Министерство образования и науки Российской Федерации
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

А.А. МАЛЯВКО, С.А. МЕНЖУЛИН

СУПЕРКОМПЬЮТЕРЫ И СИСТЕМЫ ПОСТРОЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ КЛАСТЕРОВ

Утверждено
Редакционно-издательским советом университета
в качестве учебного пособия

НОВОСИБИРСК
2018

УДК 004.272.43:004.382.2(075.8)
М 219

Рецензенты:
канд. техн. наук, доцент *В.Г. Качальский*
канд. техн. наук, доцент *С.П. Ильиных*

Работа подготовлена на кафедре вычислительной техники для студентов, обучающихся по направлениям 09.03.01 – Информатика и вычислительная техника и 09.03.04 – Программная инженерия

Малявко А.А.

М 219 Суперкомпьютеры и системы. Построение вычислительных кластеров: учебное пособие / А.А. Малявко, С.А. Менжулин. – Новосибирск: Изд-во НГТУ, 2018. – 96 с.

ISBN 978-5-7782-3633-2

В пособии содержатся материалы, которые можно использовать при изучении структурной и функциональной организации кластерных вычислительных систем, а также для освоения принципов их использования. Описываются архитектурные решения четырех актуальных программных продуктов, имеющих практическую ценность для организации параллельных вычислений: PelicanHPC, Torque, HTCondor, Slurm.

Пособие адресовано студентам, обучающимся по направлениям 09.03.01 «Информатика и вычислительная техника» и 09.03.04 «Программная инженерия», а также может быть полезно студентам-старшекурсникам и аспирантам других специальностей и преподавателям смежных дисциплин.

УДК 004.272.43:004.382.2(075.8)

ISBN 978-5-7782-3633-2

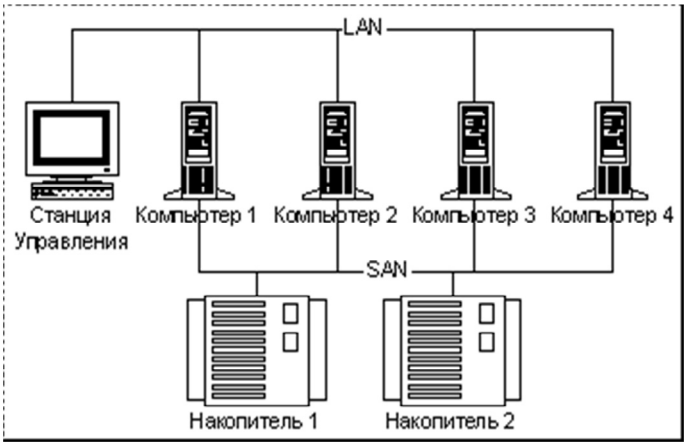
ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Кластер [1] – это группа вычислительных машин, которые связаны между собой и функционируют как один узел обработки информации. Как правило, кластер представляет собой группу однородных вычислительных узлов, соединенных высокоскоростным каналом связи.

B1. Концепция кластерных систем

Согласно концепции определения кластерной системы структура типичного кластера представлена на рисунке, где LAN – Local Area Network, локальная сеть; SAN – Storage Area Network, сеть хранения данных.



Кластерная система

Как правило, кластерные системы применяются при решении сложных научных задач. Это могут быть задачи физики, математики, механики и т. д. Активное применение кластерные системы находят в решении задач моделирования и Data Mining. Анализ больших объемов данных требует высокой скорости их обработки. В связи с этим задачи анализа данных могут также решаться при помощи вычислительных кластеров.

Для обеспечения высокой производительности и эффективности работы вычислительных кластеров требуются самые производительные и современные компоненты. В силу специфики предметной области изготовление такого оборудования требует повышенного внимания к таким показателям, как высокая производительность и надежность. Наиболее известные производители аппаратного обеспечения имеют специализированные линейки оборудования, включающие в себя исключительно аппаратное обеспечение для вычислительных кластеров (графические карты NVidia Tesla, AMD FirePro; кластерные системы HP, IBM, Fujitsu). В сфере производства высокопроизводительных решений для суперкомпьютерных кластеров присутствуют компании, специализирующиеся на поставках оборудования для суперкомпьютеров (Т-платформы, Ниагара).

B2. Требования к аппаратному обеспечению

Оборудование суперкомпьютерных кластеров должно отвечать ряду требований для обеспечения заданного уровня производительности. К таким требованиям можно отнести:

- высокую готовность;
- высокое быстродействие;
- масштабирование;
- общий доступ к ресурсам;
- удобство обслуживания.

Следует учитывать, что в зависимости от решаемых задач отдельным пунктам из этого списка будет назначен больший приоритет.

В3. Пути достижения параллелизма

Под параллельными вычислениями понимаются процессы обработки данных, в которых одновременно могут выполняться несколько машинных операций. Достижение параллелизма возможно только при выполнении следующих требований к архитектурным принципам построения вычислительной системы:

- независимость функционирования отдельных устройств вычислительной системы – требование, относящееся в равной степени ко всем ее основным компонентам: к устройствам ввода-вывода, к обрабатывающим процессорам и к устройствам памяти;
- избыточность элементов вычислительной системы; организация избыточности может осуществляться в следующих основных формах:

– использование специализированных устройств, например, таких как отдельные процессоры для целочисленной и вещественной арифметики, устройства многоуровневой памяти (регистры, кэш);

– дублирование устройств ЭВМ путем использования, например, нескольких однотипных обрабатывающих процессоров или нескольких устройств оперативной памяти.

Дополнительной формой обеспечения параллелизма может служить конвейерная реализация обрабатывающих устройств, при которой выполнение операций в устройствах представляется в виде исполнения последовательности составляющих операцию подкоманд; как результат, при вычислениях на таких устройствах могут находиться на разных стадиях обработки одновременно несколько различных элементов данных.

При рассмотрении проблемы организации параллельных вычислений следует различать следующие возможные режимы выполнения независимых частей программы:

- многозадачный режим (режим разделения времени), при котором для выполнения процессов используется единственный процессор; данный режим является псевдопараллельным, когда активным (исполняемым) может быть один-единственный процесс, а все остальные процессы находятся в состоянии ожидания своей очереди на использование процессора. Использование режима разделения времени может повысить эффективность организации вычислений (например, если

один из процессов не может выполняться из-за ожидания вводимых данных). Процессор может быть задействован для исполнения готового процесса. Кроме того, в данном режиме проявляются многие эффекты параллельных вычислений (необходимость взаимоисключения и синхронизации процессов и др.), и, как результат, этот режим может быть использован при начальной подготовке параллельных программ;

- параллельное выполнение – когда в одно и то же время может выполняться несколько команд обработки данных; такой режим вычислений может быть обеспечен не только при наличии нескольких процессоров, но и реализуем при помощи конвейерных и векторных обрабатывающих устройств;
- «распределенные вычисления» – термин, который обычно применяется для указания параллельной обработки данных, при которой используется несколько обрабатывающих устройств, достаточно удаленных друг от друга и в которых передача данных по линиям связи приводит к существенным временным задержкам. В результате эффективная обработка данных при таком способе организации вычислений возможна только для параллельных алгоритмов с низкой интенсивностью потоков межпроцессорных передач данных. Перечисленные условия являются характерными, например, при организации вычислений в многомашинных вычислительных комплексах, образуемых объединением нескольких отдельных ЭВМ с помощью каналов связи локальных или глобальных информационных сетей.

В4. Методы построения вычислительных кластеров

Высокопроизводительные вычисления на основе кластеров стали появляться сравнительно недавно. До конца 1980-х гг. практически все суперкомпьютеры представляли собой большой массив соединенных между собой процессоров. Подобные разработки чаще всего были уникальными и имели огромную стоимость не только в смысле приобретения, но и поддержки. Поэтому в 1990-х гг. все более широкое распространение стали получать кластерные системы, которые в качестве основы используют недорогие однотипные вычислительные узлы.

К основным преимуществам кластеров можно отнести относительно низкую стоимость вычислительных узлов и легкую расширяемость.

Основной их недостаток – сложность отладки параллельных программ для систем с разделенной памятью.

Как уже было отмечено, кластеры представляют собой системы с разделенной памятью, поэтому типичное параллельное приложение является совокупностью нескольких процессов, исполняемых на разных вычислительных узлах и взаимодействующих по сети. В принципе, разработчик может полностью взять на себя программирование распределенного приложения и самостоятельно реализовать общение по сети, например, на основе сокетов. Однако в настоящее время существует довольно много технологий, упрощающих создание параллельных приложений для кластеров (например, MPI). Эти технологии существуют уже достаточно продолжительное время, в течение которого они доказали свою состоятельность и легли в основу огромного числа параллельных приложений.

Кластеры стали фактическим стандартом в области высокопроизводительных вычислений. Можно с большой долей уверенности сказать, что этот подход будет актуален всегда: сколь бы совершенен не был один компьютер, кластер из узлов такого типа справится с любой задачей гораздо быстрее.

B5. Типы кластеров

Кластерные системы могут иметь различную организационную структуру. По физической реализации кластеры бывают следующими.

- Кластеры специальной разработки с быстродействием $10^{11}…10^{12}$ плавающих операций в секунду (терафлопсовый диапазон). К таким кластерам относится СКИФ (**С**упер**к**омпьютерная **и**нициатива **Ф**еникс), созданный по программе Союза Беларусь–Россия. Появление этого кластера и сам процесс разработки подготовили основу для развития параллельных вычислений, вследствие чего возникла проблема широкой подготовки программистов для решения параллельных задач.

- Кластеры, которые строятся на базе уже имеющихся локальных сетей из персональных компьютеров. Для создания таких кластеров

требуется только дополнительное программное обеспечение (ПО), поэтому их можно организовать в вузах и небольших организациях. Подобные кластеры имеют относительно небольшое быстродействие, но их удобно использовать для обучения основам параллельного программирования и подготовки параллельных программ, которые затем могут выполняться на больших кластерах.

По способу представления вычислительных узлов можно выделить следующие группы кластеров.

- Кластер, расположенный в виртуальной среде. Все узлы вычислительного кластера являются виртуальными машинами. Как правило, используются гипервизоры на основе базовой ОС (Microsoft Virtual PC, VMware Workstation, QEMU, Parallels, VirtualBox) или же гибридные гипервизоры (Microsoft Virtual Server, Sun Logical Domains, Xen, Citrix XenServer, Microsoft Hyper-V). Иногда могут применяться автономные гипервизоры, не имеющие базовой операционной системы. Такие гипервизоры используют собственные драйверы и планировщик, поэтому они не зависят от базовой ОС. Так как автономный гипервизор работает непосредственно на оборудовании, то он более производителен. В качестве оборудования, на котором функционируют виртуальные машины, может выступать один или несколько высокопроизводительных серверов или персональный компьютер.
- Кластер на основе физических серверов. При такой реализации в узлах кластера расположены только серверы или персональные компьютеры. В связи с этим производительность кластера ограничена только производительностью или количеством его узлов.

По однородности вычислительных узлов кластеры могут быть следующими.

Однородные. Все узлы кластера являются однотипными. Центральный процессор, оперативная память, дисковая подсистема, сетевые адаптеры – все эти компоненты узла являются однотипными. При такой организации узлы являются взаимозаменяемыми. Замена производится либо на аналогичный компонент вычислительного узла, либо производится замена узла целиком. Такая организация вычислительного кластера позволяет свести к минимуму задержки, связанные с не-

равномерной загрузкой суперкомпьютера. Как правило, не возникает ситуаций, при которых одни вычислительные узлы простаивают в связи с тем, что на других еще производится обработка данных.

Смешанного типа. При такой организации в качестве узла кластера может выступать любой вычислитель, который поддерживает программное обеспечение и имеет общий доступ к разделяемым ресурсам. В связи с этим может наблюдаться неоднородность загрузки вычислительного кластера. Более производительные узлы будут заканчивать работу раньше, чем менее производительные. При работе на таком кластере требуется уделять больше внимания при написании программ.

В6. Программное обеспечение вычислительных кластеров

От используемого программного обеспечения (ПО) в вычислительном кластере зависит производительность, надежность и специфика решаемых задач. Именно программное обеспечение определяет круг задач, которые могут решаться на кластере. Основу ПО составляет операционная система. Прочие составляющие программного обеспечения кластеров будут рассмотрены в разделах 2–4 настоящего учебного пособия.

В7. Операционные системы

Операционная система – комплекс управляющих и обрабатывающих программ, которые, с одной стороны, выступают как интерфейс между устройствами вычислительной системы и прикладными программами, а с другой – предназначены для управления устройствами, вычислительными процессами, а также для эффективного распределения вычислительных ресурсов между вычислительными процессами и для организации надежных вычислений. Это определение применимо к большинству современных операционных систем общего назначения.

В качестве операционных систем для кластеров, как правило, используются серверные версии UNIX-подобных операционных систем (SUSE Server, RedHat (CentOS), Ubuntu Server). До недавнего времени эти системы были фактически стандартом для вычислительных кластеров. После 2006 г. в связи с выходом Microsoft HPC Cluster Server стало возможным создавать вычислительные кластеры на основе операционных систем Microsoft.

При выборе операционной системы следует основываться прежде всего на рекомендациях разработчиков программного обеспечения. Однако, если есть выбор, то при прочих равных условиях следует отдать предпочтение Linux.

Для Linux доступно огромное количество серверного программного обеспечения, компиляторов, библиотек, средств отладки и пр. Большое количество программного обеспечения имеется в свободном доступе, для многих программ есть исходные коды и обширная документация. Достоинством Linux является «прозрачность» для пользователя и системного администратора, что позволяет быстрее и проще разрешать все возникающие проблемы.

Основой кластера является не операционная система, а коммуникационная среда (PVM, MPI), обеспечивающая возможность частям параллельной программы, выполняющимся на разных компьютерах, эффективно взаимодействовать между собой.

Рассмотренные ранее средства для построения кластера (PVM, MPI) имеют реализации как для операционных систем семейства UNIX (Linux, FreeBSD и т. п.), так и для систем фирмы Майкрософт. Кластер может быть создан и под ОС Windows, причем трудозатраты на установку коммуникационной среды будут такими же, как и в варианте с UNIX, т. е. небольшими. Основная трудность будет заключаться в том, чтобы научиться писать параллельные программы.

Однако следует заметить, что подавляющее большинство высокопроизводительных кластеров в мире работает все же в среде UNIX. Такой выбор обусловлен небольшими трудозатратами по настройке и установке кластера, более низкими требованиями ОС по сравнению с ОС Windows и большим количеством поддерживаемого программного обеспечения. Поскольку библиотеки для параллельных вычислений

MPICH/MPI являются кросс-платформенными, то выбор операционной системы (Windows vs Linux) не важен. Однако следует учесть тот факт, что Linux является заметно менее ресурсоемкой системой. Например, при использовании PelicanHPC GNU Linux система занимает в оперативной памяти не более 40 Мб! Вся остальная память доступна параллельной программе. Это является очень важным фактором в том случае, когда кластер используется с целью моделирования процессов на как можно более подробной сетке.

В настоящее время основной операционной системой, используемой при проведении учебных занятий в вузах, является операционная система Windows. При всех достоинствах системы ей присущи некоторые недостатки, существенно затрудняющие ее использование. К таким недостаткам можно отнести:

1) малую защищенность системы от неквалифицированных действий пользователей;

2) подверженность системы различного рода «взломам» при сетевом использовании и подверженность вирусам;

3) неустойчивость работы системы, проявляющаяся в зависаниях и потере информации;

4) большую стоимость лицензий на использование систем;

5) закрытость операционной системы, затрудняющей написание учебных программ в ее среде и обучение;

6) большие требования к возможностям компьютера (память, быстродействие);

7) частую смену версий ОС (примерно каждые два года).

На данный момент наиболее востребованной ОС для кластеров является Linux в различных вариациях, хотя и в линейках серверных ОС компании Microsoft появляются дистрибутивы, ориентированные на высокопроизводительные расчеты.

B8. Учебно-лабораторное техническое обеспечение

Построение вычислительного кластера подразумевает использование двух и более вычислительных систем, для чего рациональным является использование технологий виртуализации [2]. Поскольку в

процессе изучения принципов построения кластерных систем важны именно задачи развертывания, настройки, изучения системы команд и подходов к формированию вычислительных заданий, то аппаратные требования в минимальной конфигурации для трех узлов будут следующими:

1) процессор с поддержкой аппаратной виртуализации и двумя вычислительным ядрами;

2) 4 ГБ оперативной памяти;

3) 40 ГБ на файловой системе.

В качестве гипервизора могут быть использованы: VirtualBox, Xen, KVM для хост-системы на базе Linux и VirtualBox, Hyper-V для хост-систем на базе Windows.

1. КЛАСТЕР PelicanHPC

1.1. Описание архитектуры и характеристики программного продукта PelicanHPC

PelicanHPC [3] – это дистрибутив операционной системы GNU/Linux для быстрого развертывания кластеров. Дистрибутив построен на пакетной базе Debian и позволяет превратить обычный офисный компьютерный парк в вычислительный кластер, загрузив один управляющий компьютер с LiveCD/LiveUSB, а остальные машины использовать в качестве загружаемых по сети через протокол PXE узлов.

Поскольку LiveCD/LiveUSB не используют жесткие диски узлов, то применение машин в качестве узлов кластера никак не затронет установленную операционную систему. Когда PelicanHPC будет выключен, то все компьютеры будут находиться в своем первоначальном состоянии и могут быть загружены снова с той операционной системой, которая была на них установлена. Опционально PelicanHPC может быть настроен для использования загрузки с жесткого диска, чтобы сохранить конфигурацию после перезагрузки.

Дистрибутив поддерживает средства централизованного управления конфигурацией посредством доступа по ssh для загружаемых по сети узлов, что позволяет использовать дистрибутив для обеспечения функционирования постоянно работающих кластеров, а не только для проведения одноразовых экспериментов. В процессе работы количество узлов кластера может быть динамически изменено.

Основные возможности кластера на базе PelicanHPC:

- реализация MPI на базе пакета OpenMPI;

- поддержка параллельных вычислений на основе MPI с использованием Fortran (77, 90), C, C + +, GNU Octave и Python;
- интерфейс мониторинга работы кластера Ganglia;
- набор для тестирования производительности кластеров Linpack;
- интерфейс пользователя, реализованный с помощью текстовой консоли или GUI: Xfce или GNOME.

Требования и ограничения:

1) вычислительные узлы должны быть загружены по сети, в связи с этим требуется поддержка протокола PXE сетевой картой;

2) поскольку терминал PelicanHPC работает как DHCP сервер, в сети должны отсутствовать дополнительные аналогичные работающие сервисы;

3) кластер PelicanHPC предназначен для использования одним человеком, т. е. только один пользователь с именем "user";

4) текущие версии только для 64-битных процессоров.

2.2. Развертывание кластера PelicanHPC

С помощью специализированного дистрибутива PelicanHPC GNU Linux можно запустить кластер в течение небольшого интервала времени. Исходная операционная система, программное обеспечение и данные всех используемых в качестве кластера компьютеров не будут модифицированы. После выключения кластера компьютеры придут в то состояние, в котором они были до начала работы кластера.

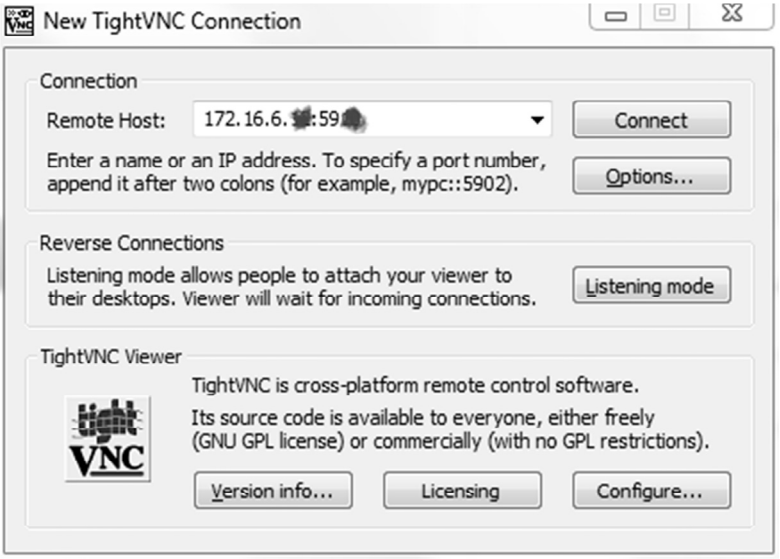
Для развертывания такого виртуального кластера потребуется один компакт-диск либо модуль флэш-памяти с версией дистрибутива PelicanHPC GNU Linux (желательно последней). С этого диска загружается операционная система кластера (без установки ее на винчестер) на компьютере, который будет играть роль консоли кластера, т. е. его управляющего узла, используемого одновременно и для связи с внешним миром, если таковая потребуется.

Остальные узлы кластера будут загружаться по сети. Для загрузки ОС вычислительных узлов кластера по сети необходимо, чтобы сетевые карты этих компьютеров умели выполнять загрузку по сети. Большинство современных карт, в том числе встроенных, это делать умеют (если же это не так, то всегда можно сделать загрузочный CD

из образа gpxe.iso и загрузить такие вычислительные узлы с этого диска).

Для создания кластера выделяется не менее трех виртуальных машин, запущенных на гипервизоре Xen. Первая из машин настроена на загрузку с образа компакт-диска, остальные настроены на загрузку по сети.

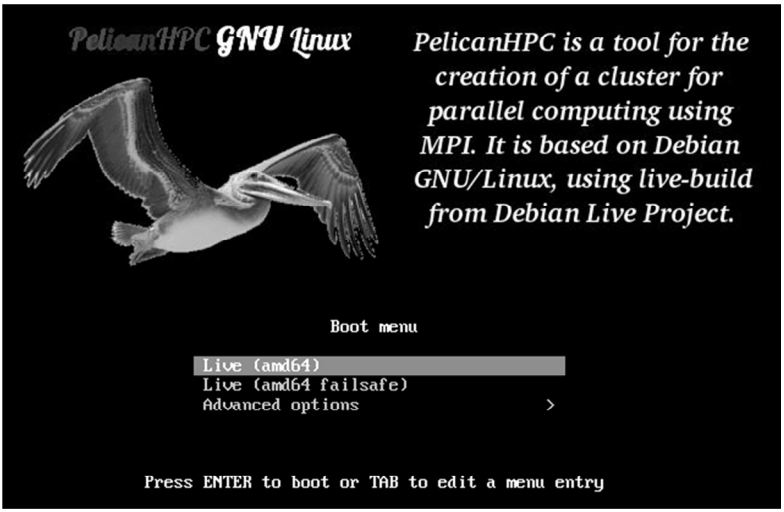
Для доступа к консоли и рабочему столу виртуальных машин используется протокол VNC, и все, что необходимо знать, – это адрес и порт, на который необходимо подключиться при помощи программы TightVNC Viewer.



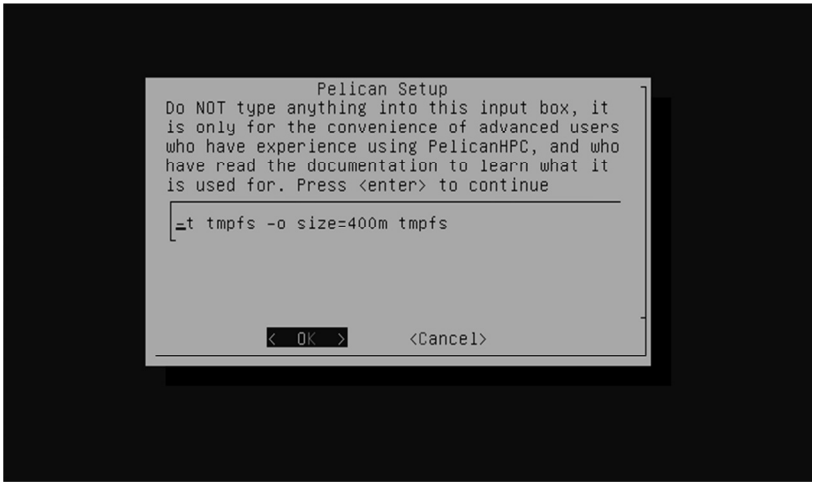
Если после подключения видно только черный экран, то вполне возможно, что все работает, но включился «хранитель экрана», и нажатие любой клавиши (например, Alt) вернет изображение. Виртуальную машину можно принудительно перезагрузить, нажав на панели меню TightVNC Viewer пиктограмму, выделенную на следующем рисунке.

Собственно загрузка кластера выполняется следующим образом.

1. Загружается консоль кластера с PelicanHPC GNU Linux Live CD.

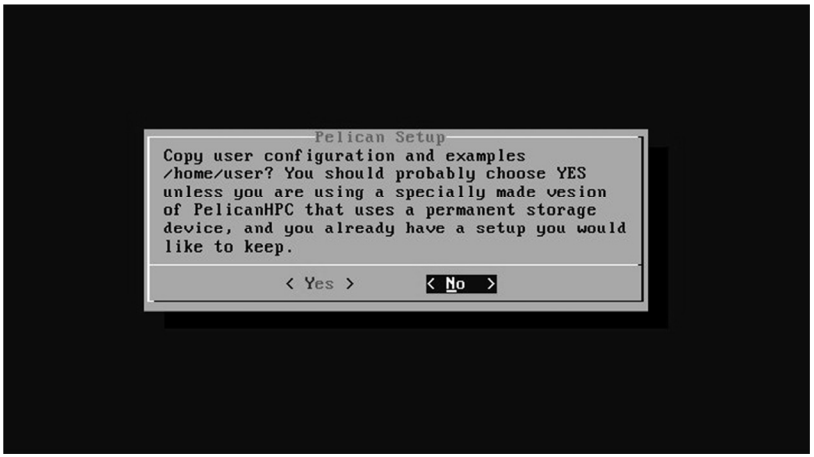


2. Через некоторое время на мониторе управляющего узла кластера появляется следующий запрос.



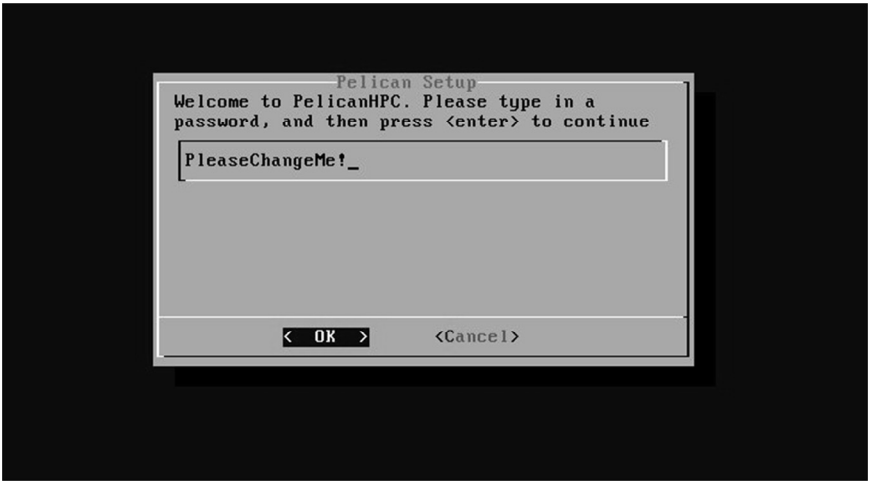
Здесь нужно указать устройство, на котором будет располагаться пользовательский каталог, т. е. тот каталог, где будут храниться программы, исходники и файлы данных. По умолчанию предложен раздел на виртуальном диске, расположенном в оперативной памяти. Это самый простой вариант, однако не самый удобный в том смысле, что после выключения компьютера все данные на этом диске будут уничтожены. Используемые виртуальные машины не имеют накопителей, и единственно верный вариант при этом – ничего не менять и нажать кнопку “Ок”.

3. Следующий вопрос, который будет задан при запуске кластера, выглядит так.



Система спрашивает, будет ли выполнена начальная конфигурация пользовательского каталога. В том случае, когда в качестве пользовательского каталога используется виртуальный диск, всегда нужно отвечать “Yes”. Если же в качестве месторасположения пользовательского каталога выбран постоянный носитель (раздел винчестера компьютера, флешка или внешний USB-винчестер), тогда ответ “Yes” следует выбрать только в самый первый раз. Во все последующие загрузки кластера необходимо выбрать ответ “No”.

4. На следующем шаге нужно указать пароль пользователя, с которым он будет подключаться к системе.

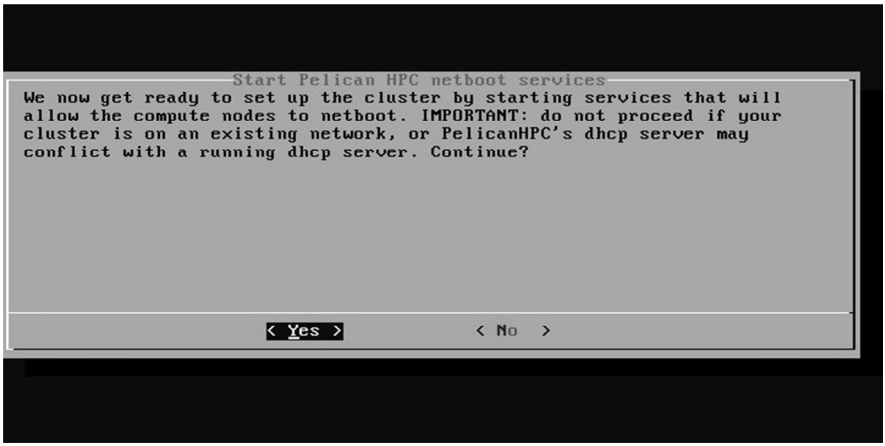


Нужно стереть фразу PleaseChangeMe! и набрать требуемый пароль (но не забыть его).

5. После задания пароля будет предложен стандартный экран входа в систему.

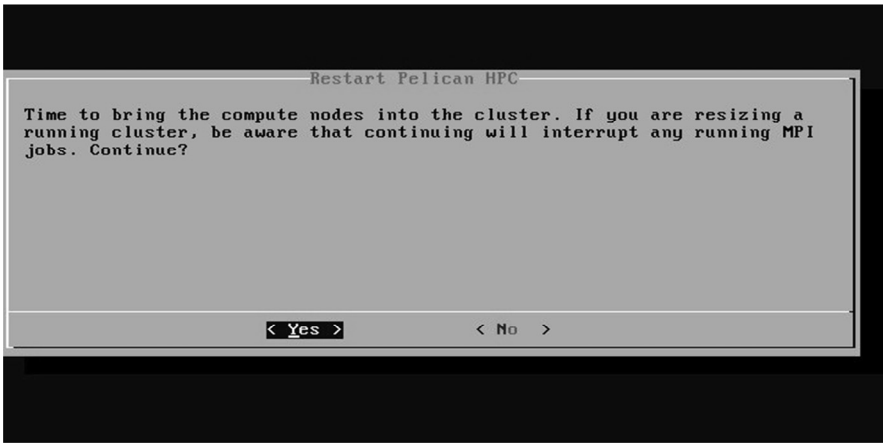
Для регистрации в системе нужно использовать логин “user” и пароль, который был задан на предыдущем шаге.

6. Теперь необходимо подключить к кластеру все собственно вычислительные узлы. Для этого должна быть запущена команда конфигурирования кластера **pelican_setup**. Первое, что спросит эта команда: будет ли конфигурироваться сетевая загрузка вычислительных узлов?

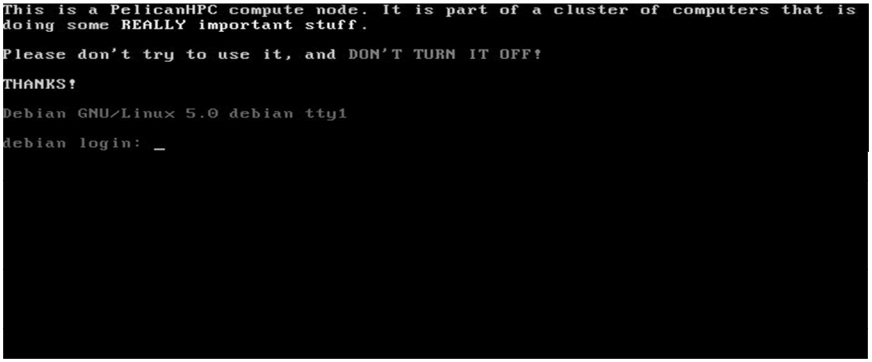


Нужно ответить “Yes”.

7. Сконфигурировав сервер сетевой загрузки, программа предложит выполнить загрузку всех стальных узлов кластера.

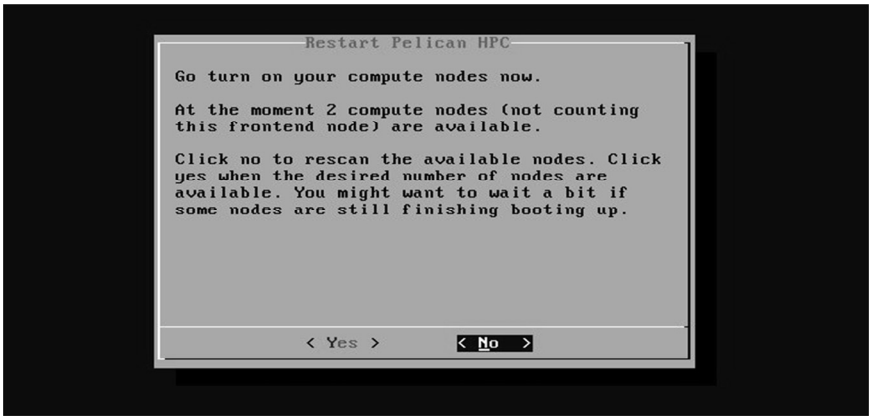


В этот момент нужно включить все остальные компьютеры кластера, у которых настройки BIOS установлены таким образом, чтобы они выполнили загрузку по сети. Затем следует дождаться, когда все вычислительные узлы закончат процедуру загрузки, о чем будет свидетельствовать следующая картинка на экранах этих компьютеров.



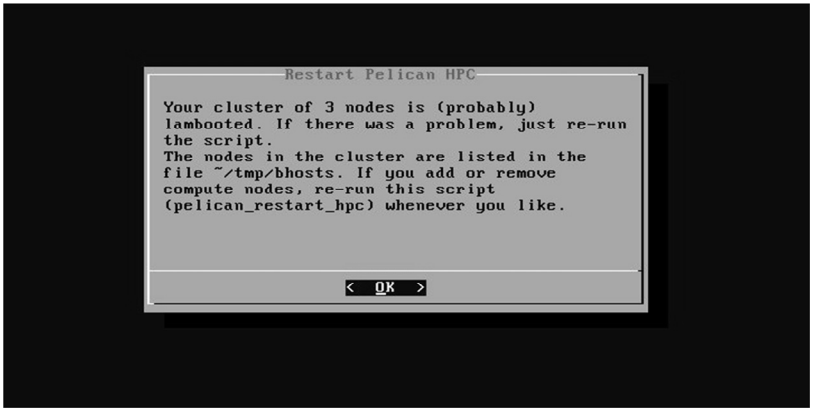
8. После того как все компьютеры будут загружены, нужно нажать кнопку “Yes” на экране управляющего узла.

9. Программа настройки попытается найти все загруженные компьютеры и включить их в конфигурацию кластера. После выполнения этого действия она выдаст на экран итоговый результат.



На этом экране программа сообщает, сколько было найдено вычислительных узлов (в данном случае два), кроме узла, который является консолью кластера. Если все нормально, нажимаем “Yes”.

10. И, наконец, программа конфигурации кластера сообщает, что все настройки выполнены и кластер готов к эксплуатации.



Остается только завершить процедуру развертывания кластера, нажав “OK”.

11. Если конфигурирование кластера завершилось с ошибкой, рекомендуется следующим шагом запустить скрипт реконфигурации: **pelican_restart_hpc**, тем самым повторив заново п. 7–9.

Теперь кластер работоспособен и можно проверить его работу на тестовой программе. В качестве стандартного теста разработчики PelicanHPC предлагают программу вычисления числа π flops.f. Ее исходный текст уже находится в пользовательском каталоге. На консоли кластера можно выполнить следующие действия.

- Компиляция программы для параллельной среды MPI с помощью команды

mpif77 flops.f -o flops

- Запуск программы на одном процессоре командой **./flops**
- Запуск программы на двух процессорах командой

mpirun –hostfile ~/tmp/bhosts –np 2 ./flops

где **'–hostfile ~/tmp/bhosts'** – это файл со списком узлов кластера и **'-np 2' –** указание, что нужно запустить две копии процесса.

Должны быть получены примерно следующие результаты:

```
user@pel1:~$ ./flops

  HPC Test ----------------------------------------
  Quantity of processors =   1
  Calculation time       =   3.22 seconds
  Cluster speed          =    559 MFLOPS
  ------------------------------------------------
  Cluster node N00 speed =    559 MFLOPS
  ------------------------------------------------

user@pel1:~$ mpirun -hostfile ~/tmp/bhosts -np 2 ./flops

  HPC Test ----------------------------------------
  Quantity of processors =   2
  Calculation time       =   1.59 seconds
  Cluster speed          =   1131 MFLOPS
  ------------------------------------------------
  Cluster node N00 speed =    565 MFLOPS
  Cluster node N01 speed =    565 MFLOPS
  ------------------------------------------------

user@pel1:~$ _
```

свидетельствующие о том, что скорость вычисления на двух процессорах примерно в два раза больше, чем на одном. Следовательно, кластер делает именно то, что от него требуется.

Вопросы для самопроверки

1. При создании кластера из двух и более узлов доступен только один узел. Перечислите все возможные причины.
2. Что произойдет, если при запуске задания указать число процессоров, превышающее их сумму в кластере?
3. Отсутствует файл flops.f. Какие ошибки были допущены?
4. Как можно получить доступ к графическому интерфейсу?
5. Опишите по шагам процедуру изменения числа узлов в уже настроенном кластере.

2. СИСТЕМА ПАКЕТНОЙ ОБРАБОТКИ Torque

2.1. Основные теоретические сведения о Torque

Несмотря на то что средства MPI сами по себе позволяют осуществлять запуск параллельных задач, обычно для этих целей используются различные менеджеры ресурсов. Одним из таких менеджеров является система Torque [4] – один из наиболее популярных и простых в использовании менеджеров. Система управления заданиями Torque предназначена для управления запуском задач на многопроцессорных вычислительных установках (в том числе кластерных). Она позволяет автоматически распределять вычислительные ресурсы между задачами, управлять порядком их запуска, временем работы, получать информацию о состоянии очередей. При невозможности запуска задач немедленно они ставятся в очередь и ожидают, пока не освободятся нужные ресурсы.

Torque главным образом используется на многопроцессорных вычислительных установках. Объединение ресурсов в вычислительных установках обычно уменьшает необходимость в постоянном управлении ресурсами для пользователей. Настроенная однажды правильно вычислительная установка абстрагируется от многих деталей, связанных с запуском и управлением заданиями. Пользователю обычно надо установить в параметрах лишь минимальные требования к задаче. Ему нет необходимости знать даже имена вычислительных узлов, на которых задача выполняется.

В том случае, если у кластера есть единственный пользователь/владелец, особой нужды в менеджере ресурсов нет. Эффективно управлять собственной задачей он может самостоятельно. Однако

если кластером пользуются несколько человек, то неизбежно возникают задачи административного характера: кто, когда и на сколько времени может занимать ресурсы кластера. Кроме того, если нет выделенного кластера, но вместо этого для решения параллельных задач используется компьютерный класс, который в дневное время предназначен для обеспечения учебного процесса, то проблема остановки «тяжеловесных» параллельных задач и освобождения ресурсов с началом рабочего дня может быть легко решена с помощью Torque.

Система Torque состоит из нескольких демонов, выполняющих различные функции по управлению потоком заданий. Вычислительная установка обязана иметь главный узел (консоль кластера), на котором запущен демон **pbs_server**. Это основной демон – менеджер, собирающий информацию о структуре кластера и запущенных заданиях. В зависимости от необходимости главный узел может быть предназначен только для управления или же исполнения роли и других компонентов системы. Например, он может быть также вычислительным узлом кластера.

Роль вычислительных узлов – выполнять поставленные задачи. На каждом из них работает демон **pbs_mom**, для того чтобы начинать, прекращать и управлять поставленными в очередь задачами. Это единственный демон, который должен быть запущен на вычислительном узле кластера.

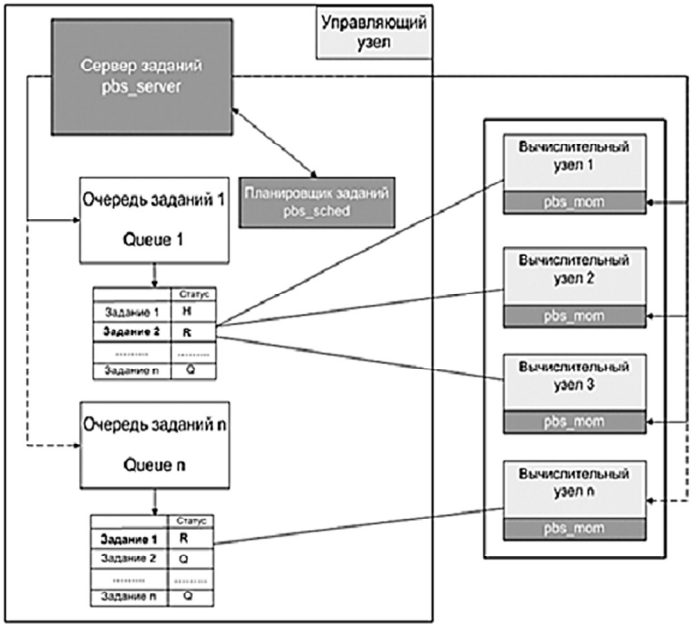
И, наконец, демон **pbs_sched**. Этот демон занимается собственно планированием запуска и остановки задач. Он должен быть запущен на главном компьютере кластера.

Torque является одной из версий стандарта системы PBS (Portable Batch System – система пакетной обработки заданий). Torque управляет загрузкой вычислительных комплексов, состоящих из определенного количества вычислительных узлов, работающих под управлением операционной системы семейства Unix. Система пакетной обработки (СПО) заданий необходима при одновременном выполнении заданий несколькими пользователями на одном вычислительном комплексе.

В результате применения СПО вычислительные ресурсы используются оптимально: сводится к минимуму как перегрузка какого-либо одного узла, так и его простой. Torque обычно применяется в областях, где высока интенсивность использования вычислительных мощностей.

Таким образом, Torque обеспечивает контроль над вычислительными ресурсами, что в конечном итоге снижает зависимость от системных администраторов и операторов и освобождает их для решения других задач. Кроме того, Torque дает возможность контролировать выполнение заданий, используя очереди и планировщик заданий.

Принцип работы Torque заключается в следующем (см. рисунок). Задания создаются и управляются сервером заданий. Клиенты СПО взаимодействуют с сервером заданий, который предоставляет соответствующие сервисы. Пользователь взаимодействует с СПО посредством утилит командной среды. Сервер заданий является демоном (daemon), который осуществляет постановку заданий в очередь, управление очередями и выполнение задания от имени клиента СПО. Сервисы, предоставляемые сервером заданий, доступны посредством утилит командной строки (batch utilities), которые запускают пользователи. В следующем разделе описаны компоненты Torque.



Взаимодействие компонентов Torque

Утилиты Torque можно запускать как из командной строки операционной системы, так и посредством графического интерфейса. Набор, синтаксис и семантика (т. е. выполняемые операции) пакетных утилит соответствуют стандарту POSIX 1003.2d. Графический интерфейс рассматриваться не будет.

2.2. Структура системы

2.2.1. Сервер заданий

Демон сервера заданий (Job Server daemon) – центральная точка Torque. Далее будет применяться термин «сервер» или имя процесса pbs_server. Все команды и другие процессы-демоны взаимодействуют с сервером посредством сети по протоколу IP. Главная задача сервера – обеспечить базовые сервисы для исполнения пакетных заданий, такие как получение/создание задания (batch job), изменение задания, обеспечение надежности функционирования системы заданий путем защиты от неполадок в системе и исполнение задания.

Обычно имеется единственный сервер, управляющий конкретным набором ресурсов. Однако в общем случае серверов может быть несколько.

2.2.2. Сервис вычислительного узла: процесс pbs_mom

Клиент, запускающий задания (Job Executor; или просто – клиент заданий), – служба (service) операционной системы, физически осуществляющая запуск задания. Эта служба, называемая pbs_mom, неформально обозначается MOM, поскольку является «предком» (mother) всех исполняемых заданий. MOM запускает задание, когда получает его копию с сервера заданий.

MOM создает новый сеанс от имени одного из пользователей (которым не является root), зарегистрированных в системе. Запуск задания производится в сеансе оболочки данного пользователя. MOM также ответствен за предоставление пользователю результатов работы задания, по умолчанию выводимых на консоль сервера. Эти результаты будут сохранены в домашнем каталоге пользователя – владельца очереди заданий. Сервис MOM запускается на всех вычислительных узлах, которые будут выполнять задания.

2.2.3. Представление вычислительных узлов

Вычислительные узлы кластера представляются в Torque определенным образом. Прежде чем обсуждать работу с вычислительными узлами (compute nodes) кластера, необходимо ввести некоторые определения.

Вычислительный узел (compute node) – это отдельная компьютерная система (или просто компьютер) с одним образом (image) операционной системы, унифицированным виртуальным адресным пространством, одним или более процессором и одним или более IP-адресом.

Часто термин «исполняющий хост» (execution host) также используется для обозначения узла.

Компьютер, содержащий несколько процессоров и работающий под управлением одной операционной системы, является одним узлом. Узлы делятся на два типа: узлы общего типа (cluster node) и разделяемые по времени узлы (timeshared).

Узел общего типа (cluster node) – узел, назначением которого является параллельное выполнение заданий. Если такой узел имеет более одного виртуального процессора, они могут быть назначены разным заданиям (англ. jobshared – распределены между заданиями) или использованы для выполнения единственного задания (англ. exclusive – эксклюзивный доступ).

Такая возможность непрерывно распределять ресурсы каждого узла важна для некоторых приложений, работающих одновременно на нескольких узлах (multi-node applications). Обратите внимание, что Torque обязывает придерживаться схемы «один к одному» при выделении виртуальных процессоров (см. далее) заданию. Таким образом, один ВП работает только с одним заданием.

Разделяемый во времени узел (timeshared node). В противоположность узлам общего типа такие узлы всегда могут обслуживать несколько заданий одновременно. Часто термин «хост» (host) используется вместо термина «узел» (node) совместно с термином «timeshared».

Разделяемый во времени узел никогда не будет эксклюзивно выделен для выполнения единственного задания.

Виртуальный процессор. Для узла может декларироваться наличие одного или нескольких виртуальных процессоров. Слово «виртуаль-

ный» используется, поскольку обозначенное число виртуальных процессоров может не соответствовать числу реальных процессоров в узле. Число ВП в узле по умолчанию равно числу реально функционирующих ядер физических процессоров.

Атрибуты узла

Вычислительные узлы кластера настраиваются в torque установкой их атрибутов. Атрибуты также используются в файле конфигурации узлов. Список основных атрибутов приведен в табл. 2.1.

Таблица 2.1

comment	Комментарий для узла
max_running	Максимальное количество заданий, которое может быть одновременно запущено на узле
max_user_run	Максимальное число заданий, принадлежащих одному пользователю, которое допускается одновременно выполнять на узле
no_multinode_jobs	Если этот атрибут имеет значение true, то задания, которые запрашивают для своего запуска несколько узлов, не будут выполняться на данном узле
np	Количество виртуальных процессоров
ntype	Задает тип узла. Типы узлов описаны выше, в предисловии к данному разделу. Значения могут быть следующими: time-shared, cluster
properties	Свойства узла, определяемые пользователем. Значением может быть любая строка, начинающаяся с буквенного символа, или такие строки, разделенные запятой
resources_available	Ресурсы, доступные на узле. Конкретный ресурс задается после символа точки «.»: resources_available.ncpus. Соответственно все ресурсы, доступные torque, можно указывать в качестве значения этого атрибута
state	При помощи этого атрибута можно задать или просмотреть статус (state) узла. Возможные значения для состояния: free, offline, down, job_busy, job_exclusive, busy, state_unknown. Первые два состояния может установить пользователь, остальные – только системные процессы

Атрибуты могут быть установлены/изменены командой qmgr.

Файл конфигурации узлов

Вычислительные узлы, где запускаются задания, определяются при взаимодействии сервера и других компонентов torque (в частности, планировщика заданий). Взаимодействие возможно благодаря файлу конфигурации nodes. Файл располагается по следующему пути:

PBS_HOME/server_priv/nodes

В файле содержатся список узлов и их атрибуты. Без списка узлов сервер не сможет взаимодействовать с MOM посредством специального потока (communication stream). У MOM также не будет возможности отчитываться о запущенных заданиях и уведомлять сервер о завершении задания. Здесь PBS_HOME – переменная окружения, содержащая путь к рабочей директории Torque. Простой файл конфигурации узлов создается в процессе установки Torque.

Этот файл содержит только название хоста, с которого была запущена инсталляция. Этот узел будет считаться разделяемым по времени (timeshared). Файл конфигурации узлов можно изменять двумя путями. Если сервер не запущен, это можно делать напрямую в текстовом редакторе. Если же сервер работает, следует использовать команду qmgr для изменения списка узлов.

Файл конфигурации представляет собой обычный текстовый файл, каждая строка которого записана в форме

node_name[:ts] [attributes]

Здесь node_name – это сетевое имя узла. Опциональный параметр «:ts» добавляется к имени и указывает, таким образом, что узел является разделяемым по времени (timeshared). Также узлы могут иметь ассоциированные с ними атрибуты. Атрибуты перечисляются в виде

attribute_name=value

Например, выражение np=<число> может быть использовано для определения числа виртуальных процессоров на узле. Если это выражение не указано для узла общего типа (cluster node), то число виртуальных процессоров будет равно 1. Пример такого файла:

node-1 np=4
node-2 np=2
node-3 np=2

node-4 np=8
node-5 np=4
node-6 np=2
node-7 np=4
node-8 np=4
node-9 np=12
node-10 np=4
node-11 np=2

Для вывода сведений об узлах используется команда pbsnodes.

2.2.4. Планировщик заданий

Демон планировщика заданий (Job scheduler daemon), процесс которого называется pbs_sched, занимается распределением ресурсов между заданиями. Он определяет, когда данное задание будет запущено и какие ресурсы ему будут выделены. Планировщик взаимодействует с MOM на узлах и запрашивает у них состояние системных ресурсов, а также с сервером заданий для получения списка заданий, доступных для выполнения. Планировщик использует файл конфигурации узлов для определения узла или узлов, где будет запущено задание.

2.3. Задания в Torque

2.3.1. Понятие задания

Задание Torque представляет собой абстрактную сущность, состоящую из набора команд и параметров. Задание представляется пользователю в виде скрипта для оболочки, содержащего требования к ресурсам, атрибуты задания и набор команд, которые необходимо выполнить.

Однажды созданный скрипт задания можно использовать столько раз, сколько необходимо. Естественно, возможна его модификация. Задание сначала необходимо поставить в очередь Torque (submit), затем из этой очереди оно будет передано на один из узлов для выполнения. Очередей заданий (batch queue) может быть несколько. Сразу после

установки torque очередей заданий еще не существует. Необходимо сначала создать очередь заданий, а затем уже ставить их в эту очередь.

Настроенный вариант системы обычно включает в себя одну очередь заданий.

Пример простого скрипта задания (номера строк не хранятся в файле):

```
1 #!/bin/bash
2 #PBS −l walltime =1:00:00
3 #PBS −l mem=400mb
4 #PBS −l ncpus=4
5 #PBS −j oe
6
7 ./subrun
```

Первая строка является стандартной для любого скрипта с описанием задания. Она определяет, какая оболочка используется для исполнения сценария. Оболочка bash используется по умолчанию для запуска сценария, но можно использовать и другую. Строки со второй по пятую являются директивами Torque. Система будет читать скрипт до тех пор, пока не найдет первую строку, которая не является валидной директивой torque, и останавливается. Это означает, что оставшаяся часть сценария содержит список команд или задач, которые пользователь желает запустить. При выполнении данного примера torque обнаружит такие команды в строках 6 и 7.

Далее будет приведено описание команды qsub, выступающей в роли командного интерпретатора. Она используется в том числе для постановки задания в очередь Torque. Любая опция, которая определяется в команде qsub, может также выступать в роли директивы внутри скрипта torque.

Строки 2–4 определяют опцию ресурса «–l», далее следует запрос определенного ресурса.

Конкретно строки 2–4 сообщают, что запрашиваются ресурсы, объем которых предполагает не более одного часа на выполнение, а также 400 Мб памяти и четыре процессора.

Строка 5 не является директивой запроса ресурса. Опция «–j oe» требует, чтобы Torque объединила (join) потоки вывода stdout и stderr в единый поток stdout.

И, наконец, строка 7 является командой для выполнения, которую пользователь хочет запустить. Этот пример запускает программу-симулятор подводной лодки, subrun. Хотя в примере имеется только одна команда, в него можно добавить необходимое количество программ и шагов.

2.3.2. Атрибуты задания

Каждое задание обладает определенным набором атрибутов, значения которых задаются при создании задания и могут быть изменены в процессе его выполнения. Примеры атрибутов задания – название задания, время выполнения, путь к выходному файлу и др.

Атрибуты задания задаются при постановке задания в очередь. Их также можно изменить уже после этого по различным причинам (например, была сделана ошибка при определении ресурсов или истекло время выполнения задания и его необходимо продлить). Однако независимо от причины изменения атрибутов для этого имеется команда qalter, которая будет описана ниже.

Большинство атрибутов может изменить владелец задания, которым может быть пользователь, поставивший задание в очередь командой qsub, либо произвольная учетная запись, указанная при постановке в очередь. Тем не менее если задание уже выполняется, то лимиты ресурсов не могут быть изменены. Такими лимитированными ресурсами являются процессорное время, обычное время, число задействованных процессоров, объем памяти.

2.3.3. Очереди заданий

Очередь Torque – это сущность, содержащая задания. Torque поддерживает два типа очередей.

1. Исполняемая очередь (execution queue), содержащая задания, готовые для выполнения. Задания могут запускаться только из очередей этого типа.

2. Очередь перемещения (routing queue), в которой находятся задания, предназначенные для перестановки в другие очереди, в том числе те, которые находятся на других серверах заданий.

Очередей обоих типов может быть несколько. Очереди также имеют свои атрибуты.

2.3.4. Возможные состояния задания

Задания в очередях могут находиться в различных состояниях (табл. 2.2).

Таблица 2.2

Сокращение	Описание
C complete	Задание успешно завершило свою работу
E exit	Прерывание работы задания
H hold	Задание заблокировано
Q queued	Задание поставлено в очередь и готово для выполнения
R running	Задание выполняется
T transiting	Задание перемещается в другое место (очередь)
W waiting	Задание ожидает, пока подойдет очередь для его выполнения, например, задание может ожидать определенного времени для своего выполнения или завершения выполнения другого задания, от которого зависит. Задание в этом состоянии не может быть выполнено
S suspended	Пауза в работе задания

2.3.5. Операции с заданиями

Операции с заданиями приведены в табл. 2.3

Таблица 2.3

Краткое описание операции	Название команды
Постановка задания в очередь. Осуществляет добавление задания с заданными параметрами в одну из существующих очередей	Команда qsub
Изменение атрибутов задания	Команда qalter
Блокировка (перевод в состояние Hold) и восстановление задания	Команды qhold и qrls
Получение информации о заданиях	Команда qstat
Перемещение заданий из одной очереди в другую	Команда qmove
Перезапуск задания	Команда qrerun
Удаление заданий	Команда qdel

2.3.6. Понятие ресурса. Типы ресурсов, управляемых Torque

Задание может запросить для запуска множество разных ресурсов, таких как процессоры, память, время (обычное и процессорное), а также может понадобиться дисковое пространство. Список ресурсов определяется с использованием опции *–l* список_ресурсов команды qsub или в скрипте задания. Таким образом, выделяются ресурсы, необходимые для выполнения задания, или определяется их лимит, который может быть выделен. Если лимит не устанавливается для какого-либо ресурса, то он считается равным бесконечности.

Аргумент список_ресурсов записывается в виде

resource_name[=[value]][,resource_name[=[value]],...]

Здесь resource_name – название ресурса; value – значение. Значения могут представляться в нескольких единицах измерения, зависящих от природы самого ресурса (например, время записывается в соответствующем формате [[часы:минуты]:секунды[.миллисекунды]; размер памяти указывается в байтах – *b*, килобайтах – *kb*, мегабайтах – *mb*).

Ресурсы могут быть следующих видов: количество процессоров, объем памяти, требуемое ПО, объем виртуальной памяти, количество времени и др.

2.4. Взаимодействие Torque с пользовательской средой

Чтобы системная среда надлежащим образом взаимодействовала с Torque, необходимо проверить несколько моментов. В большинстве случаев среда настраивается системным администратором.

Чтобы Torque работала правильно, необходимо выполнение следующих условий:

- все скрипты запуска оболочки должны быть корректными;
- пользователь должен иметь учетную запись, отличную от root на всех вычислительных узлах.

2.4.1. Настройка пользовательской среды на примере оболочки csh

В выполнении пользовательского задания могут возникнуть сложности, если скрипты запуска пользовательской оболочки (например, для оболочки csh – это файлы .cshrc, .login или .profile; для оболочки bash – .bashrc) содержат команды, которые пытаются использовать стандартные потоки. Подобная последовательность команд в таких файлах должна быть пропущена путем проверки переменной окружения PBS_ENVIRONMENT. Вот пример использования подобной методики в файле .login:

```
...
set env MANPATH /usr/man:/usr/local/man:$MANPATH
if ( ! $?PBS\_ENVIRONMENT ) then
        использование стандартных потоков (например, вывод на консоль)
endif
```

Нужно внимательно относиться к тем командам в сеансовых файлах пользователя, которые выводят на консоль какой-либо текст при работе в Torque. Как и в предыдущем примере, команды, которые выводят текст в поток stdout, не должны быть выполнены при запуске через Torque. Это достигается так же, как и в приведенном примере с файлом .login, а именно:

```
...
set env MANPATH /usr/man:/usr/local/man:$MANPATH
if ( ! $?PBS\_ENVIRONMENT ) then
        команды , выполняющие стандартный вывод
endif
```

При запуске задания Torque «выходное состояние» («exit status») последней команды, выполненной в задании, являющееся отчетом для оболочки выполнения, будет таковым и для Torque. Это важно для зависимых заданий и для построения цепочек заданий. Однако

последняя исполненная команда может не быть последней командой в задании. Такое может иметь место, если задание выполняется в оболочке csh на хосте и там имеется файл .logout. В данной ситуации последняя команда, выполненная из файла .logout, не является командой задания.

Чтобы предотвратить это, необходимо сохранить выходное состояние в файле .logout путем запоминания его в начале файла, а затем выполнить выход с этим статусом в конце, как показано ниже:

set EXITVAL = $status
содержимое файла .logout
exit $EXITVAL

2.4.2. Переменные окружения

Для любого задания в системе Torque существует некоторое количество переменных окружения. Одни переменные берутся из пользовательской среды и передаются заданию, другие создаются самой Torque, третьи могут явно создаваться пользователем для эксклюзивного использования заданием torque.

Примечание: переменные окружения Torque существуют в сеансе, создаваемом командой qsub. В обычной оболочке, из которой происходит запуск qsub, эти переменные не видны.

Все переменные, существующие в задании, имеют имена, начинающиеся с «PBS_». Некоторые из них также предваряются заглавной O: «PBS_O_», что говорит о происхождении переменной из среды выполнения задания (например, пользовательской).

Вот короткий пример, демонстрирующий использование наиболее полезных переменных и их типичных значений:

PBS_O_HOME=/home/test
PBS_O_LOGNAME=test
PBS_O_PATH=/usr/new/bin:/usr/local/bin:/bin
PBS_O_WORKDIR=/share/hpl/bin/

Полный список переменных окружения Torque включает:

PBS_JOBNAME=env
PBS_ENVIRONMENT=PBS_BATCH
PBS_O_WORKDIR=/home/ test
PBS_TASKNUM=1
PBS_O_HOME=/home/ test
PBS_MOMPORT=15003
PBS_O_QUEUE=batch
PBS_O_LOGNAME=test
PBS_O_LANG=en_US .UTF−8
PBS_JOBCOOKIE=903088939E7FAA7F4414578D7A806955
PBS_NODENUM=0
PBS_O_SHELL=/bin/bash
PBS_JOBID=93.user.localdomain
PBS_O_HOST=user.localdomain
PBS_VNODENUM=0
PBS_QUEUE=batch
PBS_O_MAIL=/var/spool/mail/test
PBS_O_PATH=/home/test/bin:/usr/local/bin:

2.5. Команды настройки Torque

2.5.1. Настройка узлов с помощью команды qmgr

Команда qmgr предоставляет пользователю интерфейс взаимодействия с сервером заданий torque. Эта команда позволяет настраивать узлы и их атрибуты. Qmgr можно рассматривать так же, как интерактивный интерфейс к менеджеру Torque.

Команда считывает директивы из стандартного потока ввода, синтаксис директив проверяется и соответствующий запрос отсылается к одному или нескольким серверам заданий. По умолчанию команду может выполнять только пользователь root.

Синтаксис команды qmgr таков:

qmgr [–a] [–c command] [–e] [–n] [–z] [server...]

Опции команды qmgr представлены в табл. 2.4.

Таблица 2.4

–a	Прервать работу qmgr в случае любых синтаксических ошибок или запросов, отклоненных сервером
–c < «команда» >	Выполнение единственной команды и завершение
–e	Перенаправить эхо-вывода в стандартный поток
–n	Только проверка синтаксиса, без выполнения команд
–z	Не выводить сообщения об ошибках в стандартный поток ошибок

Если команда qmgr запускается без опции *–c* и стандартный поток вывода ассоциирован с терминалом, qmgr выведет приглашение и директивы будут считываться с клавиатуры (стандартного потока ввода).

Создание и удаление узлов

Указав опцию *–c* при выполнении qmgr, можно задать команду создания нового или удаления существующего узла. Указанные операции необходимы при помощи qmgr всегда, если процесс pbs_server запущен.

Для добавления нового узла используйте подкоманду "create": qmgr "create node node_name [<атрибут>=<значение>]"

Например:

qmgr -c "create node pel73 np=12"

Для изменения параметров узла после его создания используйте подкоманду set:

set node node_name [attribute[+|-]=value]

Символы «+» и «-» следует использовать, если атрибут допускает несколько значений.

Для удаления узлов используется подкоманда delete:

qmgr -c "delete node mars"

Примеры работы с командой qmgr

Вывод информации об объектах типа «server». В этом примере qmgr используется с опцией *–c*.

```
[user@pel1 ~]$ sudo qmgr -c "list server"
Server user.localdomain
server_state = Active
scheduling = True
total_jobs = 0
state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
default_queue = batch
log_events = 511
mail_from = adm
scheduler_iteration = 600
node_check_rate = 150
tcp_timeout = 6
pbs_version = 2.1.8
```

В следующем примере изменяется тип узла в интерактивном режиме команды qmgr :

```
[user@pel1 ~]$ #qmgr
Max open servers: 4
Qmgr: set node node-32 ntype=time-shared
Qmgr:
```

Для вывода информации об объекте типа «queue» (очередь), который называется «batch», можно ввести с консоли такую команду:

```
[user@pel1 ~]$  qmgr -c "list queue batch"
Queue batch
queue_type = Execution
total_jobs = 0
state_count = Transit:0 Queued:0 Held:0 Waiting:0 Running:0 Exiting:0
mtime = Tue Sep 4 15:36:09 2007
enabled = True
started = True
```

2.5.2. Команда pbsnodes

Команда pbsnodes может быть использована для получения сведений об узлах и изменения их состояния. Синтаксис команды pbsnodes следующий:

pbsnodes [-a|-l|-s][-c узлы][-d узлы][-o узлы][-r узлы][узел1 узел2 . . .]
Пример запуска команды без опций:
[user@pel1 ~]$ #pbsnodes
node-1
state = down
np = 4
ntype = cluster
node-2
state = down
np=4
ntype = cluster

Если указаны только наименования узлов без дополнительных опций, то выводится состояние этих узлов.

Опции команды pbsnodes представлены в табл. 2.5.

Таблица 2.5

–a	Выводит список всех узлов и значения каждого атрибута узла
–c узлы	Изменяет состояние down или offline на free, т. е. узел становится доступным для выполнения заданий
–d узлы	Устанавливает состояние down для указанных узлов. Эти узлы будут в дальнейшем не доступны для выполнения заданий. Если команда приводится без списка узлов, то все узлы переводятся в состояние down
–l	Выводит список всех узлов
–o узлы	Переводит указанные узлы в состояние offline, даже если они на данный момент используются. Это состояние не может быть изменено никакими автоматическими, не зависимыми от пользователя средствами
-r узлы	Отменяет перевод в состояние offline для указанных узлов
-s	Определяет сервер заданий, на который будет послан запрос

Пример результата выполнения команды с опцией *–o*:

[user@pel1 ~]$ #pbsnodes -o node-1
[user@pel1 ~]$ #pbsnodes
node-1
state = down, offline
np = 4
ntype = cluster
node-2
state = down
np=4
ntype = cluster

2.6. Установка и настройка Torque на вычислительном кластере

Установка системы Torque может быть выполнена путем компиляции из исходников, которые доступны для загрузки с официального сайта проекта. Во многих дистрибутивах операционной системы Linux (Debian, Ubuntu,…) менеджер Torque имеется в репозиториях пакетов и может быть установлен в систему обычным образом – с помощью команды apt-get или менеджера пакетов Synaptic.

В этой работе должна быть выполнена установка системы Torque из исходных кодов. Скачать свежую версию можно с сайта разработчиков: http://www.adaptivecomputing.com/support/download-center/torque-download/

Архив программы необходимо загрузить на сервер будущего центрального узла при помощи утилиты pscp, например, в каталог '/usr/src'. После чего можно перейти в каталог с архивом и распаковать его, так как он скорее всего упакован при помощи двух архиваторов – gzip (сжатие, обычно обозначается расширением .gz) и tar (сохранение файлов и каталогов в одном файле без сжатия):

gunzip xxxx.tar.gz
tar -xf xxxx.tar

Для сборки понадобятся некоторые дополнительные пакеты, которые необходимо установить при помощи менеджера пакетов yum:

yum install gcc
yum install gcc-*c++*
yum install libtool
yum install openssl-devel
yum install libxml2-devel
yum install boost-devel

После чего внутри этого каталога необходимо выполнить обычную для данного действия последовательность команд:

./configure --prefix=/usr
make
make install

После установки Torque можно установить систему MPI, чтобы она знала о существовании менеджера ресурсов и могла взаимодействовать с ним:

yum install mpich
yum install mpich-autoload

Установив все необходимые компоненты, требуется выполнить перезагрузку компьютера при помощи команды *reboot* или как минимум запустить утилиту ldconfig без параметров, чтобы обновились пути к установленным общим библиотекам *.so

Настройка начальной очереди заданий будет следующим шагом. Для этого необходимо создать пользователя, от имени которого будут выполняться задания: *adduser user*

Затем следует выполнить (в каталоге с исходниками) команду ./torque.setup <Имя очереди>. Обычно <Имя очереди> совпадает с именем (логином) пользователя, который будет администратором Torque, например user:

./torque.setup user

Установка менеджера ресурсов на вычислительных узлах кластера не требует полной установки системы Torque. Вместо этого на главном

компьютере необходимо создать саморазворачивающийся архив системы с помощью команды

make package

запущеной в каталоге с исходными кодами Torque. В результате будет создано несколько исполняемых скриптов. Из них нужен только скрипт torque-package-mom-linux-x86_64.sh. Его нужно скопировать на все вычислительные узлы кластера при помощи утилиты scp из пакета openssh-clients

yum install openssh-clients (на всех узлах)
scp torque-package-mom-linux-x86_64.sh root@ip-адрес-узла:/opt

и запустить его на исполнение на этих узлах:

cd /opt && ./torque-package-mom-linux-x86_64.sh --install

Конфигурация Torque на вычислительных узлах кластера в минимальном варианте заключается в указании доверенного сервера, который будет иметь право управлять заданиями на узле. Выполнить это можно, записав в файле /var/spool/torque/server_name имя хоста вашего сервера (консоли кластера). Обычно этого не требуется, поскольку делается автоматически. Единственно, что требуется, так это проверить, правильно ли выполняется преобразование имени в IP-адрес. Нужно посмотреть, что записано в этом файле (например, там будет записано «supergate») и пропинговать этот хост. Если что-то не так, то имя этого хоста надо добавить в файл /etc/hosts.

Настройка Torque-сервера заключается в задании списка вычислительных узлов кластера в файле /var/spool/torque/server_priv/nodes (который надо создать). В этом файле должны быть перечислены имена всех вычислительных узлов, один узел – одна строчка. После этого необходимо остановить сервер, чтобы при следующем его запуске список узлов вступил в силу. Для этого нужно выполнить от имени суперпользователя команду qterm без параметров.

Проверка системы. Для проверки работоспособности системы на всех вычислительных узлах следует запустить демон pbs_mom. Далее на главном компьютере нужно запустить сервер Torque – демон

pbs_server. Проверка узлов осуществляется командой pbsnodes -a, которая выдаст на консоль нечто вроде этого:

mpiuser@supergate:~$ pbsnodes -a
supergate
state = free
np = 1
ntype = cluster
status = opsys=linux,uname=Linux supergate 2.6.28-14-generic #47-Ubuntu SMP Sat Jul 25 00:28:35
UTC 2009 i686,sessions=2524 13213 9091 9136 9353 9473 9521 8948 9868 9899 10149 10692 10101 13099
13242 13259 13268 13277 13288 13341 13359 13383 28122 32075 32076 32077 32078 32079 32080 32081
32082 32083 32084 32085 32086 32087 32091 32093,nsessions=38,nusers=8,idletime=380,totmem=3123024kb,

availmem=2583100kb,physmem=1018552kb,ncpus=2,loadave=0.00,netload=2253226200,state=free,jobs=,varattr=,
rectime=1250488593

modo
state = free
np = 1
ntype = cluster
status = opsys=linux,uname=Linux modo 2.6.28-14-generic #47-Ubuntu SMP Sat Jul 25 00:28:35
UTC 2009 i686,sessions=14835 2176 18833 2603 2455 3053 14938 14949 15015 15024 15034 15103
15126 15170 15181 15603,nsessions=16,nusers=4,idletime=1437,totmem=3026692kb,availmem=2362384kb,

physmem=1026640kb,ncpus=2,loadave=0.23,netload=1604969560,state=free,jobs=,varattr=,rectime=1250488595

mpiuser@supergate:~$

Состояние узлов state = free говорит о том, что узлы работоспособны и готовы к принятию заданий.

Проверка очереди заданий может быть осуществлена путем добавления в очередь простой задачи:

echo "sleep 30" | qsub

Задача эта ничего не делает, кроме того как ждет 30 с, по завершении которых благополучно умирает. Проверить состояние очереди можно командой qstat, которая покажет что-то вроде этого:

[user@pel1 ~] qstat
Job id Name User Time Use S Queue
------------------------ ---------------- --------------- -------- - -----
0.mail STDIN mpiuser 0 Q batch
dima@supergate:~$

Статус задания (предпоследняя колонка) имеет значение “Q”, которое говорит о том, что задание находится в очереди и ждет, когда ему будет разрешено начать работу. Ждать оно будет долго, до тех пор, пока планировщик не решит, что пришло время для выполнения задачи. Однако планировщик в системе отсутствует.

Принудительно запустить выполнение задания можно при помощи команды *qrun*, выполненной на главном узле кластера. Для автоматического управления заданиями необходимо запустить планировщик командой *pbs_sched*. После чего можно поместить новое задание в очередь, и через какое-то время задание начнет работать:

[user@pel1 ~]$ qstat
Job id Name User Time Use S Queue
------------------------ ---------------- --------------- -------- - -----
0.mail STDIN mpiuser 0 R batch
[user@pel1 ~]$

Статус задания измеился с **“Q”** (Queued) на **“R”** (Runned), а еще немного погодя статус изменится на **“C”** (Completed):

[user@pel1 ~]$ qstat
Job id Name User Time Use S Queue
------------------------ ---------------- --------------- -------- - -----
0.mail STDIN mpiuser 00:00:00 C batch
[user@pel1 ~]$

Если все прошло именно так, как описано, то система Torque работает и ее можно использовать для запуска параллельных задач.

Каждая параллельная задача, которую нужно поставить в очередь на исполнение, должна быть оформлена в виде пакета. Пакет представляет собой набор параметров запуска и инструкции, что конкретно требуется запустить (см. раздел 2).

Предположим, что нужно запускать тестовую программу **./**flops при следующих условиях.

1. Должны использоваться три вычислительных узла.

2. На каждом из них нужно занять по одному доступному процессору.

3. Задача должна выполняться не более 10 ч (по истечении которых она должна быть снята со счета).

4. На вычислительных узлах должно быть доступно не менее 100 Мб оперативной памяти.

5. О событиях запуска задачи и ее завершении (с ошибкой или без) нужно уведомлять сообщениями, направленными на электронную почту.

Для описания такого задания можно создать следующий скрипт (пусть его имя будет flops.pbs):

```
#!/bin/sh
#
#PBS -l nodes=pel1:ppn=1+2:ppn=1
#PBS -N Flops_TEST
#PBS -m abe
#PBS -M user@localhost
#PBS -l pmem=100mb
#PBS -l pcput=10:00:00
cd /home/mpiuser/mpi
mpirun ./flops
```

Количество задействованных процессоров описывается параметром *nodes=supergate:ppn=1+2:ppn=1*. Важно указать, что первым узлом для параллельной задачи будет хост *pel1*. Именно так назван центральный узел кластера. Кроме него задаче будет выделено еще два узла. Для каждого из этих трех узлов запрашивается по одному процессору (*ppn=1*).

Параметр *-N Flops_TEST* – это просто название задания. Так оно будет отображено в очереди.

Параметры *-m abe* и *user@localhost* указывают, какие должны быть уведомления и куда их посылать.

Последние два параметра (*pmem=100mb* и *pcput=10:00:00*) определяют запрашиваемый размер оперативной памяти и максимальное время исполнения программы.

Остальные строчки скрипта – это собственно задача, которая должна быть исполнена на кластере: переход в каталог с программой и запуск ее на параллельное исполнение скриптом MPICH.

Запуск задания, точнее, размещение его в очереди на исполнение, осуществляется командой qsub, единственным параметром которой будет имя скрипта:

qsub flops.pbs

В разделе 1 рассматривался способ запуска параллельной программы в среде MPICH, где в качестве одного из параметров команды запуска указывался файл со списком вычислительных узлов кластера. При использовании менеджера ресурсов Torque этот параметр не нужен. Конфигурацию кластера и список узлов диспетчер MPICH берет непосредственно из Torque.

Стандартные потоки вывода (STDOUT, STDERR) после завершения программы можно найти в том каталоге, из которого программа была запущена, в файлах с именами Flops_TEST.o45 и Flops_TEST.e45. Цифры – это номер очереди задания, а название перед расширением – это имя, которое указано в скрипте запуска.

Вопросы для самопроверки

1. В списке узлов пусто, все необходимые процессы запущены, опишите алгоритм поиска решения.

2. Узлы видны, но находятся в состоянии «down». В чем может быть причина?

3. Задание находится в очереди, но не выполняется. Опишите процедуру диагностики этой проблемы.

4. Что понимается под «ресурсами»?

5. Что произойдет, если в очередь заданий помещается задание запросами на выделение ресурсов, которые гарантированно не могут быть удовлетворены?

6. Каким образом можно запустить задачу в режиме PVM на кластере под управлением Torque?

7. Вы пытаетесь поставить задание от имени суперпользователя (root). Какова будет реакция системы?

8. Может ли центральный узел кластера выполнять роль вычислительного?

9. При запуске процессов кластера появляется сообщение об отсутствии файлов с расширением *.so. В чем причина этого и как это исправить?

10. Какими правами нужно обладать, для того чтобы монопольно использовать все доступные ресурсы кластера?

3. ВЫЧИСЛИТЕЛЬНЫЙ КЛАСТЕР CONDOR

3.1. Основные теоретические сведения
Возможности Condor

Система планирования загрузки вычислительного кластера Condor [5] (*www.cs.wisc.edu/condor/downloads*) была создана группой разработчиков университета Wisconsin-Madison, где и была развернута первая конфигурация. В настоящее время система свободно распространяется в загрузочных модулях для следующих платформ (см. таблицу).

Архитектура	*Операционная система*
Intel x86	– RedHat Enterprise Linux 6
	– Все версии Windows Vista или выше (с ограничениями)
x86_64	– Red Hat Enterprise Linux 6
	– Red Hat Enterprise Linux 7
	– Debian Linux 7.0 (wheezy)
	– Debian Linux 8.0 (jessie)
	– Macintosh OS X 10.7 through 10.10 (clipped)
	– Ubuntu 12.04; Precise Pangolin (clipped)
	– Ubuntu 14.04; Trusty Tahr

Для некоторых платформ Condor имеет ограничения – не поддерживает контрольные точки (checkpoint) и удаленные системные вызовы (работа только в режиме «Vanilla»).

Во многих случаях, особенно если речь идет о расчетных задачах, пользователям намного важнее не быстрота выполнения одного задания, а число заданий, которые можно выполнить за достаточно про-

должительное время. Такой постулат лежит в основе технологии эффективного использования имеющихся компьютерных ресурсов – High Throughput Computing (HTC). Система Condor – это ПО для поддержки среды HTC, образованной станциями на платформе UNIX и NT. Несмотря на то что Condor может управлять специализированными кластерами из рабочих станций, его ключевое преимущество – способность распределять обычные компьютерные ресурсы, доступные в любой лаборатории или офисе. Иногда Condor называют «охотником за свободными станциями»: вместо того чтобы запускать задания на своей машине, пользователь обращается к системе, которая ищет временно свободные машины в сети и запускает на них задания. Когда машина перестанет быть свободной (вернувшийся с обеда пользователь нажал клавишу или кнопку мыши, либо машина получила команду выгрузки ОС), Condor прерывает выполнение заданий, осуществляет их миграцию на другую свободную машину и перезапускает задание на ней с прерванного места. Если нет свободных машин, то задание помещается в очередь и ждет свободных ресурсов. Предусмотренный в Condor механизм управления заданиями предполагает ведение очередей, составление расписаний, схему приоритетов и классификацию ресурсов и позволяет пользователю быть в курсе дел о состоянии своих заданий, не заботясь об их дальнейшей судьбе.

Предложенная в Condor дисциплина работы ориентирована на выполнение продолжительных заданий, время счета которых исчисляется часами. Например, если заданию необходимо пять часов счета, то оно может начать работу на машине «A» и после трех часов мигрировать на машину «B», где через два часа оно завершится, о чем пользователь будет оповещен. Возможно и более глубокое деление общего времени выполнения, например, на 250 различных квантов. При этом Condor не требует включения машин в состав сетевых файловых систем, таких как NFS или AFS, и доступные станции могут размещаться на разных доменах. Кроме обычного режима обработки контрольных точек с запоминанием состояния, Condor может периодически сохранять текущее состояние, что особенно полезно для обеспечения надежности работы в случае сбоя компьютеров из пула или страховки от более

прозаических вещей типа непроизвольного выключения станции без операции «shutdown».

Кроме выполнения миграции на свободные машины, Condor обеспечивает управление распределенными ресурсами. В отличие от других систем СУПО, в которых администратору или пользователю требуется вручную редактировать реквизиты задания в очереди, чтобы соответствовать требованиям выполняющего компьютера и протолкнуть задание на счет, Condor полностью автоматизирует процесс распределения заданий, используя для этого объектную технологию ClassAds. Все машины, доступные пулу Condor, объявляют свои возможности в рубрике «Предложение»: объем памяти на диске, тип и быстродействие процессора, объем виртуальной памяти, средняя загрузка и т. п. Вместе с тем пользователь описывает потребности своего задания в рубрике «Запрос ресурсов». Condor работает в роли брокера, удовлетворяющего заявки из рубрики «Запрос ресурсов» ресурсами, представленными в рубрике «Предложение». Одновременно система обеспечивает ведение дисциплины нескольких уровней приоритетов рассмотрения заявок: приоритет назначения ресурсов, приоритет использования и приоритет среди машин, удовлетворяющих одинаковым заявкам.

При работе с системой Condor от пользователя не требуется специальной регистрации на тех машинах, где будет выполняться его задание, – система сама выполняет регистрацию с помощью технологии удаленного вызова RSC (Remote System Call), позволяющей перехватывать вызовы ОС при выполнении операций чтения/записи файлов и пересылать их на ту машину, откуда было запущено задание. Система не требует использования какого-либо специального ПО и способна выполнять обычные UNIX и NT программы, а пользователю нужно только перекомпилировать свою задачу с библиотеками Condor. Права хозяев рабочих станций, включенных в пул Condor, ни в коей мере не ущемляются – они могут использовать свои машины без каких-либо ограничений, и при этом с их стороны не требуется специальных усилий по дополнительному администрированию.

3.2. Архитектура Condor

3.2.1. Аппаратная структура

Аппаратная архитектура системы включает в себя три типа компьютеров (см. рисунок), объединенных в единый пул Condor: центральный менеджер (Central Manager), выполняющие машины (Execute Machine), запускающие машины (Submit Machine).

Архитектура Condor

Центральный сервер. Один компьютер в пуле должен выполнять функции менеджера, собирающего информацию о всех машинах и выступающего в роли брокера между имеющимися ресурсами и пользователем. Сбой в работе компьютера, на котором установлен центральный менеджер, приведет к остановке всего комплекса.

Выполняющая машина. Любая машина в пуле, в том числе центральный сервер и запускающая машина, могут быть сконфигурированы для выполнения заданий Condor.

Запускающая машина. Любая машина из пула, с которой осуществляется запуск задания. Желательно, чтобы на данной машине было достаточно ресурсов для выполнения управляющих демонов.

Каждое прерываемое для обработки контрольной точки задание имеет ассоциированный файл размером с файл задания, поэтому на диске запускающей машины должно быть достаточно свободного места для сохранения этого файла.

Кроме этих машин в пуле может быть «Сервер контрольных точек» (Checkpoint Server), на котором хранятся все рабочие файлы для обработки контрольных точек прерванных заданий.

3.2.2. Программная архитектура

Condor_master. Демон, контролирующий работу всех демонов Condor, запущенных на каждой машине пула, и выполняющий административные команды системы. Данный демон должен быть запущен на каждой машине из пула.

Condor_startd. Демон для представления ресурсов в пуле. Работает на каждой выполняющей машине и в случае ее готовности к выполнению задания запускает демон condor_starter.

Condor_starter. Демон для запуска заданий и мониторинга процесса его выполнения на конкретной машине. После завершения задания демон посылает уведомление запускающей машине и прекращает свою работу.

Condor_schedd. Демон управления ресурсами, необходимыми для пула Condor, должен работать на каждой запускающей машине. При запуске задания происходит обращение к schedd, который размещает задание в очереди заданий. При сбое демона schedd никакая дальнейшая работа невозможна.

Condor_shadow. Демон, работающий на запускающей машине и выполняющий функции менеджера ресурсов. Все системные вызовы с удаленных машин пересылаются демону condor_shadow, где они выполняются, а результаты отправляются обратно удаленной машине и заданию.

Condor_collector. Сбор информации о состоянии пула Condor. Все другие демоны периодически посылают этому демону данные о своем состоянии.

Condor_negotiator. Демон управления состоянием Condor. Периодически демон запускает цикл согласования, в течение которого собираются данные о состоянии ресурсов, текущем положении каждого демона condor_schedd с целью изменения приоритетов и т. п.

Condor_ckpt_server. Демон, реализующий функции сервера обработки контрольных точек. В его задачу входит сохранение текущего состояния задания.

Для инсталляции Condor требуется прежде всего развернуть центральный сервер, на котором будут выполняться демоны condor_ collector и condor_negotiator, выполняющие функции связного между всеми имеющимися машинами и заданиями, ожидающими выполнения. В случае сбоя этого сервера все активные задания будут выполняться вплоть до завершения, но новые задачи запускаться не будут. Более того, большинство инструментов Condor станут недоступными.

Все демоны Condor должны работать с правами суперпользователя root, в противном случае система не гарантирует безопасной работы. Полезно также создать пользователя с именем condor на всех машинах пула – демоны будут создавать файлы (например, log-файл), владельцем которых будет этот пользователь, а каталог home будет применяться для указания местоположения файлов Condor. На каждой машине пула должны быть подкаталоги spool, log и execute каталога, указанного в параметре LOCAL DIR файла конфигурации. Каталог execute используется для хранения заданий Condor, выполняемых на данной машине, spool необходим для хранения очередей и файлов истории, а также файлов контрольных точек для всех заданий от запускающей машины, log – место размещения log-файлов каждого демона Condor.

3.2.3. Управление ресурсами

Система Condor работает по типу брокера между продавцами – пулом свободных машин и покупателями – пользователями, запускающими свои задания. Таким образом, кроме имеющихся ресурсов

машины сообщают, при каких условиях они буду выполнять задания от Condor и какой тип задач предпочтителен. Например, машина с именем «dec8400» будет обрабатывать задания только ночью при физическом отсутствии пользователей. Кроме того, предпочтительное право занимать своими заданиями эту машину будет, например, принадлежать сотрудникам подразделения № 666.

Со стороны покупателя также выставляются требования к товару (машине), например, наличие процессоров, хорошо работающих с вещественной арифметикой либо имеющих память не менее 128 Мбайт. Все эти требования помещаются в описание задания ClassAd и доводятся до сведения Condor. Информацию из ClassAd можно вывести на экран станции пользователя с помощью статусных команд.

3.2.4. Подготовка задания

Процесс подготовки задания предусматривает описание системного окружения (Universe) вычислительной среды и, возможно, сборку задачи вместе с библиотеками Condor с помощью специальных команд компиляции.

Прежде всего надо учесть, что Condor выполняет задание в автоматическом режиме на фоне работы вычислительного комплекса, поэтому надо убедиться, что программа способна работать без вмешательства пользователя. Может быть организовано переназначение потоков ввода-вывода stdout, stderr и stdin. В системе Condor можно очень просто организовать многократное выполнение задания на разных входных данных – для этого надо специальным образом оформить задание так, чтобы для каждого запуска было предусмотрено свое множество входных и выходных файлов.

При запуске заданий в Condor можно использовать пять различных режимов работы: «Standard», «Vanilla», «PVM», «MPI» и «Globus». Режим «Standard» обеспечивает максимально возможные средства миграции и надежности выполнения задания, однако при его использовании имеются некоторые ограничения на тип запускаемых программ.

Режим «Vanilla» предоставляет более бедные возможности, но и не накладывает на программы существенных ограничений. Режим «PVM» предназначен для программ, использующих интерфейс Parallel Virtual Machine, а «MPI» нужен для программ, работающих под MPICH. Режим «Globus» позволяет пользователям вызывать систему Globus через интерфейс Condor.

Standard. В этом режиме осуществляется поддержка контрольных точек и вызов удаленных системных функций (RSC), что позволяет использовать ресурсы из всего компьютерного пула Condor через унифицированный интерфейс. Однако для этого необходимо собрать задание с библиотеками Condor. Можно использовать компиляторы gcc, g++, g77, cc, acc, c89, f77, fort77.

Вызов удаленных системных функций позволяет создать для задания максимально комфортное окружение так, как если бы речь шла о его выполнении на машине пользователя. При выполнении задания на другой машине на запускающей стартует второй процесс condor_ shadow, который перехватывает каждую попытку задания обратиться к системным функциям и выполняет все сам, возвращая заданию соответствующий результат. Таким образом, если заданию надо, например, открыть файл, который имеется только на запускающей машине, демон condor_shadow найдет этот файл и переадресует обращение к нему на выполняющую машину, на которой в данный момент идет счет задания.

Vanilla. Данный режим предназначен для программ, которые нельзя пересобрать заново с библиотеками Condor (например, по причине отсутствия исходных текстов). В этом режиме невозможно использование контрольных точек и RSC. В случае сбоя или выключения выполняющей машины задание будет либо ожидать возможности возобновления ее работы, либо будет заново запущено на другой. Передача данных возможна через NFS или AFS.

Режим Globus. Режим необходим для подключения к стандартному интерфейсу пакета Globus Toolkit путем трансляции очереди в строку на языке Globus RSL. Имя выполняемой программы является аргументом команды globusrun.

3.2.5. Обращение к Condor

С помощью команды «submit» пользователь размещает свое задание в Condor и указывает выполняемый файл, имена файлов потокового ввода/вывода для клавиатуры и экрана (stdin и stdout) и адрес email для оповещения о завершении задания. Можно также указать, сколько раз надо выполнять программу, какие данные при этом следует использовать, какой тип машины требуется для запуска задания, имя выполняемой задачи, исходные рабочие каталоги, командную строку. На основе данной информации создается новый объект ClassAd, который передается демону condor_schedd, выполняемому на запускающей машине.

Если требуется получить описание всех возможных атрибутов для конкретной машины из пула Condor, можно вызвать команду «status -l “имя машины”», например «status -l alfred» (все описанные ниже возможности имеют эквивалентные способы вызова из веб-интерфейса):

MyType = «Machine»
TargetType = «Job»
Name = «dmitrii.cs.wisc.org»
Machine = «dmitrii.cs.wisc.org»
StartdIpAddr = «<128.105.83.11:32780>»
Arch = «INTEL»
OpSys = «SOLARIS251»
UidDomain = «cs.wisc.org»
FileSystemDomain = «cs.wisc.org»
State = «Unclaimed»
EnteredCurrentState = 892191963
Activity = «Idle»
EnteredCurrentActivity = 892191062
VirtualMemory = 185264
Disk = 35259
KFlops = 19992
Mips = 201

LoadAvg = 0.019531
CondorLoadAvg = 0.000000
KeyboardIdle = 5124
ConsoleIdle = 27592
Cpus = 1
Memory = 64
AFSCell = «cs.wisc.org»
*START = LoadAvg - CondorLoadAvg <= 0.300000 && KeyboardIdle > 15 * 60*
Requirements = TRUE

Поле «Activity» обозначает состояние:

- Idle – машина свободна;
- Busy : занята;
- Suspended : задание приостановлено;
- Vacating : задание в контрольной точке;
- Killing – задание прервано.

Поле состояния:

- «Owner» недоступна для Condor;
- «Unclaimed»: доступно с низким приоритетом;
- «Matched»: машина подходит для работы, но пока Condor планировщик не включил ее в список ресурсов;
- «Claimed» – машина взята в пул;
- «Preempting»: задание выгружено (возможно, после контрольной точки) с другой машины.

Поле «Arch» – архитектура машины: «INTEL», «ALPHA», «SGI», «SUN4u» (Sun ULTRASPARC), «HPPA1» (PA-RISC 1.x PA-RISC 7000), «HPPA2» (PA-RISC 2.x PA-RISC 8000) и т.п.

Кроме этого, указывается средняя загрузка машины, количество процессоров, объем памяти на диске, объем оперативной памяти, производительность в KFLOPS (по тесту Linpack) и MIPS (по тесту Dhrystone), имя машины, тип ОС, IP адрес.

Система Condor не обеспечивает автоматическую конвертацию на другие архитектуры, и по умолчанию предполагается запуск задания

на выполняющей машине, имеющей ту же архитектуру, что и запускающая машина. Если имеет место различие архитектур, например, задание было запущено с Intel LINUX, а в пуле Condor большинство станций SPARC Solaris, то нужно организовать компиляцию и сборку задания для выполнения на другой архитектуре. Для изменения нужно включить в описание строку: Arch = «SUN4x» && OpSys== «SOLARIS251».

3.2.6. Управление процессом выполнения задания

В процессе выполнения задания пользователь может наблюдать за ходом работы с помощью команд «q» и «status» и, возможно, модифицировать задание. Если потребуется, Condor может также посылать уведомления о всех изменениях состояния задания: обработка контрольной точки, миграция на другую машину, размещение в очередь ожидания и т. п. Полный список доступных машин можно получить по команде «status-submitters»:

%condor_status -submitters
Name Machine Running IdleJobs MaxJobsRunning
ashoks@jules.ncsa.ui jules.ncsa 74 54 200
breach@cs.wisc.edu neufchatel
23 0 500
d.vlk@raven.cs.wisc raven.cs.w
1 48 200
ashoks@jules.ncsa.ui 74 54
jbasney@cs.wisc.edu 0 1
Total 109 103

Для идентификации выполняемых работ можно использовать команду «%condor_q»:

– Submitter: froth.cs.wisc.edu :
<128.105.73.44:33847> :
froth.cs.wisc.edu
ID OWNER SUBMITTED CPU_USAGE ST

PRI SIZE CMD
125.0 jbasney 4/10 15:35
0+00:00:00 U -10 1.2 hello.remote
127.0 raman 4/11 15:35 0+00:00:00
R 0 1.4 hello
128.0 raman 4/11 15:35 0+00:02:33
I 0 1.4 hello
3 jobs; 1 unexpanded, 1 idle,
1 running, 0 malformed

Колонка “ST” (status) показывает состояние задания в очереди: “U” – выполнение без контрольных точек; “R” – выполнение в данный момент; “I” – выполнялось ранее, а сейчас в контрольной точке и ожидания свободной машины. Эту же информацию можно получить в файле состояния “log”.

Можно посмотреть все машины, выполняющие конкретное задание с указанной машины, например «d.vlk@cs.wisc.org,»:

% condor_status -constraint
?RemoteUser == “d.vlk@cs.wisc.org”?
Name Arch OpSys State Activity
LoadAv Mem ActvtyTime
biron.cs.w INTEL SOLARIS251
Claimed Busy 1.000 128 0+01:10:00
cambridge. INTEL LINUX Claimed
Busy 0.988 64 0+00:15:00
falcons.cs INTEL SOLARIS251
Claimed Busy 0.996 32 0+02:05:03
istat03.st INTEL SOLARIS27
Claimed Busy 0.883 64 0+06:45:01

При выполнении задания пользователю назначается определенный приоритет, который может быть изменен командой «userprio». Чем меньше значение приоритета, тем лучше и тем больше машин будет доступно для выполнения. Первоначально приоритет равен 0,5 и изме-

няется в зависимости от запрашиваемых ресурсов. Чем меньше подходящих машин в пуле, тем выше значение приоритета. Изменение происходит автоматически, однако при конфигурации можно задавать периодичность коррекции приоритетов. При появлении в пуле пользователей с большим приоритетом (меньшим его значением) задания пользователя с меньшим приоритетом (большим его значением) будут немедленно выгружены с освобождением ресурсов для нового пользователя. В дополнение к приоритету пользователя можно изменить приоритет задания внутри своего пакета задач. Кроме того, при запуске задания могут быть помечены как “nice” – их выполнят на машинах, на которых нет других Condor-заданий.

После завершения работы Condor уведомляет пользователя через email, сообщая ему число процессоров, задействованных при счете, и общее время нахождения задания в пуле. Можно удалить задание, не дожидаясь завершения (команда “rm”).

3.3. Выполнение параллельных приложений

Система Condor может быть использована для автоматического создания среды выполнения параллельных заданий по технологии PVM путем динамического предоставления свободных в данный момент машин. При этом Condor-PVM действует в роли менеджера ресурсов для демона PVM, и как только PVM-программе потребуются дополнительные узлы, будет сформирован запрос для Condor на поиск свободных машин из пула и их размещение в виртуальную машину PVM версии 3.3.11.

Среди наиболее общих парадигм параллельных программ в системе Condor реализована наиболее общая – «мастер – рабочий». Один узел выполняет роль мастера, контролирующего процесс выполнения параллельных приложений и распределяющего наряды на работу другим узлам (рабочим). Рабочие выполняют вычисления и отсылают результат обратно мастеру. Роль мастера выполняет запускающая машина, а роль рабочих – все остальные машины из пула. Если рабочий не в

состоянии выполнить работу по причине отключения или занятости, то мастер передает ее другому. Если мастер замечает, что число рабочих недостаточно для выполнения всего объема работ, он (через вызов pvm addhosts()) запрашивает новых рабочих данной квалификации (нужной архитектуры) либо пытается перераспределить работы между имеющимися рабочими, каждый из которых обладает своей производительностью и «навыками».

Для работы в режиме Condor-PVM не требуется изменений программы – используется существующий стандартный интерфейс вызовов PVM, таких как pvm addhosts() и pvm notify(). При описании задания пользователь указывает режим «universe = PVM», а система сама создает программу-мастер и условия для запуска рабочих с помощью команды pvm spawn. Фактически обычные PVM-программы имеют двоичную совместимость с Condor.

Кроме PVM, система Condor поддерживает выполнение параллельных приложений, использующих MPI в версии от Argonne National Labs (*http://www-unix.mcs.anl.gov/mpi/mpich/*.

В отличие от PVM текущие реализации MPICH не допускают динамического распределения ресурсов, поэтому если задание запущено, например, на четырех узлах, то ни один из них не может быть выгружен другими заданиями Condor. Кроме этого, пока в реализации Condor MPICH не существует диспечеризации, поэтому возможна ситуация, когда ни одна из задач не сможет выполняться (deadlock). Создание такого диспетчера – основная задача следующей версии Condor.

Машины, предназначенные для выполнения заданий MPI, должны иметь разделяемую файловую систему – нельзя использовать RSC, как это предусмотрено в режиме Standard.

3.4. Ограничения Condor

Изначально разработка Condor носила исследовательский характер, и только совсем недавно под влиянием возрастающего интереса к распределенным вычислениям (метакомпьютингу) разработчики получи-

ли достаточное финансирование, позволяющее существенно развить функциональность системы и повысить ее надежность. А пока в текущих версиях имеется ряд существенных ограничений.

На платформах HPUX и Digital Unix (OSF/1) не поддерживаются разделяемые библиотеки, поэтому возможны только статические задания (на платформах IRIX, Solaris и LINUX работа с контрольными точками поддерживается для разделяемых библиотек).

В задании нельзя указывать вызовы fork(), exec(), system() – допускается работа только одного процесса. Нельзя организовать взаимодействие процессов через семафоры, конвейеры и разделяемую память. Не поддерживаются такие функции, как таймеры, будильники и ждущие процессы: alarm(), getitimer() и sleep(). Несмотря на то что в Condor поддерживаются сигналы и их дескрипторы, имеются зарезервированные сигналы SIGUSR2 и SIGT-STP, которые нельзя использовать в коде пользовательских программ.

В системе не поддерживаются команды межпроцессорного взаимодействия (IPC): socket(), send(), recv(). Не допускается использование множественных потоков на уровне ядра и применение функций отображения файлов в память mmap() и munmap(). Не допускается блокировка файлов, а все файловые операции должны быть идемпотентными – «только чтение» и «только запись», поэтому программы, которые одновременно читают и пишут в один файл, могут работать некорректно.

Если текущий рабочий каталог на запускающей машине доступен через NFS, то у системы могут быть проблемы с автоматическим монтированием дисков (automounter), если потребуется размонтировать каталог до завершения задания.

3.5. Безопасность

По умолчанию система Condor конфигурируется так, чтобы позволить любому желающему видеть состояние работы в пуле, однако при необходимости этот режим можно изменить, предусмотрев, например,

различные уровни доступа. В системе можно установить следующие уровни доступа: READ, WRITE, ADMINISTRATOR, OWNER, NEGOTIATOR.

READ. Машины с таким доступом могут только читать, ничего не меняя, информацию из Condor, например, статус пула, очередь заданий и т. п. **WRITE** – доступ для записи, что означает возможность обновления ClassAd и инициацию других машин на выполнение задания. Кроме того, машины с уровнем **WRITE** могут запрашивать демон startd для периодического выполнения записи по контрольной точке любого задания. **ADMINISTRATOR** – изменение приоритетов пользователя, активация или исключение машины из пула, запрос на реконфигурацию, рестарт и т. п. **OWNER** – уровень пользователя-владельца машины. **NEGOTIATOR** – специальный уровень доступа, означающий, что специальные команды работы с пулом (например, опрос состояния) будут обрабатывать, только если они пришли от центрального сервера. Можно указать перечень команд, разрешенных для исполнения. Уровень **NEGOTIATOR** автоматически назначается системой, остальные четыре уровня обычно прописываются в файле конфигурации. Права **ADMINISTRATOR** по умолчанию принадлежат только центральному менеджеру, **OWNER** – выполняющей машине.

3.6. Примеры файлов описания задания

Пример 1. Выполнение одной копии программы “f_r30”.

Система будет пытаться выполнить задание на машине с той же архитектурой и ОС, что и запускающая машина.

Executable = f_r30
Queue

Пример. 2. Создание очереди для двух копий программы “mathematica”. Первая копия выполняется в каталоге “run 1”, вторая – в “run 2”.

В обоих случаях задаются имена файлов для обработки stdin, *stdout и stderr.*

Executable = mathematica
Universe = vanilla
input = test.data
output = loop.out
error = loop.error
Initialdir = run_1
Queue
Initialdir = run_2
Queue

Пример 3. Запуск программы “irbis” 150 раз.

Программа должна быть компилирована и собрана на станции Silicon Graphics под ОС IRIX 6.x с памятью не менее 32 Мбайт и запушена на машине с памятью не менее 64 Мбайт. Заданы файлы stdin, stdout и stderr: “in.xxx”, “out.xxx” и “err.xxx” где xxx – номер запуска. Файл состояния “irbis.log” будет содержать данные о миграциях и прерываниях задания.

Executable = irbis
Requirements = Memory >= 32 && OpSys == «IRIX6» &&
Arch ==»SGI»
Rank = Memory >= 64
Image_Size = 28 Meg
Error = err.$(Process)
Input = in.$(Process)
Output = out.$(Process)
Log = irbis.log
Queue 150

3.7. Развертывание кластера Condor

3.7.1. Настройка главного и вычислительного узлов Condor на ОС Linux

1. Получить у преподавателя адреса, логины и пароли для доступа к системам на ОС Linux Centos 6, на которых будет производиться настройка центрального и вычислительного узлов кластера. Машины доступны через протокол ssh по стандартному порту.

2. Выполнить настройку главного узла

– Устанавливаем программу wget:

yum install wget

– Подключаем репозиторий Condor:

cd /etc/yum.repos.d/
wget http://research.cs.wisc.edu/htcondor/yum/repo.d/htcondor-stable-rhel7.repo

– Устанавливаем необходимое ПО:

yum install condor-all.x86_64

– Разрешаем внешние подключения от вычислительных узлов:

iptables -I INPUT -s 172.16.7.x -j ACCEPT

где *172.16.7.x* – адрес вычислительного узла.

– Указываем соответствие имен и адресов в файле /etc/hosts

Имя узла можно узнать, выполнив на нем команду *hostname*

– Производим настройку конфигурационного файла Condor, расположенного по адресу /etc/condor/condor_config

ALLOW_WRITE = $(ALLOW_WRITE), $(CONDOR_HOST), ***имя_вычислительного_узла1****, ...*
CONDOR_HOST = ***имя_центрального узла***
DAEMON_LIST = COLLECTOR, MASTER, NEGOTIATOR, SCHEDD, STARTD

– Запускаем демоны Condor:

service condor start

Если все выполнено *правильно, то программа condor_status будет иметь некоторый не пустой текст на выходе.*

3. Выполнить настройку вычислительного узла

– Устанавливаем программу wget:

yum install wget

– Подключаем репозиторий Condor:

cd /etc/yum.repos.d/
wget http://research.cs.wisc.edu/htcondor/yum/repo.d/htcondor-stable-rhel7.repo

– Устанавливаем необходимое ПО:

yum install condor-all.x86_64

– Разрешаем внешние подключения от центрального узла:

iptables -I INPUT -s 172.16.7.x -j ACCEPT

где *172.16.7.x* – адрес центрального узла.

– Указываем соответствие имен и адресов в файле /etc/hosts

Имя узла можно узнать, выполнив на нем команду *hostname*

– Производим настройку конфигурационного файла condor, расположенного по адрусу /etc/condor/condor_config

ALLOW_WRITE = $(ALLOW_WRITE), $(CONDOR_HOST)
CONDOR_HOST = ***имя_центрального узла***
DAEMON_LIST = MASTER, STARTD

– Запускаем демоны Condor:

service condor start

Если все выполнено правильно, то программа condor_status, выполненная на центральном узле кластера, сообщит нам, что в кла-

стер был добавлен еще один узел. Добавление происходит не мгновенно, а с некоторой периодичностью, и стоит подождать две-три минуты.

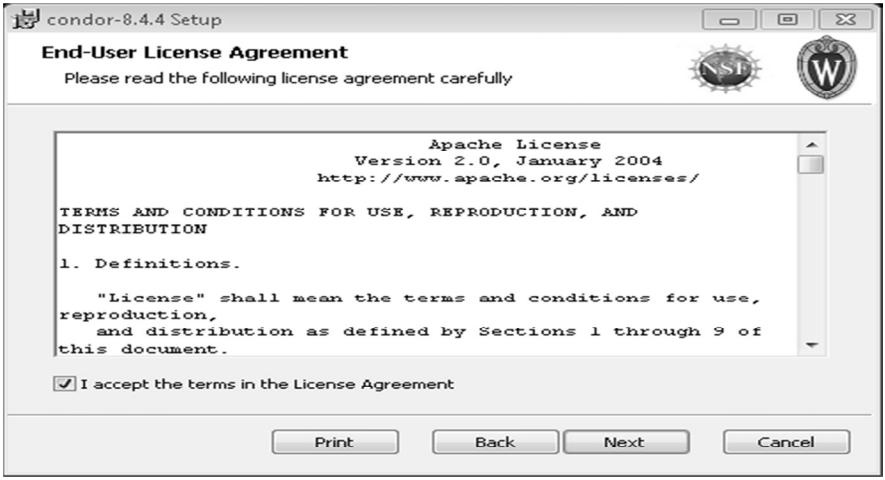
3.7.2. Настройка вычислительного узла Condor на ОС Windows

1. Получить у преподавателя адрес, логин и пароль для доступа к системе на ОС Windows 7, на которых будет производиться настройка вычислительного узла кластера.

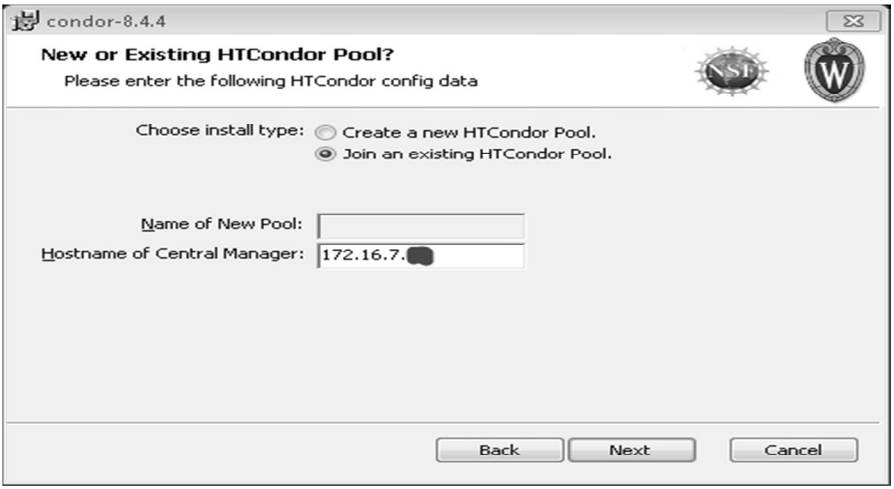
2. Используя программу mstsc, подключиться к системе и выполнить подготовку к настройке кластера путем установки Java VM. Для доступа в Интернет с компьютеров кафедры ВТ необходимо указать в настройках браузера HTTP Proxy со следующими параметрами: 217.71.138.1:8080. Установочный пакет JavaRE можно получить по адресу http://java.com.

3. Установить пакет Condor на машину с ОС Windows

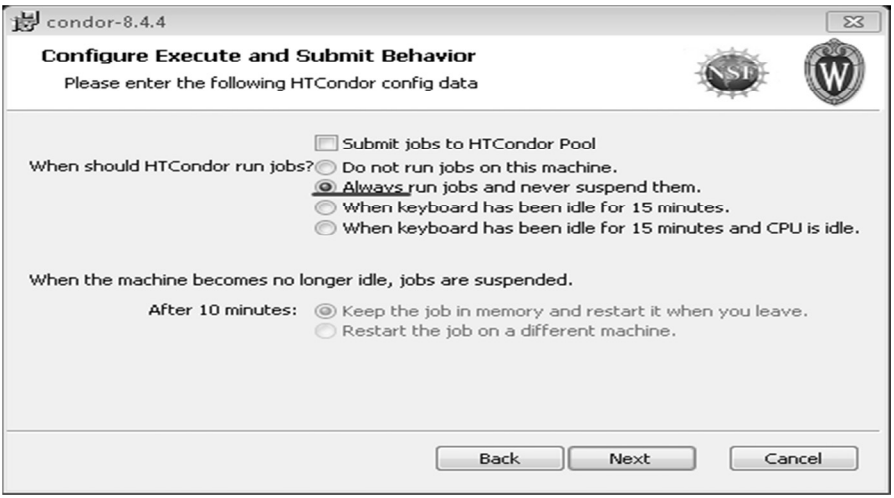
Кондор приветствует нас.



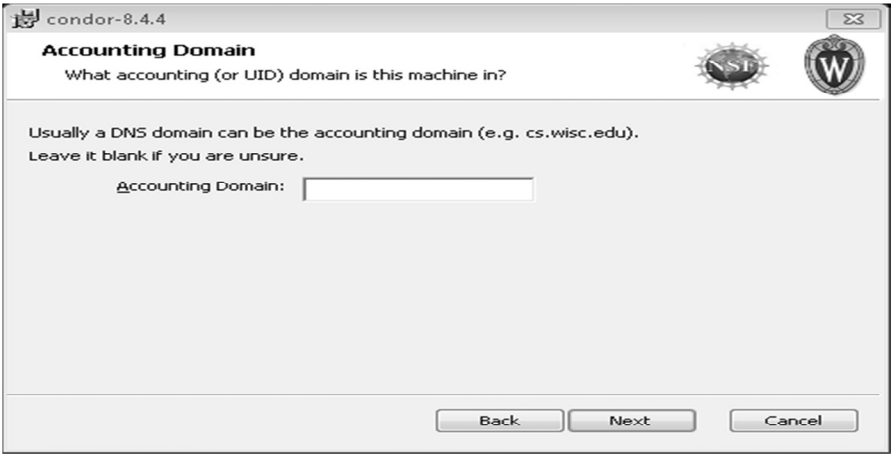
Соглашаемся с условиями лицензионного соглашения.



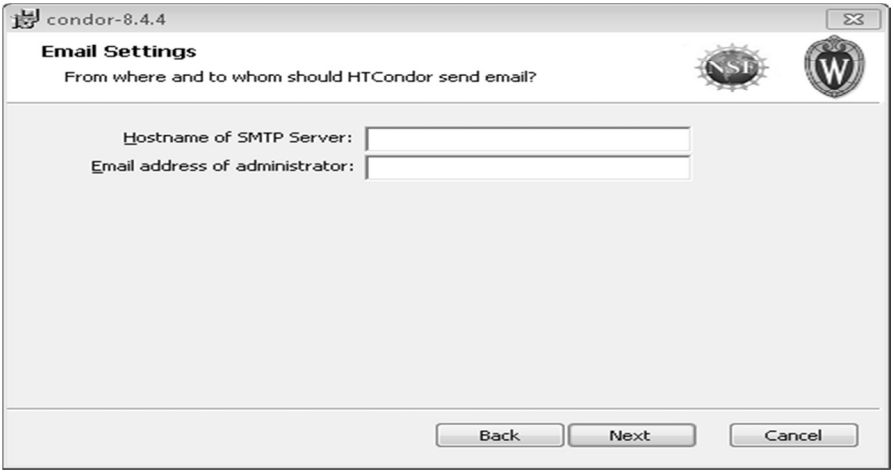
Сообщаем, что будем присоединять данную машину к существующему пулу, и указываем адрес центрального узла кластера.



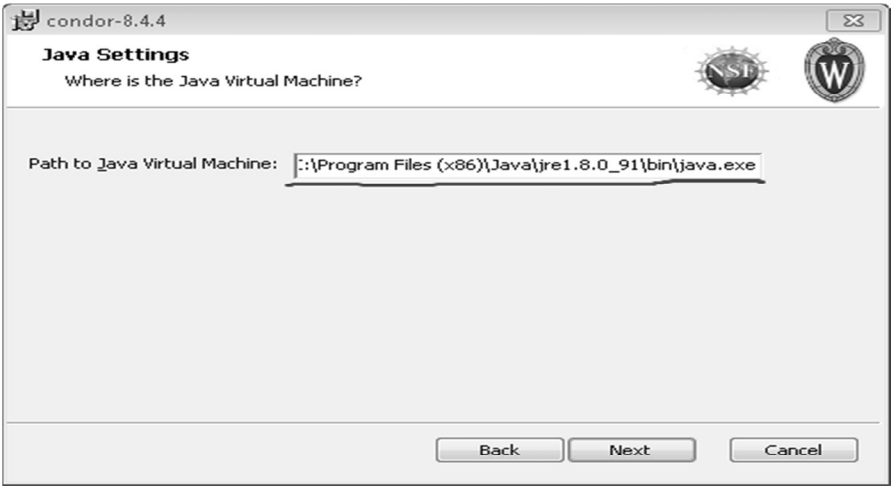
Настраиваем поведение узла, сообщая, что ресурсы могут быть выделены по первому требованию, а не только во время простоя машины.



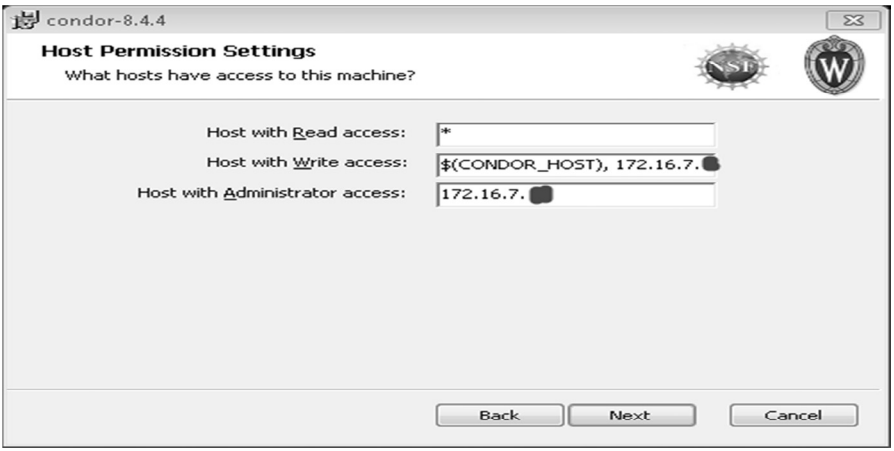
Домен в данной конфигурации не используется, Next будет подходящим ответом.



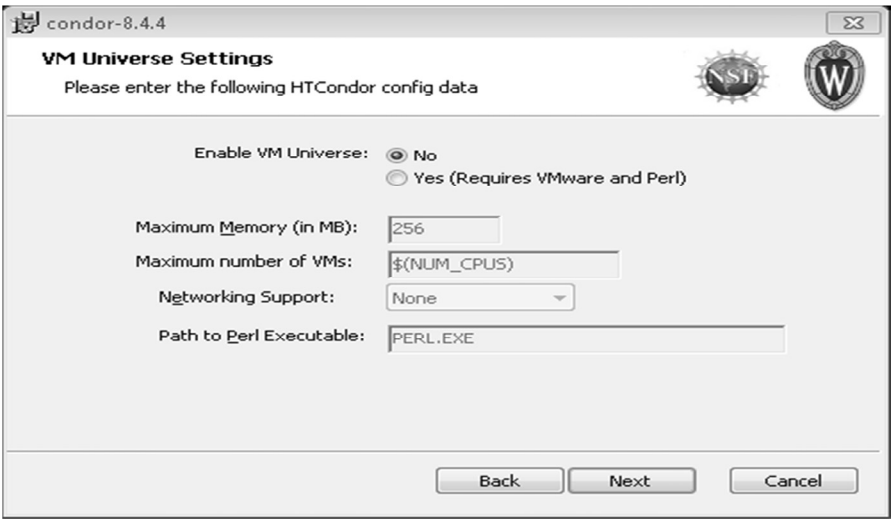
Настройка сервера для исходящей почты (в данной работе не используется, Next).



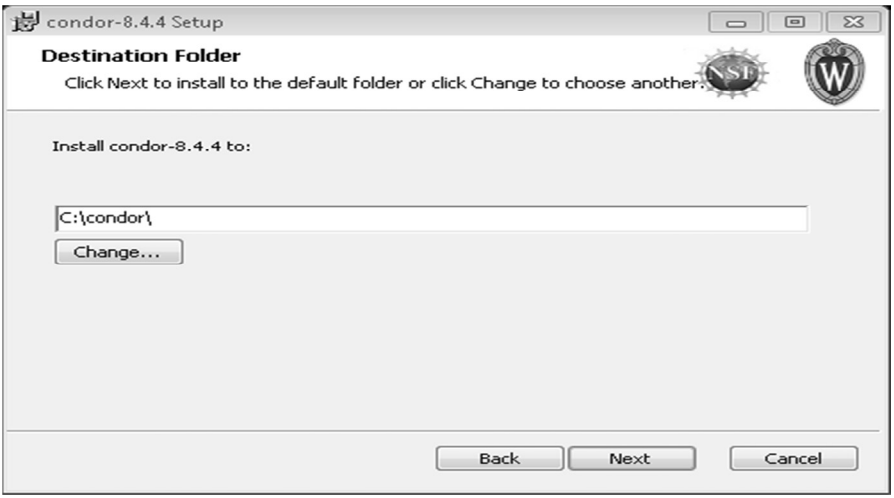
Путь к Java VM, установленной на этапе подготовки машины (эти сведения можно получить в панели управления, разделе Java).



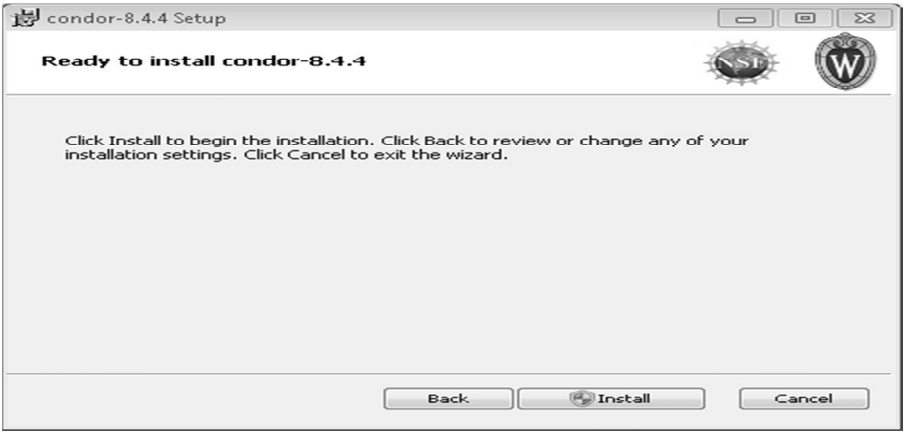
Настройка разрешений (здесь необходимо указать как минимум IP адреса центрального узла кластера. $(CONDOR_HOST) – это переменная, ее можно оставить).



VM Universe в данной работе не используется, Next.



Путь, по которому будут размещаться файлы кластера, можно оставить без изменений.



Все необходимые параметры установки заданы, пора установить Condor на данную машину.

После установки будет предложено перезагрузиться, необходимо с этим согласиться. После перезагрузки следует зайти в управление сервисами и запустить службу Condor. Альтернативный вариант – выполнить от имени администратора cmd, в открывшемся окне консоли выполнить команду:

net start condor

После чего следует добавить в конфигурационный файл condor_config на центральном узле кластера новую запись в параметр ALLOW_WRITE с именем машины для нового узла. Соответствие имени и ее адреса указывается в файле /etc/hosts на центральном узле.

Если все выполнено правильно, то в выводе команды condor_status появится следующая информация, относящаяся к нашему узлу:

sysadm-win7-0 WINDOWS X86_64 Unclaimed Idle 0.000 1007 0+00:39:39

Вопросы для самопроверки

1. В списке узлов пусто, все необходимые процессы запущены. Опишите алгоритм поиска решения.
2. Какие ограничения накладываются на задание при выборе запускающего узла?
3. Опишите режим работы задания “Vanilla”.
4. Что такое “deadlock” и как с ним бороться?
5. Перечислите основные типы ресурсов, которыми оперирует кластер на основе Condor.
6. Каким образом используются контрольные точки и в каких случаях они невозможны?
7. Опишите режим работы задания “Standard”.
8. Как можно манипулировать приоритетом задания и какими полномочиями необходимо обладать?
9. Для некоторых режимов работы необходима поддержка NFS. Что понимается под этой аббревиатурой?
10. Как можно манипулировать приоритетом задания и какими полномочиями необходимо обладать?

4. МЕНЕДЖЕР КЛАСТЕРОВ И ПЛАНИРОВЩИК ЗАДАНИЙ SLURM

4.1. Основные теоретические сведения

SLURM [6] – это высокомасштабируемый отказоустойчивый менеджер кластеров и планировщик заданий для больших кластеров вычислительных узлов. SLURM поддерживает очередь ожидающих заданий и управляет общей загрузкой ресурсов в процессе выполнения работы. Также SLURM управляет доступными вычислительными узлами в эксклюзивной или неэксклюзивной форме (как функция потребности в ресурсах). Наконец, в дополнение к мониторингу параллельных заданий вплоть до их завершения SLURM распределяет нагрузку по выделенным узлам. Не вдаваясь в подробности, можно сказать, что SLURM – это надежный, переносимый, масштабируемый на большое количество узлов, отказоустойчивый и, что самое важное, открытый менеджер кластеров (ориентированный больше на необходимые функции, чем на обеспечение дополнительных возможностей). SLURM начинал совместно разрабатываться несколькими компаниями (в их число входила Ливерморская национальная лаборатория имени Э. Лоуренса) как Open Source-менеджер ресурсов. Сегодня SLURM является лидером среди менеджеров ресурсов и используется на многих самых мощных суперкомпьютерах.

Поддерживаемые операционные системы:

- **FreeBSD** – полностью поддерживается;
- **Linux** – полностью поддерживается на архитектурах i386, ia64, x86_64;
- **NetBSD** – полностью поддерживается;
- **Solaris** – OpenSolaris полностью поддерживается.

4.2. Архитектура SLURM

В SLURM реализована типичная архитектура управления кластером (рис. 4.1). Верхний уровень управления – это резервированная пара контроллеров кластера (хотя резервирование не является обязательным). Эти контроллеры управляют вычислительным кластером и содержат демон управления под названием slurmctld. Демон slurmctld следит за вычислительными ресурсами, но, что более важно, он занимается распределением этих ресурсов между разными заданиями.

Каждый вычислительный узел содержит демон под названием slurmd. Демон slurmd управляет узлом, в котором он запущен, в том числе занимается мониторингом выполняющихся на узле заданий, получением заданий от контроллера и их распределением по ядрам внутри узла. Кроме того, slurmd останавливает выполнение заданий по запросу контроллера.

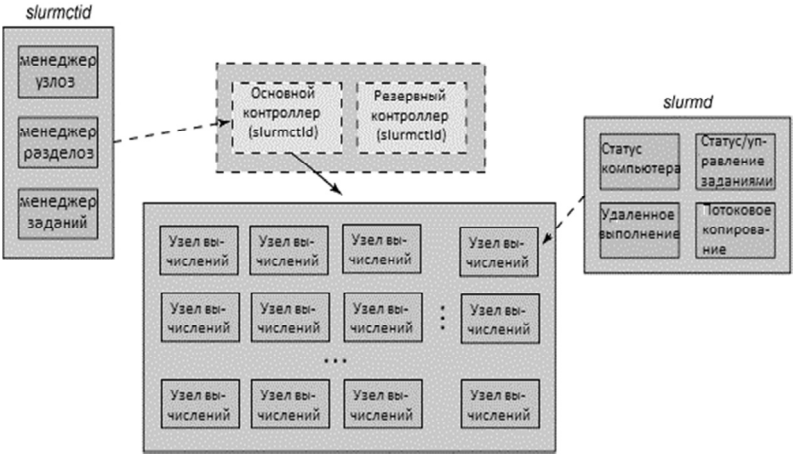


Рис. 4.1. Высокоуровневое представление архитектуры SLURM

Внутри архитектуры существуют и другие демоны (например, демоны для безопасной аутентификации). Тем не менее кластер – это не просто случайный набор узлов, поскольку в каждый момент времени часть из них логически связана с определенной задачей параллельных вычислений.

Набор узлов можно объединить в логическую группу, называющуюся *разделом* и обычно включающую в себя очередь входящих заданий. Для разделов можно задавать ограничения (например, указывать пользователей, которые могут их использовать, размер задания или предельный срок обработки). Другая особенность раздела заключается в том, что пользователю на определенный период времени выделяется часть узлов для выполнения работы, которая называется *заданием*. Каждое задание содержит один или несколько *шагов*, представляющих собой наборы задач, выполняющихся на выделенных узлах.

Эта иерархия, более подробно показывающая разделение ресурсов на разделы в SLURM, изображена на рис. 4.2. Обратите внимание на то, что при таком разделении учитывается близость ресурсов, что позволяет обеспечить низкие задержки при взаимодействии совместно работающих узлов.

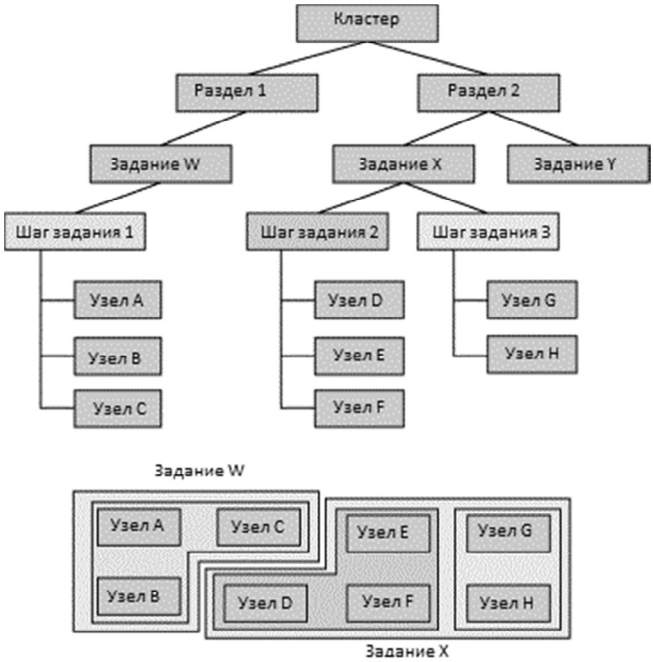


Рис. 4.2. Разделение на разделы в SLURM

4.3. Основные команды SLURM

4.3.1. Команды запуска задач

Система SLURM [7] позволяет с помощью команд **srun** и **sbatch** работать соответственно в *интерактивном* и *пакетном* режиме. ***Пакетный режим является основным в работе с кластером***.

Команда **srun** для запуска *интерактивной* программы имеет вид

srun -n <число процессов> -t <время> <имя программы> [<параметры программы>] [&]

Команда **sbatch** для запуска программы в пакетном режиме имеет вид

sbatch -n <число процессов> -t <время> <имя скрипта>

Или

sbatch -n <число процессов> -t <время> --wrap="srun <имя программы> [<параметры программы>]"

Библиотека **OpenMPI** работает только с командой **sbatch** и при этом перед именем программы вместо srun добавляется **orterun**:

sbatch -n <число процессов> -t <время> --wrap="orterun <имя программы> [<параметры программы>]"

Параметры**:**

-n – задает число процессов; если не задано, то по умолчанию **n=1**;

-t – заказывает время для решения задачи; при отсутствии **t** выделяется **30 мин** (по умолчанию);

20 ч – максимальное время, выделяемое для счета задачи в будние дни (с 9 ч понедельника до 16 ч пятницы);

85 ч – максимальное время, выделяемое для счета задачи на выходные (с 16 ч пятницы до 9 ч понедельника плюс 20 ч), т. е. на 85 ч задача запустится, если она войдет в решение в 16 ч ближайшей пятницы, иначе будет ждать следующей; аналогично на 84 ч есть шанс запустить до 17 ч пятницы, и так до 9 ч понедельника;

100 ч – максимальное время, если к выходным добавляются праздничные дни;

Время может быть задано в виде

минуты, **минуты:секунды**, **часы:минуты:секунды, дни-часы**, **дни-часы:минуты, дни-часы:минуты:секунды**

& – позволяет запустить интерактивную задачу в фоновом режиме, при котором пользователю доступна работа в командной строке и одновременно выдача результатов работы интерактивной задачи идет на экран.

Опция -**N** позволяет задать число узлов (nodes) для задачи, если пользователю это важно:

-N <кол-во узлов>

Для увеличения оперативной памяти можно воспользоваться опциями:

--mem-per-cpu=<MB> – задает минимальную память в расчете на одно ядро в мегабайтах; если не задано, то по умолчанию 1 ГБ;

--mem=<MB> – задает память на узле в мегабайтах.

Эти опции взаимно исключают друг друга.

Опция **--gres** с указанием требуемого количества **GPU** в виде

--gres=gpu:<кол-во GPU на одном узле>

задает запуск программы на узлах с GPU.

Если gres не задан, то количество GPU=0, т. е. GPU при счете не используются.

Опция **-p** позволяет указать раздел (partition) кластера для запуска задачи:

-p <раздел> или --partition=<раздел>

Если раздел не задан пользователем явно, то по умолчанию будет выбран раздел **all**, но при отсутствии в нем нужного числа свободных процессоров будут задействованы разделы **sis-0_p2** и **apollo**.

Список разделов выдается командой **sinfo -s**, при этом **all** как раздел по умолчанию помечен *****. Разделы **all**, **sis-0_p2**, **apollo** и **debug** покрывают весь кластер и взаимно не пересекаются, т. е. содержат разные узлы.

Опция **-p debug** позволяет запускать задачи в специально выделенном **для отладки** программ разделе **debug** с максимальным временем счета **20 мин**.

Примеры команд запуска с заданием раздела.

srun -p debug mytest
mqrun -np 360 -maxtime 20 -p apollo myprog

или, используя соответствующие опции srun,

mqrun -n 360 -t 20 -p apollo myprog

Опции **mqrun** выдаются по команде mqrun -help.

Пример. В результате интерактивного запуска

srun hostname

выдается имя узла, на котором запущен соответствующий процесс, например:

sis-v1

При запуске в пакетном режиме команда запуска программы задается либо в скрипте, либо через --wrap, например:

sbatch mybat

или

sbatch -n 2 --wrap "srun hostname"

где скрипт mybat

#!/bin/sh
#SBATCH -n 2
srun hostname &
wait

Команда **srun** внутри скрипта может запрашивать ресурсы только в тех пределах, которые установлены командой **sbatch**.

Скрипт запускается только на первом из выделенных узлов.

Запуск нескольких процессов осуществляется командой srun. При этом **все опции**, указанные в командной строке или самом скрипте в строках **#SBATCH**, приписываются к каждой команде **srun** данного скрипта, если не переопределены в ней. Так, результирующий файл приведенного примера будет содержать две строки с именами узлов (возможно, одинаковых), на которых выполнятся два процесса задачи, сформированные командой srun.

Если команды srun запускаются в фоновом режиме (символ **&** в конце строки), то они при наличии ресурсов могут выполняться одновременно.

По умолчанию стандартный вывод пакетной задачи и стандартный поток ошибок направляются в файл с именем slurm-%j.out, где %j заменяется уникальным идентификатором (номером) задачи.

Перенаправление ввода-вывода можно выполнить, указав программе sbatch опции

--error=<filename pattern>, --input=<filename pattern>,
--output=<filename pattern>.

При задании имени файла (filename pattern) можно использовать символы замены, в частности, **%j**.

Выдаваемые результаты конкретной команды **srun** можно поместить вместо стандартного в указанный файл, добавив после команды символ перенаправления вывода (**>**)

srun mytest > out_mytest &

Можно (чаще – при интерактивном запуске) параллельно просматривать результаты и сохранять их в файле, например:

srun --mem 40000 hostname | tee out_hostname

Описание всех опций и примеры команд можно посмотреть в **man**-руководстве с помощью команд:

man sbatch
man srun

Примеры постановки задач в очередь

user0@sis-0:~$ sbatch -n 3 --wrap="srun mytest1 3 5.1" – сформирована пакетная задача с запуском трех процессов mytest1 с двумя параметрами;

Submitted batch job 776 – задаче присвоен уникальный идентификатор 776;

user0@sis-0:~$ srun -N 2 sleep 30 & – сформирована интерактивная задача в фоновом режиме;

[1] 22313– [1] – номер фоновой задачи в текущем сеансе, 22313 – pid процесса srun на управляющей машине. Уникальный идентификатор можно узнать с помощью команд **squeue**, **sacct**.

Удаление задачи

Для отмены выполнения задачи служит команда **scancel** (убрать из очереди):

- *scancel <id1,id2,...,idn>* – снимает задачи с уникальными идентификаторами id1,id2,...,idn
- *scancel -u user0* – снимает все задачи пользователя user0

Снять со счёта (выполняется аналогично)

- *scancel --state=RUNNING 1,2,3* – снимает со счета уже стартовавшие задачи с идентификаторами 1,2 и 3
- *CTRL+C* – снимает интерактивную задачу без фонового режима.

Пример

user0@sis-0:~$ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
977 tesla sleep user0 R 1:32 2 tesla[1-2]
user0@sis-0:~$ scancel 977 – сняли со счета интерактивную задачу, считающуюся в фоновом режиме.
srun: Force Terminated job 977
user0@sis-0:~$ srun: Job step aborted: Waiting up to 2 seconds for job step to finish.
*slurmd[tesla1]: *** STEP 977.0 CANCELLED AT 2017-05-20T18:04:45 ****
srun: error: tesla1: task 0: Terminated
srun: error: tesla2: task 1: Terminated
Enter – нажать
[1]+ Exit 15 srun -p tesla -N 2 sleep 1h

4.3.2. Информационные команды

squeue – просмотр очереди (информации о задачах, находящихся в счете или в очереди на счет); возможно использование ключей, например:

squeue --user=`whoami` – посмотреть только свои задачи;
squeue --states=RUNNING – посмотреть считающиеся задачи;
squeue --long – выдать более подробную информацию.

Пример

user0@sis-0:~$ srun -N 2 sleep 30 &
[1] 22313
user0@sis-0:~$ squeue
JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
777 all sleep user0 R 0:23 2 sis-0[10,15]

JOBID – уникальный идентификатор задачи; никогда не используется повторно;
PARTITION – название раздела, где считается задача;
NAME – имя задачи пользователя;
USER – логин пользователя;
ST – состояние задачи (**R** – выполняется, **PD** – в очереди);
TIME – текущее время счета;
NODES – количество узлов для задачи;
NODELIST(REASON) – список выделенных узлов.

sacct – просмотр задач текущего пользователя за сутки (с начала текущего дня); возможно использование ключей, например:
sacct -a --starttime 2017-01-01 – посмотреть все задачи с начала года.

Пример

user0@sis-0:~$ sacct
JobID JobName Partition Account AllocCPUS State ExitCode
------------ ---------- ---------- ---------- ---------- --------------- ----------
522 sbatch tesla 2 COMPLETED 0:0
522.batch batch 1 COMPLETED 0:0
777 sleep all 2 CANCELLED+ 0:0
780 sbatch tesla 2 FAILED 0:0
780.batch batch 1 FAILED 127:0
783 sleep tesla 2 RUNNING 0:0

JobID – уникальный идентификатор задачи, повторно не используется;
JobName – имя задачи пользователя;
Partition – название раздела, где считается задача;
State – состояние задачи: **RUNNING** – выполняется;
PENDING – ждет в очереди;
COMPLETED – закончилась;

FAILED – закончилась по ошибке;
CANCELLED+ – снята пользователем;
ExitCode – код возврата.
sinfo – просмотр информации об узлах (прежде всего о состоянии узлов: доступны, заняты, свободны,...).
sinfo -s – выдача суммарной информации о разделах кластера без детализации по узлам.

Пример

user0@sis-0:~$ sinfo
PARTITION AVAIL TIMELIMIT NODES STATE NODELIST
sis-0 up 8:00:00 4 down* sis-0[59,92,139,201]*
sis-0 up 8:00:00 203 idle sis-0[1-58,60-91,93-118,120-138,140-200,202-208]*
sis-0 up 8:00:00 1 down sis-0119*
tesla up 8:00:00 1 alloc tesla2
tesla up 8:00:00 18 idle tesla[3-20]
tesla up 8:00:00 1 down tesla1
PARTITION – название раздела, где считаются задачи,
***** – указывает на раздел по умолчанию;
AVAIL – состояние раздела узлов: **up** – есть доступ, **down** – нет доступа;
TIMELIMIT – максимальное время, выделяемое для счета задачи;
NODES – количество узлов;
STATE – состояние (в сокращенной форме):
idle – свободен, **alloc** – используется процессом, **mix** – частично занят, частично свободен;
down, drain, drng – заблокирован;
comp – все задания, связанные с этим узлом, находятся в процессе завершения;
***** – обозначает узлы, которые в настоящее время не отвечают (not responding);
NODELIST – список узлов.

Пример выдачи **sinfo** из команд **mqinfo** и **mps:**

PARTITION SOCKET CORE CPU THREAD GRES TIMELIMIT CPUS(A/I/O/T)
sis-0 2 4 8 1 20:00:00 1203/53/408/1664
tesla 2 6 12 1 gpu:8 infinite 322/2/36/360

all 2 4+ 8+ 1 20:00:00 1525/55/444/2024*
PARTITION – название раздела: sis-0, tesla, all; ***** отмечен раздел по умолчанию;
SOCKET – число процессоров на узле;
CORE – число ядер в процессоре;
CPU – число ядер на узле;
THREAD – число нитей на ядро;
GRES – число общих для узла ресурсов, где gpu – графический ускоритель;
TIMELIMIT – максимальное время, выделяемое для счета задачи;
CPUS(A/I/O/T) – число ядер:
A (alloc) – заняты, **I** (idle) – свободны, **O** (other) – заблокированы, **T** (total) – всего.
scontrol – выдача детальной информации об узлах, разделах, задачах:
scontrol show node tesla34 – об узле, в частности, причине состояния drain, down;
scontrol show partition – о разделах;
scontrol show job 174457 – о задаче.

Информацию о **технических характеристиках GPU** выдает программа pgaccelinfo, которая входит в поставку компилятора PGI. Поэтому предварительно надо установить соответствующий модуль, выполнив команду mpiset, например:

mpiset 7
srun --gres=gpu:1 pgaccelinfo

Опция вида -w 'tesla21' позволяет выдать эту информацию для конкретного (в данном случае 21) узла:

srun -w 'tesla21' --gres=gpu:1 pgaccelinfo

4.4. Установка SLURM

Установка MariaDB

Slurm использует SQL сервер MariaDB для хранения отчетности, его необходимо установить только на центральном узле кластера как минимум для успешной сборки пакета Slurm.

yum install mariadb-server mariadb-devel

Создание глобальных пользователей

SLURM и Munge требуют наличия согласующихся UID (идентификатор пользователя) и GUID (идентификатор группы) на каждом узле кластера:

```
export MUNGEUSER=991
groupadd -g $MUNGEUSER munge
useradd  -m -c "MUNGE Uid 'N' Gid Emporium" -d /var/lib/munge -u $MUNGEUSER -g munge  -s /sbin/nologin munge
export SLURMUSER=992
groupadd -g $SLURMUSER slurm
useradd  -m -c "SLURM workload manager" -d /var/lib/slurm -u $SLURMUSER -g slurm  -s /bin/bash slurm
```

Установка Munge

Сервис аутентификации Munge может быть установлен из репозитория EPEL, который необходимо предварительно подключить:

```
yum install epel-release
```

Теперь можно устанавливать Munge:

```
yum install munge munge-libs munge-devel
```

После установки Munge необходимо создать секретный ключ сервера, для чего на центральном узле кластера требуется установить набор вспомогательных утилит для работы со случайными числами:

```
yum install rng-tools
rngd -r /dev/urandom
```

Теперь можно создать ключ:

```
/usr/sbin/create-munge-key -r
dd if=/dev/urandom bs=1 count=1024 > /etc/munge/munge.key
chown munge: /etc/munge/munge.key
chmod 400 /etc/munge/munge.key
```

После того как ключ создан, его необходимо скопировать на каждый узел кластера:

```
scp /etc/munge/munge.key root@172.16.7.61:/etc/munge
scp /etc/munge/munge.key root@172.16.7.71:/etc/munge
```

После чего можно последовательно подключиться к каждому узлу кластера и запустить сервис Munge:

chown -R munge: /etc/munge/ /var/log/munge/
chmod 0700 /etc/munge/ /var/log/munge/
service munge start

Чтобы проверить работоспособность Munge, можно попытаться подключиться с центрального узла на вычислительный:

munge -n && munge -n | unmunge
munge -n | ssh sis-sub-x unmunge
remunge

Если это получилось, значит, сервис Munge работает корректно.

Сборка установочных пакетов SLURM

Перед установкой непосредственно SLURM необходимо установить некоторые пакеты:

yum install openssh openssl openssl-devel pam-devel numactl numactl-devel hwloc hwloc-devel lua lua-devel readline-devel rrdtool-devel ncurses-devel man2html libibmad libibumad rpm-build

Исходный код Slurm может быть получен с официального сайта проекта https://www.schedmd.com/downloads.php

cd /usr/src
wget https://download.schedmd.com/slurm/slurm-17.11.5.tar.bz2
rpmbuild -ta slurm-17.11.5.tar.bz2

Собранные установочные пакеты будут размещены в каталоге:

cd /root/rpmbuild/RPMS/x86_64

Полученные файлы необходимо скопировать на каждый из вычислительных узлов кластера при помощи утилиты scp:

scp slurm-17.11.5-1.el7.centos.x86_64.rpm root@sis-sub-x:/usr/src/
scp slurm--17.11.5-1.el7.centos.x86_64.rpm root@sis-sub-x:/usr/src/*

Завершив процесс копирования, необходимо на каждом узле выполнить установку пакетов Slurm:

yum --nogpgcheck localinstall slurm-17.11.5-1.el7.centos.x86_64.rpm slurm--17.11.5-1.el7.centos.x86_64.rpm*

После того как все пакеты установлены, можно приступить к конфигурированию кластера.

По адресу https://slurm.schedmd.com/configurator.easy.html расположен генератор файлов конфигурации для Slurm. Использование конфигуратора не является обязательным, но в некоторых случаях может облегчить работу.

Например, можно указать некоторые параметры, оставив остальные в значении по умолчанию:

ControlMachine: sis-1
ControlAddr: 172.16.7.51
NodeName: sis-sub-[1-2]
CPUs: 1
StateSaveLocation: /var/spool/slurmctld
SlurmctldLogFile: /var/log/slurmctld.log
SlurmdLogFile: /var/log/slurmd.log
ClusterName: sis-x

В ответ на отправленную форму конфигуратор вернет полный текст файла конфигурации Slurm.

На узле, выполняющем роль центрального, например sis-1:

cd /etc/slurm
vim slurm.conf

Скопируйте текст файла конфигурации из формы на сайте и вставьте его в файл slurm.conf. Так как это лишь заготовка, то потребуется сделать в нем некоторые изменения.

В секции файла slurm.conf "# COMPUTE NODES" можно увидеть, что Slurm пытается определить IP-адреса при помощи одной строки, по DNS-записи, используя шаблон имени:

NodeName=sis-sub-[1-2] CPUs=1 State=UNKOWN

Можно указать имена и ip-адреса узлов явно, используя несколько строк:

1 NodeName=sis-1 NodeAddr=172.16.7.51 CPUs=1 State=UNKNOWN
2 NodeName=sis-sub-1 NodeAddr=172.16.7.61 CPUs=1

3 State=UNKNOWN
NodeName=sis-sub-2 NodeAddr=172.16.7.71 CPUs=1 State=UNKNOWN

Больше дополнительно ничего изменять не нужно, поэтому можно сохранить файлы и выйти из редактора.

Пример полного файла slurm.conf:

```
# slurm.conf file generated by configurator easy.html.
# Put this file on all nodes of your cluster.
# See the slurm.conf man page for more information.
#
ControlMachine=sis-1
ControlAddr=172.16.7.51
MpiDefault=none
ProctrackType=proctrack/pgid
ReturnToService=1
SlurmctldPidFile=/var/run/slurmctld.pid
SlurmdPidFile=/var/run/slurmd.pid
SlurmdSpoolDir=/var/spool/slurmd
SlurmUser=slurm
StateSaveLocation=/var/spool/slurmctld
SwitchType=switch/none
TaskPlugin=task/none
# SCHEDULING
FastSchedule=1
SchedulerType=sched/backfill
SelectType=select/linear
# LOGGING AND ACCOUNTING
AccountingStorageType=accounting_storage/none
ClusterName=sis-x
JobAcctGatherType=jobacct_gather/none
SlurmctldLogFile=/var/log/slurmctld.log
SlurmdLogFile=/var/log/slurmd.log
# COMPUTE NODES
NodeName=sis-1 NodeAddr=172.16.7.51 CPUs=1 State=UNKNOWN
NodeName=sis-sub-1 NodeAddr=172.16.7.61 CPUs=1 State=UNKNOWN
NodeName=sis-sub-2 NodeAddr=172.16.7.71 CPUs=1 State=UNKNOWN
```

PartitionName=debug Nodes=sis-1,sis-sub-1,sis-sub-2 Default=YES MaxTime=INFINITE State=UP

Готовый файл конфигурации slurm.conf необходимо тиражировать на остальные узлы:

scp slurm.conf root@172.16.7.61:/etc/slurm/slurm.conf
scp slurm.conf root@172.16.7.71:/etc/slurm/slurm.conf

На центральном узле sis-1 необходимо создать каталоги для хранения журналов и назначить им требуемые права доступа:

mkdir /var/spool/slurmctld
chown slurm: /var/spool/slurmctld
chmod 755 /var/spool/slurmctld
touch /var/log/slurmctld.log
chown slurm: /var/log/slurmctld.log
touch /var/log/slurm_jobacct.log /var/log/slurm_jobcomp.log
chown slurm: /var/log/slurm_jobacct.log /var/log/slurm_jobcomp.log
mkdir /var/spool/slurmd
chown slurm: /var/spool/slurmd
chmod 755 /var/spool/slurmd

На вычислительных узлах также необходимо создать некоторые каталоги и установить корректные права доступа:

mkdir /var/spool/slurmd
chown slurm: /var/spool/slurmd
chmod 755 /var/spool/slurmd
touch /var/log/slurmd.log
chown slurm: /var/log/slurmd.log

Команда для проверки синтаксиса файлов конфигурации поможет понять, что нет явных опечаток:

slurmd -C

Результат должен быть примерно таким:

ClusterName=(null) NodeName=sis-1 CPUs=1 Boards=1 SocketsPerBoard=1 CoresPerSocket=1 ThreadsPerCore=1 RealMemory=7822 TmpDisk=45753
UpTime=13-14:27:52

Поскольку сетевой экран включен по умолчанию, то он может блокировать входящие сетевые соединения и мешать нормальной работе Slurm. Можно выключить его полностью:

systemctl stop firewalld
systemctl disable firewalld

или добавить правила, разрешающие сетевой трафик по портам, используемым Slurm:

firewall-cmd --permanent --zone=public --add-port=6817/udp
firewall-cmd --permanent --zone=public --add-port=6817/tcp
firewall-cmd --permanent --zone=public --add-port=6818/tcp
firewall-cmd --permanent --zone=public --add-port=6818/tcp
firewall-cmd --permanent --zone=public --add-port=7321/tcp
firewall-cmd --permanent --zone=public --add-port=7321/tcp
firewall-cmd --reload

Последняя настройка заключается в синхронизации часов на каждом из узлов кластера, для чего можно настроить синхронизацию с сервером времени по протоколу ntp на каждом узле кластера:

yum install ntp -y
chkconfig ntpd on
ntpdate pool.ntp.org
systemctl start ntpd

Теперь ничто не мешает запустить кластер Slurm. На всех вычислительных узлах это делается командами:

systemctl enable slurmd.service
systemctl start slurmd.service
systemctl status slurmd.service

На центральном узле:

systemctl enable slurmctld.service
systemctl start slurmctld.service
systemctl status slurmctld.service
systemctl enable slurmd.service

systemctl start slurmd.service
systemctl status slurmd.service

Успешность запуска можно проверить по результату, возвращаемому командой systemctl status …, если есть проблемы, то найти их помогут журналы служб.

Вопросы для самопроверки

1. В списке узлов пусто, все необходимые процессы запущены, опишите алгоритм поиска решения.
2. Перечислите основные типы ресурсов, которыми оперирует кластер на основе Slurm.
3. Система аутентификации настроена, но не работает должным образом, опишите алгоритм диагностики проблемы.
4. Как можно манипулировать приоритетом задания в кластере Slurm и какими полномочиями необходимо обладать?
5. Что понимается под «разделом» в кластере?
6. Чем отличается режим отладки от прочих режимов работы задания?
7. Каким образом можно задействовать аппаратную акселерацию вычислений в кластере Slurm?
8. Что произойдет, если при запуске задания указать число процессоров, превышающее их сумму в кластере?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Малявко А.А*. Суперкомпьютеры и системы. Мультикомпьютеры : учебное пособие / А.А. Малявко, С.А. Менжулин. – Новосибирск : Изд-во НГТУ, 2017.

2. *Таненбаум Э., Бос Х*. Современные операционные системы. 4-е изд. – СПб.: Питер, 2015. – 1120 с.: ил. – (Серия «Классика computer science»).

3. PelicanHPC. – URL: https://www.pelicanhpc.org (дата обращения: 06.04.2018).

4. Torque Resource Manager. – URL: http://www.adaptivecomputing.com/products/open-source/torque (дата обращения: 06.04.2018).

5. HTCondor. – URL: https://research.cs.wisc.edu/htcondor/ (дата обращения: 06.04.2018).

6. Slurm workload manager. – URL: https://slurm.schedmd.com/ (дата обращения: 06.04.2018).

7. *Джонс М.* Оптимизация управления ресурсами суперкомпьютеров с помощью SLURM. – URL: https://www.ibm.com/developerworks/ru/library/l-slurm-utility/index.html (дата обращения: 06.04.2018).

Малявко Александр Антонович
Менжулин Сергей Алексеевич

СУПЕРКОМПЬЮТЕРЫ И СИСТЕМЫ

ПОСТРОЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ КЛАСТЕРОВ

Учебное пособие

Редактор *Л.Н. Ветчакова*
Выпускающий редактор *И.П. Бровановa*
Корректор *Л.Н. Кинит*
Дизайн обложки *А.В. Ладыжская*
Компьютерная верстка *Н.В. Гаврилова*

Налоговая льгота – Общероссийский классификатор продукции
Издание соответствует коду 95 3000 ОК 005-93 (ОКП)

Подписано в печать 25.07.2018. Формат 60 × 84 1/16. Бумага офсетная
Тираж 100 экз. Уч.-изд. л. 5,58. Печ. л. 6,0. Изд. 123. Заказ № 1065
Цена договорная

Отпечатано в типографии
Новосибирского государственного технического университета
630073, г. Новосибирск, пр. К. Маркса, 20