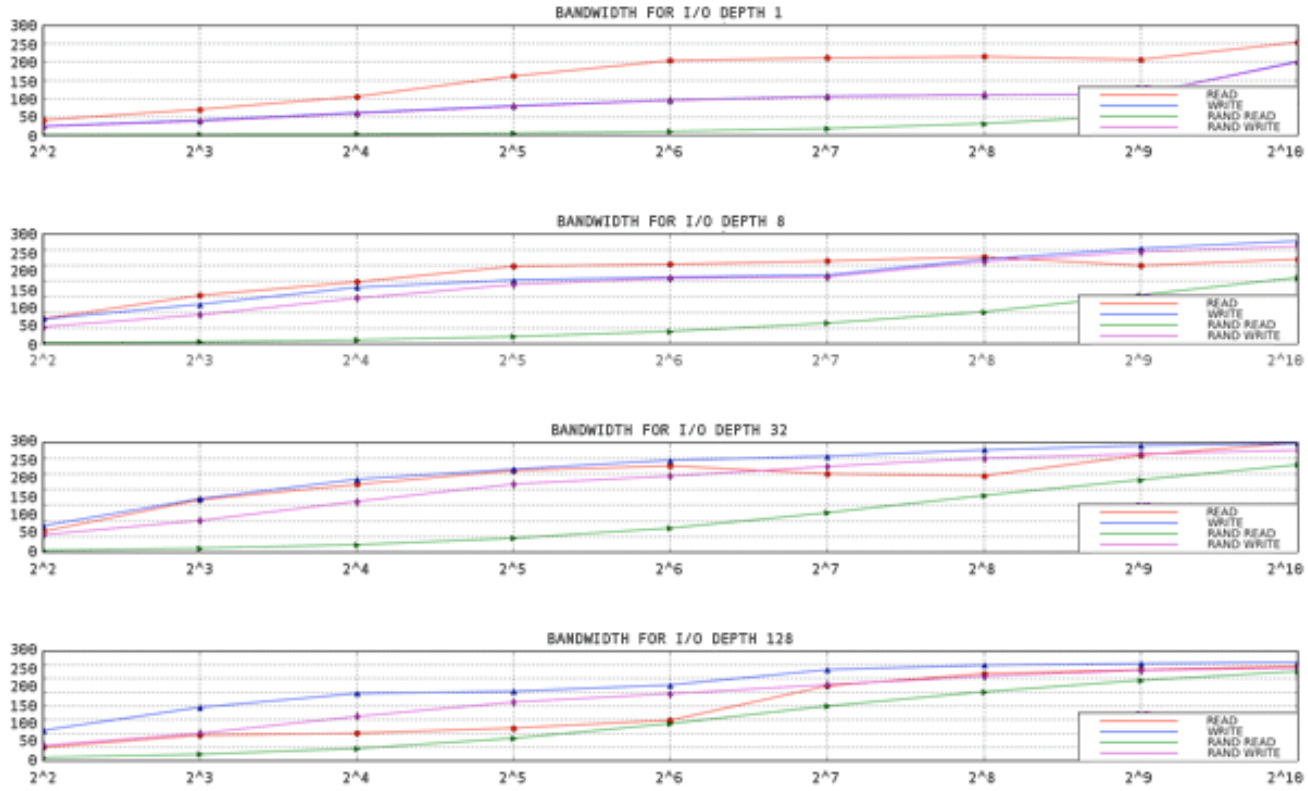
- Твердотельные накопители (SSD, Solid State Disks) теперь определяются автоматически, что положительно сказывается на производительности планировщика ввода-вывода в силу эффективного использования высокоскоростного ввода-вывода SSD-дисков.
- Ядро теперь поддерживает освобождение блоков при удалении файлов, что позволяет передавать информацию о незанятых блоках накопителю. Это увеличивает эффективность работы накопителей, поддерживающих инициализацию логических блоков (виртуальную адресацию) за счет более точного контроля используемого пространства.
- Улучшена производительность барьеров файловой системы.
- Вместо **pdflush** теперь используются потоки очистки отдельных устройств, что значительно облегчает масштабирование в схемах с большим количеством LUN.

6.2. АНАЛИЗ

Оптимизация производительности стека данных требует понимания схемы движения данных в системе, а также конфигурации накопителей и их эффективности при разных уровнях нагрузки.

При развертывании новой системы рекомендуется создать полный профиль схемы хранения данных. Начните с физических дисков и LUN, оцените их производительность при обработке прямого ввода-вывода (без участия кэша страниц ядра). Результаты этой проверки будут служить точкой отсчета. После этого можно приступить к тестированию базовой нагрузки, например при помощи **aio-stress**, что будет генерировать набор случайных операций чтения и записи.

Ниже приведены графики результатов прогонов **aio-stress**, каждый из которых содержит четыре стадии: последовательная запись, последовательное чтение, случайная запись, случайное чтение. Так, в приведенных примерах тестирование проводится для разных размеров записей (ось X), а графики иллюстрируют результаты для очередей разной длины.



Вертикальная ось отражает скорость обслуживания в мегабайтах в секунду, горизонтальная — размер данных в килобайтах.

Рисунок 6.1. Вывод aio-stress для 1 потока и 1 файла

Обратите внимание на рост кривой — производительность возрастает при росте числа обрабатываемых запросов ввода-вывода.

После выполнения нескольких тестов вы будете знать, как меняется производительность в зависимости от нагрузки. Сохраните результаты, чтобы в будущем их можно было сравнить с новыми данными.

После этого можно приступить к тестированию многопутевых схем. Если наблюдается существенное снижение производительности, определите, является ли это оправданным. Так, например, снижение производительности при добавлении уровня проверки контрольной суммы RAID в стек является ожидаемым в отличие от неупорядоченных операций ввода-вывода. Red Hat Enterprise Linux по умолчанию стремится оптимизировать сопоставление разделов и метаданных Device Mapper, но в некоторых случаях может требоваться ручная коррекция.

Наконец, выполняется тестирование файловой системы с прямым вводом-выводом. Сравните полученные результаты с результатами предыдущих тестов и убедитесь, что для всех случаев снижения производительности есть объяснение. Обычно скорость прямого ввода-вывода выше для предварительно выделенных файлов, поэтому рекомендуется заранее выделить пространство и уже потом приступить к тестированию.

Следующие инструменты искусственно генерируют нагрузку:

- **aio-stress**,
- **iozone**,
- **fio**.

6.3. ИНСТРУМЕНТЫ

Существует множество инструментов диагностики производительности в подсистемах ввода-вывода. Так, **vmstat** возвращает общую информацию о производительности. Следующие столбцы имеют отношение к вводу-выводу: ***si*** (в пространство подкачки), ***so*** (из пространства подкачки), ***bi*** (в блочное устройство), ***bo*** (из блочного устройства), ***wa*** (время ожидания). ***si*** и ***so*** служат индикаторами нагрузки на память, особенно если пространство подкачки расположено на том же устройстве что и раздел данных. ***si*** и ***bi*** содержат информацию об операциях чтения, а ***so*** и ***bo*** об операциях записи. В качестве единиц измерения используются килобайты. ***wa*** возвращает время простоя и сообщает, какая часть очереди ожидает завершения операции ввода-вывода.

Анализ статистики **vmstat** поможет подтвердить или опровергнуть факт потери производительности на уровне подсистемы ввода-вывода. Так, если значения ***cache*** и ***bo*** увеличиваются, после чего ***cache*** уменьшается, а ***free*** увеличивается, это означает, что система осуществляет запись и освобождает кэш страниц.

Вывод **vmstat** объединяет статистику всех устройств. Определив слабое место, можно провести более тщательный анализ с помощью **iostat**, который покажет статистику по устройствам, включая средний размер запросов, число операций ввода и вывода в секунду и т.п.

Знание среднего размера запросов и очередей (**avgqu-sz**) поможет оценить производительность. Так, если средний размер очереди равен 1, а размер запросов — 4 КБ, производительность не может быть высокой.

Более детальный анализ можно провести с помощью **blktrace**. Вывод **blktrace** может быть обработан другими утилитами, такими как **blkparse**.

blkparse представляет вывод **blktrace** в удобном для чтения формате.

Пример вывода **blktrace**:

```
8,64   3        1     0.000000000  4162  Q  RM 73992 + 8 [fs_mark]
8,64   3        0     0.000012707     0  m   N cfq4162S / alloced
8,64   3        2     0.000013433  4162  G  RM 73992 + 8 [fs_mark]
8,64   3        3     0.000015813  4162  P   N [fs_mark]
8,64   3        4     0.000017347  4162  I   R 73992 + 8 [fs_mark]
8,64   3        0     0.000018632     0  m   N cfq4162S / insert_request
8,64   3        0     0.000019655     0  m   N cfq4162S / add_to_rr
8,64   3        0     0.000021945     0  m   N cfq4162S / idle=0
8,64   3        5     0.000023460  4162  U   N [fs_mark] 1
8,64   3        0     0.000025761     0  m   N cfq workload slice:300
8,64   3        0     0.000027137     0  m   N cfq4162S / set_active
wl_prio:0 wl_type:2
8,64   3        0     0.000028588     0  m   N cfq4162S / fifo=(null)
8,64   3        0     0.000029468     0  m   N cfq4162S / dispatch_insert
8,64   3        0     0.000031359     0  m   N cfq4162S / dispatched a
request
8,64   3        0     0.000032306     0  m   N cfq4162S / activate rq,
drv=1
8,64   3        6     0.000032735  4162  D   R 73992 + 8 [fs_mark]
8,64   1        1     0.004276637     0  C   R 73992 + 8 [0]
```

Как видно из примера, такой вывод не очень удобно читать. Фрагмент вывода **blkparse** будет выглядеть так:

```
Total (sde):
Reads Queued:          19,       76KiB  Writes Queued:     142,183,
568,732KiB
Read Dispatches:       19,       76KiB  Write Dispatches:   25,440,
568,732KiB
Reads Requeued:         0               Writes Requeued:       125
Reads Completed:       19,       76KiB  Writes Completed:   25,315,
568,732KiB
Read Merges:            0,        0KiB  Write Merges:      116,868,
467,472KiB
IO unplugs:        20,087               Timer unplugs:           0
```

Существует несколько утилит для интерпретации слишком подробного вывода **blkparse**.

Так, **btt** получает информацию о том, сколько времени было затрачено на выполнение операций в стеке ввода-вывода.

- Q — запросы к блочному устройству поставлены в очередь;
- G — запрос получен;

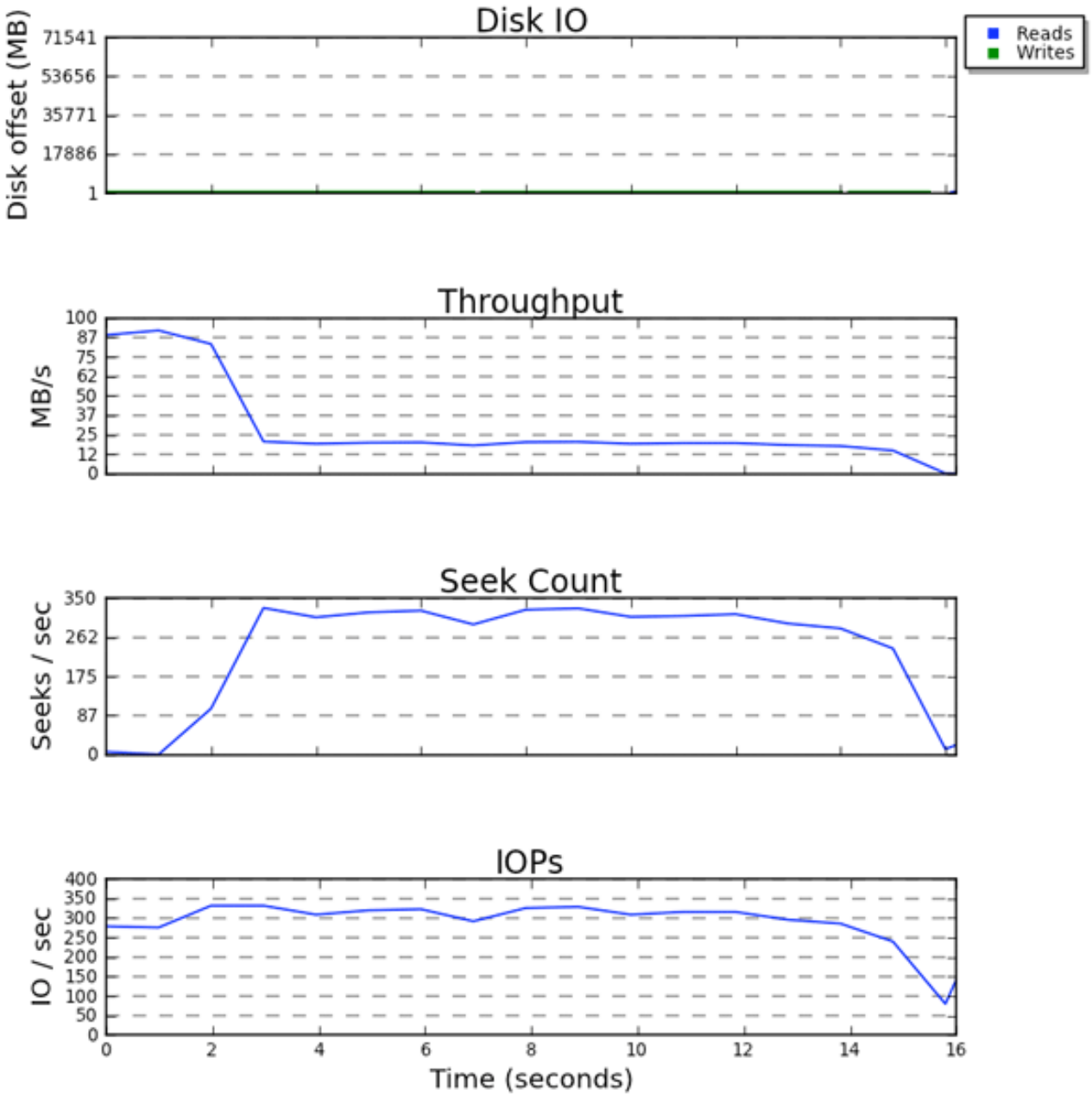
  Полученный запрос не может быть объединен с существующим и будет обслуживаться отдельно.
- M — запрос объединен с существующим.
- I — запрос добавлен в очередь к устройству;
- D — запрос передан устройству;
- C — запрос обработан;
- P — очередь к блочному устройству закрыта с целью объединения запросов;
- U — очередь к блочному устройству открыта, передача составных запросов устройству разрешена.

btt разбивает информацию по времени:

- Q2Q — интервал передачи запросов блочной подсистеме;
- Q2G — время от добавления операции в очередь блочного ввода-вывода до выдачи запроса обслуживания;
- G2I — время от выдачи запроса до его добавления в очередь устройства;
- Q2M — время с момента добавления операции в очередь блочного ввода-вывода до его слияния с существующим запросом;
- I2D — время нахождения запроса в очереди устройства (с момента поступления запроса в очередь до его передачи устройству);
- M2D — время с момента слияния запросов до их передачи устройству;
- D2C — время обслуживания запроса устройством;
- Q2C — общее время нахождения запроса в блочной подсистеме.

На основе данных из приведенной выше таблицы можно сделать много выводов. Например, если Q2Q значительно превышает Q2C, это значит, что приложение генерирует запросы ввода-вывода слишком медленно. Слишком высокое значение D2C означает, что устройство слишком долго обрабатывает запросы в силу слишком большой или неоптимальной нагрузки. Высокое значение Q2G сообщает о том, что в очередь одновременно поступает большое число запросов, и возможно, подсистема хранения не справляется с нагрузкой.

Еще одна утилита — **seekwatcher** — использует двоичные данные **blktrace** для построения диаграмм обработки ввода-вывода.



Avg Seeks/s	Avg MB/s	Avg IO/s	Run time (s)
252.57	31.72	305.99	16.21

Рисунок 6.2. Пример вывода seekwatcher

Горизонтальная ось содержит время. Кривые чтения и записи представлены разными цветами. Обратите внимание на зависимость производительности от скорости поиска данных — производительность накопителей, интенсивно выполняющих операции seek, снижается.

6.4. ПЛАНИРОВЩИКИ

Одним из первых решений при проектировании подсистемы ввода-вывода, которое вам надо будет принять, является выбор планировщика. В этой секции рассматриваются планировщики, позволяющие улучшить производительность подсистемы ввода-вывода.

6.4.1. CFQ

Планировщик (CFQ, Completely Fair Queuing) равномерно распределяет ресурсы между запросами в соответствии с их приоритетом. Процессы подразделяются на три класса: RT (realtime), BE (best effort) и бездействующие (idle). Приоритет может быть присвоен процессу вручную с помощью команды **`ionice`** или программно с помощью **`ioprio_set`**. По умолчанию процессы обслуживаются в порядке очереди (best effort). Процессы классов RT и BE дополнительно могут иметь приоритет от 0 до 7 (чем меньше значение, тем выше приоритет). Процессы класса RT имеют наивысший приоритет, поэтому нужно его использовать с большой осторожностью, потому что это может значительно ограничить время обслуживания менее приоритетных процессов. Процессы BE имеют приоритет 4. Бездействующие процессы (idle) обслуживаются в последнюю очередь.

CFQ выделяет процессам интервалы времени на обслуживание. Во время обслуживания по умолчанию процесс может отправлять 8 запросов ввода-вывода. Планировщик пытается предсказать число запросов исходя из прошлой статистики. Так, если ожидается поступление дополнительных запросов, CFQ перейдет в режим ожидания, даже если в это время другие процессы ожидают обработки.

Однако ожидание приводит к простою оборудования, что особенно ощутимо на высокоскоростных накопителях (SSD и внешних массивах даннхы). Поэтому при использовании CFQ может потребоваться откорректировать настройки в **`/sys/block/`*`устройство`*`/queue/iosched/`**.

```
slice_idle = 0
quantum = 64
group_idle = 1
```

Даже если ***`group_idle`***=1, вероятность простоев не исключена, но это будет происходить намного реже.

Как уже упоминалось выше, CFQ может находиться в режиме ожидания запросов, в то время как другие процессы ожидают обслуживания. Как следствие, использование нескольких таких планировщиков может значительно снизить производительность, как например, CFQ в комбинации с аппаратным RAID-контроллером. В этом случае RAID-контроллер может использовать собственный планировщик, что только ухудшит ситуацию. Дело в том, что планировщик нижнего уровня получает только те данные, которые проходят через планировщик верхнего уровня, что не отражает действительный уровень нагрузки.

Параметры

`back_seek_max`

Обратный поиск характеризуется более длительными задержками и негативно влияет на производительность. Этот параметр ограничивает расстояние для обратного поиска (в килобайтах). По умолчанию составляет **`16`** КБ.

`back_seek_penalty`

Так как обратный поиск характеризуется низкой эффективностью, за его выполнение будет назначаться штраф. Представим, например, что головка диска расположена на позиции 1024 КБ, а очередь обслуживания содержит два запроса — к позициям 1008 КБ и 1040 КБ. Оба

запроса одинаково удалены от текущего расположения. Однако с учетом штрафа на обратный поиск (по умолчанию 2), запрос обращения к позиции 1040 КБ будет выполнен первым.

fifo_expire_async

Максимальное время ожидания обслуживания асинхронной записи (через буфер). По умолчанию равно **250** миллисекундам. После его истечения один запрос будет перенесен в очередь для передачи.

fifo_expire_sync

Аналогично ***fifo_expire_async***, но для синхронных запросов. По умолчанию равно **125** мс.

group_idle

Переводит CFQ в режим ожидания после обслуживания последнего процесса в cgroup. Должен быть равен **1** при использовании равноценных групп cgroup, и при этом ***slice_idle*** равен **0**.

group_isolation

1 включает изоляцию групп, даже если при этом пострадает производительность. Изоляция групп позволяет равномерно обрабатывать нагрузку, которая поступает последовательно и случайно. Если изоляция отключена (**0**), нагрузка будет обрабатываться последовательно. Подробную информацию можно найти в **Documentation/cgroups/blkio-controller.txt**.

low_latency

Если равно **1** (по умолчанию), CFQ предоставляет процессам максимальное время ожидания (300 миллисекунд). Это позволяет обслуживать запросы более равномерно, но за счет снижения скорости. Значение **0** отключает ожидание.

quantum

Этот параметр контролирует число запросов ввода-вывода, передаваемых хранилищу за один раз, и тем самым ограничивает длину очереди. По умолчанию равен **8**. Следует соблюдать осторожность при увеличении значения, так как это может отрицательно повлиять на производительность при высоких уровнях нагрузки.

slice_async

Контролирует интервал времени, предоставляемый процессам, запрашивающим асинхронный ввод-вывод. По умолчанию равен **40** миллисекундам.

slice_idle

Определяет время ожидания поступления запросов. В Red Hat Enterprise Linux 6.1 и предыдущих выпусках равно **8** миллисекундам, а начиная с Red Hat Enterprise Linux 6.2 —**0**. Нулевое значение может ухудшить производительность внешних RAID-накопителей, так как возрастет общее число операций поиска. Для других типов накопителей рекомендуется использовать положительное значение.

slice_sync

Контролирует интервал времени, предоставляемый процессам, запрашивающему синхронный вводв-вывод. По умолчанию равен **100** миллисекундам.

6.4.2. Deadline

Планировщик Deadline ограничивает продолжительность задержки. Задержка измеряется с момента получения запроса планировщиком, что довольно важно, так как приложение может перейти в состояние бездействия при ожидании освобождения дескрипторов. По умолчанию операции чтения обслуживаются в первую очередь, так как вероятность блокирования приложений при выполнении чтения более высока.

Deadline обрабатывает запросы не отдельно, а в форме пакетов, содержащих последовательность запросов. После обработки пакета планировщик проверяет время ожидания запросов записи и принимает решение о том, стоит ли приступить к их обработке или получить новый пакет. В начале обработки пакета очередь FIFO будет проверяться только на предмет истекших запросов, и уже после этого будет выбрано направление (чтение или запись). Если была выбрана запись, но в то же время существует истекший запрос чтения, его обслуживание будет отложено до завершения записи.

Параметры

fifo_batch

Число операций чтения и записи в пакете. По умолчанию равно **16**. Увеличение значения может улучшить производительность, но и увеличить задержку.

front_merges

Значение **0** может быть присвоено, если вы уверены, что уровень нагрузки не будет требовать объединения запросов в начале очереди. В противном случае рекомендуется оставить **1**.

read_expire

Максимальное время ожидания обслуживания запроса чтения (в миллисекундах). По умолчанию равно **500** мс.

write_expire

Максимальное время ожидания обслуживания запроса записи (в миллисекундах). По умолчанию равно **5000** мс.

writes_starved

Определяет приоритет операций чтения, то есть число пакетов чтения, которые будут обслужены, прежде чем будет обработан пакет записи.

6.4.3. Noop

Noop помещает запросы в простую очередь типа FIFO (First In First Out). Слияние запросов происходит на блочном уровне. Этот планировщик особенно эффективен для высокоскоростных накопителей.

Ниже рассмотрены его параметры.

Параметры в /sys/block/sdX/queue

add_random

В некоторых случаях издержки обслуживания ввода-вывода для **/dev/random** измеримы. Тогда этот параметр рекомендуется обнулить.

max_sectors_kb

Этот параметр изменяет максимальный размер запроса и по умолчанию составляет **512** килобайт. Минимально допустимый размер ограничивается размером логического блока, а максимальный размер ограничивается значением ***max_hw_sectors_kb***. Производительность некоторых SSD-дисков может снизиться, если размер запросов превышает размер внутренних блоков SSD. В таких случаях рекомендуется уменьшить значение ***max_hw_sectors_kb*** до размера блока. Производительность на этом уровне можно протестировать с помощью **iozone** и **aio-stress**, изменяя размер записи, например с **512** Б до **1** МБ.

nomerges

Отключает слияние запросов, что используется для отладки. Слияние запросов способствует улучшению производительности, но его отключение поможет оценить число операций ввода-вывода, которое может быть обработано без необходимости отключения предварительного чтения или выполнения случайного ввода-вывода.

nr_requests

Число запросов чтения или записи в очереди. По умолчанию равно **128**, то есть очередь может содержать 128 запросов чтения и 128 запросов записи. Если очередь заполнена, процесс перейдет в состояние ожидания, а при освобождении места в очереди он будет пробужден.

При проектировании приложений, чувствительных к задержкам, рекомендуется уменьшить ***nr_requests***, тем самым ограничив длину очереди команд. После выделения числа запросов, определенных параметром ***nr_requests***, процессы, все еще ожидающие обслуживания, будут переведены в состояние ожидания. Это обеспечивает их равномерное обслуживание и не позволяет одному процессу использовать все ресурсы.

optimal_io_size

Иногда накопители предоставляют информацию об оптимальном размере сегмента ввода-вывода. Это типично для программных и аппаратных RAID, где оптимальный размер равен размеру сегмента чередования. Следует придерживаться значений, кратных рекомендуемым.

read_ahead_kb

Если приложение осуществляет чтение данных из файлов последовательно, операционная система может использовать алгоритм упреждения чтения. Этот алгоритм считывает больше данных, чем было запрошено, и помещает их в кэш, что экономит время на обращение к диску. Недостаток этого подхода состоит в том, что существует вероятность того, что данные не понадобятся, но они уже занимают место в кэше и будут вытеснены только тогда, когда нагрузка на память возрастет. В свою очередь, при наличии нескольких подобных процессов нагрузка на память возрастает значительно быстрее.

Для многопутевых устройств рекомендуется увеличить это значение, например до **8192** килобайт, так как они обычно включают в свой состав несколько физических устройств. Для начала можно присвоить ***read_ahead_kb*** значение **128** и корректировать его по мере необходимости.

rotational

SSD-диски не используют вращающиеся пластины в отличие от традиционных жестких дисков. Если устройство не определяет флаг, сообщающий о наличии движущихся деталей, может потребоваться обнулить параметр ***rotational*** вручную. Нулевое значение отключает использование алгоритмов снижения времени поиска данных, так как SSD-диски в этом не нуждаются.

rq_affinity

Значение **1** принуждает обработку операций на том же процессоре, где они были сгенерированы. Это может повысить эффективность кэширования данных.

ГЛАВА 7. ФАЙЛОВЫЕ СИСТЕМЫ

Эта глава содержит обзор файловых систем, поддерживаемых Red Hat Enterprise Linux, и способов оптимизации их производительности.

7.1. ХАРАКТЕРИСТИКИ ПРОИЗВОДИТЕЛЬНОСТИ ФАЙЛОВЫХ СИСТЕМ

Основные характеристики файловых систем, которые влияют на производительность включают параметры монтирования, выбранный метод форматирования, а также индивидуальные свойства приложений.

7.1.1. Параметры форматирования

Размер блока файловой системы

Размер блока может быть задан на стадии `mkfs`. При этом верхний предел определяется максимальным размером страниц, а нижний предел зависит от типа файловой системы. Обычно стандартного размера должно быть достаточно.

Если заранее известно, что в большинстве случае размер файлов будет меньше размера блока, можно уменьшить блок. Это позволит более эффективно использовать пространство, но в то же время увеличивает время обработки данных, особенно для больших файлов, занимающих несколько блоков.

Геометрия

Если в основе системы лежат дисковые массивы (например, RAID5), производительность может быть улучшена за счет выравнивания метаданных в соответствии с геометрией массива. Для программных RAID-массивов (LVM и MD) это будет сделано автоматически, в то время как в других ситуациях администратор должен будет определить геометрию вручную в строке команды `mkfs`.

Подробную информацию о файловых системах можно найти в *руководстве по управлению накопителями*.

Внешние журналы

В файловых системах с журналированием (ext4 и XFS) при интенсивном чтении и записи метаданных обращение к журналам возрастает. Чтобы сократить время поиска, можно разместить журнал на отдельном накопителе. Однако если его скорость меньше скорости диска файловой системы, выигрыш будет минимален.

ПРЕДУПРЕЖДЕНИЕ

Журналы должны храниться в надежном месте. Их потеря может привести к порче файловой системы.

Внешние журналы создаются на стадии **mkfs**, а устройства журналов определяются во время монтирования. За подробной информацией обратитесь к справочным страницам **mke2fs(8)**, **mkfs.xfs(8)** и **mount(8)**.

7.1.2. Параметры монтирования

Барьеры

Барьер записи — механизм ядра, отвечающий за порядок записи метаданных. При отключении энергии файловые системы с барьерами сохраняют состояние данных, переданных при помощи **fsync()**. Red Hat Enterprise Linux включает барьеры по умолчанию.

Активация барьеров может замедлить работу некоторых приложений, особенно тех, которые активно используют **fsync()** или часто создают и удаляют небольшие файлы. Для стабильных накопителей, не использующих непостоянный кэш, а также для некритичных приложений функциональность барьеров можно отключить с помощью параметра ***nobarrier***. Более подробно об этом рассказывается в *руководстве по управлению накопителями*.

Время доступа (noatime)

Обычно время доступа для чтения файлов (***atime***) можно изменить в метаданных дескриптора inode, что требует выполнения дополнительной операции записи. Если нет необходимости в точном определении ***atime***, при монтировании файловой системы можно добавить параметр ***noatime***. В большинстве случаев, однако, ***atime*** не добавляет значительную задержку, так в ядре Red Hat Enterprise Linux 6 уже по умолчанию включен режим ***relatime***. В этом режиме ***atime*** будет обновляться , если предыдущее время доступа меньше времени изменения файла (***mtime***) или времени изменения статуса (***ctime***).

ПРИМЕЧАНИЕ

При активации ***noatime*** будет автоматически включен параметр ***nodiratime***, поэтому отдельно его устанавливать не требуется.

Поддержка чтения с упреждением

Упреждающее чтение ускоряет доступ за счет предварительного чтения данных и размещения их в кэше страниц. При интенсивной нагрузке это может значительно улучшить производительность.

Утилита **tuned** и чередование LVM особенно эффективно используют функции чтения с упреждением, но этого не всегда бывает достаточно. Помимо этого, Red Hat Enterprise Linux не всегда может установить оптимальное значение для чтения с упреждением. Например, если высокопроизводительный массив представлен как единственный LUN, операционная система не будет его использовать как массив и не сможет воспользоваться всеми преимуществами чтения с упреждением.

Для просмотра настроек упреждающего чтения для указанного устройства выполните следующую команду:

```
# blockdev -getra устройство
```

Ниже это значение будет изменено (*N* — число блоков размером 512 байт).

```
# blockdev -setra N устройство
```

Заданное с помощью **blockdev** значение не будет сохраняться между перезагрузками. Рекомендуется создать сценарий **init.d**, чтобы восстановить его при запуске системы.

7.1.3. Обслуживание файловой системы

Удаление незанятых блоков

Блоки, не используемые файловой системой, можно освободить. Это помогает освободить пространство в файловых системах на SSD-дисках и в динамически созданных томах.

Операции удаления пакетов блоков выполняются пользователем: команда **fstrim** удаляет неиспользуемые блоки в соответствии с заданными критериями. Эти операции можно использовать в файловых системах XFS и ext4 в версиях Red Hat Enterprise Linux 6.2 и выше при условии, что они поддерживаются на уровне физических устройств. Параметр **/sys/block/*устройство*/queue/discard_max_bytes** поможет это проверить: если его значение не равно нулю, удаление поддерживается.

Чтобы разрешить удаление блоков, при монтировании файловой системы надо добавить параметр **-o discard** (или указать его в строке команды **mount** в **/etc/fstab**). Удаление будет выполняться без участия пользователя. Блоки, переходящие из занятого состояние в свободное, будут автоматически удаляться в ext4 (начиная с Red Hat Enterprise Linux 6.2) и в XFS (начиная с Red Hat Enterprise Linux 6.4).

Red Hat не рекомендует отключать эту функциональность, если в этом нет необходимости.

7.1.4. Производительность приложений

Выделение пространства

Файловые системы ext4, XFS позволяют заранее выделить пространство при помощи вызова **fallocate(2)**. Это поможет уменьшить фрагментацию, тем самым улучшив производительность. **fallocate(2)** отмечает заданный диапазон как выделенный и инициализирует его нулями.

7.2. ПРОФИЛИ ПРОИЗВОДИТЕЛЬНОСТИ

tuned-adm позволяет выбрать профиль для улучшения производительности системы. Некоторые профили будут рассмотрены ниже.

latency-performance

Профиль для коррекции задержки ответа сервера. Отключает механизмы **tuned** и **ktune**, изменяет режим **cpuspeed** на **performance**. Для контроля дискового ввода-вывода используется планировщик с алгоритмом **deadline**. В качестве обязательного значения **cpu_dma_latency** используется **0**.

throughput-performance

Профиль для коррекции производительности обработки данных. Рекомендуется при отсутствии доступа к большому хранилищу данных. Эквивалентно **latency-performance** за исключением:

- Минимальный интервал ***kernel.sched_min_granularity_ns*** равен **10** миллисекундам.
- Значение ***kernel.sched_wakeup_granularity_ns*** равно **15** миллисекундам.

- Значение ***vm.dirty_ratio*** равно 40%.
- Включены возможности прозрачного использования страниц большого размера.

enterprise-storage

Этот профиль рекомендуется для конфигурации пространства данных для крупных серверов уровня предприятия, что включает защиту кэша за счет переключения на питание от батареи и управление кэшем на диске. Аналогичен профилю **throughput-performance** за исключением следующих факторов:

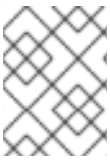
- значение ***readahead*** равно **4x**;
- файловые системы, которые не являются корневыми и загрузочными, монтируются с параметром **barrier=0**.

За дальнейшей информацией обратитесь к справочной странице **man tuned-adm** и к *руководству по управлению энергопотреблением* на сайте http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

7.3. ФАЙЛОВЫЕ СИСТЕМЫ

7.3.1. Ext4

Файловая система ext4 в Red Hat Enterprise Linux 6 используется по умолчанию. Red Hat Enterprise Linux 6 поддерживает файловые системы с максимальным размером 16 ТБ, файлы размером до 16 ТБ и снимает ограничение на число подкаталогов (равное 32000 в ext3).



ПРИМЕЧАНИЕ

Если размер файловой системы превышает 16 ТБ, рекомендуется выбрать другой тип, например XFS (см. Раздел 7.3.2, «XFS»).

В большинстве случаев стандартной конфигурации ext4 должно быть достаточно, но иногда может потребоваться откорректировать ее с целью повышения производительности.

Инициализация таблицы inode

В файловых системах большого размера инициализация таблиц inode в ходе работы **mkfs.ext4** может занять длительное время. Параметр ***-E lazy_itable_init=1*** может отложить его выполнение. При этом процессы ядра будут по-прежнему инициализировать файловую систему после ее монтирования. Скорость инициализации контролируется с помощью параметра ***-o init_itable=n*** команды **mount**. Значение **n** по умолчанию равно **10**.

Автоматическая синхронизация fsync

В некоторых случаях приложения не могут корректно выполнить синхронизацию **fsync()** после переименования файла, его изменения или перезаписи; в таких случаях ext4 будет по умолчанию синхронизировать файлы. Операции **fsync()** могут занимать много времени, поэтому не рекомендуется автоматически использовать этот вид синхронизации. Параметр ***-o noauto_da_alloc*** команды **mount** ее отключает, после чего синхронизацию надо будет выполнять вручную при помощи **fsync()**.

Приоритет ввода-вывода для журналов

Обычно операции обращения к журналам имеют более высокий приоритет по сравнению с обычным вводом-выводом. Параметр ***journal_ioprio=n*** команды **mount** контролирует это поведение. Допустимые значения включают от 0 до 7 (чем меньше значение, тем выше приоритет). По умолчанию ***journal_ioprio=3***.

Подробную информацию можно найти в справочных страницах **mkfs.ext4(8)**, **mount(8)** и в файле документации **Documentation/filesystems/ext4.txt** для пакета kernel-doc.

7.3.2. XFS

XFS — высокопроизводительная 64-разрядная файловая система, предназначенная для использования на дисках большого размера. Число файлов в XFS ограничивается лишь доступным пространством.

XFS включает возможности журналирования метаданных, что помогает ускорить процесс восстановления поврежденной файловой системы, допускает динамическое изменение размера и дефрагментацию подключенных файловых систем. Red Hat Enterprise Linux 6 дополнительно поддерживает специфичные для XFS утилиты резервного копирования и восстановления.

Особенности XFS включают предварительное и отложенное выделение места в форме экстентов. Использование экстентов облегчает отслеживание пространства и упрощает работу с большими файлами за счет уменьшения фрагментации. Отложенное выделение пространства гарантирует, что для файл будет записан последовательно, что положительно скажется на производительности. Предварительное выделение эффективно, если приложению заранее известно, сколько потребуется места для записи.

XFS характеризуется высоким уровнем масштабирования благодаря индексированию данных и метаданных в структуре B-дерева. Число объектов увеличивается, так как операции над индексами наследуют логарифмические характеристики масштабирования их B-деревьев. Некоторые параметры оптимизации в команде **mkfs** изменяют ширину дерева, что может изменить характеристики масштабирования подсистем.

7.3.2.1. Основы коррекции производительности XFS

Обычно стандартной конфигурации XFS должно быть достаточно, и Red Hat рекомендует ее придерживаться. В отдельных случаях можно ее изменить в соответствии с индивидуальными требованиями. Например, при наличии программного RAID-массива **mkfs.xfs** автоматически подберет размер сегмента чередования, но возможно, его надо будет определить вручную на аппаратных массивов.

При монтировании файловых систем большого размера рекомендуется добавить параметр **inode64**. Исключение составляют файловые системы NFS-сервера и устаревшие 32-разрядные клиенты NFS, которым нужен доступ к файловой системе.

Для интенсивно меняющихся файловых систем параметру **logbsize** рекомендуется присвоить значение 256 КБ (максимум). По умолчанию он равен **MAX** (32 КБ).

7.3.2.2. Дополнительные функции коррекции производительности XFS

Прежде чем приступить к изменению конфигурации XFS, необходимо точно знать, почему текущие настройки приводят к снижению производительности. Это включает понимание того, как работает приложение, и как файловая система реагирует на его действия.

Чаще всего производительность страдает из-за фрагментации и состязания за ресурсы. Существуют разные способы решения подобных задач оптимизации, включая коррекцию производительности на уровне приложений.

Если у вас нет опыта в этой области, рекомендуется обратиться в службу поддержки Red Hat.

Оптимизация для большого числа файлов

XFS косвенно накладывает ограничения на число файлов в файловой системе, но этот предел настолько высокий, что практически не может быть достигнут. Если заранее известно, что стандартного размера будет недостаточно, его можно увеличить с помощью **`mkfs.xfs`**. В свою очередь, размер существующей файловой системы можно нарастить при помощи **`xfs_growfs`**.

Оптимизация для большого числа файлов в одном каталоге

Размер блока каталогов имеет фиксированную величину, которая определяется в строке **`mkfs`**, и не может быть изменен. Минимальный размер равен размеру блока файловой системы, который по умолчанию равен **`MAX`** (4 КБ). Обычно в его уменьшении нет необходимости.

Так как схема каталогов построена на основе B-дерева, изменение размера блока повлияет на размер данных о каталогах, обрабатываемых операциями ввода-вывода. Если размер блока остается неизменным, то с ростом каталога число необходимых запросов ввода-вывода для обработки одной операции будет увеличиваться.

Однако при увеличении размера блока операции модификации будут использовать больше процессорных ресурсов. Увеличение размера блока для небольших каталогов может отрицательно сказаться на производительности, но при обращении к большим каталогам это может ускорить работу.

Стандартный размер блоков (4 КБ для файловой системы, 4 КБ для каталогов) подходит для каталогов с максимальным числом записей 1-2 миллиона с именами, длина которых составляет 20-40 байт. Если число записей каталогов достигает 10 миллионов, оптимальный размер блоков для файловой системы составляет 16 КБ, а начиная с 10 миллионов и выше — 64 КБ.

Если операции чтения каталогов выполняются намного чаще операций записи, вышеперечисленные значения будут на порядок меньше.

Параллелизм

В отличие от других файловых систем, XFS может параллельно выполнять операции выделения и освобождения экстентов и дескрипторов inode при условии, что они выполняются в разных линейных группах.

Число линейных групп имеет большое значение при выполнении многопоточных программ в многопроцессорных системах. При наличии лишь четырех групп степень параллелизма ограничивается этими группами. В больших файловых системах, размер которых исчисляется десятками терабайт, в ходе форматирования будет автоматически создано достаточное число групп.

Приложения, одновременно создающие и удаляющие большое число файлов, должны избегать размещения файлов в одном каталоге. Создаваемые каталоги помещаются в разные линейные группы, что упрощает масштабирование.

Оптимизация приложений с расширенными атрибутами

XFS может хранить некоторые атрибуты в дескрипторе inode. Чтение и изменение таких атрибутов осуществляется вместе с чтением дескриптора и не требует дополнительных операций, что положительно сказывается на производительности.

Так, стандартный дескриптор размером 256 байт может содержать примерно 100 байт атрибутов.

Увеличение размера inode во время создания файловой системы поможет увеличить пространство атрибутов. Так, в дескрипторе размером 512 байт будет доступно приблизительно 350 байт, а в дескрипторе размером 2 КБ — 1900 байт.

На размер отдельных атрибутов, которые могут размещаться внутри inode, накладываются ограничения. Так, максимальный размер атрибута и его значения составляет 254 байта. Если атрибут или его значение превышает этот размер, то он не может располагаться в inode.

Изменение метаданных

Изменение метаданных со временем приводит к росту журнала, размер которого служит своего рода критерием в определении допустимой частоты их изменения. Дело в том, что ведение журналов осуществляется циклически, то есть прежде чем новые данные смогут быть добавлены в журнал, существующие данные должны быть сохранены на диск. Поиск несохраненной информации и ее запись занимает некоторое время. Исходный размер журнала зависит от размера файловой системы и обычно не нуждается в изменении.

Небольшой размер устройства журналирования означает более частую запись метаданных в силу необходимости освобождения пространства для записи обновляемых метаданных. Это может привести к снижению скорости работы.

Увеличение размера журнала сократит частоту его записи на диск, что позволит лучше организовать записываемые метаданные. Однако недостатком является то, что для отслеживания всех изменений потребуется больше памяти.

Если объем памяти компьютера ограничен, использование больших журналов не рекомендуется. В этом случае небольшой размер журналов может быть более предпочтителен в силу более эффективной записи данных на диск.

Для устройств MD и DM команда **mkfs** по умолчанию размещает журнал в начале сегмента чередования массива, на основе которого построена файловая система. В свою очередь, для аппаратных RAID-массивов эту информацию надо будет определить. Такое расположение журнала исключает необходимость выполнения лишних операций при записи изменений на диск.

Настройки журналов можно корректировать при помощи параметров монтирования. Так, параметр **logbsize** позволяет изменить размер буфера в памяти. По умолчанию он равен **MAX** (32 КБ), а максимальный размер составляет 256 КБ. В большинстве случаев большой размер помогает улучшить производительность, но при интенсивном выполнении fsync буфер меньшего размера может оказаться эффективнее.

Параметр **delaylog** уменьшает число операций записи в журнал, накапливая изменения и сохраняя их за один раз. Несмотря на то что этот подход требует больше памяти для хранения изменений и увеличивает риск их потери в случае сбоя, он значительно улучшает скорость изменения метаданных и облегчает масштабирование. Этот параметр не нарушает целостность метаданных при выполнении **fsync**, **fdatasync** и **sync**.

7.4. КЛАСТЕРИЗАЦИЯ

Кластерное хранилище позволяет серверам в составе кластера обращаться к накопителям в его составе как к единому целому. Наличие единой файловой системы упрощает администрирование, так как отпадает необходимость в установке, обновлении приложений и поддержке резервных копий в разных файловых системах.

Комплект Red Hat High Availability в совокупности с Red Hat Global File System 2 предоставляет инструменты для управления кластерным хранилищем.

7.4.1. GFS 2

Файловая система Red Hat GFS2 (Global File System 2) напрямую взаимодействует с интерфейсом файловой системы ядра Linux и служит для организации совместного пространства хранения данных в кластере. Настройки GFS2 определяются автоматически, но можно корректировать их вручную, что и будет рассмотрено в этой секции.

В Red Hat Enterprise Linux 6.3 одновременная запись файлов разными процессами приводила к фрагментации, что замедляло работу, особенно при обслуживании больших файлов. В Red Hat Enterprise Linux 6.4 эта функциональность была оптимизирована.

Несмотря на то что Red Hat Enterprise Linux не предоставляет средства для дефрагментации GFS2, можно выполнить дефрагментацию отдельных файлов вручную. Команда **filefrag** поможет определить фрагментированные файлы, которые можно скопировать во временные и переименовать, заменив исходные файлы.

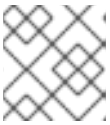
GFS2 использует глобальный механизм блокирования, требующий взаимодействия узлов в кластере. Чтобы сократить число обращений, при проектировании системы следует избегать ситуаций, приводящих к состязанию узлов за ресурсы. Для этого можно:

- Заранее выделить место для файлов и каталогов при помощи **`fallocate`**.
- Минимизировать области, которые будут использоваться совместно. Например, если несколько узлов монтируют одну и ту же файловую систему, но обращаются к разным каталогам, перемещение одного каталога в другую файловую систему поможет улучшить производительность.
- Выбрать оптимальный размер и число групп ресурсов в зависимости от среднего размера файлов и свободного пространства в системе. Это определяет вероятность одновременного обращения узлов к группе ресурсов. Но следует помнить, что слишком большое число групп может замедлить выделение блоков, в то время как недостаточное их количество приведет к конкуренции во время освобождения блоков. Обычно эти характеристики подбираются экспериментальным путем.

Другие способы повышения производительности GFS2:

- При выборе оборудования следует принимать во внимание требования файловой системы и особенности ввода-вывода кластерных узлов.
- Использовать SSD-диски.
- Подобрать размер файловой системы в соответствии с ожидаемым уровнем нагрузки таким образом, чтобы ее занятость не превышала 80%. Небольшие файловые системы не требуют много времени на проверку и резервное копирование, но при больших нагрузках подвергаются фрагментации.
- Для интенсивной нагрузки рекомендуется увеличить размер журналов. Несмотря на то что журналы будут использовать больше памяти, производительность улучшится, так как запись на диск будет осуществляться реже.
- Синхронизация часов на узлах GFS2 позволит избежать проблем в работе сетевых приложений. Для этой цели рекомендуется выбрать протокол NTP.

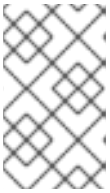
- Если время доступа к файлам и каталогам некритично, файловая система должна быть смонтирована с параметрами **noatime** и **nodiratime**.



ПРИМЕЧАНИЕ

Для GFS2 рекомендуется выбрать **noatime**.

- При наличии квот следует уменьшить частоту их синхронизации или предпочесть грубую синхронизацию.



ПРИМЕЧАНИЕ

Нестрогие квоты допускают превышение заданного порога. Чтобы его минимизировать, GFS2 будет динамически сокращать интервал синхронизации по мере приближения к предельному значению.

За дальнейшей информацией обратитесь к *руководству GFS2* по адресу http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

ГЛАВА 8. СЕТЕВОЕ ОКРУЖЕНИЕ

Сетевой стек Red Hat Enterprise Linux оптимизирован для разных уровней нагрузки, поэтому в большинстве ситуаций автоматическая конфигурация обеспечивает оптимальную производительность.

Обычно снижение производительности сети наблюдается при сбое оборудования или нарушениях функциональности инфраструктуры. Обсуждение подобных ситуаций выходит за рамки этого документа, поэтому далее будут рассмотрены методы оптимизации рабочих окружений.

Элементы сетевой подсистемы могут включать чувствительные соединения, поэтому и Red Hat, и открытое сообщество уделяют много внимания улучшению возможностей автоматической коррекции производительности.

8.1. ПРОИЗВОДИТЕЛЬНОСТЬ СЕТИ

Ниже перечислены основные функции оптимизации производительности сети в Red Hat Enterprise Linux 6.1.

8.1.1. RPS (Receive Packet Steering)

Механизм RPS распределяет нагрузку **softirq**, поступающую в очередь **rx** сетевой карты, между процессорами, что позволяет обслуживать запросы параллельно.

Для активации RPS необходимо указать имена процессоров в **/sys/class/net/*ethX*/queues/*rx-N*/rps_cpus**, заменив ***ethX*** именем сетевого устройства (например, **eth1**, **eth2**), а ***rx-N*** обозначением очереди. Пакеты, поступающие в очередь ***rx-N*** устройства ***ethX***, будут обрабатываться на заданных процессорах. При выборе процессоров следует учитывать схему *привязки кэша*[4].

8.1.2. RFS (Receive Flow Steering)

RFS является дополнением RPS и позволяет контролировать выбор ресурсов на основе хэш-таблицы. Таблица заполняется автоматически при обращении к приложениям из сети и регистрирует, на каких процессорах выполняются приложения, получающие пакеты из сети.

На основе этой информации для обслуживания пакетов можно выбрать тот же процессор, на котором выполняется приложение. Параметры конфигурации RFS включают:

/proc/sys/net/core/rps_sock_flow_entries

Максимальное число потоков, перенаправляемых заданному процессору. Это общесистемное значение.

/sys/class/net/*ethX*/queues/*rx-N*/rps_flow_cnt

Максимальное число потоков, направляемых в очередь ***rx-N*** устройства ***ethX***. Общее число потоков не может превышать **/proc/sys/net/core/rps_sock_flow_entries**.

RFS допускает совместное использование процессора при получении пакетов из очереди устройства и при обслуживании пакетов из приложения, что в некоторых случаях может улучшить производительность. Однако это зависит и от множества других факторов — иерархии кэша, нагрузки приложений и т.п.

8.1.3. Поддержка тонких потоков TCP в getsockopt

Термин *тонкий поток* используется для описания обработки пакетов небольшими блоками, что приводит к тому, что механизмы повторной передачи TCP не заполняются полностью. Такая передача используется приложениями, для которых время отклика имеет критическое значение: онлайн-играми, управляющими системами, системами биржевой торговли.

Потеря пакетов для таких приложений недопустима, поэтому вызов **getsockopt** теперь поддерживает два новых флага:

TCP_THIN_DUPACK

Разрешает динамический переход в режим обработки тонких потоков после получения одного подтверждения dupACK.

TCP_THIN_LINEAR_TIMEOUTS

Включает динамический таймаут для тонких потоков.

Оба параметра должны быть установлены приложением. За дальнейшей информацией обратитесь к **file:///usr/share/doc/kernel-doc-*версия*/Documentation/networking/ip-sysctl.txt** и **file:///usr/share/doc/kernel-doc-*версия*/Documentation/networking/tcp-thin.txt**.

8.1.4. Поддержка прозрачного прокси

Ядро разрешает подключение сокетов UDP и TCP к прозрачным прокси. Для этого надо будет настроить правила iptables и откорректировать правила маршрутизации.

Подробную информацию можно найти в **file:///usr/share/doc/kernel-doc-*версия*/Documentation/networking/tproxy.txt**.

8.2. ОПТИМИЗАЦИЯ ПАРАМЕТРОВ СЕТИ

Уровень производительности иногда корректируется экспериментальным путем: если изменение параметров не помогает достичь ожидаемого результата, администратор будет искать другие способы. За точку отсчета принимается стандартная конфигурация системы и подразумевается, что она может быть улучшена.

Как уже говорилось, сетевой стек сам оптимизирует свою работу, поэтому дальнейшие изменения требуют точного понимания его работы и требований ресурсов. Неверная конфигурация может только ухудшить производительность.

Например, увеличение длины очереди буфера может привести к возникновению заторов в обработке TCP-соединений. Такие соединения будут иметь слишком большие значения RTT, так как пакеты будут проводить слишком много времени в очереди. Это приведет к снижению скорости обработки.

Если возможно, рекомендуется использовать стандартные настройки и корректировать их, только если производительность сети заметно упала. В любом случае, прежде чем приступить к изменению конфигурации, надо тщательно изучить проблему.

Ниже перечислены инструменты, которые помогут выполнить анализ производительности сети.

netstat

Утилита командной строки, возвращающая информацию о соединениях, таблицах маршрутизации, замаскированных подключениях, многоадресных схемах и статистику интерфейсов из **/proc/net/**.

- **/proc/net/dev** (информация об устройстве)
- **/proc/net/tcp** (информация о TCP-сокете)
- **/proc/net/unix** (информация о сокете домена Unix)

Подробную информацию можно найти на справочной странице **man netstat**.

dropwatch

Утилита для отслеживания потери пакетов. Подробную информацию можно найти на справочной странице **man dropwatch**.

ip

Утилита для управления и наблюдения за устройствами, правилами маршрутизации и туннелями. Подробную информацию можно найти на справочной странице **man ip**.

ethtool

Утилита для просмотра и изменения настроек сетевой карты. Подробную информацию можно найти на справочной странице **man ethtool**.

/proc/net/snmp

Содержит управляющую информацию IP, ICMP, TCP, UDP для агента **snmp** в формате ASCII.

Документ под названием *Введение в SystemTap* содержит образцы сценариев для мониторинга и профилирования производительности сети. Его можно найти на странице документации по адресу http://access.redhat.com/site/documentation/Red_Hat_Enterprise_Linux/.

После сбора и анализа статистики сети можно принять решение по оптимизации ее работы.[5]Например, рост числа ошибок UDP в **/proc/net/snmp** служит знаком переполнения очереди сокетов.

Из этого можно сделать вывод, что либо очередь обрабатывает пакеты медленно, либо объем поступающих пакетов возрос. Проверьте журналы приложений, интенсивно обращающихся к сети, — если число поступающих запросов действительно превышает длину очереди, следует откорректировать настройки приложения.

8.2.1. Размер буфера сокета

Размер буфера поступающих и отправляемых данных сокета изменяется динамически, поэтому обычно в его изменении нет необходимости. Если же в ходе анализа производительности выяснилось, что скорость освобождения очереди сокета слишком низкая, можно увеличить ее длину. Для этого надо откорректировать следующие параметры:

rmem_default

Определяет размер буфера, который будет использоваться по умолчанию. Чтобы его изменить, выполните команду:

```
sysctl -w net.core.rmem_default=N
```

В этой команде ***N*** — размер буфера в байтах. Значение установленного параметра можно получить из **/proc/sys/net/core/rmem_default**. Стоит помнить, что стандартный размер не может быть больше **rmem_max** (в **/proc/sys/net/core/rmem_max**). Если же он должен быть больше текущего максимума, надо изменить **rmem_max**.

SO_RCVBUF

Параметр сокета, определяющий размер буфера поступающих данных. Подробную информацию можно найти на справочной странице **man 7 socket**.

Его значение можно просмотреть с помощью **getsockopt** и изменить с помощью **setsockopt**. Подробную информацию можно найти на справочной странице **man setsockopt**.

8.3. ОБЗОР ПОЛУЧЕНИЯ ПАКЕТОВ

Чтобы облегчить идентификацию слабых мест в сети, необходимо знать, как осуществляется передача и получение пакетов. Оптимизация получения пакетов имеет большое значение при коррекции работы сети, так как потеря пакетов чаще всего происходит именно на стадии их получения.

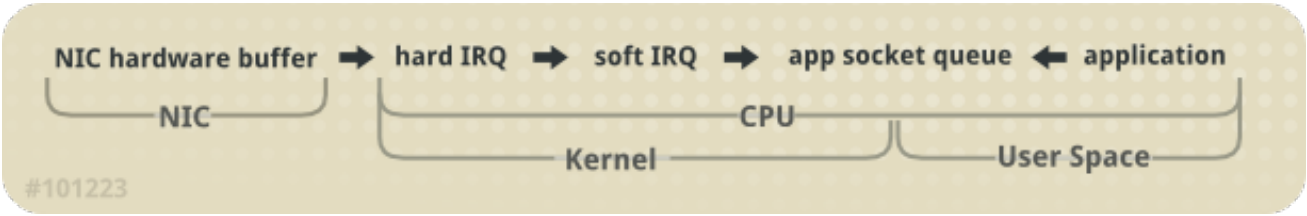


Рисунок 8.1. Прохождение данных из сети

Процесс получения пакета из сети можно разбить на несколько этапов.

1. *Оборудование получает пакет*. Сетевая карта получает пакет из сети. В зависимости от конфигурации драйвера он может быть помещен в буфер внутренней памяти устройства или в кольцевой буфер.

2. *Аппаратное прерывание*. Сетевая карта генерирует запрос прерывания, после чего будет назначено программное прерывание.

3. *Программное прерывание*. На этом этапе, собственно, и начинается процесс получения пакета в контексте **softirq**.

   Обслуживание прерывания на том же процессоре, где обрабатывается аппаратный запрос, минимизирует нагрузку. Ядро перемещает пакет из буфера в сетевой стек, откуда он может быть перенаправлен, удален или передан ожидающему сокету.

   Полученный сокетом пакет будет добавлен к приложению, привязанному к сокету. Этот процесс будет повторяться до тех пор, пока не будет достигнут предел **dev_weight**, или до тех пор, пока буфер сетевой карты не будет освобожден (см. Раздел 8.4.1, «Буфер сетевой карты»).

4. *Получение пакета приложением*. Приложение извлекает пакет из очереди сокета при помощи вызовов POSIX (**read**, **recv**, **recvfrom**). На этом этапе данные удаляются из сетевого стека.

8.3.1. Привязка процессоров и кэша

Для поддержки достаточной пропускной способности при получении пакетов необходимо минимизировать задержку ответа кэша L2. Как уже говорилось, пакеты из сетевого буфера обслуживаются на том же процессоре, что и соответствующее прерывание. Таким образом, данные, которые хранятся в буфере, будут использовать кэш L2 того же процессора.

Если заранее известно, что приложение будет получать большинство данных через сетевую карту, использующую кэш L2 определенного процессора, можно привязать приложение к этому процессору. Это увеличит шансы попадания в кэш, тем самым улучшив производительность.

8.4. ДИАГНОСТИКА ПОТЕРЬ ПАКЕТОВ

Наиболее распространенной причиной потери пакетов является переполнение очереди. Ядро накладывает ограничения на ее длину, и в некоторых случаях заполнение очереди происходит быстрее чем ее освобождение, что со временем приведет к переполнению и потере пакетов.

Рисунок 8.1, «Прохождение данных из сети» демонстрирует две основные очереди: аппаратный буфер сетевой карты и очередь сокета. Обе очереди должны быть настроены так, чтобы минимизировать риск переполнения.

8.4.1. Буфер сетевой карты

Пакеты, поступающие в буфер сетевой карты, извлекаются посредством **softirq**. Опросить состояние очереди можно следующим образом:

```
ethtool -S ethX
```

Замените ***ethX*** именем устройства. Эта команда вернет число потерянных на ***ethX*** пакетов, что обычно является следствием переполнения очереди.

К решению этой проблемы можно подойти с разных сторон.

Входящий трафик

Можно замедлить входящий трафик (добавить фильтры, уменьшить число многоадресных групп и т.п.)

Длина очереди

Можно увеличить длину очереди. При этом также потребуется увеличить число буферов в очереди. Для этого надо изменить параметры **rx** и **tx** для ***ethX***:

```
ethtool --set-ring ethX
```

Подробную информацию можно найти на справочной странице **man ethtool**.

Вес устройства

Скорость освобождения очереди можно контролировать с помощью весового значения устройства. Вес определяет максимальное число пакетов, получаемых сетевой картой в

контексте одного прерывания **softirq**. Его значение хранится в **/proc/sys/net/core/dev_weight**.

Обычно администраторы предпочитают последний вариант. Однако надо помнить, что увеличение числа пакетов, получаемых сетевой картой за один раз, означает увеличение процессорных циклов, во время выполнения которых процессор не будет обслуживать приложения.

8.4.2. Очередь сокета

Очередь к сокету заполняется сетевым стеком в ходе обработки **softirq**. Приложения получают данные из очереди при помощи вызовов **read**, **recvfrom** и т.п.

Состояние очереди может быть проверено с помощью **netstat**. Так, например, столбец **Recv-Q** содержит длину очереди. Переполнение очереди предотвращается уже рассмотренными способами (см. Раздел 8.4.1, «Буфер сетевой карты»).

Входящий трафик

Этот подход предусматривает снижение скорости поступления пакетов (добавить фильтры, заранее отбрасывать пакеты и т.п.). Дополнительно можно уменьшить весовое значение устройства[6].

Длина очереди

Чтобы увеличить длину очереди, надо изменить параметр ядра **rmem_default** или параметр сокета **SO_RCVBUF** (см. Раздел 8.2, «Оптимизация параметров сети»).

Частота программных вызовов

По возможности следует сократить число вызовов из приложения. Это можно сделать за счет увеличения вызовов POSIX (**recv**, **read** и т.п.).

Обычно администраторы предпочитают второй вариант, который предоставляет простое, но краткосрочное решение. С развитием сетевых технологий очереди сокетов будут заполняться все быстрее, что снова будет требовать изменения их длины.

Оптимальное решение заключается в изменении конфигурации приложения и увеличении скорости получения данных из ядра. Это обеспечивает более гибкое хранение данных, даже если это требует подкачки страниц.

8.5. МНОГОАДРЕСНАЯ РАССЫЛКА

Если группа многоадресной рассылки прослушивается несколькими приложениями, соответствующий код ядра должен дублировать сетевые данные для каждого сокета. Дублирование происходит в рамках **softirq** и занимает продолжительное время.

Чем больше приложений прослушивают группу многоадресной рассылки, тем ниже скорость обслуживания, так как это требует создания дополнительной копии каждого поступающего пакета.

При интенсивной нагрузке это может привести к потере пакетов в сетевом буфере и очереди сокета. Увеличение времени обслуживания **softirq** снижает число обслуживаемых приложений в загруженных системах, поэтому с увеличением потери пакетов увеличивается

число прослушивающих приложений.

Снизить вероятность потери пакетов можно за счет оптимизации очередей сокетов и сетевых буферов (см. Раздел 8.4.2, «Очередь сокета», Раздел 8.4.1, «Буфер сетевой карты»). Дополнительно можно оптимизировать использование сокетов приложениями и более эффективно распределять полученные данные между приложениями пространства пользователя.

---

[4] Привязка кэша процессора и сетевого устройства возможна за счет совместного использования кэша L2 (см. Раздел 8.3, «Обзор получения пакетов».

[5] Раздел 8.3, «Обзор получения пакетов» содержит информацию о передаче пакетов, что поможет в определении слабых мест в сетевом стеке.

[6] Вес устройства определяется в **`/proc/sys/net/core/dev_weight`** (см. Раздел 8.4.1, «Буфер сетевой карты»).

ПРИЛОЖЕНИЕ A. ИСТОРИЯ ПЕРЕИЗДАНИЯ

Издание 4.0-22.2.400 **2013-10-31** **Rüdiger Landmann**
Rebuild with publican 4.0.0

Издание 4.0-22.2 **Mon Jul 1 2013** **Yuliya Poyarkova**
Перевод на русский язык.

Издание 4.0-22.1 **Thu Apr 18 2013** **Chester Cheng**
Translation files synchronised with XML sources 4.0-22

Издание 4.0-22 **Fri Feb 15 2013** **Laura Bailey**
Опубликован в Red Hat Enterprise Linux 6.4.

Издание 4.0-19 **Wed Jan 16 2013** **Laura Bailey**
Проверка согласованности (BZ#868404).

Издание 4.0-18 **Tue Nov 27 2012** **Laura Bailey**
Опубликован в Red Hat Enterprise Linux 6.4 Beta.

Издание 4.0-17 **Mon Nov 19 2012** **Laura Bailey**
Обновлена секция numad (BZ#868404).

Издание 4.0-16 **Thu Nov 08 2012** **Laura Bailey**
Добавлен черновой вариант секции о numad (BZ#868404).

Издание 4.0-15 **Wed Oct 17 2012** **Laura Bailey**
Изменения и перенос секции в раздел параметров монтирования (Applying SME feedback to block discard discussion and moved section to under Mount Options (BZ#852990).
Обновлено описание профилей производительности (BZ#858220).

Издание 4.0-13 **Wed Oct 17 2012** **Laura Bailey**
Обновлено описание профилей производительности (BZ#858220).

Издание 4.0-12 **Tue Oct 16 2012** **Laura Bailey**
Улучшена навигация в документе (BZ#854082).
Откорректировано определение ***file-max*** (BZ#854094).
Откорректировано определение ***threads-max*** (BZ#856861).

Издание 4.0-9 **Tue Oct 9 2012** **Laura Bailey**
Добавлена рекомендация FSTRIM в главе файловых систем (BZ#852990).
Обновлено описание параметра ***threads-max*** (BZ#856861).
Обновлено примечание об оптимизации управления фрагментацией GFS2 (BZ#857782).

Издание 4.0-6 **Thu Oct 4 2012** **Laura Bailey**
Добавлена секция numastat (BZ#853274).

Издание 4.0-3 **Tue Sep 18 2012** **Laura Bailey**
Добавлено примечание о новых функциях perf (BZ#854082).
Откорректировано описание file-max (BZ#854094).

Издание 4.0-2 **Mon Sep 10 2012** **Laura Bailey**
Добавлена секция BTRFS и введение для файловой системы (BZ#852978).
Примечание об интеграции Valgrind с GDB (BZ#853279).

Издание 3.0-15 **Thursday March 22 2012** **Laura Bailey**

Добавлено и обновлено описание профилей tuned-adm (BZ#803552).

Издание 3.0-10 **Friday March 02 2012** **Laura Bailey**

Обновлено описание параметров threads-max и file-max (BZ#752825).

Обновлено исходное значение параметра slice_idle (BZ#785054).

Издание 3.0-8 **Thursday February 02 2012** **Laura Bailey**

Обновлена структура и добавлена дополнительная информация о taskset и сопоставлении процессоров и памяти для numactl (BZ#639784).

Откорректировано использование внутренних ссылок (BZ#786099).

Издание 3.0-5 **Tuesday January 17 2012** **Laura Bailey**

Раздел 5.3, «Профилирование памяти при помощи Valgrind» подверглась дополнительным изменениям (BZ#639793).

Издание 3.0-3 **Wednesday January 11 2012** **Laura Bailey**

Согласованы внешние и внутренние ссылки (BZ#752796).

Добавлен Раздел 5.3, «Профилирование памяти при помощи Valgrind» (BZ#639793).

Добавлен Раздел 4.1.2, «Коррекция производительности процессора» и обновлена Глава 4, *Процессор* (BZ#639784).

Издание 1.0-0 **Friday December 02 2011** **Laura Bailey**

Подготовлено к выпуску Red Hat Enterprise Linux 6.2.