Установка кластера в полевых условиях или кластер своими руками

Максим Щербак
email: <scherbak@karelia.ru>
23 марта 2001

Эта статья не является полностью точным и достоверным руководством по созданию кастера на базе существующего парка машин. Это лишь описание тех процедур, которые проделал автор для создания такой системы и некоторые дополнительные комментарии к ним. Во многом эта статья основывается на документах из проекта Linux Documentation Project (LDP).

1 Выбор операционной системы.

Этот вопрос является первоочередным, т. к. от его решения зависят выбор программных средств и дальнейшая специфика работы. Из всего многообразия можно выделить лишь два самых доступных и функциональных варианта: Windows 9X или Linux. И под Windows и под Linux существуют пакеты программ, позволяющие организовать кластер на базе существующего парка рабочих станций в сети. Для обеих операционных систем существуют реализации наиболее распространенных языков программирования FORTRAN и C/C++. Для обеих существуют коммуникационные библиотеки, позволяющие организовывать внутрипрограммный обмен данными между локальными машинами кластера. Но выбор этих двух возможных вариантов основывался не только на том, существуют ли для операционной системы необходимые программные решения, но и

в том, как широко могут быть использованы программы, разрабатывающиеся в рамках диссертационной работы. Linux и Windows являются двумя самыми распространенными операционными системами, которые используются в научной работе. Выбор между Linux и Windows легко решется в пользу Linux по ряду весьма важных критериев. Самым большим преимуществом Linux является то, что и сама операционная система, и огромное количество прикладных программ распространяются бесплатно. Для Linux сущестуют бесплатные компиляторы, средства автоматического распараллеливания программ, отладчики, коммуникационные библиотеки и другое программное обеспечение, необходимое для разработки параллельных приложений. Если же подобный комплекс собирать под Windows, то покупка операционной системы программных средств обойдется в значительную сумму. Благодаря бесплатности программ для Linux, можно позволить себе использовать очень широкий круг инструментов, что благотворно скажется на конечных результатах.

Также, нельзя обойти вниманием тот факт, что будет необходима легкая переносимость программного обеспечения кластера, т.к. на базе компьютерного парка Петрозаводского университета не будет возможности создать постоянно действующий кластер. Для работы кластера необходимо чтобы на каждой локальной машине был запущен необходимый набор программ. В случае с Windows на каждой машине необходимо будет установить физически этот набор программ, а каждая копия программного обеспечения занимает не один десяток мегабайт. Важным требованием также является неизменяемость настроек систем кластера на каждой машине. Безусловно, в существующей ситуации с компьютерными классами университета, эти требования не могут быть выполнены при использовании программ на базе Windows.

Linux же предоставляет уникальные возможности для решения этой проблемы. На базе этой операционной системы можно организовать множество бездисковых рабочих станций, которые будут загружаться с одной, единожды установленной системы на сервере. Этим одновременно решаются две проблемы: во-первых, нужно держать всего одну копию установленной системы, а во-вторых, решается проблема постоянства настроек системы на разных хостах. Используя эту возможность, можно не прикрепляться к какому-либо компьютерному классу, для этого достаточно носить серверную установку системы на собственном жестком диске.

Кроме того, у Linux есть еще ряд преимуществ перед Windows. Прежде всего, необходимо остановиться на большей «прогрессивности» Linux-сообщества из-за открытости разработки программных средств. Например, коммуникационные библиотеки PVM и MPI пишутся именно под Linux, а уж потом портируются под Windows. Отсюда и отставание в последних версиях этих программ для разных операционных систем. Еще одним плюсом ОС Linux является безусловно большие возможности на-

стройки системы, нежели Windows. Использование модульности позволяет выбрать именно тот набор программного обеспечения, который будет необходим в работе, и не устанавливать ничего лишнего. Но все-таки необходимо упомянуть еще и о недостатках Linux, основным из которых является относительная сложность настройки и эксплуатации. Чтобы со всем разобраться, необходимо перечитать большое количество сопроводительной информации, но нельзя сказать, что это не идет на пользу.

Исходя из вышесказанного, становится очевидным, что более разумным выбором будет именно Linux.

Теперь необходимо определиться с другим важным вопросом, каким языком программирования пользоваться. Можно рассмотреть несколько распространенных сейчас языков программирования: C/C++, perl и Фортран. Доступный инструментарий средств разработки для этих языков очень широк, и на первый взгляд может показаться более логичным использование perl, как наиболее динамично развивающегося языка программирования. Но нельзя забывать об области применения этого языка и о его происхождении. Perl в основном используется для написания серверных приложений для Intrernet/WWW, правда, в последние несколько лет, он стал использоваться и C-программистами для написания полноценных приложений. Соответственно средства разработки для этого языка появляются исходя из его области применения, поэтому инструментарий для применения Perl'а в параллельном программировании весьма ограничен. Кроме этой причины есть еще и ряд других ограничений применения perl в параллельном программировании.

Исторически сложилось так, что большинство численных алгоритмов, используемых в научной работе, было написано на языке FORTRAN. Поэтому, для FORTRAN существует огромное количество научной литературы и различной документации. Кроме того, в редакции языка 1990 года, в него были заложены возможности, которые предполагалось использовать в будущем, для программирования параллельных задач. Тем не менее, существует практика переписывания FORTRAN-программ на C/C++ и аргументируется это тем, что код на C более эффективный. Это действительно так, C более приближен к машинному коду по логике работы, но в то же время в C существуют черты, которые при параллельном программировании становятся серьезными недостатками. В C/C++, а также и в их последователе perl, используются указатели и ссылки, которые заметно усложняют определение зависимостей в данных программы. Это делает анализ кода для автоматического распараллеливания и профилирования программ чрезвычайно сложной задачей. Ведь эти операции необходимо делать часто, т.к. обычно программу нужно профилировать и компилировать для каждого конкретного кластера заново. Поэтому для C еще не создано некоммерческих средств автоматического распараллеливания программ. Всилу этого ряда причин, для программирования параллельных программ рекомендуют применение именно FORTRAN.

2 Установка программного обеспечения.

2.1 Выбор дистрибутива.

Существует несколько распространенных дистрибутивов OS Linux, среди которых можно назвать RedHat, Mandrake, SUSE, SlackWare, TurboLinux, Caldera и др. Для сервера и для бездисковых рабочих станций я выбрал дистрибутив RedHat 7.0. Достоинствами его являются:

- Наличие программы автоматического обнаружения и установки нового оборудования. Эта возможность сильно поможет при переносе на жестком диске установленной серверной операционной системы в другой компьютерный класс.
- RedHat является одним из самых распространенных дистрибутивов в России и за рубежом.
- Он наиболее доступен для приобретения в магазинах или у знакомых.
- Широкий спектр поддерживаемого оборудования, что будет особенно важно при переносе кластера в другой компьютерный класс.

Но, как и всегда, имеются недостатки

- Фирма RedHat сильно поторопилась с выпуском новой версии 7.0 и в ней обнаружено очень большое количество ошибок. Как известно, стабильность и сетевая безопасность являются коньком Lunux и, безусловно, этого нельзя сказать о RH7.0. Но ошибки эти, в основном, касаются сетевой безопасности, что в моем случае не очень актуально.

RedHat, Mandrake и многие другие дистрибутивы направлены на сектор обычных пользователей и пытаются сделать «замену» OS Windows, что очень актуально за рубежом ввиду большой разницы в цене. Поэтому в этих дистрибутивах максимально упрощены процедуры настройки и интерфейс пользователя (XFree86)

2.2 Установка операционной системы.

RedHat Linux 7.0 распространяется на трех компакт-дисках. Первый является установочным и загрузочным, второй содержит дополнительные пакеты с исходными кодами программ, а третий - справочную информацию. Для операционной системы был выделен отдельный жесткий диск емкостью 10 Gb.

Загрузочный компакт-диск облегчает и ускоряет процедуру установки, т.к. загрузка облегченного варианта операционной системы Linux, под

управлением которой происходит установка, происходит непосредственно с компакт-диска, необходимо лишь сделать соответствующие установки в BIOS компьютера.

Во время установки, будет предложено разбить жесткий диск на разделы. Для нормальной работы, Linux'у необходимо минимум два раздела: раздел корневой файловой системы (root filesystem /) и раздел подкачки (swap partition). В руководстве Beowulf-Installation-HOWTO рекомендуется разбить жесткий диск следующим образом:

- 500 Mb под каталоги `/`, `/bin`, `/boot`, `/dev`, `/etc`, `/proc`, `/sbin`, `/var`, `/tmp` и каталог, в котором будут находиться NFS-файловые системы бездисковых клиентов `/tftpboot`. Очень важно, чтобы `/` и `/tftpboot` находились на одном разделе, т.к. схема, описываемая ниже, не сработает, невозможно будет установить жесткие ссылки (hard links) на файлы клиентских файловых систем.

- 1.5 Gb раздел под каталог `/usr`. Именно на этот раздел будет устанавливаться почти все программное обеспечение кластера.

- 500 Mb – 1.5 Gb для каталога `/usr/local` (или любой другой в который вы будете устанавливать дополнительные программы, касающиеся кластера).

- Swap. Авторы HOWTO рекомендуют задать размер swap partition равным удвоенному количеству оперативной памяти. В нашем случае неизвестно, каким будет объем оперативной памяти, поэтому можно заранее принять размер раздела подкачки равным 200 Mb.

Такое деление на разделы диктуется соображениями безопасности и, вероятно, традициями UNIX. В моем случае, необходимо было на том же жестком диске иметь еще и разделы MS Windows, но количество главных разделов ограничено четырьмя. Поэтому, я не стал выделять несколько linux-разделов, а создал только один размером 4.5 Gb.

При профилировании операционной системы, необходимо будет выбрать из списка программное обеспечение, которое будет использоваться. Обязательно потребуется выбрать пункты Kernel Development и пункты, касающиеся работы в сети: XX. Рекомендуется установить графическую оболочку X Windows и один из менеджеров к ней Gnome или KDE. Графический интерфейс скорее всего понадобится, так как под X Windows уже созданы удобные средства программирования, отладки и мониторинга кластера. Но, все же, используя графический интерфейс, невозможно будет полностью отказаться от использования текстового режима, т.к. в Linux большая часть задач, касающихся администрирования, традиционно решается в текстовом режиме. В моем случае установка и настройка программного обеспечения кластера проходила без использования X Windows.

После копирования файлов на диск, будет предложено сделать загрузочный диск, если читатель не уверен, то ему лучше сделать его, т.к. потом он сможет избежать множества сложностей, если система откажется загружаться.

После установки системы следующим этапом является настройка бездисковых клиентов и сервера для них.

2.3 Сборка ядра для сервера.

Прежде чем приступать к этой процедуре читателю, не уверенному в своих силах рекомендуется прочитать инструкцию Kernel HOWTO

Если во время установки операционной системы был отмечен пункт Kernel Development, то все, необходимые для пересборки ядра пакеты, будут установлены. Если же их все же нужно устанавливать вручную, то все они находятся на установочном диске. В дистрибутиве RedHat для распространения, установки и удаления программ используется программа rpm (RedHat Packet Manager), поэтому файл `*.rpm`, в котором содержится программа называется пакетом. Установить пакет можно прямо с компакт-диска командой (в linux регистр символов *имеет* значение):

```
rpm -i название_пакета.rpm
```

Для обращения к компакт-диску его необходимо сначала подмонтировать. Дело в том, что в Linux любое внешнее устройство должно быть сначала смонтировано командой mount, после чего оно становится для системы как бы несъемным, да и пользователь, например, не сможет достать смонтированный диск из привода CD-ROM, для этого его нужно размонтировать команой umount. Команды необходимые для монтирования и размонтирования компакт-диска:

```
mount -t iso9660 /dev/cdrom /mnt/cdrom
umount /mnt/cdrom
```

Для более подробной информации можно посмотреть `man mount`, `man fstab`.

Список минимального набора пакетов, необходимых для компиляции ядра, приводится ниже. Более подробную и свежую информацию можно прочитать в файле: `путь_к_исходникам/Documentation/Changes`.

Название	Версия	Команда, которой можно проверить
Kernel modutils	2.1.121	`insmod -V`
Gnu C	2.7.2.3	`gcc -version`
Binutils	2.8.1.0.23	`ld -v`
Linux libc5 C Library	5.4.46	`ls -l /lib/libc*`
Linux libc6 C Library	2.0.7pre6	`ls -l /lib/libc*`
Dynamic Linker (ld.so)	1.9.9	`ldd -version or ldd -v`
Linux C++ Library	2.7.2.8	`ls -l /usr/lib/libg++.so.*`
Procps	2.0.3	`ps -version`
Procinfo	16	`procinfo -v`
Psmisc	17	`pstree -V`
Net-tools	1.52	`hostname -V`
Loadlin	1.6a	
Sh-utils	1.16	`basename -v`
Autofs	3.1.3	`automount -version`
NFS	2.2beta40	`showmount -version`
Bash	1.14.7	`bash -version`
Ncpfs	2.2.0	`ncpmount -v`
Pcmcia-cs	3.0.14	`cardmgr -V`
PPP	2.3.10	`pppd -version`
Util-linux	2.9z	`chsh -v`
isdn4k-utils	v3.1beta7	`isdnctrl 2>&1|grep version`

В общем случае, ядро для сервера, установленное по умолчанию, не требует изменения, все необходимые возможности уже либо включены в ядро, либо могут быть подгружены по необходимости из внешних модулей. Например, в состав ядра RH7.0 по умолчанию не входит модуль RARP, необходимый для идентификации бездискового клиента. «Подгрузить» его можно используя команду:

```
insmod rarp
```

Чтобы этот модуль загружался автоматически при старте системы, необходимо изменить файл `/etc/modules.conf`, добавив в него, например, следующую строку:

```
alias rarp-ident rarp
```

Для ядра сервера очень важными элементами являются скорость работы и занимаемая память, поэтому имеет смысл отказаться от некоторых ненужных вещей, которые входят по умолчанию в ядро стандартной поставки. Например, поддержка звука, аппаратного видеоускорения, контроллеров USB, ненужных контроллеров IDE или SCSI, в зависимости от конфигурации, заведомо не нужных сетей - ArcNet, AppleTalk и

TokenRing и т.п. Особое внимание следует обратить на пункты (см. ниже), касающиеся поддержки сетевых карт, протоколов и сетевых файловых систем. Обязательно должна поддерживаться файловая система NFS (Network File System), необходимая для выделения корневых каталогов бездисковым клиентам.

Сама процедура пересборки ядра включает в себя несколько этапов: подготовка исходных текстов ядра, его конфигурация и компиляция. Все исходные тексты ядра находятся в каталоге `/usr/src/linux`. Для подготовки файлов достаточно из этого каталога выполнить команду make mrproper (эту команду очень важно выполнить, иначе при компиляции ядра будут возникать ошибки), после этого можно приступать к конфигурации ядра - `make menuconfig` или `make config`. Первая предоставляет удобный псевдографический интерфейс, основанный на меню, а вторая команда будет выводить те же вопросы последовательно.

После установки исходных текстов ядра, перед началом дальнейших действий, необходимо выполнить команду:

```
make mrproper
```

При конфигурировании ядра предлагается выбрать или отказаться от определенных характеристик будущего ядра. Существует три возможности для выбора того или иного пункта: (Y) - пункт выбран, и он будет включен в состав нового ядра, (N) - пункт не выбран, и не будет включен, для некоторых характеристик существует (M) - означающая, что эта возможность будет реализована в виде подгружаемого по мере необходимости модуля.

После сохранения новой конфигурации необходимо выполнить команды

```
make dep
make clean
```

после чего можно запустить компиляцию ядра командой

```
make bzImage
```

Общая последовательность команд должна быть такой:

```
menuconfig
make dep
make clean
make bzImage
```

Сборка ядра на компьютере Duron 650 занимала время порядка десяти минут.

Скомпилированное ядро будет помещено в файл `/usr/src/arch/boot/bzImage`. Теперь это ядро необходимо установить так, чтобы система грузилась

именно с него. В linux используется специальная программа-загрузчик lilo (linux loader), которая одновременно является своего рода менеджером загрузки. В целях *безопасности* лучше установить новое ядро в качестве второго варианта загрузки, не удаляя старое ядро. Для этого необходимо переименовать файл `bzImage`, например, `bw-server`, скопировать его в каталог `/boot` и отредактировать файл `/etc/lilo.conf`, добавив в него следующие строки:

```
image=/boot/vmlinuz-2.2.16-22 # строка загрузки
      label=linux             # ядра по умолчанию
      read-only
      root=/dev/hda3
image=/boot/bw-server    #добавляем запись для
      label=nfs          #загрузки нового ядра
      read-only
      root=/dev/hda3
```

Чтобы изменения вступили в силу, необходимо выполнить команду:

```
lilo
```

При успешном выполнении команды и при правильном синтаксисе файла lilo.conf, результат команды должен быть следующим (знак `*` обозначает вариант загрузки по умолчанию):

```
* Added linux
  Added nfs
```

При перезагрузке будет предложено выбрать вариант загрузки. В RedHat 7.0 это будет в виде графического меню, а в более ранних версиях это будет приглашение вида:

```
lilo boot:
```

Здесь необходимо ввести соответствующее имя варианта загрузки (поле `label=` в файле `lilo.conf`).

Если при загрузке нового ядра возникли проблемы, то можно загрузить старое и исправить проблемы. Вероятнее всего они могут быть вызваны неправильной конфигурацией ядра или вынесением каких-то функций в модули при отсутствии этих модулей. Наверняка при конфигурации ядра Если ошибки касаются модулей, то необходимо из каталога с исходными файлами ядра выполнить следующее:

```
make modules
make modules_install
```

Описаная выше процедура изменения конфигурации ядра не является обязательной, так как современные дистрибутивы распространяются максимально универсальными, но из-за этого ядро может оказаться «перегруженным» ненужными для конкретных задач возможностями. Поэтому, для экономии ресурсов эта процедура желательна.

2.4 Компиляция ядра и подготовка загрузочной дискеты для клиентов.

Компиляция ядра производится также как и для сервера в предыдущем пункте с некоторыми отличиями в конфигурации. В ядро бездисковой рабочей станции желательно включить драйвера большого количества сетевых карт, особенно тех типов, которые будут установлены на рабочих станциях. В нашем случае пока неизвестно, какие сетевые карты будут использоваться, но известно, что тип их будет либо 10Mb/s, либо 100Mb/s; для шины ISA или PCI, под неэкранированную витую пару (UTP) либо тонкий коаксиальный кабель. *Обязательно* надо включить опции `CONFIG_ROOT_NFS`, `CONFIG_RNFS_BOOTP`, `CONFIG_RNFS_RARP`, также понадобится включенная в ядро (не ввиде модуля) поддержка сетевой файловой системы NFS.

Модули BOOTP и RARP необходимы для идентификации бездискового клиента в сети, т.е. получение IP-адреса, адреса сервера, маски подсети. После загрузки ядро ищет поддерживаемые устройства, и если поддержка необходимых сетевых карт включена, то оно должно обнаружить их и сопоставить с интерфейсами eth0, eth1 и т.д., в зависимости от количества установленных сетевых карт. После этого в сеть, через интерфейс eth0 (в моем случае) передаются специальные сигналы BOOTP и RARP, смысл которых заключается в запросе «кто я». На сервере должна быть запущена программа-сервис, которая слушала бы такие запросы в сети и посылала бы ответы.

Ядро бездискового клиента должно быть самодостаточным до момента выделения ему файловых систем nfs. Поэтому, все необходимые для загрузки драйверы, должны быть «прошиты» в ядро, т.е. должны быть включены в ядро непосредственно, а не в виде модулей. Особое внимание следует обратить на следующие пункты:

- `Networking options/IP: Kernel level autoconfiguration`
  `[*]  RARP support`
  Конфигурация сетевого интерфейса (ip-адрес) выполняется через RARP.

- `Networking options/IP: Reverse ARP`
  Этот пункт отвечает за поддержку ядром системы RARP, по умолчанию этот пункт отмечен как модуль, т.е. эта функция подключается как модуль (`rarp.o`). Для бездискового клиента эта функция

должна быть включена в ядро непосредственно (значок "*" напротив).

- `Network device support/Ethernet (10 or 100Mbit)` — драйверы сетевых карт.
- `Filesystems/Network filesystems/NFS filesystem support`
- `Filesystems/Network filesystems/Root filesystem on NFS`
- `Block devices/Network block device support`

Множество остальных пунктов можно просто отключить за ненадобностью, но с некоторыми разделами надо быть осторожным (см. Kernel-HOWTO). Здесь можно дать простую рекомендацию: «Если вы не уверены в чем-то, то не изменяйте настроек по умолчанию». Но все же, таких пунктов не много, т.к. большинство разделов имеет ясные названия и иерархическую организацию. Так, вряд ли понадобится поддержка 1000Mb, оптоволоконных или радио сетевых адаптеров. Также вряд ли понадобится поддержка таких сетевых архитектур как AppleTalk, ArcNet, TokenRing и пр. Безболезненно можно отказаться от всех блочных устройств, относящихся к жестким дискам разных интерфейсов, ленточных устройств (необходимо лишь оставить гибкие диски); поддержки звука, видео, джойстиков и пр.

При конфигурации модулей для ядра необходимо всегда помнить, что эти модули будут доступны только лишь после монтирования удаленных файловых систем NFS, поэтому те, которые потребуются до этого момента (см. `man init`, `/usr/share/doc/initscripts-*.*`) необходимо включить внутрь ядра.

Полученный после сборки образ ядра (файл `/usr/src/arch/boot/bzImage`) теперь необходимо переписать на загрузочную дискету, но не просто скопировать, а переписать как образ, т.к. внутри него содержится boot-сектор, базовые устройства и прочие необходимые данные. По умолчанию в системе linux дисководом считается устройство `/dev/fd0`. Поэтому команда записи образа на дискету дожна быть следующей:

```
dd if=bzImage of=/dev/fd0
```

По умолчанию, после загрузки с полученной дискеты, ядро будет искать корневую файловую систему на том устройстве, с которого оно было скомпилировано (например, /dev/hda1). Но нам надо, чтобы система искала ее в сети, через NFS. Для этого нам необходимо создать устройство, на котором ядро стало бы искать коневую файловую систему и затем указать на него:

```
mknod /dev/nfsroot b 0 255
rdev /dev/fd0 /dev/nfsroot
```

Если делать несколько загрузочных дискет, то удобно сначала изменить сам образ, а потом просто копировать на дискеты:

```
rdev bzImage /dev/nfsroot
dd if=bzImage of=/dev/fd0
```

Если все было сделано правильно, то конфигурация клиента на этом заканчивается, далее вся конфигурация будет касаться только лишь сервера.

2.5 Конфигурация сервера.

В этом разделе будет описана настройка небходимых программ-сервисов. Они должны обеспечивать

2.5.1 Настройка системного окружения и сервисов.

В этом разделе будет описана настройка небходимых программ-сервисов. Они должны обеспечивать работу сервисов, требуемых для работы бездисковых клиентов и работы коммуникационных библиотек. Кроме тех, которые будут перечислены, в системе могут присутствовать еще множество других сервисов, выполняющие служебные и различные дополнительные функции. Нам потребуются четыре: DNS и сервис named, сервис rarpd, nfsd, rshd.

Выделение имени и адреса хосту.

Для этого достаточно изменить или создать файл `/etc/sysconfig/network` примерно следующего содержания:

```
NETWORKING=yes
HOSTNAME=node1.cluster.ptz.ru
```

Настройка сервиса named.

Для старта этого сервиса достаточно в каталоге `/etc/rc.d/rc.3` переименовать файл `K$named` в `S$named`, где `$` это число от 00 до 99. Если по умолчанию установлен runlevel 5 (загрузка системы заканчивается стартом Xfree86) переименовывать файл надо в каталоге `/etc/rc.d/rc.5` Дело в том, что призагрузке система входит на так называемый runlevel, номер которого указан в `/etc/inittab` (`id:3:initdefault:` или `id:5:initdefault:`). А скрипт tpcom/etc/rc.d/rc из каталога `rc$.d`

выбирает файлы начинающиеся с S (start), поэтому для старта сервиса достаточно переименовать его файл-ссылку.

Конфигурационными файлами для named являются /etc/hosts /etc/resolv.conf В файл /etc/hosts заносятся ip-адреса, полные имена хостов и короткие имена без доменной части. Например, в нашем случае, имя домена cluster.ptz.ru а имена хостов - node1, node2, node3 и т.д., тогда файл /etc/hosts должен выглядеть следующим образом:

```
127.0.0.1 localhost loopback
192.168.10.11 node1.cluster.ptz.ru node1
192.168.10.12 node2.cluster.ptz.ru node2
192.168.10.13 node3.cluster.ptz.ru node3
192.168.10.14 node4.cluster.ptz.ru node4
192.168.10.15 node5.cluster.ptz.ru node5
192.168.10.16 node6.cluster.ptz.ru node6
192.168.10.17 node7.cluster.ptz.ru node7
192.168.10.18 node8.cluster.ptz.ru node8
```

Адреса, использующиеся здесь, 192.168.0.0/255.255.0.0 а также адреса 10.0.0.0/8, являются специальными адресами, предназначенными для частных сетей и поэтому могут использоваться здесь беспрепятственно (RFC 1918 http://www.alternic.net/nic/rfcs/1900/rfc1918.txt.html).
Файл /etc/resolv.conf:

```
search cluster.ptz.ru
nameserver 127.0.0.1
nameserver 192.168.10.11
```

Настройка сервиса nfsd.

Этот сервис обеспечивает NFS-подключение локальных файловых систем для удаленных хостам. Нам он необходим, т.к. через него бездисковые клиенты будут получать доступ к своим корневым файловым системам. Для автоматического запуска его в процессе загрузки достаточно переименовать файл /etc/rc.d/rc$.d/K**nfs в S**nfs ($ —номер текущего runlevel, ** — порядковый загрузочный номер). Конфигурационным файлом для nfsd является /etc/exports, в нем приводятся сведения кому, как и к чему можно предоставлять доступ (man expotrs, man nfsd). В моем случае этот файл выглядел так:

```
/tftpboot/192.168.10.12/ node2(rw,no_root_squash)
/usr  node2(ro,no_root_squash)
/bin  node2(ro,no_root_squash)
/sbin node2(ro,no_root_squash)
/lib  node2(ro,no_root_squash)
```

```
/home node2(rw,no_root_squash)
/tftpboot/192.168.10.13/ node3(rw,no_root_squash)
/usr  node3(ro,no_root_squash)
/bin  node3(ro,no_root_squash)
/sbin node3(ro,no_root_squash)
/lib  node3(ro,no_root_squash)
/home node3(rw,no_root_squash)
...
```

Для хоста node2.cluster.ptz.ru (192.168.10.12) выделяются несколько каталогов сервера: `/tftpboot/192.168.10.12/` — корневая файловая система для 192.168.10.12 в режиме чтение/запись, `/lib` — необходимые общие библиотеки в режиме только для чтения, `/usr`, `/sbin`, `/bin` и `/home`. Далее, процедура записи в файл `/etc/exports` будет автоматизирована, она будет производиться скрпитом клонирующим файловые системы клиентов из шаблона. Здесь необходимо остановиться на одном важном обстоятельстве. В этом варианте конфигурации сервиса nfsd NFS-клиенты идентифицированы по своим коротким именам, а не по IP-адресам, поэтому nfsd будет пытаться найти адрес клиента с помощью программы resolver. При этом, nfsd не читает содержимое файла `/etc/hosts.conf`, так как этот файл является конфигурационным для resolver'a. Сам resolver в свою очередь, является частью сервиса named, поэтому для такой схемы конфигурации сервис named необходим. Но без него можно обойтись, если в файле `/etc/exports` вместо имен клиентов указать их IP-адреса.

Сервис RARP.

Сервис RARP не является самостоятельной программой как, например, named, а входит в состав ядра либо явно, либо в виде модуля `rarp.o`. Он также может запускаться при старте системы автоматически. В RH7.0 такой сервис не предусмотрен по умолчанию, поэтому необходимо будет подключить модуль и сделать скрипт стартующий при запуске системы. В стандартном ядре RH7.0 сервис rarp предусмотрен в виде модуля и, если ядро для сервера не изменялось, то этот модуль необходимо «подгрузить» (см. п. 2.3). Для этого достаточно в файл `/etc/modules.conf` добавить строку `alias rarp-ident rarp`. При следующей перезагрузке модуль rarp будет установлен автоматически. Для старта самого сервиса для клиентов, необходимо «добавить» их описания. Это можно сделать используя стандартные механизмы UNIX Init V, использующиеся в linux. Можно создать файл `/etc/rc.d/init.d/rarp` и сделать на него символическую ссылку:

```
cd /etc/rc.d/init.d
touch rarp
```

```
chmod 755 rarp
cd /etc/rc.d/rc3.d           # в моем случае runlevel=3
ln -s ../init.d/rarp S64rarp # вместо 64 может быть
                             # другое число
```

Структура этого файла проста, он последовательно выполняет команды вида:

```
rarp -s ip-addr X:X:X:X:X:X
```

ip-addr — это ip-адрес, который будет присвоен интерфейсу (сетевой карте — обычно `eth0`) клиента, аппаратный номер которого X:X:X:X:X:X. Аппаратный номер сетевой карты уникален, стандарт этой нумерации поддерживается всеми производителями сетевых карт. Изменение этого файла впоследствии тоже будет автоматически выполняться скриптом клонирования клиентов.

Сервис Remote Shell - rshd.

С помощью rhs можно работать с компьютером удаленно, работать с файлами, запускать программы на удаленной машине. Этот сервис по умолчанию необходим для коммуникационных библиотек MPI, входящих в состав пакета MPICH (MPICHameleon). Также с помощью rsh удобно управлять бездисковыми клиентами, особенно если у них отсутствуют монитор, видеокарта и даже клавиатура. Наша задача здесь состоит в том, чтобы этот сервис допускал управление с некоторых машин, не запрашивая ни логина, ни пароля. Проверить это можно попытавшись выполнить какую-либо команду на интересующем нас хосте, например:

```
rsh node2 uname -a
```

Если результатом этой команды будет `Permission denied`, то это будет означать, удаленный компьютер запрашивает пароль. Для определения правил доступа программа-сервис rshd использует файлы `/etc/hosts.equiv` и `/etc/pam.d/rshd`. В установке по умолчанию файл `/etc/hosts.equiv` отсутствует потому, что он определяет так называемые «дружественные» хосты и негативно сказывается на общей безопасности системы. В нашем случае это не очень важно, т.к. кластер практически замкнут и не работает постоянно, к тому же работать будет лишь ограниченный круг пользователей. Файл `/etc/hosts.equiv` необходимо создать и внести в него следующие записи:

```
+localhost
+node1.cluster.ptz.ru
```

Важно, чтобы такой же файл присутствовал и в *файловых системах клиентов*, ведь нам необходимо, чтобы клиенты безусловно подчинялись командам с сервера.

Кроме указания имен «дружественных» хостов в файле `/etc/hosts.equiv` Сервис rsh использует еще и другую систему для регулирования прав доступа — PAM (Plugable Authentication Modules) и правила эти описываются в файле `/etc/pam.d/rshd`:

```
#auth required /lib/security/pam_rhosts_auth.so
auth required /lib/security/pam_nologin.so
account required /lib/security/pam_stack.so service=system-auth
session required /lib/security/pam_stack.so service=system-auth
```

Здесь необходимо закомментировать первую строку. Эти файлы должны быть такими же и у клиентов, иначе с сервера невозможно будет управлять рабочими станциями напрямую, без передачи пароля.

В состав стандартной поставки входит еще множество сервисов, которые не нужны и просто занимают ресурсы. Например, можно отключить сервис ssh (Secure Shell), т.к. это более защищенная замена rsh, вероятней всего не понадобится сервис httpd (Web-сервер Apache). Все ненужные сервисы можно не загружать при старте, для этого достаточно переименовать соответствующие файлы `S*` в `K*` в каталоге `/etc/rc.d/rc$.d`.

2.6 Создание файловых систем для клиентов.

Для каждого клиента необходимо выделить свою корневую файловую систему на сервере, в которой должны находиться конфигурационные файлы этого клиента, файлы-устройства, необходимый минимум программ и библиотек. Это лишь необходимый минимум для загрузки системы, но ведь кроме этого нам необходимо, чтобы клиенты имели доступ к разрабатываемым программам, библиотекам, специфичным для параллельного программирования. Все это занимает довольно много места на диске, и я предполагаю, что файловую систему каждого клиента можно будет ужать до 50 Mb. Но планируется подключать 7-10 рабочих станций и в сумме это может занять до 500 Mb.

Авторы серии Beowulf - HOWTO разработали методику позволяющую сильно сократить требуемое дисковое пространство сервера. Легко заметить, что большинство файлов, требующихся в файловой системе каждого бездискового клиента, одинаковы для каждого из них и не требуют доступа на запись. Для экономии места на диске такие файлы можно сделать общими для всех клиентов копируя не сам файл, а жесткую ссылку на него. Внешне такая ссылка выглядит как отдельная копия файла и использующие ее программы также будут воспринимать как свою копию файла, но сама ссылка занимает очень мало места на диске и, поэтому позволяет сократить объем клиентской файловой системы на порядок. Остаются лишь несколько файлов, уникальных для каждого кли-

ента: /etc/fstab, /etc/mtab, /etc/sysconfig/network, каталоги /var/lock/subsys, /var/run и т.п.

По такой технологии файловуюсистему можно ужать до 5-10 Mb. Кроме самой методики авторы разработали и средства ее автоматизации. Был создан скрипт, формирующий каталог-шаблон для файловых систем клиентов, в котором содержится физическая копия файловой системы, но без некоторых необходимых файлов (см. выше). Другой скрипт (adcn) создает каталоги для клиента, полностью повторяя структуру каталога-шаблона и заполняет их жесткими ссылками на соответствующие файлы шаблона.

К недостатку можно отнести то, что все эти средства были разработаны для RH6.2 и они не полностью совместимы с RH7.0 Например, в RH7.0 немного поменялась структура каталога /etc, все инициализационные скрипты обращаются теперь не к /etc/rc.d/init.d, а к /etc/init.d, которая является символической ссылкой на /etc/rc.d/init.d. Поэтому, для нормального функционирования такая ссылка в файловой системе клиента необходима, а скрипты создания этих файловых систем не делают их. Поэтому необходимо изменить этот скрипт или скопировать эти ссылки вручную.

Созданную файловую систему скриптом sdct (по умолчанию она находится в каталоге (/tftpboot/Template)) необходимо поправить. В системе устанавливаются немного «кривые» shutdown-скрипты. Они не расчитаны на то, чтобы корневой каталог был nfs-разделом, при остановке системы эти скрипты пытаются отключить все смонтированные nfs-файловые системы вместе со всеми дополнительными. Но ведь так как один из них является корневым для бездискового клиента, то они должны отмонтироваться одними из самых последних, поэтому, попытка отмонтировать корневую файловую систему раньше времени заканчивается неудачей, выдается сообщение об ошибке и процесс остановки системы прекращается.

Для ясности происходящего, здесь необходимо остановиться на процессе выделения корневой файловой системы для бездискового клиента. При старте системы клиента с дискеты, его ядро, после собственной инициализации, настраивает сетевую карту как физическое устройство и, затем, передает через нее в сеть запрос RARP. Получив ответ от сервера, ядро клиента настраивает сетевую карту как сетевой интерфейс (присваивает ему ip-адрес). Далее, ядро посылает запрос на монтирование корневой файловой системы nfs в режиме только для чтения. Ядро, по умолчанию, пытатся найти его в каталоге /tftpboot/%i на сервере (%i — ip-адрес, выделенный клиенту). Этот каталог прописан в файле fs/nfs/nfsroot.c в исходных текстах ядра и его легко можно изменить. Надо заметить здесь, что на этом этапе информация из файла /etc/fstab никак не учитывается. После монтирования, считывается файл /sbin/init (так он представляется клиенту, на сервере этот файл

находится в каталоге /tftpboot/%i/sbin/init) и скрипты инициализации, находящиеся в /etc. В режим чтение-запись корневая файловая система переводится этими скриптами после, но ведь у нас еще есть файловые системы nfs разделенные непосредственно с сервером: /usr, /lib, /home и др. (см. стр. 13). Непосредственно в корневом каталоге эти каталоги пустые (для экономии места) поэтому если их не подключить, то они так и останутся пустыми и маловероятно, что система клиента загрузится. Подключение этих разделов производится при старте сервиса nfs, который вызывается по стандартной схеме UNIX System V. Понятно, что этот сервис должен вызываться как можно раньше. Если все эти процедуры выполнились успешно, то система должна загрузиться без особых проблем.

При остановке системы init при помощи своих скриптов пытается выполнить эти действия симметрично, но получается, что корневая файловая система, в которую включены подключенные позже каталоги /usr, /lib, /home, размонтируется раньше времени.

В [3] дается рекомендация по этому поводу, они предлагают просто не отмонтировать nfs-разделы перед остановкой или перезагрузкой бездискового клиента. Для этого необходимо исправить скрипты в системе инициализации клиентов: /etc/init.d/reboot и /etc/init.d/netfs. Необходимо заострить внимание на том, что это путь для клиента, на сервере они находятся в каталоге /tftpboot/Template, а серверные скрипты лучше не изменять.

Файл /etc/init.d/netfs ответственнен за запуск и остановку сарвиса nfs, именно он, руководствуясь /etc/fstab и /etc/mtab подключает и отклчает nfs-разделы. При остановке системы, этот скрипт первым пытается отключить их и необходимо изменить его так, чтобы пресечь такие действия. Для этого достаточно обнулить один счетчик в цикле (см. файл):

```
...
stop)
   [ -n "$NFSMTAB" ] && {
sig=
# Warning! The following modification prevents execution
# of next do..done cycle.
# retry value was set to 0 (original value = 3).
# This modification needed to prevent root NFS partition
# to be unmounted when shutdown scripts are running!
# (for diskless NFS-Root clients)
retry=0
sleep 5
...
```

Файле /etc/init.d/reboot отключает все оставшиеся подключенными разделы (корневой и /proc в том числе). Его тоже надо изменить

так, чтобы он пропускал nfs-разделы и не размонтировал их. Здесь тоже файл правится довольно легко, надо к команде `umount` добавить еще один параметр `nonfs`.

```
...
while [ -n "$remaining" -a "$retry" -gt 0 ]
do
  if [ "$retry" -lt 3 ]; then
#   runcmd "Unmounting file systems (retry):
#                   "  umount -a -f -t noproc
    runcmd "Unmounting file systems (retry):
                    "  umount -a -f -t noproc,nonfs
    else
#     runcmd "Unmounting file systems:
#                   "  umount -a -f -t noproc
      runcmd "Unmounting file systems:
                    "  umount -a -f -t noproc,nonfs
  fi
  sleep 2
  remaining=`awk '!/(^#|proc|loopfs|^none|^\/dev\/root| \/ )/ {print $2}'
...
```

В таком варианте этот цикл будет выполняться 3 раза с паузой 2 секунды. Если это неудобно, то можно изменить его так, чтобы он выполнялся только один раз (`retry=1`, как и в предыдущем варианте)

... Продолжение следует ...

Список литературы

[1] Beowulf – HOWTO из LDP (Linux Documentation Project)

[2] Beowulf Installation And Administration – HOWTO из LDP (Linux Documentation Project)

[3] NFS-HOWTO из проекта LDP (Linux Documentation Project)