Серия
Суперкомпьютерное
Образование

С. А. Жуматий, К. С. Стефанов

Суперкомпьютеры: администрирование

Суперкомпьютерный
консорциум
университетов России

12+

Константин Сергеевич Стефанов
Сергей Анатольевич Жуматий

Суперкомпьютеры: администрирование

http://www.litres.ru/pages/biblio_book/?art=60372236
SelfPub; 2020

Аннотация

Как стать администратором суперкомпьютера? Что нужно знать и уметь? Какие подводные камни ждут на этом нелёгком пути? В книге есть ответы на эти и некоторые другие вопросы. Материал поможет имеющим опыт системного администрирования повысить свою квалификацию, а тем, кто пока не имеет такого опыта, разобраться в том, что нужно изучить. Издание подготовлено при поддержке издательства МАКС-Пресс. ISBN 978-5-317-05877-7

Содержание

Введение

Здравствуй, читатель!

Эта книга написана для того, чтобы помочь начинающему или уже «продолжающему» системному администратору стать администратором вычислительного кластера или суперкомпьютера. Именно помочь, так как научить этому никакой книжке не под силу. Тем, у кого уже есть опыт администрирования Linux, учиться придётся меньше, но всё равно придётся обязательно. Тем, кто такого опыта не имеет, советуем почитать книги по администрированию Linux и потренироваться, например, на виртуальной машине. В этой книге мы коснёмся основ Linux, но лишь поверхностно.

Рассматривать будем **только** кластеры на базе Linux – это стандарт de-facto на настоящее время. Кластеры строят и на базе других ОС, например Windows, AIX и других, но здесь о них говорить не будем. Под суперкомпьютером мы понимаем вычислительный кластер, хотя большинство информации в этой книге применимо не только к кластерам. В тексте нами часто будет использоваться более широкое понятие – вычислительный комплекс. Все суперкомпьютеры разные, а уж кластеры и подавно – каждый со своими особенностями, требованиями и капризами. А значит, и навыки для каждого нужны свои.

Здесь мы собрали всё то, что, на наш взгляд, должно по-

мочь в обучении системного администратора суперкомпьютера. Конечно, только прочитав книгу, нельзя сразу же стать настоящим администратором суперкомпьютера, но знания, заложенные в ней, помогут стать им намного быстрее.

Нами даже не ставилась цель охватить весь спектр технологий, программ, архитектур, которые применяются в суперкомпьютерах. Это не только невозможно, но и бесполезно: они изменяются, устаревают, сменяются новыми с такой скоростью, что книга безнадёжно устарела бы уже через несколько лет. В мире суперкомпьютеров ещё больше, чем в мире IT в целом действуют законы Льюиса Кэрролла: нужно бежать со всех ног, чтобы только оставаться на месте, а чтобы куда-то попасть, надо бежать как минимум вдвое быстрее.

Наша задача – рассмотреть самые распространённые на момент написания книги технологии, чтобы дать понятие об основных принципах, приёмах работы с ними. Это позволит с небольшими затратами начать их использовать, изучить более глубоко, освоить более новые версии, а также совсем новые технологии, архитектуры, программы. Чтобы всё-таки дать хотя бы небольшую практическую базу, мы будем приводить самые важные примеры прямо в тексте, а в последних трёх главах сжато изложены инструкции, приёмы и справочные данные рассмотренным по технологиям.

Главное, что авторам хотелось бы показать в книге, это то, что суперкомпьютер – не просто набор серверов, коммутато-

ров, дисков… Это единый комплекс – не только идеологически, но и по сути. Все компоненты его тесно связаны, и самая важная задача администратора – понять, осознать эти связи, значение каждой и её влияние на комплекс в целом. Конечно же, этого нельзя сделать, не умея контролировать все части комплекса, поэтому надо изучить особенности (хотя бы основные) настройки и мониторинга всех компонент конкретного кластера. Однако не следует думать, что, запомнив значение всех «галочек» в административных интерфейсах всех «железок», можно получить полный контроль над суперкомпьютером. Поскольку масштаб даже небольшого вычислительного кластера значительно отличается от десятка серверов, настоятельно (очень настоятельно) рекомендуем отнестись к изучению возможностей командной строки. Если работать с десятком серверов в графическом режиме ещё можно, хотя и очень утомительно, то с сотней – уже просто нереально.

Как выяснить, на каких серверах определился не весь объём оперативной памяти при последнем включении? Запустить на каждом «системный монитор»? Зайти на вкладку «система» и посмотреть объём ОЗУ? На это уйдёт весь рабочий день. А вот выполнив на каждом узле, например с помощью `pdsh`, команду типа

```
grep MemTotal /proc/meminfo | awk '{print $2}'
```

можно получить этот самый объём ОЗУ за секунды. Доба-

вив ещё пару команд shell, можно сравнить полученное значение с эталоном (даже с учётом допусков) и выдать имена узлов, не прошедших проверку. Магия, вызываемая заклинанием? В чём-то – да, магия, но с понятными законами и вполне осваиваемая.

Нередко очень непростые действия можно выполнить с помощью комбинации стандартных команд. К счастью, это практически всегда возможно без большого труда. Труд потребуется для начального освоения этих команд, а потом – вся магия Linux будет в ваших руках! Очень советуем изучить «Advanced Bash Scripting Guide» (в Интернете есть хороший русский перевод). Это пособие позволит использовать огромную мощь инструмента, который всегда под рукой, – оболочки `bash` (практически всё работает и для `zsh`). Добавив в свой арсенал несколько простых приёмов `sed` и `awk` (а если захочется абсолютной магии, то и `perl`, а может быть, `python` или `ruby`), узнав возможности `find`, `ps` и подобных команд, вы многократно повысите эффективность своей работы.

В этой книге приведены также базовые знания о работе с командной строкой и основные понятия Linux, для того чтобы дать хороший старт тем, кто совсем с ними не знаком или знаком поверхностно. Без этих знаний и навыков невозможно понять работу системы в целом. Опытные администраторы Linux могут просмотреть эти главы бегло: для них там будет мало нового. А новичку они обязательны: нель-

зя быть администратором суперкомпьютера, не будучи хорошим Linux-администратором.

В книге затронута ещё одна важная тема, на первый взгляд не относящаяся к системному администрированию, – тема поддержки пользователей. Как известно, поддержка пользователей суперкомпьютера во многом отличается от поддержки пользователей компьютеров, которые работают в соседних комнатах. Мы постарались подготовить начинающего администратора к тем сложностям, которые ожидают его на этом пути. В каждой главе даны основные знания и понятия по той или иной теме. В некоторых из них раскрываются разные стороны одного и того же понятия. В конце каждой главы дано краткое резюме материала и ключевые слова для поиска в Интернете по теме главы.

Соглашения и обозначения, принятые в книге

Код скриптов, текст конфигурационных файлов выделяются так:

```
# Это текст скрипта
```

Предупреждения, важные моменты, о которых необходимо помнить:

Внимание! Не наступайте на одни грабли дважды!

Термины или важные понятия даны **полужирным** шрифтом.

Короткие команды выделяются в тексте так: `ls -la`.

Материал для новичков, который можно пропустить опытным специалистам, выделяется так:

Для начала работы включите компьютер в сеть.

В книге часто используются сокращения и распространённые термины. Для многих они привычны, для кого-то – пока нет. В отдельном словарике в конце пособия бóльшая их часть дана с расшифровками. Многие термины широко применяются как на русском, так и на английском языках. В основном нами используются русские варианты, а для того, чтобы не запутать читателя, в конце книги приведён список

соответствий терминов, где также даны и русские «кальки», так как, к сожалению, многие начинающие используют только их и не знают корректного перевода.

Свои отзывы и пожелания, пожалуйста, присылайте на адрес `superbook@parallel.ru`.

Авторы выражают искреннюю благодарность:

Владимиру Воеводину за идеи и критику,

Александру Наумову и **Антону Коржу** за предоставленный материал и консультации,

Виктору Дацюку, Павлу Костенецкому, Алексею Лацису и **Юрию Хребтову** за важные замечания.

Вадиму Кузнецову (dikbsd) за неоценимую помощь в конвертации книги в электронный формат.

Глава 1. Что же такое «супер»?

Общие понятия о параллельной обработке и параллельных программах

Все современные суперкомпьютеры используют параллельную обработку данных. С самого начала компьютерной эры именно этот путь был и остаётся наиболее важным для достижения высокой производительности. Сейчас даже самый простой настольный компьютер если не многоядерный, то уж почти наверняка с технологией simultaneous multithreading (например, HyperThreading от Intel или AMD SMT), позволяющей работать двум (иногда более) программным потокам одновременно. Даже мобильные телефоны и фотоаппараты становятся параллельными и многоядерными.

Принцип параллельной обработки данных прост: если две или более операции **независимы** (т. е. результаты их выполнения не влияют на входные данные друг друга), то эти операции можно выполнить одновременно, т. е. **параллельно**. В аппаратуре традиционно есть два варианта воплощения этого принципа – параллелизм и конвейеризация.

Параллелизм – параллельное выполнение машинных команд разными устройствами. Например, команды `x=a+b` и `y=b*c`, где `a`, `b` и `c` – регистры, процессор может выполнить независимо, если он имеет раздельные устройства сложения и умножения (см. рис. 1). Этот принцип воплощён в большинстве современных процессоров.

Конвейеризация – разделение команд на этапы, каждый из которых выполняется быстро отдельным элементом аппаратуры, и выполнение этих этапов происходит по принципу конвейера: друг за другом. Таким образом, одновременно может выполняться несколько команд на разных стадиях конвейера. Наиболее часто этот принцип используется в **векторизации** – выполнении однотипной операции над векторами, т. е. массивами данных, расположенными в памяти регулярно (см. рис. 2).

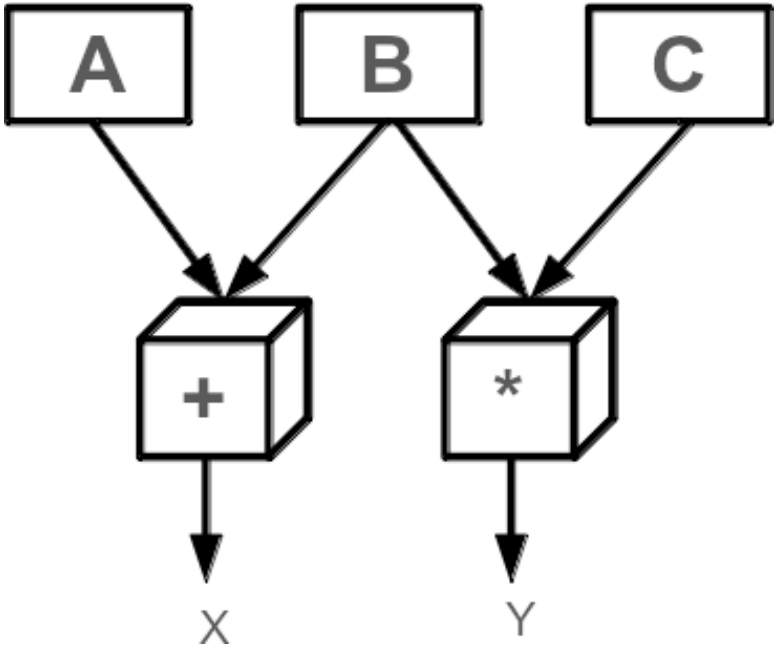

Рис. 1: Параллельное выполнение

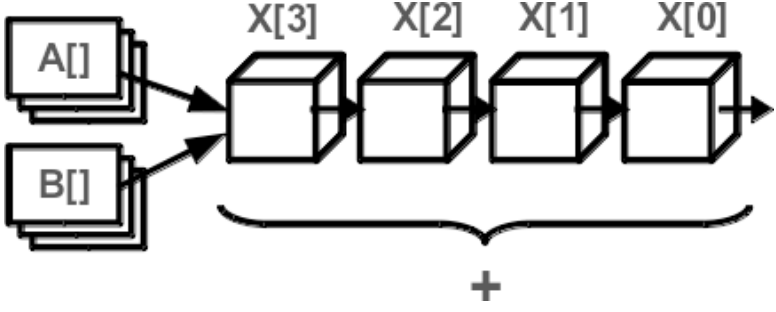

Рис. 2: Векторизация + конвейер

Чаще всего элементы вектора расположены друг за другом. Типичный пример – сложение векторов. Так как операция, выполняющаяся над векторами, одна и та же, то её разбивают на фазы – ступени конвейера. Например, загрузка элементов из памяти, нормализация мантисс, сложение, коррекция, запись в память. После выполнения первой ступени над первым элементом вектора эту стадию можно сразу же выполнить над вторым элементом, не дожидаясь завершения всей операции над первым. После завершения каждой ступени над одним элементом можно выполнить её над следующим. Таким образом, если самая медленная ступень конвейера выполняется за `K` тактов, а все ступени – за `S` тактов, то вектор из `N` элементов обработается за `K*(N-1)+S`.

Первый элемент обработается за положенные `S` тактов (это называется «время разгона конвейера»), а потом устройство будет выдавать по одному результату в `K` тактов. В современных процессорах чаще всего `K=1`. Однако конвейеризация не обязательно подразумевает векторизацию и наоборот. Например, если устройство сложения конвейерное и мы выполняем несколько обычных сложений подряд, они могут отлично задействовать конвейер.

Для системного администратора не всегда нужно доскональное понимание работы процессора, языка ассемблера и умение оптимизировать пользовательские программы, но

понимание того, как устроено параллельное выполнение в процессоре, очень важно. Многие современные процессоры многоядерные, т.е. содержат несколько полноценных (или почти полноценных) процессоров на одном кристалле (в одной микросхеме). Естественно, что они могут работать параллельно. Есть ещё параллелизм на уровне доступа к памяти, когда разные банки памяти могут работать независимо, а значит, отдавать или записывать данные быстрее. Есть также параллелизм на уровне работы с устройствами – можно заранее сформировать в памяти блок для записи на диск или для отправки по сети и «скомандовать» контроллеру выполнить запись/передачу. Далее процессор может выполнять другие действия, а данные из памяти в это время будут записываться на диск или пересылаться по сети.

Это параллелизм, заложенный в «железе». Для того, чтобы задействовать его максимально, заставить вычислители (ядра, процессоры, вычислительные узлы…) работать параллельно, необходимо составить программу таким образом, чтобы она использовала все эти ресурсы. Т. е. написать **параллельную программу**. Этим мы заниматься не будем (по крайней мере, в рамках этой книги), но иметь дело с параллельными программами нам придётся постоянно, и знать, как они работают, нам необходимо. [1]

[1] Можем порекомендовать отличную книгу по параллельному программированию: Антонов А.С. Технологии параллельного программирования MPI и OpenMP: учеб. пособие. Предисл.: В.А. Садовничий. – М.: Издательство Московского университета, 2012. – 344 с. – (Серия «Суперкомпьютерное образова-

Если я написал параллельную программу, значит ли это, что она тут же заработает быстрее обычной (последовательной)? Нет. Более того, она может работать даже медленнее. Ведь части, которые должны исполняться параллельно, в реальности могут конфликтовать друг с другом. Например, две нити обращаются к разным участкам памяти и не дают эффективно работать кэшу процессора. Или параллельные процессы постоянно вынуждены ждать данных от самого медленного из них. Или…

Вариантов неэффективного параллельного кода много, и если не удаётся достичь хорошего ускорения программы на суперкомпьютере, то, возможно, она неэффективно использует параллелизм. Выяснить реальную причину очень трудно, для этого необходимо использовать «отладчики производительности» – параллельные профилировщики, трассировщики или хотя бы мониторинг вычислительных узлов, по данным которого можно судить о том, что происходит во время работы программы.

Для нас в первую очередь интересен параллелизм на уровне процессов UNIX (в том числе на разных узлах) и нитей. Именно здесь заложен основной потенциал параллельных приложений. Именно такой параллелизм используют наиболее популярные среды параллельного программирования MPI и OpenMP. Это не значит, что на других уровнях этого потенциала нет, но именно отсюда всегда нужно начи-

---

ние»).

нать. Среды (технологии) параллельного программирования представляют собой библиотеки или языки программирования, позволяющие упростить написание параллельных программ.

Для администратора важно знать, как устроена каждая среда, как она реализована технически, так как при возникновении проблем с программами потребуется понять, в чём причина неполадки. Даже если причина – ошибка в программе, нужно уметь показать это пользователю и подсказать ему путь решения проблемы. Параллельные программы, как правило, пишутся в терминах нитей или параллельных процессов (или всего вместе). То есть один и тот же код программы выполняется в разных нитях одного процесса (на разных процессорных ядрах) или в разных процессах, которые могут работать и на разных узлах.

Такой подход позволяет максимально задействовать все процессорные ядра – на каждом выполняется свой процесс или поток. Действия разных процессов в одной программе необходимо согласовать, для этого в средах параллельного программирования предусмотрены разные механизмы: в MPI – передача сообщений, в OpenMP – общие переменные и автоматическое распараллеливание циклов и т. д. Технологии типа MPI – это не только указание особых функций и инструкций в коде, но и среда запуска программы. Особенно это относится к средам, использующим несколько вычислительных узлов, – ведь на каждом узле надо запустить

экземпляр (а то и не один) программы и «подружить» его с остальными запущенными экземплярами той же программы (но не соседней). Вот тут и начинаются заботы системного администратора. Все установленные параллельные среды надо оптимально настроить, а при возникновении проблем – уметь расшифровать их диагностику.

Например, в кластере используется скоростная коммуникационная сеть (InfiniBand или другая) и обычный Ethernet для управления. Установленная среда MPI работает, но эффективность работы программ низкая. Нередко причиной является неверная настройка, в результате которой MPI использует медленную управляющую сеть вместо скоростной.

Виды кластеров

Когда говорят «кластер», подразумевают множество компьютеров, объединённых в нечто единое. Но вариантов этого «нечто» может быть несколько. Они отличаются целью и, как следствие, – реализацией.

Первый вид кластеров – **High-Availability**, или кластеры высокой доступности. Их задача – предоставить доступ к какому-то ресурсу с максимальной скоростью и минимальной задержкой. Ресурсом обычно выступают web-сайт, база данных или другой сервис. В таком кластере при выходе из строя одного узла работоспособность всего ресурса сохраняется – клиенты сбойного узла переподключаются и получают доступ к ресурсу с другого узла кластера. Очень похожий принцип применяется в «облачных» технологиях: вы не знаете, на каком именно узле будет работать ваше приложение или образ операционной системы, облако само подберёт свободные ресурсы.

Другой вид кластеров – **High Productivity**. Этот тип похож на предыдущий, но в данном случае все узлы кластера уже работают над одним заданием, разбитым на части. Если какой-то узел отказал, его часть задания отправляется другому; если в кластер добавляются новые узлы, им выделяются не посчитанные ещё части, и общий счёт идёт быстрее. В качестве примеров можно назвать GRID, программы типа

Seti@home, Folding@Home. Однако с помощью таких кластеров может быть решён только узкий класс задач. Да и сам кластер для таких задач нередко становится не нужен, можно воспользоваться домашними компьютерами или серверами, связав их через локальную сеть или Интернет.

Третий вид – **High Performance** (HPC – High Performance Computing). Именно он интересен нам. В отличие от остальных, выход из строя одного из узлов кластера, как правило, ведёт к аварийному завершению параллельной программы, только в редких случаях выполнение программы автоматически продолжается с сохранённой ранее контрольной точки. Именно поэтому, в отличие от предыдущих видов, HPC-кластеры менее устойчивы в работе, и без должного контроля и мониторинга использовать их просто не получится.

Важное отличие этого вида кластеров от остальных – тесная связность всех узлов. Это и самые быстрые сети, соединяющие узлы, и высокопроизводительные параллельные файловые системы, и средства дополнительной синхронизации узлов, и другие средства, важные для параллельных программ. Приложения, работающие на таких кластерах, как правило, работают в модели передачи сообщений между параллельно запущенными процессами. Если запустить их на множестве компьютеров, соединённых медленной сетью, то они бóльшую часть времени потратят на ожидание информации друг от друга.

Идеал, к которой стремятся все производители класте-

ров, – создать виртуальный компьютер с большой памятью и огромным числом вычислительных ядер. К сожалению, реальность ещё очень далека от идеала, и сейчас любой вычислительный кластер – это всё-таки множество отдельных вычислительных узлов, соединённых быстрой сетью. От сети в таком кластере требуется не только скорость (пропускная способность), но и низкая величина задержек или накладных расходов (латентность). Большинство параллельных программ обмениваются сообщениями часто, а значит, время на инициализацию отправки и приёма сообщения начинает играть большую роль. На сети с большой латентностью некоторые программы могут работать в разы медленнее, чем на сети, где латентность низкая.

Кластеры и суперкомпьютеры – общее и разное

Мы только что поговорили о кластерах. Но всегда ли слово «суперкомпьютер» означает кластер? Нет, не всегда. Важная черта кластера – возможность сборки из серийных общедоступных компонентов. Т. е. можно купить все компоненты кластера в магазине и, обладая достаточным опытом, собрать его самостоятельно.

Суперкомпьютер в общем случае – изделие с уникальными компонентами, производимое одним поставщиком. В качестве примера приведём серию Blue Gene компании IBM – архитектура этих машин похожа на кластер, на них доступны те же программные средства, что и на вычислительных кластерах, но купить Blue Gene можно только у IBM или их дистрибьюторов.

Построить Blue Gene самостоятельно невозможно: ключевые компоненты отдельно не продаются. И дело не в марке, а в уникальных технологиях. Кроме Blue Gene есть множество иных серий, иных уникальных разработок. Обратный пример – «вычислительные фермы», т. е. группы компьютеров, работающих над одной задачей, но обычно даже не передающие данные друг другу, или кластеры класса «BeoWulf[2]»,

[2] Подробнее см. http://parallel.ru/computers/reviews/beowulf.html.

т. е. собранные практически из подручных средств.

Как видим, грань между понятиями «кластер» и «не-кластер» достаточно чёткая, но какой кластер считать суперкомпьютером, а какой нет – вопрос размытый. Часто вместо «кластер» говорят более тактично: «обладающий кластерной архитектурой». В этой книге мы будем рассматривать технологии, доступные для всех или большинства. Следовательно, большинство из них будет относится именно к кластерам. Но это не значит, что в вычислительных комплексах, которые мы формально не относим к кластерам, этих технологий не встретится. Большинство современных суперкомпьютеров используют те же наработки, что и кластеры, более того, почти все они построены как кластеры с добавлением особо быстрых сетей, техник работы с общей памятью, синхронизации или иных технологий. А значит, все знания о кластерах вам только помогут.

Что означает «супер» для администратора суперкомпьютера

На первый взгляд, большой кластер ничем не отличается от множества офисных компьютеров, объединённых локальной сетью, и нескольких стандартных серверов – дискового хранилища и т. п. На самом деле отличия есть, и очень важные. Начнём с оборудования – для кластера требования намного выше. Если в локальной сети можно временно заменить сломанный коммутатор на более простой или даже на несколько дней нарушить связность сети (ну, придётся отчёты печатать на втором этаже, потерпите), то в кластере это недопустимо. Заменив IB-коммутатор на GigabitEthernet или узел с 8ГБ памяти на узел с 4ГБ, мы получим неработающий кластер или работающий так, что все пользователи завалят нас жалобами.

Настоятельно рекомендуем иметь ЗИП (аварийный запас) всех ключевых компонент оборудования, если у них нет аппаратного дублирования, и сервисный договор о замене оборудования в чётко оговорённый срок.

Ещё вспомним о том, что кластер, в отличие от офисных компьютеров, упакован на нескольких квадратных метрах (большой – на нескольких десятках, реже – сотнях). Поэтому требования к охлаждению для него намного выше, тут открытым окном или бытовым кондиционером не обойтись.

Электричества на суперкомпьютер уходит гораздо больше, чем на много офисных ПК, и бытовых UPS тут тоже не хватит, да и в бытовую розетку и даже в десяток его не включишь.

В современных кластерах вычислительная часть может занимать меньше четверти от всей площади установки, всё остальное занимает климатическое и энергетическое оборудование. А контроль и управление этим оборудованием (но не обслуживание) – тоже часть работы администратора. Более того, в отличие от офиса, если вычислительный узел, кондиционер или UPS вышли из строя, то об этом нельзя узнать от прибежавшего сотрудника, у которого «горит отчёт, а монитор не включается». Хуже всего, если об этом придётся узнать от пользователей, у которых программа перестала работать как надо или запускается два раза из трёх. Эту задачу решает мониторинг всего и вся. Очень важно знать как можно больше о состоянии кластера. На этом отличия не заканчиваются. Одно из самых важных связано с режимом работы. В офисе нагрузка на компьютеры не высока: большая мощность от них требуется несколько минут в день, чтобы отобразить большой документ или проиграть видеоролик новой рекламы продукта. 99% времени эти компьютеры ждут клика мышкой или нажатия на клавишу. В кластере всё принципиально иначе, его нормальный режим работы – 80–100% загрузки каждого узла постоянно.

В офисе даже пиковая нагрузка одного или двух компью-

теров не будет заметна на общем фоне. Но каждый опытный администратор знает, что такое «все компьютеры схватили какой-то вирус» – нагрузка на сеть возрастает в сотни раз, сетевое хранилище не справляется с потоком запросов, всё начинает жутко «тормозить»… А в кластере ситуация, когда все узлы, занятые под одно задание, начинают обмениваться данными или писать промежуточные данные на сетевой диск – это не вирус, а совершенно нормальная ситуация. Особый тип пиковой нагрузки – включение. В офисе всё происходит само собой: утром все приходят, кто-то пораньше, кто-то попозже, включают компьютеры, подключают ноутбуки… Для суперкомпьютера же процедура включения означает резкое увеличение энергопотребления на десятки, а то и тысячи киловатт, дружное обращение вычислительных узлов к дисковому хранилищу, сервисным серверам. Если включить всё разом, то, скорее всего, установка просто сгорит. И даже «плавное» включение узлов одного за одним с интервалом в несколько секунд может привести к сетевым конфликтам, перегрузке какого-то сервиса запросами.

Для примера: в больших дисковых массивах (из нескольких стоек) полки и диски запускаются поочерёдно в определённой последовательности не только из-за больших пусковых токов, а ещё и для того, чтобы не раскачивалась стойка от раскручивающихся дисков. Другой пример: серверы организованы в коридор – стойки стоят напротив друг друга

и серверы выдувают горячий воздух внутрь получившегося коридора, тогда и включать их надо пáрами, чтобы не перегреть ещё не включённые серверы.

Многое зависит от того, как спроектирован конкретный суперкомпьютер, поэтому хорошо изучите его структуру и процедуру старта. Конечно, эти и другие проблемы касаются и больших офисов, но в кластере они возрастают многократно. Все эти проблемы решаемы с той или иной степенью эффективности, но нередко методы их решения отличаются от «офисных». Во многом всё зависит от оборудования – при планировании суперкомпьютера очень важно помнить о пиковых нагрузках. Тут они – серая повседневность, поэтому изначально надо закладывать решения, позволяющие их выдерживать.

Кроме чисто аппаратных решений важны и программные: если один ключевой сервис поставить даже на супермощный сервер, то он всё равно может не справиться с нагрузкой, и, возможно, надо подумать о дублировании или разделении нагрузки. Если же при планировании по какой-то причине не удалось всё учесть и под нашей опекой оказался кластер с «бутылочным горлышком», то нужно суметь найти способ расширить или совсем ликвидировать это горлышко, например, заменив часть оборудования и/или программного обеспечения, но это обычно непросто.

Централизованное управление вычислительным комплексом

Как мы увидим далее, аспектов управления вычислительным комплексом масштаба вычислительного кластера очень много. Тут и развёртывание системы, и обновление ПО, и контроль учётных записей, удалённого доступа, управление доступом и заданиями, мониторинг, резервное копирование и многое другое. Каждая из этих задач по отдельности решается, а как – мы покажем в этой книге. Однако объём действий, совершаемый администратором при массовых операциях, таких, как заведение групп пользователей с присвоением им специфических прав, изменения в настройках сетевых устройств или узлов, становится весьма внушительным.

Тут на помощь вам придут знания скриптовых языков – большинство таких действий автоматизируется скриптами. Но, к сожалению, не все действия можно выполнить набором скриптов. В нелёгкие будни системный администратор большого вычислительного комплекса всё чаще задумывается об удобной «консоли», в которой можно сделать всё, что требуется, не запуская лишних программ, скриптов, не копируя промежуточных файлов и текста с экрана терминала. Особенно часто такие мысли возникают при виде продуктов типа HP OpenView или Zenoss. «Вот она – панацея!» – так и хочется воскликнуть. И правда, такие продукты нацелены на

решение очень похожих задач. Они сами инвентаризируют оборудование, ведут учёт пользователей и ПО, делают массу автоматизированных действий… Более того, их действительно можно (а если есть возможность, то и нужно) приспособить для решения части ваших задач.

Увы, лишь части. Подобные продукты, как коммерческие, так и свободные, нацелены именно на похожие, пересекающиеся с нашими, но всё же другие задачи. Заставить их делать то, чего они не умеют, но что нужно нам, иногда можно, но это требует колоссальных затрат – людских, финансовых, времени… А как только конфигурация вашего суперкомпьютера поменяется, всё это придётся делать заново. По нашему личному опыту и опыту многих администраторов суперкомпьютеров, с которыми мы общались, универсальных решений нет. К сожалению, создание таких инструментов востребовано только узким кругом администраторов, при этом оно дорого в разработке и поддержке. Именно поэтому мы хотим обратить ваше внимание на важность системного подхода ко всем учётным и организационным действиям с вычислительным комплексом. Однако это не значит, что нужно выбирать как можно более интегрированные решения. Это значит, что все действия должны быть хорошо описаны – не для того, чтобы не забыть, а для того, чтобы видеть картину в целом и в изменившихся условиях быстро адаптировать к ним устоявшиеся процессы.

Старайтесь использовать гибкие и расширяемые инстру-

менты. И не забывайте учиться новому, применять адекватные (а не только самые модные) технологии к решению всего комплекса задач администрирования суперкомпьютера!

Краткое резюме

Суперкомпьютер очень похож на «много-много обычных серверов», но в то же время особенностей работы с ним намного больше, чем с множеством серверов. Очень многие серверные технологии тут используются для решения стандартных задач, но не для всех они применимы. Кроме того, есть множество специфичных задач и технологий, применяемых только в области супервычислений.

Ключевые слова для поиска

`HPC, beowulf, supercomputer.`

Глава 2. Как устроен суперкомпьютер

Рассмотрим «анатомию» вычислительного кластера: из каких компонент он состоит? В зависимости от размера и архитектуры конкретного кластера некоторые компоненты могут объединяться. Далее мы часто будем писать «узел» – это синоним слова «сервер», но в HPC так принято.

Итак, обязательная часть любого кластера – **вычислительные узлы**, или так называемое **счётное поле**. Это серверы, на которых будут считаться задания. Кроме вычислительных узлов должен быть как минимум один **управляющий узел**, в больших системах к нему добавляются дополнительные **служебные узлы**, их может быть несколько десятков. Для эффективной совместной работы вычислительных узлов необходимы **сети**:

• **коммуникационная**, по которой происходит обмен данными вычислительных заданий;

• **управляющая**, по которой происходит удалённый доступ на узлы, запуск заданий и т. п.;

• одна или несколько **служебных** – для доступа к сетевой файловой системе, управления через протоколы IPMI или iKVM, дополнительной синхронизации (прерываний, тактовой частоты, барьеров и т. п.) и, возможно, другие.

Обязательный компонент современного вычислительного кластера – **сетевая файловая система**.

Для работы всего комплекса обязательно необходимо наличие **инфраструктуры**: систем энергообеспечения, климатических систем. Для большой установки они могут занимать в несколько раз больше места, чем вычислительные узлы. Как правило, обслуживание инфраструктуры не входит в обязанности администратора, но он должен по возможности осуществлять контроль её состояния.

Управляющий узел

Все узлы любого кластера делятся на вычислительные и служебные. Один служебный узел присутствует всегда – это управляющий узел. Именно с него выполняется управление всеми подсистемами (или с него выполняется вход в управление ими), как правило, на него же попадают пользователи по ssh. В небольших кластерах он может совмещать функции всех служебных серверов.

Вычислительный узел

«Рабочая лошадка» кластера – счётное поле. Как правило, тут все узлы одинаковой конфигурации, но иногда в поле могут входить узлы двух и более конфигураций. Чем однороднее состав вычислительных узлов, тем проще ими управлять, тем проще работать планировщику. Создавать смешанные конфигурации стоит только в тех случаях, когда вы уверены, что все(!) они будут активно использоваться заданиями.

Аппаратная начинка вычислительного узла полностью определяется характером заданий, которые будут решаться на кластере, но всегда нужно стараться сбалансировать состав «железа», чтобы не возникло узких мест, например, таких, как большое число ядер при узком канале в память, недостаточная ширина канала в вычислительную сеть и т. п. Наличие жёсткого диска имеет как плюсы, так и минусы. Минусы – дополнительное место и энергопотребление с тепловыделением, а также высокая вероятность выхода из строя. В блейд-конфигурациях всё это особенно актуально. Плюсы – возможность установить локальную копию ОС, что сильно упрощает процедуру включения, ускоряет загрузку системных библиотек (а значит, и старт программ), а также возможность добавить swap-пространство и локальный каталог /tmp. Это значительно повышает эффективность работы

памяти.

При установке локальной копии ОС следует быть очень осторожным при обновлениях ПО и локальном хранении учётных данных. Для повышения эффективности конфигурация ПО должна быть максимально облегчена: чем меньше лишних сервисов, тем лучше.

На вычислительном узле вполне можно отказаться от таких сервисов, как **почта** (можно отправлять сообщения через головной узел), **cron** (самые важные задания можно выполнять по ssh также с головного узла), **udev**, **acpid** и т. п. Оставьте только самые необходимые, а вместо udev, если возможно, используйте заранее созданные файлы устройств – они всё равно не будут меняться со временем. Самые важные сервисы для вычислительного узла – sshd и клиент сетевой файловой системы. Очень желательно настроить мониторинг работы узла. В некоторых современных дистрибутивах отключить udev невозможно: от него зависят важные сервисы (**systemd**, например). В этом случае оставьте его, не пытайтесь «обмануть» систему. Как правило, все вычислительные узлы логически объединяются в разделы (или очереди) в рамках системы управления заданиями. Если в поле есть узлы разных конфигураций, то удобно создать разделы для каждой конфигурации отдельно. Иногда бывает полезным объединить несколько вычислительных узлов в один раздел для запуска небольших тестовых заданий (тестовая очередь), при этом полезно ограничить время счёта таких

тестовых заданий (например, 15–20 минут).

Служебные узлы

Все узлы, не включённые в счётное поле, – служебные. Совмещать функции вычислительного и служебного узла (например, NFS-сервера) крайне не рекомендуется, так как это наверняка приведёт к разбалансировке работы заданий и повышению вероятности отказа сервиса. Существует несколько ролей, которые выполняют служебные узлы, но часто один сервер выполняет несколько ролей, а то и все сразу.

Рассмотрим типичные роли. В больших вычислительных комплексах не всегда бывает удобно нагружать управляющие узлы пользовательскими и служебными системными процессами. Например, если установлены вычислительные узлы с разными версиями операционных систем, то совсем неудобно производить сборку пользовательских программ на управляющем узле, логичнее выделить несколько узлов для компиляции программ (**узлы компиляции**).

Для защиты от несанкционированного доступа к системным службам и чувствительным данным (например, база данных паролей пользователей) обычно функции управляющих узлов разносят на две группы: узлы доступа и узлы управления. **Узлы доступа** предназначены для входа пользователей и их дальнейшей работы в системе, а **узлы управления** – для работы системы управления заданиями.

Практически в любом кластере есть сетевая **файловая система**, а значит, и сервер для неё, а нередко – целая ферма, если файловая система распределённая. Довольно распространённым служебным узлом является **лицензионный сервер**, на котором располагаются специальные службы, отвечающие за лицензирование коммерческих программ и утилит. Например, может использоваться сервер лицензий FlexLM для нескольких коммерческих пакетов. Расположение лицензионных служб на отдельной машине оправдано как с точки зрения безопасности (защита от кражи лицензионных файлов), так и с точки зрения повышения отказоустойчивости комплекса в целом. Обязательно запишите MAC-адрес этого сервера, при его внезапной замене для большинства программ будет достаточно установить на новом сервере старый MAC-адрес. И не забудьте запросить перевыпуск лицензии для нового сервера, конечно, с его настоящим MAC-адресом.

В современных вычислительных комплексах довольно часто встречаются **узлы подготовки входных и обработки выходных данных** (так называемые узлы пред/постобработки, от англ. pre- и postprocessing). Такие узлы отличаются бóльшим объёмом оперативной памяти, чем на остальных узлах (256 Гбайт и более), что крайне важно для подготовки больши́х заданий и обработки результатов расчётов.

Часто полезными являются так называемые **узлы визуализации**. Обычно это выделенные серверы со специаль-

ными графическими картами для обработки визуальной информации и выдачи готовой картинки через сеть удалённому пользователю. Это бывает удобным, в частности, для удалённой подготовки заданий к расчёту (например, для визуализации сеток и иных входных данных). Узлы визуализации могут играть роль узлов пре/постобработки.

Для организации распределённого хранилища данных могут быть использованы **узлы хранения данных**. К каждому такому узлу подключается своё собственное дисковое хранилище, а все узлы хранения объединяются в единую сеть с общим доступом к файловой системе со всех узлов (подробнее об этом – в следующем разделе).

Среди служебных узлов также могут быть выделенные узлы:

- резервного копирования;
- удалённой загрузки;
- развёртывания ПО;
- авторизации и аутентификации;
- удалённого журналирования;
- сбора и обработки данных мониторинга;
- сбора и отображения статистики и состояния оборудования;
- служебных баз данных;
- и др.

Всё зависит от того, какие нужды у пользователей и администраторов вычислительного комплекса.

Сетевое оборудование

Компьютерные сети позволяют организовать взаимодействия компьютеров между собой. Для их построения применяется специальное оборудование: это сетевые карты и коммутаторы. В кластерах, как правило, имеются как минимум две сети. Одна, называемая служебной, выполняет те же функции, что и обычная локальная компьютерная сеть, другая обеспечивает обмен данными между вычислительными заданиями на разных узлах.

Наиболее серьёзные требования предъявляются к коммуникационной сети. Для характеристик возможностей сетей используются два основных параметра: пропускная способность и латентность.

Пропускная способность характеризует, какой наибольший объём информации может быть передан в единицу времени (чаще всего это секунда). Производители сетевого оборудования нередко указывают пиковую пропускную способность. В реальных приложениях, как правило, наблюдается скорость в 1,5–2 раза ниже пиковой. Термин **латентность** (задержка) – это чистое время на передачу сообщения нулевой длины. Оно в первую очередь зависит от времени, затрачиваемого сетевыми устройствами и системой на подготовку к передаче и получению информации.

Пропускная способность и латентность позволяют оце-

нить, насколько эффективно будут считаться задания на кластере. Если задание требует частого обмена данными между узлами, то использование сетевого оборудования с большой латентностью приведёт к тому, что бóльшая часть времени будет тратиться не на передачу данных, а на подготовку, а узлы будут простаивать. При малой пропускной способности обмен данными между узлами не будет успевать за скоростью счёта задания, что тоже скажется отрицательно на производительности: узлы будут тратить много времени на ожидание данных по сети.

Латентность и пропускная способность сети в первую очередь определяются используемой технологией передачи данных. Наиболее широко распространённой сетевой технологией является Ethernet, но её параметры удовлетворяют только требованиям, предъявляемым для организации служебной сети кластера, для сетей обмена данными используются менее известные, но более высокоскоростные сети.

Технология	Пропускная способность (мксек)	Латентность (мксек)	Латентность MPI
Gigabit Ethernet	**Пиковая** — 1 Gbit/s (125 MB/s),**MPI** — 60-120 MB/sec	30-100	50
10-Gigabit Ethernet	**Пиковая** — 10 Gbit/s (1,2 GB/s),**MPI** — 700-900 MB/s	9	25-30
Myrinet 2000, Myrinet-10G	**Пиковая** — 2 Gbit/s (10 Gbit/s), полный дуплекс.**TCP/IP около** 1,7-1,9 Gbit/s (9,6 Gbit/s).**MPI** — до 200 MB/s (до 400 MB/s на дуплексных операциях)	2	10
InfiniBand	От 10 до 312 Gbit/s и выше, зависит от реализации	0,1-0,2	0,5-2,6

Таблица 1: некоторые характеристики сетевых технологий

В таблице 1 приведены наиболее применяемые в кластерах сетевые технологии и их типичные характеристики. При проектировании сетей для вычислительных кластеров следует рассмотреть ещё один немаловажный вопрос – цену. Если не вдаваться в детали, то каждая сетевая карта высокоскоростной сети стоит около 1 000$, а цена коммуникатора может колебаться от 10 000$ до 1 000 000$ и выше. На сегодняшний день наиболее популярной технологией при построении кластеров для создания сетей обмена данными является технология InfiniBand. Причины её популярности связаны с хорошим соотношением между ценой и возможностями оборудования, доступностью программного обеспечения.

Некоторые сети могут использовать лишь один вариант топологии (способа коммутации узлов сети). Например, GigabitEthernet поддерживает только топологию «звезда», но так как в реальных приложениях она используется только совместно с TCP/IP, то допускается объединять несколько «звёзд» каналами, настроив маршрутизацию.

InfiniBand позволяет использовать практически любые топологии, которые поддерживаются установленным Subnet Manager. Стандартные реализации Subnet Manager поддерживают топологии «звезда», «дерево», «толстое дерево», «гиперкуб», но появляются и новые реализации. За счёт того, что в InfiniBand допускаются множественные маршру-

ты, для средних конфигураций неплохо подходит топология «толстое дерево», которая хорошо использует дублирующиеся каналы. Топология – важный фактор эффективности сети. Наличие «узких мест» в топологии может свести на нет высокую скорость сети. Например, два GigabitEthernet-коммутатора, соединённых одним каналом, – явно не лучшее решение. А если соединять их несколькими каналами, то необходимо позаботиться о том, чтобы они объединялись на уровне коммутатора. Такое объединение поддерживается многими видами сетевого оборудования, существуют стандартные технологии, например EtherChannel, bonding, trunking. Важно заранее убедиться, что все стороны, участвующие в таком объединении, используют одинаковые стандарты (например, bonding может быть реализован по-разному у разных производителей).

InfiniBand

Мы отдельно останавливаемся на описании сетевой технологии InfiniBand, так как, с одной стороны, эта технология является широко распространённой в мире высокопроизводительных вычислений, и многим администраторам HPC-кластеров приходится в своей деятельности сталкиваться с этой технологией, а с другой стороны, InfiniBand довольно сильно отличается от привычных большинству администраторов сетей Ethernet, и при первом знакомстве возникает множество затруднений. При этом информации по InfiniBand немного, особенно на русском языке, хотя в последнее время ситуация улучшается.

Развитием InfiniBand занимается альянс InfiniBand Trade Association, InfiniBand – это открытая технология, стандарты которой опубликованы и доступны. Также есть набор программного обеспечения с открытым исходным кодом **OFED** (OpenFabrics Enterprise Distribution), в котором содержится все необходимое для работы в сетях, построенных на основе InfiniBand (возможно, кроме драйверов адаптеров). Компании-производители оборудования InfiniBand могут выпускать и свои версии стека программного обеспечения. Чаще всего они включают в себя OFED и дополнительные компоненты, ориентированные на работу с оборудованием конкретного производителя.

Связь (link) в сети InfiniBand состоит из нескольких линий (lanes), работающих параллельно. Каждая линия работает как последовательный двунаправленный канал связи. Чаще всего используются связи 4x (четыре линии, работающие параллельно). Связи 12x используются для связи отдельных элементов, чаще всего микросхем коммутаторов, внутри одного большого коммутатора. Скорость передачи данных по линии зависит от поколения стандарта InfiniBand. Для передачи данных могут использоваться соединения на печатной плате, медные провода (для небольших расстояний) и оптические кабели, часто продающиеся уже с трансмиттерами. Сведения о скоростях передачи данных приведены в таблице 2.

В материалах по сетям InfiniBand обычно указывается «сырая скорость» (raw speed) передачи данных, т. е. та скорость, с которой данные передаются физически по среде передачи. При этом данные пользователя перед передачей кодируются для восстановления при возможных ошибках на линии. Для поколений SDR-QDR 8 бит пользовательских данных превращаются в 10 бит, которые надо передать, для поколений FDR-EDR используется кодирование 64/66. Поэтому доступная для передачи данных пользователя пропускная способность будет ниже, чем указанная в спецификации.

	SDR	DDR	QDR	FDR	EDR
«Сырая» скорость передачи данных, Гбит/с	2,5	5	10	14,0625	25,78125
Теоретическая эффективная пропускная способность связи 4x, Гбит/с	8	16	32	54,4	163,2
Пропускная способность связи 12x, Гбит/с	24	48	96	163,2	240
Метод кодирования	8/10	8/10	8/10	64/66	64/66

Таблица 2: производительность сетей InfiniBand

В каждое устройство, подключённое к сети InfiniBand (узел кластера, который в материалах по InfiniBand часто называют процессорным узлом, Processor Node, сервер системы хранения и т. п.), устанавливается адаптер канала хоста InfiniBand (**HCA**, Host Channel Adapter). Стандарт предусматривает упрощённый вариант HCA, называемый TCA (Target Channel Adapter), который предполагалось использовать для подключения систем хранения данных, но этот вид адаптеров не получил распространения.

Адаптер может иметь несколько **портов** (ports) для подключения к сети. Сеть InfiniBand (ещё говорят про **фабрику** InfiniBand, InfiniBand Fabric) состоит из адаптеров, которые соединяются при помощи коммутаторов (switches) и маршрутизаторов (routers). Коммутаторы и маршрутизаторы всегда имеют более одного порта. На каждом коммутаторе выделяется виртуальный порт 0, через который коммутатором можно управлять.

Порты, на которые могут быть направлены пакеты, называются оконечными (endports). Набор адаптеров, со-

единённых при помощи коммутаторов, составляет подсеть (subnet). Подсети имеют ограничение на количество устройств, которое в ней может содержаться, – не более 215 + 214 – 1 = 49 151 оконечных портов и коммутаторов. Подсети соединяются при помощи маршрутизаторов, позволяя создавать фабрики InfiniBand практически неограниченных размеров.

Идентификация компонентов и адресация в сетях InfiniBand

Компоненты сети InfiniBand имеют идентификаторы, которые называются GUID (Globally Unique ID, глобально уникальный идентификатор), длиной в 64 бита. В зависимости от типа устройства, таких идентификаторов может быть несколько. GUID назначаются производителем устройства, хотя могут иметься средства для их изменения. Каждый адаптер имеет NodeGUID и по одному PortGUID на каждый порт адаптера. Один из PortGUID может совпадать с NodeGUID адаптера. Коммутатор также имеет NodeGUID и PortGUID, однако все PortGUID должны быть одинаковыми для всех портов коммутатора.

Также есть идентификатор, называемый SystemImage GUID. Его назначение – дать возможность определить, какие компоненты составляют единую систему (находятся под управлением одной программы). Для многопортовых ком-

мутаторов, например, этот параметр одинаков для всех элементарных коммутаторов, составляющих один большой составной коммутатор. Для адаптеров, установленных в один сервер, этот параметр будет различаться, так как каждый адаптер имеет независимую управляющую программу (то, что по-английски называется firmware). SystemImage GUID может быть равен NodeGUID одного из компонентов, составляющих единую систему, или быть нулевым, если компонент не входит ни в какую систему (или производитель не хочет дать возможность определения компонентов своей системы).

GUID используются для идентификации компонентов сети InfiniBand, то есть для того, чтобы отличить один компонент от другого. Они не используются как адреса при передаче данных. В качестве адресов при передаче данных внутри подсети используются LID (Local ID, локальный идентификатор). Для передачи данных между подсетями в качестве адресов используются GID (Global ID, глобальный идентификатор). GID могут использоваться и для передачи данных внутри одной подсети, но адресация при помощи GID требует присутствия в пакете с данными дополнительного заголовка GRH (Global Routing Header, заголовок глобальной маршрутизации), что увеличивает размер служебной информации в пакете данных.

Локальный идентификатор LID имеет длину в 16 бит. Значение LID = 0 зарезервировано и не может быть ис-

пользовано; LID от 1 до 0xBFFF предназначены для обычных LID, используемых при передаче точка-точка (unicast): LID от 0xC000 до 0xFFFE предназначены для организации многоадресной передачи (multicast); LID = 0xFFFF – так называемый разрешительный LID (permissive LID), пакет, адресованный такому LID, будет обработан первым портом, его получившим. Каждому оконечному порту и каждому коммутатору (LID назначается коммутатору в целом, а не отдельным его портам) в подсети во время её инициализации назначается минимум один LID, при этом внутри одной подсети LID не могут повторяться.

Из доступного количества LID и вытекает ограничение на количество устройств в подсети. Именно при помощи LID получателя пакета коммутаторы определяют, на какой порт надо передавать полученный пакет: записи в таблице форвардинга коммутаторов (forwarding table) в качестве ключа используют именно LID. Для упрощения обработки подсетей, в которых имеется много возможных альтернативных маршрутов между заданными парами точек, порту или коммутатору может назначаться несколько LID. В этом случае назначается базовый LID (Base LID) и LMC (LID Mask Control, управляющая маска LID). LMC – это число от 0 до 128.

Младшие LMC бит базового LID должны быть равны нолю, и считается, что порту назначены 2LMC подряд идущих значений LID, т. е. значения от Base LID до Base LID +

2LMC – 1. Если на порт назначается только один LID, то LMC = 0. Обычно в одной подсети используются не более двух значений LMC: одно для назначения LID портам адаптеров и другое (чаще всего ноль) для назначения LID коммутаторам.

Глобальный идентификатор GID имеет длину в 128 бит. Он назначается каждому оконечному порту. Фактически, GID – это адрес IPv6, в котором младшие 64 бита – это GUID порта, которому назначен этот GID. Старшие 64 бита GID (GID Prefix, префикс GID) по умолчанию равны 0xFE80::/64 (подробности по текстовому представлению адресов и префиксов IPv6 см. в RFC2373). Область действия этого префикса – подсеть (link-local scope). Пакеты данных с GID назначения, имеющим такой префикс, не будут переданы маршрутизаторами между подсетями, т. е. GID с такими префиксами могут использоваться только для передачи данных внутри подсети. При инициализации подсети может быть назначен один (или ни одного) префикс GID, отличный от префикса по умолчанию. При этом GID с префиксом по умолчанию все равно должен работать как GID порта.

Префикс 0xFEC0::/64 – префикс с сайтовой областью действия (site-local scope). Пакеты данных, предназначенные таким адресам, могут передаваться маршрутизаторами из подсети в подсеть, но не должны покидать пределы сайта (области с единым администрированием адресного пространства). Также может быть назначен префикс с глобальной об-

ластью действия (global scope), он должен выбираться по правилам, установленным для адресов IPv6. Кроме одноадресных (unicast) GID есть и GID, предназначенные для многоадресной (multicast) передачи данных. Префикс многоадресных GID имеет старший байт `0xFF`, про значение остальных бит префикса можно подробнее прочитать в RFC2373 и RFC2375.

Кроме адресации при помощи LID и GID есть еще один способ адресации, адресация при помощи направленного маршрута (Directed Route). Этим способом можно адресовать только пакеты управления подсетью (Subnet Management Packet, SMP). Он используется в основном при начальной инициализации подсети, когда портам еще не назначены LID и не установлены таблицы форвардинга коммутаторов, или после перезагрузки адаптера или коммутатора, когда доступ к ним при помощи LID ещё невозможен. В режиме адресации при помощи направленного маршрута в пакете перечисляется список портов коммутаторов, через которые должен пройти пакет данных (Initial Path). Также в пакете есть счётчик количества пересылок (hop count), который указывает число элементов в списке портов, указатель на текущий элемент в списке портов (hop pointer), указатель направления D (Direction, 0 – пакет пересылается от источника к адресату запроса, 1 – пакет содержит ответ и пересылается по направлению к источнику исходного запроса) и обратный маршрут (reverse path).

Получив пакет с полем D = 0, коммутатор при помощи указателя на текущий элемент Hop Pointer определяет порт, в который следует направить полученный пакет, записывает номер порта, через который пакет получен, в поле для сохранения обратного маршрута Reverse Path, и увеличивает поле hop pointer на единицу. Если список закончился, то получатель обрабатывает пакет, формирует ответ, меняет указатель направления на обратный (устанавливает поле D в 1) и посылает ответ. При получении пакета, в котором указатель направления установлен на обратное, коммутаторы используют обратный маршрут для определения порта для пересылки, соответственно не записывают нового обратного маршрута, и на каждом шаге уменьшают значение hop pointer на единицу.

Кроме чистого направленного маршрута возможен вариант, когда указывается LID коммутатора, до которого пакет должен быть направлен при помощи обычной адресации (по LID), и LID получателя, которому пакет должен быть направлен после того, как будет пройден путь, определяемый направленным маршрутом. Очевидно, что при этом части фабрики до и после пути, определяемого направленным маршрутом, должны быть уже инициализированы и поддерживать пересылки при помощи LID.

Управление подсетью InfiniBand

Как было сказано выше, для нормальной работы подсеть InfiniBand должна быть настроена: назначены LID портам адаптеров и коммутаторов, настроены таблицы форвардинга коммутаторов (в отличие от сетей Ethernet, в сетях InfiniBand коммутаторы не формируют свою таблицу форвардинга сами, она должна настраиваться извне).

Компонентом, который отвечает за такую настройку, а затем за поддержание подсети в рабочем состоянии, является менеджер подсети (Subnet Manager). Менеджер подсети – это программа, которая может работать на компьютере с адаптером InfiniBand или на коммутаторе (не все коммутаторы InfiniBand поддерживают запуск менеджера подсети). Для надёжности в подсети может быть запущено несколько менеджеров, в этом случае один из них является главным (master), а остальные – запасными (standby). В случае, если главный менеджер перестаёт работать, его функции берет на себя один из запасных. Также главный менеджер может явно передать роль главного одному из запасных менеджеров, например, в процессе нормальной остановки.

После запуска менеджер подсети при помощи пакетов управления подсетью, передаваемых по направленным маршрутам, выясняет структуру подсети: какие есть адаптеры, коммутаторы, маршрутизаторы, и какие между ними

есть связи. Если после определения структуры подсети выяснится, что других, более приоритетных менеджеров подсети в этой подсети нет, данный менеджер становится активным и осуществляет настройку подсети, т. е. назначает всем конечным портам LID, каждому конечному порту сообщает LID порта, на котором работает сам менеджер подсети, устанавливает таблицы форвардинга коммутаторов и делает некоторые другие настройки. После этого подсеть готова к работе. В процессе работы подсети менеджер время от времени собирает информацию об изменениях её структуры (этот процесс называется Sweeping) и соответствующим образом меняет конфигурацию.

Запасные менеджеры время от времени опрашивают главного, и если тот перестаёт отвечать на запросы, один из запасных становится главным и перенастраивает подсеть, указывая ей расположение нового менеджера подсети.

IP через InfiniBand (IP over IB, IPoIB)

Работа стека протоколов TCP/IP поверх InfiniBand не является частью спецификации InfiniBand, она определена в соответствующих документах RFC. Работа InfiniBand вполне возможна и без IPoIB. Однако некоторые программы и библиотеки хотя и предназначены для работы поверх InfiniBand, требуют также работающего IP поверх InfiniBand. Чаще всего при помощи IpoIB определя-

ют InfiniBand-идентификаторы (LID, GID) процессов, работающих на других вычислительных узлах, а после определения дальнейшие коммуникации осуществляются без участия стека TCP/IP.

Настройка IP поверх InfiniBand, в общем, не отличается от настройки IP поверх Ethernet. Есть только несколько моментов, на которые следует обратить внимание. Интерфейсы IPoIB в системе называются `ib0, ib1` и т. д. (по одному интерфейсу на порт InfiniBand). Адреса лучше назначать статически, прописывая их в конфигурационных файлах серверов и вычислительных узлов. Работа протокола DHCP поверх IPoIB возможна, но для надёжности мы рекомендуем его не использовать.

Адрес канального уровня (link layer address), который в сетях Ethernet называется MAC-адрес или hardware address, для IPoIB имеет длину в 20 байт. Поэтому некоторые утилиты, в частности, широко применяемая утилита `ifconfig`, в которых жёстко прописана длина MAC-адреса Ethernet в 6 байт, не могут корректно работать и отображать адреса канального уровня для IPoIB. Утилита `ip`, рекомендуемая для замены `ifconfig`, такого недостатка лишена. В адресе канального уровня содержится GID порта, номер пары очередей (Queue Pair Number, QPN, аналог номера порта в TCP для InfiniBand) и флаги, указывающие, какие протоколы транспортного уровня InfiniBand могут использоваться для передачи IP.

Утилиты для просмотра информации по сетям InfiniBand

В этом разделе мы приводим примеры выдачи некоторых утилит из комплекта OFED с объяснениями выдаваемой информации. Эти данные помогут сориентироваться в том, что происходит в сети InfiniBand, и диагностировать некоторые ошибки в её работе.

Команда `ibstat` показывает состояние всех портов на всех адаптерах InfiniBand, установленных на узле, где она запущена

```
CA 'mlx5_0'
CA type: MT4113
Number of ports: 2
Firmware version: 10.12.1100
Hardware version: 0
Node GUID: 0x00265802000073f0
System image GUID: 0x00265802000073f0
Port 1:
  State: Active
  Physical state: LinkUp
  Rate: 56
  Base lid: 913
  LMC: 0
  SM lid: 43
  Capability mask: 0x26516848
  Port GUID: 0x00265802000073f0
  Link layer: InfiniBand
Port 2:
  State: Active
  Physical state: LinkUp
  Rate: 56
  Base lid: 1361
  LMC: 0
  SM lid: 698
  Capability mask: 0x26516848
  Port GUID: 0x00265802000073f8
  Link layer: InfiniBand
```

Сначала выводится информация по адаптеру: его имя (`mlx5_0`), тип адаптера (название модели), количество портов, версии встроенного программного (firmware) и аппаратного обеспечения, а также идентификаторы Node GUID и System Image GUID.

Для каждого порта в строке `Link layer` выводится тип подключения: InfiiniBand или Ethernet. Некоторые адаптеры InfiniBand позволяют подключаться как к сети InfiniBand, так и к Ethernet. Тип подключения определяется установленным трансивером. Строка `Port GUID` показывает GUID порта. `Base lid` – первый LID, присвоенный данному порту. Всего порту присвоено, как говорилось выше, 2LMC подряд идущих LID. `SM lid` – LID порта, на котором работает менеджер данной подсети. Rate – скорость передачи данных, на которой работает порт (56 в данном случае – это режим 4x FDR).

`Physical state` – состояние физического уровня передачи данных. Нормальное состояние – `LinkUp`. Также может быть `Disabled`, `Polling` (в это состояние порт переходит после включения), `Configuration` (согласование режимов работы с другой стороной связи), `Recovery` (восстановление после сбоя связи). Есть и другие состояния, но их появление означает серьёзный сбой в работе оборудования, и мы их здесь описывать не будем.

`State` – состояние канального уровня передачи данных.

`Active` – состояние нормального функционирования, возможна передача любых типов данных. `Down` – передача данных невозможна (физический уровень ещё не перешёл в состояние `LinkUp`). `Initialize` – состояние, в которое канальный уровень переходит сразу после того, как физический уровень перешёл в состояние `LinkUp`. В этом состоянии возможны приём и передача только пакетов управления подсетью (SMP, Subnet Management Packets). В этом состоянии менеджер подсети должен настроить порт (задать LID и прочие параметры) и перевести порт в состояние `Active`. Есть и другие состояния, но порт не должен находиться в них долгое время, поэтому мы опустим их описания.

`Capability mask` – набор флагов, описывающих поддерживаемые портом режимы работы (скорости и т. п.).

Команда `ibstatus` также выводит информацию обо всех портах, но немного в другом формате, и выдаёт частично отличающийся набор данных:

```
Infiniband device 'mlx5_0' port 1 status:
default gid:     fe90:0000:0000:0000:0026:5802:0000:73f0
base lid:        0x391
sm lid:          0x2b
state:           4: ACTIVE
phys state:      5: LinkUp
rate:            56 Gb/sec (4X FDR)
link_layer:      InfiniBand

Infiniband device 'mlx5_0' port 2 status:
default gid:     fea0:0000:0000:0000:0026:5802:0000:73f8
base lid:        0x551
sm lid:          0x2ba
state:           4: ACTIVE
phys state:      5: LinkUp
rate:            56 Gb/sec (4X FDR)
link_layer:      InfiniBand
```

Обратите внимание, что информация о базовом LID и LID менеджера подсети дана в шестнадцатеричном виде. Более подробно дана информация о скорости, на которой работает порт. Ещё добавлена строка `default gid`, в которой указан GID для данного порта.

Иногда нужно узнать, какой машине назначен конкретный LID. Для этого можно применить утилиту `smpquery`. Вообще эта утилита предназначена для посылки пакетов управления подсетью SMP (Subnet Management Packet) и выдачи ответов в понятной человеку форме. В нашем случае нам нужен запрос описания узла (node description). Вот пример выдачи команды `smpquery nodedesc 914` (запрос описания узла с LID `914`):

```
Node Description:.................n51001 HCA-1
```

Узел ответил, что LID 914 назначен адаптеру HCA-1 вычислительного узла с именем n51001.

При помощи smpquery доступна информация о том узле, которому адресован запрос. В то же время менеджер подсети имеет информацию обо всех узлах подсети. Запросить информацию у менеджера подсети можно при помощи утилиты saquery. Информацию об узле подсети с LID 914 можно запросить командой saquery 914. Вот пример выдачи такой команды:

```
NodeRecord dump:
lid.....................914
reserved................0x0
base_version............0x1
class_version...........0x1
node_type...............Channel Adapter
num_ports...............2
sys_guid................0x0026580200003740
node_guid...............0x0026580200003740
port_guid...............0x0026580200003740
partition_cap...........0x80
device_id...............0x1011
revision................0x0
port_num................1
vendor_id...............0x2C9
NodeDescription.........n51001 HCA-1
```

В последней строке указано описание узла, включающее имя хоста. Также приводится дополнительная информация. Ещё раз обращаем внимание, что команда `smpdump` позволяет запрашивать информацию об узле в сети InfiniBand у самого этого узла, а команда `saquery` – у менеджера подсети. Если результаты этих запросов различаются или если команда `saquery` выдаёт ошибку – это свидетельство того, что имеются проблемы с менеджером подсети. Ещё две полезные утилиты при диагностике сетей InfiniBand – утилиты `ibnetdiscover` и `ibdiagnet`. Утилита `ibnetdiscover` пытается обнаружить все компоненты подсети: конечные узлы, коммутаторы, маршрутизаторы и связи между ними, и выводит информацию обо всех найденных компонентах. Утилита `ibdiagnet` также пытается найти все компоненты подсети, но кроме этого она ещё и пытается обнаружить ошибки в конфигурации подсети, такие как совпадающие GUID, скорости портов и т. п.

Мы не будем приводить примеры выдачи этих утилит, так как они достаточно объёмны, а для `ibdiagnet` ещё и состоят из нескольких файлов. Мы упоминаем эти утилиты, чтобы иметь представление, какие средства можно использовать при диагностике проблем с сетью InfiniBand.

Утилиты, которые посылают информацию в сеть, имеют ключи для выбора адаптера и порта, с которым следует работать (напомним, что в разных подсетях один и тот же LID

может относиться к разным устройствам). Ключ `-C` предназначен для указания адаптера (например, `mlx4_0` в примерах выше), а ключ `-P` позволяет указать номер порта заданного адаптера (порты нумеруются, начиная с 1).

Хранение данных

В каждый узел – управляющий, вычислительный или служебный – могут быть установлены локальные жёсткие диски. Наряду с этим возможно подключение внешних дисковых подсистем, доступ к которым будет производиться со всех узлов одновременно.

Локальные жёсткие диски могут использоваться для загрузки операционной системы, как виртуальная память (область подкачки) и для хранения временных данных. Конечно, вычислительные узлы могут и не иметь локальных дисков, если загрузка операционной системы на них организована через сеть, хотя даже в этом случае локальный диск полезен для области подкачки и хранения временных данных. На управляющем узле локальные жёсткие диски обычно устанавливаются, а сетевая загрузка при этом не предусматривается.

На внешних системах хранения данных (далее – СХД) обычно располагаются программные пакеты и утилиты, запуск которых требуется на всех узлах, а также домашние каталоги пользователей, временные хранилища общего доступа (для хранения временных данных расчётов) и прочие данные, которые должны быть доступны со всех узлов. Внешние СХД обычно различаются по внутреннему устройству и по способу доступа, от чего зависит уровень надёжности хране-

ния данных и скорость доступа к ним. Внутреннее устройство СХД мы разбирать здесь не будем, упомянем лишь различные способы доступа.

По способу доступа СХД разделяются как минимум на три типа:

• непосредственно подключённая СХД – Direct Attached Storage или **DAS**;

• СХД с доступом по локальной сети или сетевое хранилище данных – Network Attached Storage, или **NAS**;

• СХД, подключённая через выделенную сеть хранения данных – Storage Area Network или **SAN** (см. рис. 3).

Непосредственно подключённая СХД подключается либо к выделенному узлу хранения данных, либо к управляющему узлу. Такая СХД всегда видна в операционной системе узла, к которому она подключена, как локально подключённое дисковое устройство (физическое подключение – по SATA, SAS, Fibre Channel).

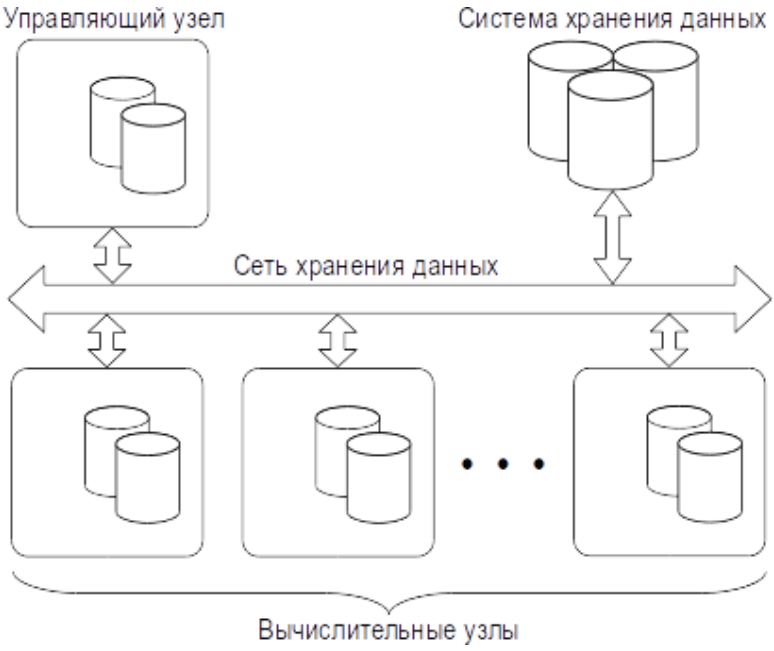

Рис. 3: сеть хранения данных (SAN)

Для обеспечения отказоустойчивости и повышения скорости работы в системах хранения нередко используют технологию RAID (redundant array of independent disks – избыточный массив независимых дисков). В рамках RAID несколько дисков равного объёма объединяются в один логический диск. Объединение происходит на уровне блоков (которые могут не совпадать с физическими блоками дисков). Один логический блок может отображаться на один

или несколько дисковых блоков.

Есть несколько «уровней», которые приняты как стандарт de-facto для RAID:

RAID-0 – логические блоки однозначно соответствуют блокам дисков, при этом они чередуются: блок0 = блок0 первого диска, блок1 = блок1 второго диска и т. д.;

RAID-1 – зеркальный массив, логический блок N соответствует логическим блокам N всех дисков, они должны иметь одинаковое содержимое;

RAID-2 – массив с избыточностью по коду Хэмминга;

RAID-3 и -4 – дисковые массивы с чередованием и выделенным диском контрольной суммы;

RAID-5 – дисковый массив с чередованием и невыделенным диском контрольной суммы;

RAID-6 – дисковый массив с чередованием, использующий две контрольные суммы, вычисляемые двумя независимыми способами.

Уровень 0 обеспечивает наибольшую скорость последовательной записи – блоки пишутся параллельно на разные диски, но не обеспечивает отказоустойчивости; уровень 1 – наибольшую отказоустойчивость, так как выход из строя N-1 диска не приводит к потере данных.

Уровни 2, 3 и 4 в реальности не используются, так как уровень 5 даёт лучшую скорость и надёжность при той же степени избыточности. В этих уровнях блоки дисков объединяются в **полосы**, или **страйпы** (англ. stripe).

В каждом страйпе один блок выделяется для хранения контрольной суммы (для уровня 6 – два страйпа), а остальные – для данных, при этом диск, используемый для контрольной суммы, чередуется у последовательных страйпов для выравнивания нагрузки на диски. При записи в любой блок рассчитывается контрольная сумма данных для всего страйпа, и записывается в блок контрольной суммы. Если один из дисков вышел из строя, то для чтения логического блока, который был на нём, производится чтение всего страйпа и по данным работающих блоков и контрольной суммы вычисляются данные блока.

Таким образом, для RAID-5 можно получить отказоустойчивость при меньшей избыточности, чем у зеркала (RAID-1), – вместо половины дисков можно отдать под избыточные данные только один диск в страйпе (два для RAID-6). Как правило, «ширина» страйпа составляет 3-5 дисков. Ценой этого становится скорость работы – для записи одного блока нужно сначала считать весь страйп, чтобы вычислить новую контрольную сумму.

Часто применяют двухуровневые схемы – RAID-массивы сами используются как диски для других RAID-массивов. В этом случае уровень RAID обозначается двумя цифрами: сначала нижний уровень, затем верхний. Наиболее часто встречаются RAID-10 (RAID-0, построенный из массивов RAID-1), RAID-50 и -60 – массивы RAID-0, построенные из массивов RAID-5 и -6 соответственно. Подробнее о

RAID читайте в литературе и Интернете.

Если используется распределённое хранение данных, например, как в Lustre (о ней мы расскажем далее), то узлов хранения данных может быть несколько, а данные, хранящиеся на такой СХД, распределяются по узлам хранения данных. СХД с доступом по локальной сети (или сетевое хранилище данных, NAS) обычно предоставляет дисковое пространство узлам по специальным протоколам, которые можно объединить под общим названием **сетевые файловые системы**. Примерами таких файловых систем могут быть NFS (Network File System), Server Message Block (SMB) или её современный вариант – Common Internet File System (CIFS).

Строго говоря, CIFS и SMB – два разных названия одной и той же сетевой файловой системы, изначально разработанной компанией IBM и активно используемой в операционных системах компании Microsoft. Сейчас CIFS может применяться практически в любой операционной системе для предоставления доступа к файлам через локальную сеть. Как правило, кроме NFS и CIFS системы NAS могут предоставлять доступ к хранимым данным и по другим протоколам, например FTP, HTTP или iSCSI.

СХД, подключённые через специальные сети хранения данных (SAN), обычно видны в операционной системе как локально подключённые дисковые устройства. Особенность SAN в том, что для формирования такой сети в целях

повышения надёжности могут использоваться дублированные коммутаторы. В этом случае каждый узел будет иметь несколько маршрутов для доступа к СХД, один из которых назначается основным, остальные – резервными. При выходе из строя одного из компонентов, через которые проходит основной маршрут, доступ будет осуществляться по резервному маршруту.

Переключение на резервный маршрут будет происходить мгновенно, и пользователь не обнаружит, что вообще что-то вышло из строя. Для того чтобы это работало, необходима поддержка множественных маршрутов (multipath) в оборудовании и ОС. Заметим, что хотя для multipath и есть стандарты, но в реальности часто встречается «капризное» оборудование, для корректной работы которого с multipath требуются нестандартные драйверы или пакеты системного ПО.

Отличие NAS от SAN довольно условное, поскольку существует протокол обмена iSCSI, позволяющий использовать обычную локальную сеть в качестве сети хранения данных. В этом случае сетевое хранилище данных будет видно в операционной системе как локально подключённое дисковое пространство. Сеть хранения данных может объединяться с высокоскоростной коммуникационной сетью. Например, в качестве SAN-сети способна выступать InfiniBand, используемая для высокоскоростного обмена данными между вычислительными узлами кластера.

Особенности аппаратной архитектуры

Ни для кого не секрет, что в самом начале компьютерной эры понятия «процессор» и «ядро» (имеется в виду вычислительное ядро процессора) были синонимичными. Точнее, понятие «ядро» к процессору не относилось вовсе, поскольку многоядерных процессоров ещё не было. В каждом компьютере устанавливался обычно один процессор, который в каждый момент времени мог исполнять лишь один процесс. Современные системы такого типа можно встретить и сейчас, но они, как правило, предназначены для решения специальных задач (контроллеры, встраиваемые системы).

Для увеличения мощности сервера или рабочей станции производители устанавливали несколько «одноядерных» процессоров (обычно от двух до восьми). Такие системы существуют и сейчас и называются симметричными многопроцессорными системами, или SMP-системами (от англ. Symmetric Multiprocessor System) (см. рис. 4).

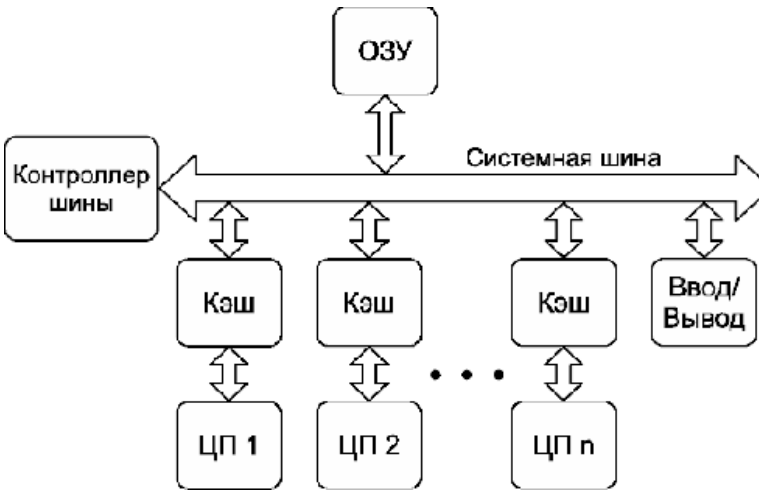

Рис. 4: симметричная многопроцессорная система (SMP)

Как видно из схемы, каждый процессор, представляющий собой одно вычислительное ядро, соединён с общей системной шиной. В такой конфигурации доступ к памяти для всех процессоров одинаков, поэтому система называется **симметричной**. В последнее время в каждом процессоре присутствует несколько ядер (обычно от 2 до 16). Каждое из таких ядер может рассматриваться как процессор в специфической SMP-системе. Конечно, многоядерная система отличается от SMP-системы, но эти отличия почти незаметны для пользователя (до тех пор, пока он не задумается о тонкой оптимизации программы).

Для ускорения работы с памятью нередко применяется

технология NUMA – Non-Uniform Memory Access. В этом случае каждый процессор имеет свой канал в память, при этом к части памяти он подсоединён напрямую, а к остальным – через общую шину. Теперь доступ к «своей» памяти будет быстрым, а к «чужой» – более медленным. При грамотном использовании такой архитектуры в приложении можно получить существенное ускорение.

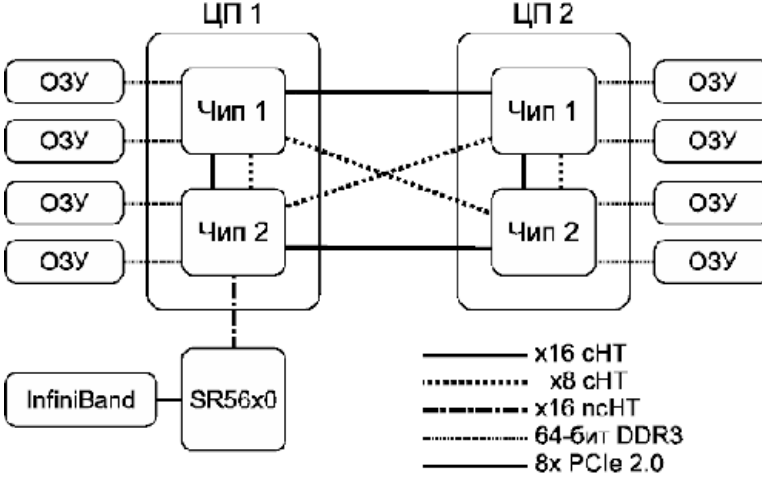

Рис. 5: схема узла NUMA на примере AMD Magny-Cours

Например, в архитектуре AMD Magny-Cours (см. рис. 5) каждый процессор состоит из двух кристаллов (логических процессоров), соединённых между собой каналами HyperTransport. Каждый кристалл (чип) содержит в себе

шесть вычислительных ядер и свой собственный двухканальный контроллер памяти. Доступ в «свою» память идёт через контроллер памяти, а в «соседнюю» – через канал HyperTransport. Как видим, построить SMP- или NUMA-систему из двух или четырёх процессоров вполне возможно, а вот с большим числом процессоров – уже непросто.

Ещё одним «камнем преткновения» в современных многоядерных системах является миграция процессов между ядрами. В общем случае для организации работы множества процессов операционная система предоставляет каждому процессу определённый период времени (обычно порядка миллисекунд), после чего процесс переводится в пассивный режим.

Планировщик выполнения заданий, переводя процесс из пассивного режима, выбирает ядро, которое не обязательно совпадает с тем, на котором процесс выполнялся до этого. Нередко получается так, что процесс «гуляет» по всем ядрам, имеющимся в системе. Даже в случае с SMP-системами влияние на скорость работы программы при такой миграции заметно, а в NUMA-системах это приводит ещё и к большим задержкам при доступе в память.

Для того, чтобы избавиться от паразитного влияния миграции процессов между ядрами, используется привязка процессов к ядрам (processor affinity, или pinning). Привязка может осуществляться как к отдельному ядру, так и к нескольким ядрам или даже к одному и более NUMA-уз-

лам. С применением привязки миграция процессов или будет происходить контролируемым образом, или будет исключена вовсе.

Аналогичная проблема присутствует и в механизме выделения памяти пользовательским процессам. Допустим, процессу, работающему на одном NUMA-узле, требуется для работы выделить дополнительную память. В какой области памяти будет выделен новый блок? А вдруг он попадёт на достаточно удалённый NUMA-узел, что резко уменьшит скорость обмена? Для того, чтобы избежать выделения памяти на сторонних узлах, есть механизм привязки процессов к памяти определённого NUMA-узла (memory affinity).

В нормальном случае каждый процесс параллельной программы привязывается к определённым NUMA-узлам как по ядрам, так и по памяти. В этом случае скорость работы параллельной программы не будет зависеть от запуска и будет достаточно стабильной. При запуске параллельных программ такая привязка не просто желательна, а обязательна. Более подробно данный вопрос рассмотрен в главе «Библиотеки поддержки параллельных вычислений», где описываются различные среды параллельного программирования.

В большинстве современных процессоров компании Intel используется технология HyperThreading. Благодаря этой технологии каждое вычислительное ядро представлено в системе как два отдельных ядра. Конечно, эффективность использования аппаратных ресурсов в этом случае сильно за-

висит от того, как написана программа и с использованием каких библиотек и каким компилятором она собрана. В большинстве случаев параллельные вычислительные программы написаны достаточно эффективно, поэтому ускорения от использования технологии HyperThreading может не быть, и даже наоборот, будет наблюдаться замедление от её использования.

На суперкомпьютерах эта технология вообще может быть отключена в BIOS каждого узла, чтобы не вносить дополнительных трудностей в работу параллельных программ. Как правило, эта технология не приносит ускорения для вычислительных программ. Если вы используете небольшой набор программ на суперкомпьютере, проверьте их работу с включённым и отключённым HyperThreading и выберите лучший вариант. Обычно мы рекомендуем включить её, но при этом указать системе управления заданиями число ядер, как с отключённым HT. Это позволяет получить дополнительные ресурсы для системных сервисов, минимально влияя на работу вычислительных заданий.

Ещё одна особенность архитектуры касается уже не отдельного, а нескольких узлов. Как мы ранее указывали, вычислительные узлы в вычислительном кластере объединены высокоскоростной коммуникационной сетью. Такая сеть может предоставлять дополнительные возможности обмена данными между процессами параллельных программ, запущенных на нескольких вычислительных узлах. В рамках од-

ного узла применяется технология прямого доступа в память (Direct Memory Access, или DMA), позволяющая устройствам узла связываться с оперативной памятью без участия процессора. Например, обмен данными с жёстким диском или с сетевым адаптером может быть организован с использованием технологии **DMA**.

Адаптер InfiniBand, используя технологию DMA, предоставляет возможность обращаться в память удалённого узла без участия процессора на удалённом узле (технология Remote Direct Memory Access, или **RDMA**). В этом случае возникнет необходимость синхронизации кэшей процессоров (данный аспект мы не будем рассматривать подробно). Применение технологии RDMA позволяет решить некоторые проблемы масштабируемости и эффективности использования ресурсов.

Существует достаточно серьёзная критика данной технологии. Считается, что модель двухстороннего приёма-передачи (two-sided Send/Receive model), применяемая в суперкомпьютерах компании Cray (коммуникационная сеть SeaStar) и в коммуникационных сетях Quadrics QsNet, Qlogic InfiniPath и Myrinet Express, более эффективна при использовании параллельной среды программирования MPI. Конечно, это не исключает эффективного использования технологии RDMA, но применение её ограничено. В большинстве практических приложений использование RDMA даёт снижение латентности, но на больших приложе-

ниях (сотни узлов) может вылиться в чрезмерное использование системной памяти.

Краткое резюме

Знание аппаратуры, основных принципов работы ваших сетей, хранилищ данных и прочих «железных» компонент очень важно для администратора суперкомпьютера. Без этих знаний часто бывает невозможно решить проблемы, возникающие в таких вычислительных комплексах.

Ключевые слова для поиска

`rdma, hpc interconnect, numa, smp, cache, latency.`

Глава 3. Как работает суперкомпьютер

Рассмотрим стек ПО, который необходим для обеспечения работы суперкомпьютера. Очевидно, что в первую очередь это операционная система, затем системное ПО, которое требуется для работы аппаратной части, – драйверы и т. п., а также ПО для файловой системы.

Следующая часть – набор ПО для организации загрузки и ПО для удалённого доступа. Далее – система контроля запуска заданий (система очередей, batch system). Потом следует ПО, необходимое для работы параллельных программ: готовые параллельные пакты и библиотеки – MPI, Cuda и т. п.

Обязательный компонент – компиляторы и дополнительные библиотеки, часто требующиеся для вычислительных программ, такие как BLAS, FFT и др. Для организации полноценного управления суперкомпьютером также потребуются ПО для организации резервного копирования, мониторинга, ведения статистики, визуализации состояния суперкомпьютера.

Как происходит типичный сеанс пользователя

Существует множество вариантов организации работы с конкретными вычислительными пакетами, которые предоставляют собственный интерфейс для работы с суперкомпьютером. Мы будем рассматривать «общий» вариант.

Итак, пользователь работает на своём компьютере – рабочей станции, ноутбуке, планшете и т. п. Для начала сеанса он запускает ssh-клиент (putty, openssh и т. п.), вводит адрес, логин, указывает пароль или файл с закрытым ключом (или загружает профиль, где всё это уже указано) и открывает соединение с суперкомпьютером. Попав на узел доступа, пользователь может отредактировать и откомпилировать собственную параллельную программу, скопировать по протоколу sftp входные данные. Для запуска программы пользователь выполняет специальную команду, которая ставит его задание в очередь. В команде он указывает число требуемых процессов, возможно, число узлов и другие предпочтения, а также свою программу и её аргументы. Пользователь может проверить статус своего задания, посмотреть список заданий в очереди. Если он понял, что в программе ошибка, то может снять её со счёта или удалить из очереди, если она ещё не запустилась.

При необходимости можно поставить в очередь и несколь-

ко заданий (например, если нужно обработать несколько наборов входных данных). После того как задание поставлено в очередь, его ввод/вывод будет перенаправлен в файлы, поэтому можно спокойно завершить сеанс и проверить состояние задания или посмотреть/скачать результаты позже, в другом сеансе. Большинство систем управления заданиями позволяют запустить задание и интерактивно, связав её ввод/вывод с терминалом пользователя. В этом случае придётся оставлять сеанс открытым до тех пор, пока задание стоит в очереди и работает.

Вся работа происходит в командной строке, поэтому пользователь должен знать минимальный набор команд Linux (как правило, это не составляет проблем). Элементарного самоучителя Linux или даже странички на сайте с описанием нужных команд обычно бывает достаточно. Для управления файлами многие пользователи применяют программу Midnight Commander (mc), которая ещё больше упрощает задачу.

Жизненный цикл задания

Типичное задание на суперкомпьютере проходит несколько фаз. Первая – постановка задания в очередь. На этом этапе пользователь указывает путь к исполняемой программе, её аргументы и параметры запуска, такие как число MPI-процессов, число узлов, требования к ним и т. д. Явно или неявно пользователь указывает также способ запуска задания – через команду mpirun (для MPI-приложений), как обычное приложение и т. д.

Система управления заданиями регулярно проверяет, можно ли запустить новую задание, просматривая очередь. Как только наше задание подойдёт к началу очереди или по каким-то иным критериям подойдёт для запуска, система управления (точнее, её планировщик) выберет набор узлов, на которых будет произведён запуск, оповестит их, возможно, выполнит скрипт инициализации (так называемый пролог) и приступит к запуску задания.

Фаза запуска может отличаться в разных системах, но общий смысл одинаков: на вычислительном или управляющем узле запускается стартовый процесс, например mpirun, которому передаётся список узлов и другие параметры. Этот процесс запускает на вычислительных узлах рабочие процессы задания – самостоятельно (через ssh) или используя помощь системы управления заданиями. С этого момента си-

стема управления заданиями считает, что задание работает. Она может отслеживать состояние рабочих процессов на узлах, если это поддерживается, или отслеживать только состояние стартового процесса. Как только стартовый процесс завершается либо задание снимается со счёта принудительно (пользователем или самой системой управления), задание переходит в фазу завершения.

В этой фазе система управления пытается корректно завершить работу задания – убедиться, что все её процессы завершились, не осталось лишних файлов во временных каталогах и т. п. Для этого часто используется отдельный скрипт, так называемый эпилог. По окончании фазы завершения задание считается завершённым. Какое-то время информация о ней может сохраняться в системе управления, но обычно данные о ней теперь можно найти только в журналах.

В описанном цикле могут быть и нестандартные действия, например изменение приоритета задания, меняющее скорость его прохождения в очереди, блокировка, временно запрещающая запуск задания, приостановка работы и некоторые другие.

Что скрыто от пользователя

Всё, что мы описали выше, – это то, что видно рядовому пользователю. Однако есть и то, что остаётся для него «за кадром», но играет важную роль для администратора. Это те сервисы, которые обеспечивают корректную работу суперкомпьютера: управление учётными записями, распределённой файловой системой, квотами, сервисы удалённого мониторинга узлов, сбора статистики и журналирования, мониторинга оборудования и инфраструктуры, экстренного оповещения и отключения, резервного копирования. Все эти сервисы работают незаметно для пользователя, но их важность трудно переоценить.

Краткое резюме

Собрать простейший вычислительный кластер можно и «на коленке»: взять два ноутбука, подключить в общую сеть, настроить беспарольный доступ по ssh, на одном из них запустить NFS-сервер, а на другом примонтировать по NFS файловую систему, и – готово, можно запускать MPI-программы. Но производительность такого кластера весьма невелика, а при попытке подключить вместо двух ноутбуков двадцать возникают проблемы: сеть не справляется с нагрузкой, NFS тормозит, один ноутбук завис, и мы полчаса выясня-

ем, что же случилось, и многое другое. Увы, если кластер не «игрушечный», а предназначен для реальных задач, то подходить к его построению и эксплуатации надо серьёзно. Мы кратко обозначили основные компоненты программного «стека» суперкомпьютера, далее попробуем рассмотреть их подробнее.

Ключевые слова

MPI, сеанс работы, ssh-клиент, NFS.

Глава 4. UNIX и Linux – основы

Если вы уже используете Linux и имеете неплохое представление о его администрировании, то **смело пропустите эту главу**. Если информация из неё будет для вас совсем новой, то для дальнейшего чтения желательно почитать дополнительную литературу, потренироваться в написании скриптов на bash.

В любом случае мы рекомендуем ознакомиться с книгами из списка ниже, в них есть масса информации, полезной даже опытным профессионалам:

Эви Немет, Гарт Снайдер, Трент Хейн, Бэн Уэйли
Unix и Linux: руководство системного администратора

Это классический учебник по Unix и Linux. В нём нередко случаются отсылки к таким древним системам, как VAX и PDP-11, тем не менее он отлично отражает суть работы UNIX и остаётся актуальным по сей день.

Томас Лимончелли, Кристина Хоган, Страта Чейлап
Системное и сетевое администрирование. Практическое руководство

Более новый учебник по Linux, содержит массу полезных примеров.

Брайан Керниган, Роб Пайк

Unix – Программное окружение

Эта книга в большей степени посвящена программированию как в оболочке bash, так и с помощью иных инструментов. Даже если вам не приходится регулярно этим заниматься, настоятельно советую прочесть эту книгу, так как она откроет вам те принципы, по которым строится работа в UNIX, вам станут более понятны многие процессы, происходящие внутри ОС.

Томас Лимончелли

Тайм-менеджмент для системных администраторов

Название говорит само за себя. В книге есть множество ситуаций, в которые попадает любой системный администратор, и практические советы, как из них выходить с минимальными потерями.

Эта глава не претендует на статус учебника по UNIX, но мы постарались собрать в ней все основные понятия, знание которых в дальнейшем вам обязательно потребуется. На сегодняшний день на суперкомпьютерах в подавляющем большинстве случаев используется UNIX-подобная операционная система. Мы говорим UNIX-подобная, так как легендарная ОС UNIX в чистом виде сейчас не развивается и практически не используется.

Краткая историческая справка. После разделения компании AT&T, которая и разработала эту ОС, товарный знак UNIX и права на оригинальный

исходный код неоднократно меняли владельцев, в частности, длительное время они принадлежали компании Novell. В 1993 г. Novell передала права на товарный знак и на сертификацию программного обеспечения на соответствие этому знаку консорциуму X/Open, который затем объединился с Open Software Foundation и сейчас называется «The Open Group». Этот консорциум занимается разработкой открытых стандартов для ОС, таких как POSIX (сейчас он переименован в Single UNIX Specification).

Согласно положению «The Open Group», название «UNIX» могут носить только системы, прошедшие сертификацию на соответствие Single UNIX Specification. В настоящее время несколько ОС прошли разные версии этой сертификации, например Solaris, AIX.

Даже те ОС, которые не проходили сертификации UNIX (например Linux), стараются соответствовать этим стандартам. Именно поэтому архитектура приложений на этих ОС очень похожа, а перенос приложения с одной ОС на другую прост, особенно если при написании программы использовались только стандартные библиотеки и функции. Именно эти качества и огромная популярность UNIX в прошлом, а также отлично зарекомендовавшие себя её наследники – Solaris, OpenBSD, FreeBSD, AIX и, конечно же, Linux – обеспечили UNIX-подобным ОС лидерство на серверах всего мира.

Вычислительные кластеры и суперкомпьютеры не исключение. Здесь стандартом de facto является Linux. Именно на эту операционную систему мы и будем ориентироваться. Несмотря на то что существует немало установок на других операционных системах, таких как Windows, FreeBSD, Solaris и других, в данной книге мы не будем останавливаться на их особенностях в классе HPC.

Процессы

Основное понятие в любой ОС – процесс. Это нечто типа контейнера (реально – описания в таблицах ОС), содержащего уникальный идентификатор (`PID`), права (владелец, группа и некоторые другие), код программы, область данных, стек, набор страниц памяти, таблицу открытых файлов и прочие атрибуты. Для ОС процесс – единица планирования процессорного времени, каждый процесс может исполняться процессором, быть в ожидании исполнения, быть в состоянии системного вызова (передать запрос к ОС и ждать ответ), быть остановленным или завершившимся. Обозначаются они как **R** (`running`), **S** (`sleeping`), **D** (`uninterruptable sleep`), **T** (`stopped`) и **Z** (`zombie`).

Например, если запустить на компьютере с 2 ядрами 10 программ расчёта числа пи, то одновременно смогут считаться только 2, но ОС будет с большой частотой (например 100 раз в секунду) приостанавливать выполнение активного процесса, помещать его в очередь и отправлять на выполнение следующий процесс из очереди (очень грубо, но суть именно такая). Для процесса это выглядит как будто он монопольно владеет процессором, просто скорость этого процессора раз в 5 ниже, чем могла бы.

Среднее число процессов в очереди обозначается как

«уровень загрузки» – **Load Average**. Если он больше числа ядер, то обычно это значит, что не всем задачам «достаётся» процессор, и они работают медленнее. Надо учесть что в очередь включаются и процессы в состоянии **D**, то есть высокий LA могут вызвать процессы, которые, например, много читают с диска или пишут (и постоянно ждут в вызове read или write). То есть высокий LA – это сигнал, что потенциально что-то не так, но хорошо бы проверить.

В состояние `stopped` процесс переводится, только если другой процесс послал ему сигнал STOP. В этом случае он «замирает» и перестаёт исполняться до тех пор, пока не получит сигнал CONT (или не будет завершён). Если процесс в состоянии **D**, то сигнал игнорируется. В принципе, сигнал STOP процесс может игнорировать, но так делается очень редко.

Состояние `zombie` возникает, когда процесс завершился, но его родитель «не подтвердил» это (не вызвал системный вызов `wait`). Это делается для того, чтобы родительский процесс мог получить данные о том, как завершился процесс. т. е. процессы в состоянии `zombie` уже не потребляют никаких ресурсов ни процессора, ни памяти. По этой же причине их нельзя принудительно завершить – они уже завершены.

Родительский процесс (`PPID`) есть у каждого процесса в системе, если родительский процесс завершился, то им становится процесс с `PID` 1 (обычно это специальный процесс

`init` в системе, мы про него поговорим ниже), который выполняет `wait` для всех таких процессов.

Посмотреть список процессов и их состояние можно с помощью команды `ps`. У неё нелёгкая судьба, т. к. в разных вариантах популярных ОС (Unix, BSD, Solaris) исторически у неё было много разных, в том числе конфликтующих опций. В результате в Linux используется вариант GNU, который пытается их сочетать. В частности, есть опции, которые обязательно надо указывать с минусом впереди, а другие – наоборот только без минуса. Ниже самые полезные с нашей точки зрения:

```
ps fax  # все-все процессы, сгруппированные в дерево,
        # можно сразу увидеть, кто чей потомок
ps aux  # все процессы с большинством полезных полей
ps -eLf # показать процессы и нити
ps -eo pid,ppid,user,vsize,pmem,pcpu,stat,wchan:32,comm
        # явно указать формат выдачи (-e = все процессы)
```

К большинству комбинаций можно добавить **w**, тогда поле имени процесса (обычно программа с аргументами) будет шире. Если добавить дважды, то будет ещё шире, а если трижды, то ограничений на ширину не будет совсем.

Бывает удобно отслеживать активность процессов в реальном времени. Тут помогут команды `top` и более новомодная `htop`. Они показывают процессы в виде таблицы, отсортированной по одному полю, и обновляют её раз в 5 секунд (можно поменять интервал). При этом показываются

только те процессы, которые поместились на экране, плюс некоторые общие данные о системе – загрузка процессора, памяти, loadaverage, число процессов в разных состояниях.

Можно переключать режимы отображения и сортировки. Для `top` есть несколько горячих клавиш, их список можно получить, нажав '**h**'. Наиболее удобные варианты сортировки и команды:

`<Shift>+<P>` – сортировать процессы по использованию процессора;

`<Shift>+<M>` – сортировать процессы по использованию памяти;

`1` – показывать загрузку каждого ядра или суммарную;

`k` – послать сигнал процессу;

`r` – изменить приоритет процесса;

`u` – фильтровать по пользователю;

`q` – выход.

У `htop` более дружественный интерфейс, по возможности она использует цветной вывод, загрузку процессора и памяти выводит в виде текстовых прогресс-баров, умеет организовывать процессы в деревья (и схлопывать их с одну строку, что иногда очень удобно). Клавиши управления выведены в нижней строке в стиле Norton Commander (Midnight Commander/FAR manager).

Мы уже не раз упомянули сигналы – это простой способ общения процессов, любой процесс может послать другому сигнал, если он принадлежит тому же пользователю (пользо-

ватель `root` может посылать всем). Сигнал – целое число, так что много информации им не передать, но его функция – попросить процесс выполнить какое-то действие. Все сигналы, кроме KILL, могут быть перехвачены и обработаны, если процесс не обрабатывает сигнал, то ОС выполняет заранее определённое действие за него.

Для большинства сигналов есть стандартные значения и действия, ниже – самые часто используемые:

Номер	Обозначение	Действие
9	KILL	немедленно завершить процесс
15	TERM	«вежливо» завершить процесс
3	QUIT	сигнал завершения с клавиатуры (если нажата комбинация **Ctrl-C**)
19	STOP	остановить выполнение
18	CONT	продолжить выполнение
4	ILL	посылается ОС, если процесс попытался выполнить некорректную инструкцию процессора. По умолчанию процесс завершается
11	SEGV	посылается ОС, если процесс обратился к несуществующему адресу в памяти. По умолчанию процесс завершается
8	FPE	посылается ОС, если произошло исключение операции с плавающей точкой. По умолчанию процесс завершается
10	USR1	пользовательский сигнал, по умолчанию игнорируется

Таблица 3: некоторые сигналы в Linux

Действия «по умолчанию» процесс может изменять (кроме сигнала KILL). Их можно обработать или игнорировать. При корректном завершении память процесса может быть записана в т. н. core-файл для того, чтобы после можно было исследовать причину ошибки отладчиком. Будет ли создан core-файл, определяется настройками ОС и лимитами (см. главу о квотах).

Послать сигнал из командной строки можно командой `kill`. Например, `kill -9 1234` принудительно завершит процесс 1234, а `kill -STOP 2345` остановит процесс 2345. Как видно, можно использовать как номер сигнала, так и его обозначение. `kill -l` покажет список всех сигналов. Иногда требуется послать сигнал не одному процессу, а многим, например всем процессам пользователя. Тогда на помощь приходит программа `pkill`: `pkill -u vasya -TERM` пошлёт сигнал `TERM` всем процессам пользователя `vasya`.

Выше мы говорили о том, что процессы, желающие выполняться, ставятся в очередь. В ней они выполняются не всегда подряд, у каждого есть приоритет и влияющий на него параметр `nice` (вежливость). Чем выше приоритет, тем быстрее процесс продвигается к началу очереди. Явно задать приоритет нельзя, но можно поменять вежливость (часто её тоже называют приоритетом для простоты, но это не совсем так). Делается это командой nice или renice, первая запускает

программу с заданным приоритетом, вторая меняет приоритет уже запущенной. Чем выше вежливость, тем ниже приоритет, программа чаще будет «пропускать» других вперёд. Исторически вежливость меняется от 20 до +20, и обычный пользователь не может указать её меньше 0. Например:

```
nice -n 15 ./my_program # запустить программу с низким приоритетом
renice -n -10 -p 3322   # повысить приоритет процессу 3322
```

Здесь мы меняем вежливость с 0 до 15 (приоритет понижается) или до -10 (приоритет повышается).

Кроме очереди на ресурсы процессора есть очередь к ресурсам жёстких дисков, несколько процессов могут одновременно читать-писать, и их запросы будут конкурировать. В этой очереди тоже есть приоритет, им управляет команда `ionice`. Есть три класса приоритетов – `idle` (выполнять запрос, если больше никого в очереди нет), `best effort` (нормальная очередь) и `real time` (запрос должен быть выполнен за заданное время). Внутри классов, кроме `idle`, есть собственные приоритеты, но в наших задачах можно ограничиться назначением класса idle процессу, занимающему много времени на дисковых операциях, но не приоритетному:

```
ionice -c 3 -p 89  # задать класс idle процессу с PID 89
```

Понятие процесса – основное в любой ОС. Не следует путать процессы и нити, важно знать, что такое реальная и виртуальная память процесса, как работают разделяемые библиотеки (shared objects, so) и динамический линкер (ld.so). Изучите документацию на эту тему в дистрибутиве своей ОС или в обширной документации в Интернете.

Права

Unix-подобные ОС изначально проектировались как многопользовательские системы, а значит, нужно защитить данные и процессы пользователей от нежелательных посягательств других пользователей. Чуть выше мы уже упомянули права процесса и отметили, что, например, нельзя послать сигнал процессу другого пользователя (если нет особых прав на это). Подобный принцип применяется и для разделения прав на прочие объекты в ОС.

Базовый механизм разделения прав в UNIX-подобных системах основан на понятиях **UID,** или **User ID** (идентификатор пользователя) и **GID,** или **Group ID** (идентификатор группы, которой принадлежит пользователь). UID и GID – числа, но обычно с ними связывают текстовые имена. Каждый процесс имеет «реальные» UID и GID (ruid/rgid), которые не меняются со временем, а также список дополнительных групп, в которые он входит. Кроме реальных, процесс имеет «эффективные» UID и GID (euid/egid), которые определяют его текущие возможности (т. е. именно по ним определяются его права), а также «сохранённые» UID и GID (suid/sgid) – в них копируются эффективные UID/GID при смене UID/GID. Смена UID/GID может происходить, если у процесса есть такое разрешение (capability) или его EUID либо SUID равен 0.

Пользователь с UID = 0 обычно имеет текстовое имя 'root' и имеет почти неограниченные права, поэтому часто его называют «суперпользователь».

Чаще всего, наверное, приходится сталкиваться с правами на файловой системе. Здесь каждый объект (файл, ссылка, каталог, устройство, сокет, канал, далее будем для краткости писать «файл») имеет владельца и группу, а также связанные с ними права – чтение, запись и исполнение. Часто для их записи используют восьмеричную запись или формат команды `ls`. Например, права с восьмеричным кодом `750` (в выдаче `ls` `rwxr-x---`) обозначают, что владельцу разрешено чтение, запись и исполнение (`rwx / 7`), группе – чтение и исполнение (`r-x / 5`), остальным – ничего (`--- / 0`).

Проверка прав производится в таком порядке – если владелец файла совпадает с EUID, берутся права владельца. Иначе, если группа или одна из дополнительных групп совпадают с группой файла, то берутся права группы, и в противном случае – права «остальные».

Для каталога право на «исполнение» означает возможность войти в каталог. При этом посмотреть список файлов в нём не гарантируется – для этого нужно право на чтение. Право на запись означает возможность создавать и удалять файлы в каталоге. Сменить права на файл (или другой объект) может только его владелец. Если нужно сменить права для группы, то владелец должен в неё входить. Ну и конечно, суперпользователь может менять любые права, а также

владельца и группу любого файла.

Как уже было сказано, право записи в каталог позволяет создавать и удалять в нём файлы. В том числе чужие. Чтобы обеспечить комфорт работы с общими каталогами, например `/tmp`, и не позволять удаление чужих файлов, был придуман дополнительный «липкий» флаг (`sticky`). Если каталог обладает им, то удаление файлов разрешается только тем, кто владеет файлом и имеет право записи в каталог (и `root`-у, конечно).

Раз уж мы коснулись файловой системы, то стоит отметить ещё два флага – `suid` и `sgid`. Если файл имеет флаг `suid`, то при его запуске EUID процесса сменится на владельца файла. Для `sgid` – аналогично, но для группы. Чаще всего он ставится на файлы, исполнение которых необходимо с правами суперпользователя, например `passwd`. Для скриптов они не работают. Если флаг `sgid` устанавливается на каталог, то созданные в нём файлы и каталоги автоматически наследуют группу. Флаг `suid` на каталогах игнорируется.

Как уже упоминалось выше, посмотреть права можно командой `ls`. В строке прав первым символом показывается тип файла ('`-`' = файл, '`d`' = каталог, '`l`' = ссылка, '`s`' = сокет и т. п.), далее три группы прав владельца, группы и остальных по три символа '`r`/`-`', '`w`/`-`' '`x`/`-`' для чтения, записи и исполнения соответственно ('`-`' означает отсутствие прав). Флаг `sticky` обозначается символом '`t`' вместо '`x`' в груп-

пе «остальные». флаги suid/sgid – символом 's' вместо 'x' в группе «владелец» или «группа» соответственно. Если за этой строчкой идёт символ '+', это значит, что на этот файл установлены acl (см. ниже).

Менять права на файл можно командой `chmod`. Сменить владельца файла – командой `chown`, группу – `chgrp`. Для того, чтобы сменить права на файл, команде chmod нужно указать новые в восьмеричном виде или в символьном. Последний вариант позволяет добавить или удалить отдельные права для владельца, группы или остальных, например:

```
chmod 660 myfile    # чтение и запись для владельца и группы
chmod g-w myfile    # сброс записи для группы
```

В символьном виде для `chmod` указывается один или более символов 'u/g/o/a' (владелец, группа, остальные, все три группы вместе), затем символ '+' или '–' для установки или сброса прав, и затем один или более символов 'r/w/x' – какие права затрагиваются. Например, `chmod go +rx myfile` добавит права на чтение и исполнение для группы и остальных файлу `myfile`.

Описанная выше система покрывает множество потребностей, но не все. В попытках улучшить её были созданы различные расширения, реализованные в файловых системах Linux. Одно из них – расширенные атрибуты файлов. Они включены по умолчанию, и их можно смотреть и изменять

командами `lsattr` и `chattr`. Самые важные для нас:

Название	Знак	Значение
append only	a	файл нельзя «затереть», только дописывать информацию
compressed	c	использовать сжатие (если поддерживается)
immutable	i	файл нельзя изменять
secure deletion	s	при удалении данные файла затираются
undeletable	u	файл нельзя удалять
no atime updates	A	не обновлять поле «последнее время обращения»

Таблица 4: некоторые расширенные атрибуты

Если расширенный атрибут запрещает какое-то действие (например удаление), то это распространяется даже на суперпользователя (в отличие от обычных атрибутов). Но суперпользователь может легко снять или установить любой из них.

Другое расширение, как правило требующее активации на файловой системе, – **ACL** (Access Control List), списки контроля доступа. Посмотреть и изменить их можно командами `getfacl` и `setfacl`. Они работают аналогично традиционным правам доступа, рассмотренным выше, однако права на чтение/запись/исполнение теперь можно устанавливать отдельным пользователям и/или группам, а также ограничивать эти права маской. Маска – набор «максимальных» прав

из правил acl (традиционные к ним не относятся), которые будут работать. Например, разрешим пользователю `vasya` чтение файла и запись в файл `test.txt`:

```
setfacl -m u:vasya:rw test.txt
getfacl test.txt
# file: test.txt
# owner: root
# group: root
user::rwx
user:vasya:rw-
group::r--
mask::rwx
other::---
```

Здесь ключ '`-m`' указывает модифицировать правила acl. Указав ключ '`--set`', можно заменить правила, т. е. удалить старые и заменить новыми (в `setfacl` можно указать несколько правил одновременно). Ключом '`-x`' правила можно удалять. Строка '`u:vasya;rw`' указывает на то, что правило относится к пользователю (u = user, g = group, o = others), его и<я `vasya,` и устанавливаются права на чтение и запись. Теперь установим маску – разрешим пользователю `vasya` (и другим, имеющим доступ через правила acl)

«не более чем» чтение:

```
setfacl -m m:r test.txt
getfacl test.txt
# file: test.txt
# owner: root
# group: root
user::rwx
user:vasya:rw- #effective:r--
group::r--
mask::r--
other::---
```

Как видим, правило осталось, но права на запись ограничены маской.

Ещё одно полезное свойство acl – наследование правил. На каталог можно установить acl «по умолчанию», они могут не совпадать с acl на сам каталог, и они будут автоматически применяться ко всем создаваемым файлам и каталогам.

Советуем подробнее прочесть о правах в Linux и опциях вышеупомянутых команд, тут мы их коснулись совсем немного.

Понятие сервиса, ключевые сервисы

Выше мы уже не раз использовали термины «сервис», «демон», «служба». Все они обозначают одно и то же: процесс или группу процессов, которые работают постоянно или автоматически запускаются по запросу. Их задача – обслуживать определённые запросы от пользователей, других процессов, других компьютеров в сети. Например, web-сервер `apache` – сервис, обслуживающий запросы по http-протоколу. SMTP-сервер отвечает за запросы на передачу почтовых сообщений и т. д. Рассмотрим сервисы, часто используемые в суперкомпьютерах. О некоторых из них мы уже говорили выше и здесь только перечислим их.

Некоторые сервисы запускаются через «супердемон» `inetd` или более новую его реализацию – `xinetd`. В этом случае в конфигурационном файле `inetd/xinetd` описываются необходимые сервисы: команда запуска, на каком порту слушать, от имени какого пользователя запускать и т. п. После запуска `inetd/xinetd` начинает слушать на указанных портах и при получении запроса запускает соответствующую команду, которой на стандартный поток ввода направляется установленное сетевое соединение. Такой принцип облегчает написание сервиса, а также позволяет более гибко настраивать политику доступа к нему. Например, `xinetd` позволяет задать диапазон адресов, с которых раз-

решён доступ, максимальное число одновременных запросов к сервису и т. п.

Чтобы узнать, работает ли конкретный сервис, можно проверить, запущен ли соответствующий процесс (за исключением сервисов, запускаемых через `inetd/xinetd`), слушает ли какой-либо процесс нужный порт (если сервис привязан к порту) командой `netstat -lpn`.

Основные (но не все) сервисы для кластера:

Имя	Порт	Описание
sshd	22	шифрованный удалённый консольный доступ
nfsd	2049/...	сетевая файловая система. Реальный набор портов определяется сервисом portmap
portmapd	111	управление сервисов RPC
smtp	25/587	приём электронной почты
dns	53	сервер доменных имён. Часто используется реализация bind (named) или dnsmasq
bootps	63	информация для начальной загрузки BOOTP, а также DHCP
tftp	69	trivial file transfer protocol — протокол для загрузки начальных файлов по сети
http	80/443	протокол Всемирной паутины — WWW
ntp	123	синхронизация времени
snmp	161	протокол управления и мониторинга сетевыми устройствами
ldap	389	облегчённый протокол для доступа к каталогам (базам данных)
syslog	514	удалённый журнал
rsync	873	серверная часть команды rsync
nfs	2049	основной сервер для NFS
nut	3493	управление UPS-ами
X11	6000..6000+N	X-сервер
X font server	7100	сервер шрифтов для X-сервера
bacula	9101..9103	резервное копирование bacula
zabbix	10050/10051	сервер мониторинга zabbix

Таблица 5: некоторые стандартные сервисы и их порты

Более полный список можно найти в файле `/etc/services` – он содержит соответствие номера порта традиционно используемому сервису. Некоторые сервисы в нём не представлены ввиду не очень широкого распространения, и, конечно, ничто не мешает запустить какой-либо сервис на нестандартном порту, не забывайте об этом. Через супердемон `inetd/xinetd` часто запускаются такие сервисы, как `tftp, echo, ftp`.

Справка

В системе UNIX несколько тысяч команд, однако в обычной работе пользователю достаточно хорошо знать несколько десятков команд. В этом пособии мы кратко рассмотрим небольшой набор наиболее употребительных команд. В первую очередь потребуются команды для работы с каталогами и файлами. Как и ранее, в квадратных скобках будем указывать необязательные параметры.

Самая главная команда, которая вам потребуется, – `man`. Её имя происходит совсем не от английского «человек» (man), а от «руководство, учебник» (manual). Это – основной источник справочной информации по командам, пакетам и многому другому в UNIX и Linux. Вся информация в `man` разделена по разделам, исторически им присвоены номера. Как правило, для получения справки по какой-то команде достаточно набрать `man имя_команды`. Man найдёт первую страницу с заданным именем и отобразит её.

Номер	Раздел
1	Пользовательские команды
2	Системные вызовы
3	Функции стандартной библиотеки Си
4	Устройства и специальные файлы
5	Форматы файлов и соглашения по форматам
6	Игры и т. п.
7	Разное
8	Системное администрирование и демоны

Таблица 6: разделы справки man

Так как в разных разделах могут быть страницы с одинаковыми именами, то иногда надо явно указать номер раздела. Например, по команде `man crontab` отобразится информация по команде `crontab` из раздела 1. Чтобы показать справку по формату файла `crontab`, надо набрать `man 5 crontab`, отобразить список файлов, в которых упомянуто нужное слово – `man -k слово`. И конечно, не забудьте выполнить `man man`.

Кроме `man`, есть ещё команда `info`, которая была призвана заменить `man`, но, несмотря на массу новых возможностей, так и не стала популярной. Но многие аспекты стандартных программ и сервисов описаны в `info` намного подробнее, чем в `man`.

Соглашения об именах файлов

В именах файлов и каталогов можно использовать любые символы, кроме '/' и '\0'. Одним из самых распространённых средств работы для UNIX является оболочка – shell. В shell некоторые символы имеют специальное значение (которое можно отменить) – это облегчает работу с файлами. Ниже приведён список спецсимволов shell:

Спецсимвол	Значение
backslash (\)	символ экранирования специального значения символов
ampersand (&)	символ исполнения команды в фоновом режиме
parentheses '(', ')'	указывает shell-у запустить команды в новом экземпляре оболочки
angle braces (< и >)	символы перенаправления ввода-вывода
space ' '	разделитель аргументов команд
question mark (?)	означает один любой символ в шаблоне
dollar sign ($)	означает подстановку значения переменной
square braces ([])	определяют диапазон символов
curly braces ({})	определяют список значений в bash
asterisk (*)	любое (включая 0) число любых символов
vertical bar (|)	оператор конвейера
colon (:)	во многих программах отделяет имя удалённого сервера от пути к файлу на этом сервере
semicolon(;)	символ, обозначающий конец команды в shell
newline (\n)	перевод строки также означает конец команды в shell

Таблица 7: спецсимволы shell

UNIX не запрещает использовать эти символы в именах файлов, но необходимо экранировать их специальное назна-

чение символом '\' или заключать их в одинарные кавычки '...'.

Соглашения о расширениях

Расширение файла – часть имени после последней точки; например, файл 'text.cc' имеет расширение '.cc'. Для большинства программ расширение не имеет принципиального значения, но его наличие облегчает понимание назначения файла. Ниже – наиболее часто встречающиеся расширения:

Расширение	Обычное значение
.c	файл с программой на C
.cc .cpp	файл с программой на C++
.h .hpp	Include-файл программы на C/C++
.f .for	файл с программой на фортране
.o	объектный файл
.a	статическая библиотека
.so	динамическая библиотека
.html	HTML-документ
.tar .cpio	архивный файл
.gz .bz2 .7z .zip	сжатый файл

Таблица 8: распространённые расширения файлов

Важно понимать, что расширение файла не имеет принципиального значения для ОС и большинства программ. Поменяв расширение файла на '.exe' или '.sh', вы не сделаете его исполняемым. А вот скрипт с именем 'do_it_now'

можно сделать исполняемым, выполнив 'chmod a+x do_it_now'. Расширения всего лишь упрощают восприятие файлов, давая понять, что это.

Имена, начинающиеся с точки (.), часто присваиваются служебным файлам и каталогам. Эти файлы и каталоги обычно игнорируются программами и файловыми менеджерами по умолчанию. Например, команда `ls` не показывает их, если не указать ключ `'-a'`.

Многие команды допускают в качестве аргумента использование списков имён файлов. Эти списки удобно формировать с помощью шаблонов shell. Рассмотрим их ниже.

Шаблоны

Стандартная оболочка (**shell**) в UNIX – очень мощный инструмент и кроме запуска команд имеет массу возможностей, упрощающих работу в консоли. Самое простое средство – шаблоны имён файлов. Например, написав команду '`ls *.c`', мы получим список всех файлов с расширением '.c' в текущем каталоге.

Важно понимать, что '`*.c`' – не один аргумент, вместо него сам shell подставит нужный список. Если в каталоге только два файла – `1.c` и `2.c`, то будет выполнена команда '`ls 1.c 2.c`'. Если ни одного файла с подходящим под шаблон именем нет, то будет подставлен сам шаблон (т. е. будет выполнена команда '`ls *.c`').

Шаблон	Значение
`*`	соответствует любой строке (кроме '.' в начале имени)
`?`	соответствует любому символу (кроме '.' в начале имени)
`[c1- c2]`	любой символ из диапазона c1..c2
`[!c1-c2]`	любой символ, кроме указанного диапазона (только в bash/zsh)
`{a,b,c}`	точный список слов через запятую (только в bash/zsh)

Таблица 9: шаблоны в shell

Все шаблоны, кроме '{}', применяются к реальному списку файлов и выбирают из него только те, которые попадают под шаблон. С помощью скобок '{}' можно конструировать более сложные шаблоны.

Например, '`ls *.{cxx,h,la}`' превратится в `ls *.cxx *.h *.la`. Более интересный приём – '`cp config{,.bak}`', который превратится в `cp config config.bak`. Второй файл не существует, он явно задан шаблоном.

Если задан шаблон символами '*', '?' или '[]', но под него не попадает ни один файл, то команде будет передан сам шаблон. Например, если каталог пуст, а мы выполняем в нём команду '`ls *.abc`', то выполнится команда '`ls *.abc`', т. е. текст шаблона будет дан команде в качестве аргумента. Будьте осторожны со случайно или намеренно созданными файлами, начинающимися с тире, так как их имена после раскрытия шаблона могут быть восприняты командой как имя управляющего ключа команды!

Чтобы отменить действие спецсимвола, достаточно поставить перед ним обратную косую черту '\' или заключить весь аргумент в одинарные кавычки. Например, если мы хотим удалить файл с именем «`--rf *.?`», то можно использовать команду:

```
rm - -rf\ \*.\?
```

или

```
rm – '-rf *.?'
```

Обратите внимание на первый аргумент '`--`' – он нередко используется в командах Linux и обозначает «здесь закончились ключи, далее только имена файлов». В данном случае он не обязателен, но, к примеру, если потребуется удалить файл с именем '`-f`', то команда '`rm -f`' не сработает, так как '`-f`' – это ключ команды `rm`. Сработает команда '`rm – -f`'.

Команды для работы с деревом каталогов

`pwd` – напечатать полное имя текущего каталога.

`cd [ dirname ]` – перейти в указанный каталог (в домашний каталог, если dirname не задано); dirname здесь – имя каталога, которое может состоять из собственно имени и пути к нему. Путь может быть абсолютным, если он начинается с символа /, и относительным, если начинается с любого другого символа.

Примеры перемещения по дереву каталогов:

`cd /export/home/user1` – переход в домашний каталог пользователя user1;

`cd /` – переход в корневой каталог файловой системы;

`cd prog/cc` – переход из текущего каталога в каталог cc, находящийся в каталоге prog;

`cd ../gosha/bin` – возврат на шаг назад и переход в каталог bin пользователя gosha;

`cd` – переход в свой домашний каталог.

Специальные имена каталогов:

. (точка) – текущий каталог;

.. (две точки) – родительский каталог по отношению к текущему.

В bash или zsh можно использовать спецсимволы, которые shell преобразует в имена каталогов:

~ (тильда) – домашний каталог;

`~name` – домашний каталог пользователя name;

– (тире) – возврат в предыдущий каталог (опция встроенной команды cd).

Команды для работы с каталогами

`mkdir [опции] имя_каталога ...` – создать новые каталоги.

Опции:

`-m mode` – задать права доступа;

`-p` – создавать при необходимости родительские каталоги.

`rmdir имя_каталога ...` – удалить каталоги (каталоги должны быть пустыми).

`ls [опции/имена]` – выводит содержимое каталога или атрибутов файлов.

`имена` – это имена каталогов или файлов. Если имена не указаны, то выводится содержание текущего каталога.

Наиболее часто используются опции:

`-a` – вывести все файлы (даже если имена начинаются с точки);

`-l` – вывести подробную информацию о файлах и папках (права доступа, имя владельца и группы, размер в блоках по 512 байт, время последней модификации, имя файла или каталога);

`-t` – имена файлов сортируются не по алфавиту, а по времени последнего изменения;

`-R` – рекурсивно пройти по всем подкаталогам, выводя по ним информацию.

Команды для работы с файлами

`touch [опции] имя_файла` – создать файл, если он не существовал, или изменить время последнего изменения файла.

`rm [опции] имя_файла ...` – удаление файлов

опции

`-i` – интерактивное удаление (с требованием подтверждения);

`-f` – без выдачи сообщений;

`-r` – рекурсивное удаление каталогов вместе с содержимым.

Примеры:

```
rm file1 file2    # удаление файлов file1 и file2
rm data  # удаление пустого каталога
rm -r data    # удаление не пустого каталога
rm /tmp/file1 # удаление файла по полному имени
```

Для задания списка файлов можно использовать шаблоны, но пользоваться ими следует крайне осторожно. Команда

`rm test*` удалит все файлы с именами, начинающимися на `test`;

`rm test *` (после test стоит пробел) удалит вообще все файлы в каталоге (кроме начинающихся на точку).

`mv [опции] источник назначение` – перемещение файлов и каталогов.

Опции:

`-i` – интерактивное перемещение (с требованием подтверждения);

`-f` – без выдачи сообщений.

Команда **mv** выполняет множество функций в зависимости от типа аргументов.

1) **Переименовывает** файлы и каталоги, если оба аргумента являются либо файлами, либо каталогами:

`mv file1 file2` – в рабочем каталоге файл `file1` получит имя `file2`;

`mv dir1 dir2` – если `dir2` не существовал в рабочем каталоге, то каталог `dir1` получит имя `dir2`; если `dir2` существовал, то каталог `dir1` будет перемещён в него.

2) **Перемещает** файл или каталог в другой каталог с тем же именем или другим:

`mv file1 dir2` – перемещает `file1` из рабочего каталога в каталог `dir2` с тем же именем;

`mv file1 dir2/file2` – перемещает `file1` из рабочего каталога в каталог `dir2` с именем `file2`.

Если источником является список файлов, а назначением – каталог, то можно использовать шаблоны:

`mv file* ../dir2` – перемещает все файлы, имена которых начинаются со строки `file`, в каталог одного уровня с рабочим.

Во всех операциях объекты, выступающие в качестве источника, исчезают: меняют имя или расположение.

`cp [опции] источник назначение` – копирование файлов и каталогов.

Опции:

`-i` – интерактивное копирование (с требованием подтверждения, если объект **назначение** уже существует);

`-f` – без выдачи сообщений;

`-r` – рекурсивное копирование каталогов вместе с содержимым;

`-p` – копирование с сохранением атрибутов файлов (прав доступа, времени модификации).

Примеры:

`cp file1 file2` – будет создана копия файла `file1` в файле с именем `file2`;

`cp file1 dir2` – будет создана копия файла `file1` в каталоге `dir2` (т. е. с именем `dir2/file1`);

`cp -r dir1 dir2` – будет создана копия каталога `dir1` в каталоге `dir2`;

`cp file1 file2 file3 /tmp` – копирует файлы с именами `file1`, `file2`, `file3` в подкаталог `tmp` корневого каталога. Это можно выполнить командой:

```
cp file* /tmp
```

```
cat [опции] [файл][файл]...
```

Команда `cat` объединяет файлы и выдаёт их на стандартный поток вывода. Если аргумент **файл** отсутствует, то команда `cat` будет принимать входной поток из стандартного файла ввода (клавиатуры). Поскольку команда работает со стандартным файлом вывода (терминалом), то чаще всего она используется для просмотра на экране содержимого файла. Не рекомендуется выдавать на экран бинарные файлы.

`cat ls.txt` – выводит содержимое файла с именем `ls.txt` на терминал;

`cat ls1.txt ls2.txt ls3.txt` – по очереди выводит на терминал содержимое файлов `ls1.txt, ls2.txt, ls3.txt`;

`cat ls1.txt ls2.txt ls3.txt > lsall.txt` – объединяет три файла в один. При этом старые файлы сохраняются. Если файл `lsall.txt` уже существовал, то он затрётся новым содержимым. Можно дописать в конец файла, если использовать для перенаправления знак >> (два знака «больше»).

Команду `cat` можно использовать для создания файла:

`cat > ls.txt` – все набранное на клавиатуре будет

записано в файл `ls.txt`. Оборвать ввод можно сочетанием клавиш `Ctrl-D`.

Команда `cat` выдаёт все содержимое на экран. Если файл большой, то на экране можно будет увидеть только последние строки.

Для просмотра текстовых файлов порциями можно напрямую использовать команды:

- `more file.txt`
- `less file.txt`

Команда `less` содержит большой набор внутренних команд для перемещения по файлу, поиска контекста и даже редактирования:

Команда	Значение
пробел	перемещение вперёд на один экран
b	перемещение назад на один экран
Return	перемещение вперёд на одну строку
Ctrl-b	перемещение назад на один экран
g	переход в начало файла
#g	переход на заданную строку
G	переход в конец файла
F	переход в режим команды tail -f; для выхода из режима нажмите Ctrl-C
/строка	поиск строки далее в файле
?строка	поиск строки в файле обратно (вверх)
n	поиск следующего вхождения ранее ведённого слова
h	список доступных команд

Таблица 10: некоторые клавиатурные команды less

`tail [опции] файл` – просмотр конца файла. По умолчанию выдаётся 10 последних строк. С помощью опций можно начать просмотр с любой позиции.

Опции:

`-n number` – сколько выдавать строк;

`-r number` – отображение в обратном порядке;

`-f` – непрерывная выдача файла по мере его заполнения.

Прерывание интерактивной выдачи комбинацией `Ctrl-C`.

`grep [опции] строка [файл][файл]...` – поиск контекста «строка» в указанных файлах.

Опции:

`-i` – поиск без учёта регистра;

`-n` – отображать номера строк, содержащих контекст;

`-v` – отображать строки, не содержащие контекста.

`find [опции] каталог выражение` – рекурсивный поиск файлов в указанном каталоге по различным атрибутам, таким как имя, размер, время модификации, права доступа.

Выражения:

`-name filename` – поиск файла с именем `filename`. Возможно использование шаблонов, но тогда надо брать их в кавычки `'test*'` либо экранировать символы шаблона `test\*`;

`-size [+|-]number` – поиск файлов с заданным раз-

мером, превышающим его (+) или меньшим (–). Размер указывается в блоках по 512 байт;

`-atime number` – поиск файлов, к которым происходил доступ number суток назад;

`-mtime number` – поиск файлов, которые были модифицированы number суток назад;

`-exec command \{\} \;` – выполнить команду `command` над списком файлов, найденных командой `find`. Здесь выражение «`{}`» будет заменяться именем найденного файла, а ';' означает конец команды. Так как эти символы обрабатываются оболочкой, то их надо экранировать, например:

```
find . -name 'core.*' -exec rm \{\} \;
```

– рекурсивно удалить все core-файлы, начиная с текущего каталога.

Следует отметить, что многие действия из перечисленных выше и связанных с манипуляциями с каталогами и файлами можно выполнять с помощью специальной программы – файлового менеджера **Midnight Commander**. Он не требует графической оболочки, вызывается в терминальном окне командой:

```
mc
```

С помощью этой программы можно перемещаться по дереву каталогов, просматривать содержимое каталогов и файлов, создавать каталоги (но не файлы), удалять, копировать, перемещать каталоги и файлы, вести поиск файлов. Для многих пользователей текстовый редактор Midnight Commander является очень хорошим выбором. Его можно вызвать отдельно командой `mcedit`.

Редактирование файлов – отдельная важная тема. Существует большое число редакторов, работающих как в текстовом, так и в графическом режимах. Нас как администраторов в первую очередь будет интересовать редактор, который может работать в самых сложных условиях – без графического интерфейса, возможно, по сети, когда функциональные клавиши недоступны или работают неверно. Таких редакторов существует несколько, например `gnu nano`. Но, на наш взгляд, самый гарантированно работающий вариант, который, ко всему прочему установлен на 99% Linux-систем, – это редактор `vi`.

Его интерфейс на первый взгляд совсем не дружественен и не логичен, на деле же большинство его команд продуманы и удобны. Главное его преимущество – возможность работы практически в любых условиях и быстрое выполнение массовых операций (поиск, замена и т. п.). Он имеет два режима работы – командный и режим вставки. Изначально файл открывается в командном режиме. Для перемещения по тексту

используйте клавиши курсора, если же они не работают, – клавиши 'h, j, k, l' (посмотрите на клавиатуру и поймёте, почему такой странный набор). Клавиши w и b перемещают вперёд и назад на слово. И конечно, «главная» команда – выход с сохранением: ':wq' или просто 'ZZ' (заглавными буквами). Выйти без сохранения можно командой ':q!'.

Полезные команды:

Команда	Значение
x	удалить символ под курсором (и скопировать его в буфер)
dd	удалить строку (и скопировать её в буфер)
yy	скопировать строку в буфер
p/P	вставить текст из буфера перед текущей строкой/ после неё
r	заменить символ
R	писать поверх старого текста
cw	заменить слово
i/a	перейти в режим вставки в текущей позиции/за ней
o/O	добавить новую строку после текущей/перед ней
/	поиск по регулярному выражению (? — поиск назад)
n	повтор поиска
.	повтор предыдущей команды
u	отмена предыдущей команды
NN<кмд>	повторить NN раз команду <кмд>
:	расширенная команда

Таблица 11: некоторые клавиатурные команды vi

Режим вставки позволяет вписывать текст в нужную пози-

цию. Выйти из него можно клавишей <ESC>. Перед любой командой можно набрать число, тогда команда будет повторена это число раз. Например, '10dd' удалит 10 строк (и поместит их вместе в буфер, потом можно будет их вставить в другом месте командой 'p'). Повтор команды вставки или замены повторит и ввод, например ввод 'cwNEW_WORD<ESC>' приведёт в замене слова после курсора на 'NEW_WORD', а если потом переместить курсор на начало другого слова и нажать '.', то оно также будет заменено на 'NEW_WORD'.

Из расширенных команд особенно удобна команда массовой замены 's'. Её синтаксис взят из команды sed. Перед командой можно указать через запятую диапазон строк, на которые она будет действовать, при этом '.' обозначает текущую строку, $ – последнюю, а знак + указывает на то, что номер дан относительно текущей строки.

Например, заменить адрес old-cluster на new-supercomputer в 10 строках после текущей включительно можно, набрав:

```
:.,+10s/old-cluster/new-supercomputer/g<Enter>
```

Очень непривычно, но очень эффективно. Обязательно почитайте учебник по vi и попробуйте использовать его для редактирования. Такие возможности, как быстрая замена слов и предложений, исправление переставленных местами букв, моментальная навигация, поддержка работы с фай-

лами огромных объёмов делают его исключительно эффективным для редактирования файлов конфигурации, журналов и многих других.

Пакеты

Во всех Linux-дистрибутивах есть прекрасная (на наш взгляд) система – упаковка ПО в так называемые «пакеты». Самих вариантов систем пакетирования много, наиболее популярны **rpm** (RedHat, Fedora, CentOS, SuSE и другие), **deb** (Debian, Ubuntu, Mint и другие), **ports** (Arch Linux и производные), **ebuild** (Gentoo и производные), **pkg** (Slackware и производные). Все они предлагают хранение дерева всех файлов некоторого ПО, например web-сервера, или его части, например модуля шифрования, в одном файле (обычно это сжатый архив). Кроме файлов в пакете хранятся метаданные, такие как название пакета, описание и другие данные. Набор метаданных в разных пакетных системах разный, поэтому и возможности тоже разные.

Самые важные особенности метаданных пакетов на наш взгляд:

• зависимости – указание других пакетов, установка которых необходима или желательна. Вместо пакетов может указываться функция (например smtp-сервер), если из метаданных её можно получить;

• хэш-суммы файлов;

• указание, какие файлы являются конфигурационными.

С помощью зависимостей установка ПО становится намного проще, можно быстро выяснить, какие дополни-

тельные пакеты необходимо установить. Часто вычисление и установку всех дополнительных пакетов берут на себя «пакетные менеджеры», такие как **yum** (RedHat), **apt** или **aptitude** (Debian), **pacman** (Arch Linux) и т. п.

В формате **deb** зависимости можно указывать гибко: от «необходимо» до «желательно» (обычно это документация и примеры) и «опционально», поэтому в интерактивном пакетном менеджере типа **aptitude** можно выбрать именно тот набор дополнительных пакетов, который нужен. В мире формата rpm нередко с пакетом автоматически устанавливается ещё десяток ненужного ПО, которое теоретически может с ним потребоваться…

Имея хэш-суммы всех файлов пакета, можно проверять целостность системы, не изменились ли какие-то важные файлы (конечно, только из тех, что входят в пакеты), в большинстве пакетных систем для этого есть отдельные команды.

Знание файлов конфигурации позволяет не затирать их при обновлении – новая версия копируется «рядом» с оригинальной, чтоб можно было проверить разницу. А в интерактивном режиме **aptitude** можно эту разницу сразу посмотреть и выбрать, какой вариант использовать, старый или из обновлённого пакета. Аналогично при удалении пакета, как правило, его файлы конфигурации остаются и при новой установке не затираются. Удалить пакет вместе с настройками обычно можно, указав дополнительный ключ при удалении. Информация об установленных на компьютере пакетах

хранится в базе данных, если она повреждается, то восстановить её очень тяжело. Поэтому не рекомендуем удалять или ставить пакеты на файловой системе, занятой на 100%, это может привести к потере базы.

Сами файлы пакетов можно просто скачать или скопировать откуда-либо, но чаще всего используются так называемые **репозитории** – индексированные каталоги пакетов. На CD/DVD-диске с дистрибутивом ОС почти всё место занимает репозиторий пакетов, после установки он будет прописан в настройках. Кроме него почти наверняка будет указан основной сетевой репозиторий ОС (или несколько). Кроме основного репозитория могут потребоваться дополнительные или даже совсем сторонние. Например, в стандартном репозитории RedHat не так уж и много пакетов, очень часто приходится подключать проверенные сторонние, типа EPEL. Некоторые программные проекты создают собственные мини-репозитории только для своего ПО. Иногда имеет смысл создавать и собственные локальные репозитории, например для компьютеров без доступа в Интернет.

Самые важные варианты использования команды `yum` и полезные ключи:

Команда	Значение
install pkg1 pkg2 ...	установить пакеты и их зависимости
erase pkg1 pkg2 ...	удалить пакеты
update	обновить все пакеты в системе, если есть новые версии
check-update	проверить обновления установленных пакетов
reinstall pkg1 pkg2...	установить пакеты заново
--enablerepo=REPO	временно включить репозиторий
--nogpgcheck	отключить проверку подписи (осторожно!)
--skip-broken	пропустить проверку зависимостей, используйте, только если на 100% уверены, что это необходимо
--downloadonly	не устанавливать, только скачать
--downloaddir=DIR	каталог для скачивания пакетов

Таблица 12: важные команды и ключи команды yum

Когда приходится работать с конкретным пакетом или rpm-файлом, то на помощь приходит команда `rpm`. Её важные ключи:

Ключ	Значение
Режимы	
-q	режим поиска — получить информацию о пакете
-i	режим установки пакета
-U	режим обновления пакета
-e	режим удаления пакета
Ключи выбора	
-a	выбрать все пакеты
-f path	найти пакет, владеющий этим файлом/каталогом
-p file	пакет по rpm-файлу
Ключи установки/проверки/удаления	
--nodeps	отключить проверку зависимостей
--force	перезаписывать файлы, если они конфликтуют с файлами других пакетов, разрешить ставить более раннюю версию пакета

Таблица 13: некоторые ключи команды rpm

Сетевые команды

Поскольку вычислительные кластеры по своей сути являются сетевыми структурами, то важную роль для администраторов кластеров играют сетевые команды. Рассмотрим наиболее важные из них.

Команда `ping` – команда для проверки соединения между двумя компьютерами в сетях, построенных на базе стека протоколов TCP/IP. Команда отправляет на другой компьютер запросы Echo-Request по протоколу ICMP и принимает поступающие ответы. Засекая время между отправкой запроса и получением ответа, программа определяет задержку в передаче пакетов по маршруту и частоту потерь пакетов, позволяя оценить качество сетевого соединения между двумя узлами.

Синтаксис команды:

```
ping [опции] имя узла или его IP-адрес
```

Пример:

```
ping host1.mynet
```

Результат работы команды:

```
PING host1.mynet (10.0.1.2) 56(84) bytes of data.
64 bytes from host1.mynet (10.0.1.2): icmp_seq=1 ttl=64 time=4.69 ms
64 bytes from host1.mynet (10.0.1.2): icmp_seq=2 ttl=64 time=0.169 ms
64 bytes from host1.mynet (10.0.1.2): icmp_seq=3 ttl=64 time=0.120 ms
--- host1.mynet ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 0.120/1.661/4.694/2.144 ms
```

При запуске без специальной опции команда ping в UNIX-подобных системах работает неограниченно долго, посылая запросы указанному узлу. Каждый отправленный запрос имеет свой номер, по которому программа определяет, дошёл он до целевого компьютера или нет. В выводе команды номер запроса показывает поле `icmp_seq`, поле `ttl` – Time To Live – определяет время жизни ответного пакета, заданное в числе узлов. Ровно столько узлов пакет может пройти, передаваясь по маршруту до узла назначения.

Каждый узел, через который проходит пакет, уменьшает величину `ttl` на единицу; если значение счётчика станет равным нулю, то пакет будет уничтожен как «заблудившийся» и не будет отправлен дальше по маршруту. Последнее поле показывает время обмена сообщениями между двумя узлами. Оборвать работу команды `ping` можно с терминала, нажав комбинацию клавиш Ctrl-C, после чего команда `ping` выведет статистику работы: сколько пакетов было отправлено, сколько получено, процент потерянных пакетов, общее время работы в миллисекундах. Кроме того, выводится минимальное, среднее и максимальное время прохождения па-

кетов.

Основные опции команды `ping`:

`-c count` ограничивает число посылаемых пакетов значением count;

`-n` отменяет преобразование IP-адреса отвечающего узла в его DNS-имя. Такой режим может ускорить работу программы и исключить проблемы с настройками DNS при диагностике сети;

`-i interval` задаёт время ожидания перед посылкой следующего пакета;

`-l size` задаёт размер пакета.

Эта команда может служить в том числе для тестирования сети InfiniBand, если на интерфейсах InfiniBand поднят протокол IPoIB (IP over InfiniBand). Если вы поняли, что удалённый узел или сеть недоступны, можно выяснить, где происходит обрыв связи. Для этого используется команда `traceroute` или её более современный аналог `tracepath`. В качестве аргумента команда принимает адрес узла.

Она посылает пакеты `ping` на этот узел со значением ttl, равным 1, затем 2 и т. д. В выдаче программы видно, какие узлы по пути следования пакета обработали факт обнуления ttl и сообщили об этом. Таким образом, мы можем отследить путь пакета.

Пример работы команды `traceroute`:

```
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
1  333.444.9.161  0.233 ms  0.223 ms  0.226 ms
2  333.444.9.1  1.262 ms  1.660 ms  2.143 ms
3  333.444.1.8  0.651 ms  0.652 ms  0.959 ms
4  333.444.0.190  0.933 ms  0.935 ms  0.943 ms
5  193.232.246.232  0.915 ms  1.187 ms  1.176 ms
6  72.14.236.220  8.712 ms  9.226 ms  9.221 ms
7  209.85.243.135  51.475 ms
8  209.85.249.79  23.483 ms  24.171 ms
9  72.14.233.168  23.33 ms  12.3 ms  24.5 ms
10  * * *
11  8.8.8.8  12.678 ms  12.509 ms  23.694 ms
```

Видно, что 10-й по счёту узел не ответил; это значит, что он просто игнорирует пакет, не уведомляя об этом отсылающего.

Полезные опции команды `traceroute`:

`-n` не преобразовывать DNS-имена узлов,

`-f N` начать с TTL с указанным номером,

`-m N` ограничить TTL указанным числом (по умолчанию 30),

`-w N` время ожидания отклика (по умолчанию 5 сек.).

Команда `route` показывает текущую таблицу маршрутизации, т. е. правила, по которым узел определяет, куда послать пакет. Типичный вывод команды:

```
# route -n
Kernel IP routing table
Destination  Gateway    Genmask         Flags Metric Ref   Use Iface
9.10.11.12   0.0.0.0    255.255.255.0   U     0      0       0 eth1
10.0.0.0     0.0.0.0    255.0.0.0       U     0      0       0 eth0
0.0.0.0      9.10.11.1  0.0.0.0         UG    0      0       0 eth1
```

Значения столбцов:

• `Destination` – адрес назначения пакета;

• `Gateway` – адрес хоста(роутера), куда будет направлен пакет;

• `Genmask` – маска адреса(destination);

• `Flags, Metric, Ref, Use` – служебная информация;

• `Iface` – имя интерфейса, куда будет передан пакет.

Если необходимо передать пакет по сети на адрес x.y.z.q, ядро последовательно проверит этот адрес по таблице: на адрес и на поле `destination` будет наложена маска (`genmask`), и если результаты совпадут, то пакет будет пересылаться на роутер (`gateway`) через сетевой интерфейс (`interface`). Наложение маски производится битовой операцией AND, т. е. все биты, установленные в маске в 0, будут в результате сброшены в 0, а биты, установленные в маске в 1, будут в результате такими же, как и у исходного адреса.

Отсюда, в частности, следует, что маска `0.0.0.0` задаёт маршрут, который сработает всегда, так как результат её применения всегда будет `0.0.0.0`. Такой маршрут часто называют `default` (по умолчанию). В нашем примере сеть `9.10.11.*` доступна через `eth1`, сеть `10.*.*.*` – через интерфейс `eth0` (это внутренняя сеть), а все остальные пакеты направляются на роутер `9.10.11.1`, который доступен через интерфейс `eth1`.

Командой `route` можно также добавлять и удалять маршруты. Для добавления маршрута к сети используйте:

```
route add -net 1.2.3.0 netmask 255.255.255.0 gw 1.2.3.1 dev eth0
```

Здесь мы добавляем маршрут для сети 1.2.3.* на интерфейсе eth0:

```
route add default gw 1.2.3.4
```

Эта команда – сокращённый вариант команды

```
route add -net 0.0.0.0 netmask 0.0.0.0 gw 1.2.3.4
```

Интерфейс определяется автоматически, если роутер (gw) доступен через другие правила. Если заменить в предыдущих командах '`add`' на '`del`', получим команду удаления маршрута. Обратите внимание, что при удалении надо также указать все параметры: `netmask, gw, dev` и т. п., даже если они очевидны, иначе команда может не отработать.

Команда `ifconfig` управляет работой сетевого интерфейса. Без аргументов она показывает состояние активных интерфейсов:

```
eth0  Link encap:Ethernet  HWaddr 00:10:21:37:40:5F
      inet addr:10.0.0.2  Bcast:10.255.255.255  Mask:255.0.0.0
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:25400911846 errors:0 dropped:5419 overruns:0 frame:0
      TX packets:22217149338 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:13736261041 (12.4 GiB)  TX bytes:10704560149 (9.7 GiB)
      Memory:d8900000-d8920000

eth1  Link encap:Ethernet  HWaddr 00:10:21:37:40:5E
      inet addr:9.10.11.12  Bcast:9.10.11.255  Mask:255.255.255.0
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:3419282263 errors:0 dropped:0 overruns:0 frame:0
      TX packets:5796890559 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:1000
      RX bytes:1111996013 (1.0 GiB)  TX bytes:7592797386 (6.9 GiB)
      Memory:d8920000-d8940000

lo    Link encap:Local Loopback
      inet addr:127.0.0.1  Mask:255.0.0.0
      UP LOOPBACK RUNNING  MTU:16436  Metric:1
      RX packets:27777839 errors:0 dropped:0 overruns:0 frame:0
      TX packets:27777839 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:0
      RX bytes:10171240527 (9.4 GiB)  TX bytes:10171240527 (9.4 GiB)
```

Тут мы видим MAC-адреса карт (HWaddr), IP-адреса интерфейсов (inet addr), широковещательный адрес сети и маску сети (Bcast, Mask), а также статистику:

`RX/TX packets` – передано/принято пакетов;

`RX/TX bytes` – передано/принято байт;

`UP BROADCAST RUNNING MULTICAST` – состояние карты;

`MTU` – размер фрейма Ethernet;

`txqueuelen` – лимит очереди пакетов;

`errors` – число ошибок;

`dropped` – число сброшенных пакетов;

`overruns` – число переполнений буфера;
`frame` – число ошибок при принятии фрейма;
`carrier` – число потери связи;
`collisions` – число коллизий при передаче.

Чтобы посмотреть данные о всех, а не только о работающих интерфейсах, запустите `ifconfig` с ключом `-a`. С правами root командой `ifconfig` можно управлять параметрами интерфейсов. Быстро отключить интерфейс `eth0` можно командой `ifconfig eth0 down`, включить обратно – `ifconfig eth0 up`. Пример быстрой настройки интерфейса и его адреса:

```
ifconfig eth0 inet 192.168.0.1 netmask 255.255.255.0
```

Эта команда задаст для интерфейса eth0 адрес `192.168.0.1` и маску `255.255.255.0`. После этого надо включить (поднять) интерфейс командой `ifconfig eth0 up`. В большинстве реализаций команда `ifconfig` автоматически создаёт правило маршрутизации.

В современном ядре Linux команды `route` и `ifconfig` считаются устаревшими, и на смену им пришла команда `ip` из пакета `iproute2`. Настоятельно рекомендуем использовать новые команды, если вы всё ещё не применяете их.

Формат команды `ip` прост:

```
ip [опции] объект команда
```

«Объектом» может быть одна из более чем десяти подсистем, здесь мы кратко рассмотрим только некоторые. «Команда» – действие, которое мы хотим выполнить. С помощью необязательных опций можно, например, ограничить действие команды только сетями ipv4 или запросить более подробный вывод.

Конец ознакомительного фрагмента.