Кластерная файловая система OCFS2

mkondrin
из hppi.troitsk.ru

22 сентября 2011 г.

Аннотация

Собираем SAN сервер своими руками

Сетевая файловая система (NFS,CIFS/SMB и др.) безусловно мощное и надёжное средство, когда нужно предоставить общий доступ к файловой системе группе компьютеров. Но бывают ситуации, когда это решение не срабатывает. Причина сбоев довольно проста: на самом деле сетевые файловые системы – это “не вполне” файловые системы, а просто один из способов, который позволяет натянуть доступ через сеть к каким-то данных на семантику локальной файловой системы. К сожалению, такие натяжки иногда оборачивается не так, как хотелось бы – где-то тонко, где-то толсто. С такой проблемой сталкиваются многие сетевые администраторы, когда перед ними возникают задачи виртуализации и кластеризации, что как правило упирается в вопрос в предоставлении единой файловой системы для всех операционных систем (будь это гостевой операционные системы в виртуальном окружении или локальные операционные системы внутри сетевого кластера). Разумеется, в такой широкой постановке, это вопрос вряд-ли имеет универсальное решение, но в том случае, если нас интересуют только операционные системы на базе Linux, то параллельная POSIX-совместимая файловая система для группы компьютеров может быть построена с помощью iSCSI SAN-сервера и разделяемой кластерной (shared-disk) файловой системы OCFS2 (Oracle Clustered File System). В качестве одной из моделей, где такая система может быть внедрена, можно представить себе высокодоступный почтовый IMAP/POP3 сервис, где, собственно, обслуживание подключений по IMAP/POP3 выполняется группой серверов с настроенной балансировкой нагрузки, а само хранилище электронной почты вынесено на отдельный файловый сервер, который используют параллельно несколько почтовых серверов. Это только один из примеров (подсказанный названием одного из типов форматирования томов OCFS2), но на самом деле это решение может оказаться полезным при создании отказоустойчивых кластеров (и тем самым заменить более традиционное решение DRBD+heartbeat [1, 2]), серверов файловых баз данных и даже небольших высокопроизводительных вычислительных кластеров.

1 Что нам понадобится для этого?

Совершенно точно, что с одной операционной системой поэкспериментировать не получится. Я буду разбирать случай, что у вас есть по крайней мере два компьютера (один сервер, второй клиент) объединённых в одну сеть с пропускной способностью как минимум 1 Гб/с. Можно заменить это одним компьютером с виртуальной машиной внутри, где хозяин (host) выступает в качестве сервера, а гостевая система (guest) – клиента, это также должно сработать. Также предполагается, что на обеих системах установлен и работает Linux с “ванильным” ядром версии в диапазоне 2.6.27-36, и все вопросы с настройкой сетевого обмена данными между этими системами уже решены. Под “ванильным” подразумевается ядро с сайта www.kernel.org без дополнительных патчей. Поскольку нам в дальнейшем потребуется латать ядро, то это требование может оказаться (а может и не оказаться, если

нужные нам патчи наложатся без проблем) весьма критичным. Также лучше всего заранее решить проблемы с оптимизацией сетевого трафика, как минимум крайне желательно настроить jumbo-frames, так чтобы реальная скорость обмена между клиентом и сервером, регистрируемая программами типа netperf (или же просто netcat) всё же более или менее походила на заявленные 1 гигабит в секунду. Можно, конечно же, добиться приемлемой производительности и на более медленных подключениях, но это скорее всего потребует дополнительной настройки как на стороне сервера так и на стороне клиентов, а учитывая, что цена гигабитных сетевых карт и 5е-кабелей не слишком бьёт по карману, то стоит ли очинка выделки?

2 SAN сервер своими руками

Центральным звеном нашей системы будет iSCSI сервер, построенный на базе компьютера с Linux на борту. Несмотря на сходство названий, iSCSI не имеет отношения к известной фирме по производству мобильников, а собственно обозначает протокол передачи команд SCSI (что само по себе расшифровывается как Small Computer System Interface) через сеть (internet). Сама по себе система команд SCSI, появившаяся ещё на заре компьютерной эпохи (в 1981 году как подсказывает Википедия [3], собственно поэтому слово Small в названии не должно вводить в заблуждение, поскольку в то время масштабы были совсем другими), оказалась очень удобна для доступа и хранения данных, и в настоящее время SCSI так или иначе присутствует в широком диапазоне интерфейсов доступа к дискам – начиная от промышленных систем, основанных на Fiber Channel, до обычной USB-флешки. Так что iSCSI – это просто ещё один интерфейс SCSI.

Несколько слов о названиях. Поскольку SCSI возник давно, ещё до возникновения клиент-серверной парадигмы, то терминология используемая там сейчас кажется несколько необычной. Сервер в терминологии SCSI называется целевой системой (target), клиент – инициатором (initiator), поскольку именно он начинает обмен данными или устанавливает сессию взаимодействия (session) с целевой системой. Целевая система может включать в себя несколько логических устройств (для простоты можно считать их дисками), идентифицируемых своим LUN (logic unit number). Хотя Linux начиная с версии 2.2.0 (1999 год прошлого века, считая с даты рождения модуля SCSI-generic) позволяет подключать SCSI-устройства, но вот для решения обратной задачи – использовать Linux компьютер в качестве целевой системы, необходимые условия появились сравнительно недавно, и само их появление происходило не вполне гладко.

> Впервые модуль STGT, представляющий, если можно так сказать, абстрактный драйвер целевого SCSI устройства под названием STGT появился в официальном ядре Линукс в 2006 году. Его работа через интерфейс iSCSI сильно зависела от пользовательского процесса ietd (iSCSI enterprise target daemon). К сожалению эта связка вызывала многочисленные нарекания пользователей своей медлительностью и ненадёжностью, что во многом и сформировало отношение системных администраторов к такого рода деятельности, что, мол, это годится только для тренировок и тестирования, а в промышленных решениях необходимо большое железо. Кто бы спорил! Но если взять в рассчёты цену большого железа, то это автоматически закрывает для технологии iSCSI многие области. Скажем, озвученная недавно на одном из популярных интернет-форумов цена за аппаратное решение “начального уровня за 150k” достаточно тяжела для академических и образовательных учреждений. Однако, уже в 2008 году Владислав Болховитин предложил альтернативный абстрактный драйвер SCST вместе с ядерным модулем, обслуживающем iSCSI интерфейс iscsi-scst, который оказался существенно быстрее имеющегося. За прошедшее с тех пор время это решение приобрело значительное количество пользователей и прошло обкатку в промышленных условиях. Долгое время неявно предполагалось, что этот драйвер заменит STGT в

ядре Линукс, но команда разработчиков Линукса, славящаяся своими, мягко говоря, оригинальными стратегическими решениями (можно вспомнить историю с Reiser4 или планировщиком Кона Коливаса) отличилась и на этот раз. В декабре 2010 года было решено, что в основную ветку ядра 2.6.38 наряду с STGT будет вставлен ещё и мало кому известный модуль LIO, впоследствии переименованный в TCM. Из последующего потом бурного обсуждения в списке рассылки разработчиков ядра (краткое изложение приведено здесь [4]), неискушённый наблюдатель мог бы сделать вывод, что единственной претензией, высказанной разработчику SCST, было его неумение работать в команде и его недостаточная отзывчивость к просьбам облечённых властью разработчиков по внесению необходимых на их взгляд изменений в интерфейс модуля. Это и привело к тому, что место SCST в основной ветке ядра занял модуль LIO/TCM.

Как бы то ни было, пока что разработка SCST продолжается как и прежде вне основной ветки ядра. Можно также упомянуть тот факт, что в апреле 2011 года, т.е. уже позже описываемых событий, дистрибутив OpenFiler (он рассматривался ранее на страницах “Системного администратора” [5]), ориентированный на создание сетевых файловых хранилищ, выпустил новую версию с ядром 2.6.32, куда были включены патчи и модули scst и iscsi-scst.

Для наших целей создания самодельного SAN-сервера мы воспользуемся пакетами Владислава Болховитина SCST [6] и iSCSI-SCST [7], поскольку для конечного администратора и пользователя решающим аргументом в их пользу будут их быстродействие и отлаженность. Можно также отметить, что если у вас имеется соответствующее оборудование, то на томже сайте [7] можно скачать драйверы для FiberChannel карт от Qlogic, Marvell и Emulex, что позволяет собрать уже вполне “взрослый” и более быстрый SAN сервер с подключением через FiberChananel. Кроме того рекомендуется также скачать пакет Scstadmin [8], который представляет из себя perl-скрипт, существенно упрощающий администрирование и настройку iSCSI сервера. Как уже упоминалось ранее, необходимо иметь исходники ванильного ядра Линукс. Мною (по ряду причин, включая необходимость наложения сторонних патчей) была выбрана версия 2.6.30, где ядерные заплатки от scst и iscsi-scst прекрасно ужились с этими сторонними патчами. Если мы собираемся устанавливать модифицированное ядро и дополнительные модули на несколько компьютеров, то желательно распаковать исходники ядра в директорию, которую можно будет экспортировать (допустим, через NFS) на все остальные компьютеры, что значительно упростит установку. Т.е. предположим, что исходный код ядра на одном из компьютеров распакован в директорию `/usr/src/linux` и эта директория видна на всех остальных компьютерах, которые мы собираемся включить в кластер, также под точкой монтирования `/usr/src`. В ту же директорию `/usr/src` распакуем пакеты scst и iscsi-scst (версий 2.0.0.1 и 2.0.0, являющиеся последними на текущий момент). После этого залатаем ядро такой последовательностью команд:

```
cd /src/linux
patch -p1 < /usr/src/scst-2.0.0.1/kernel/readahead-2.6.30.patch
patch -p1 < /usr/src/scst-2.0.0.1/kernel/readahead-context-2.6.30.patch
patch -p1 < /usr/src/scst-2.0.0.1/kernel/scst_exec_req_fifo-2.6.30.patch
patch -p1 < /usr/src/iscsi-scst-2.0.0/kernel/patches/put_page_callback-2.6.30.patch
```

Строго говоря, эти заплатки из пакетов SCST и iSCSI-SCST не являются обязательными, но существенно улучшают быстродействие. Вам только нужно будет подобрать версию патчей под свою версию ядра. После этого нужно будет сконфигурировать ядро. Поскольку вопрос конфигурации ядра сложный, включает в себя много нюансов, которые сложно подробно осветить в одной статье, но по-крайней мере нужно убедиться, что отмечены пункты касающиеся выбора как доступных, так и основного планировщиков ввода/вывода (см. Рис. 1), а также, что в ядре выбраны модули для OCFS2 (они входят в состав ванильного ядра, Рис. 2) и отмечен пункт, позволяющий использовать zero copy для iscsi (этот пункт меню добавлен заплаткой из пакта iscsi-scst, Рис. 3). После этого можно собрать ядро командой `make`, а после её успешного завершения установить ядро на каждую

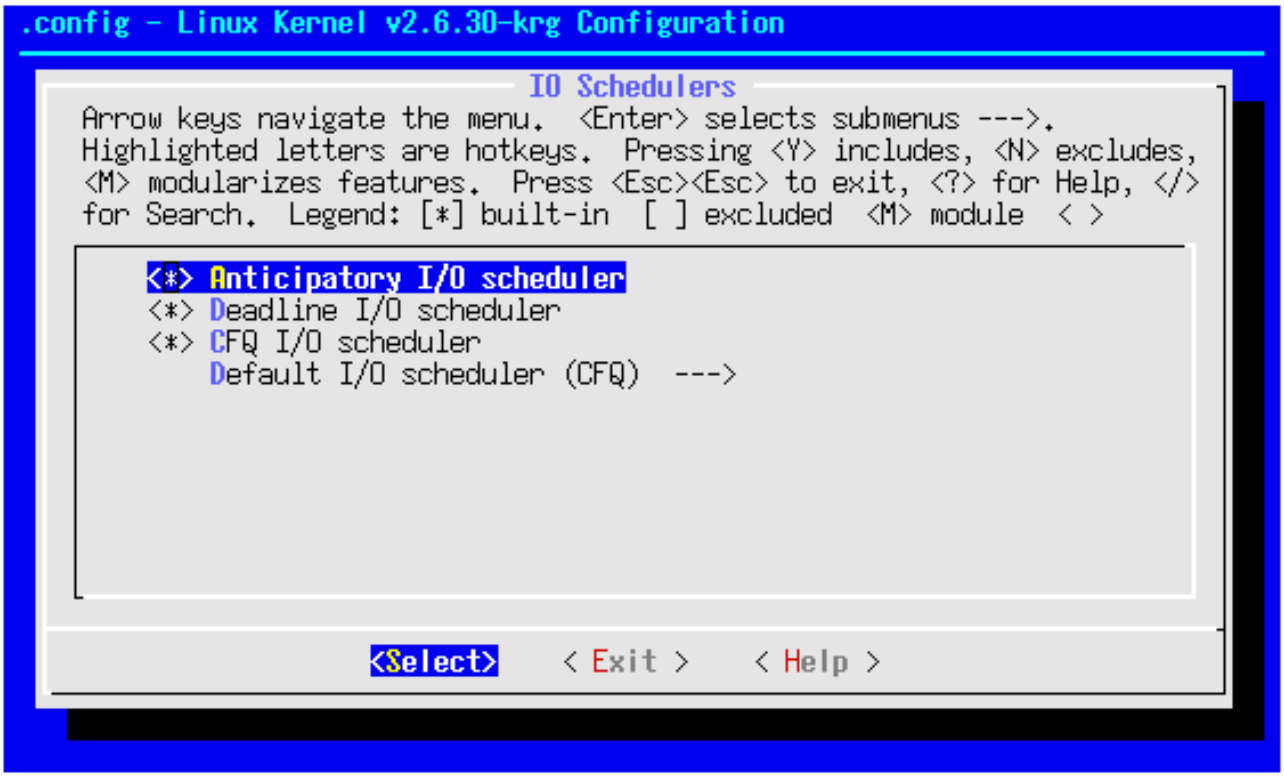

Рис. 1: Настройка планировщиков ввода-вывода ядра Linux (меню Enable the block layer -> IO Schedulers)

из операционных систем, войдя из-под неё в директорию `/usr/src/linux` и выполнив там серию команд `make install ; make modules_install`. Перегрузив компьютеры и загрузившись с новым ядром, можно приступать к установке модулей scst и iscsi-scst. Собственно особенно ничем от вышеописанной процедуры эта установка не отличается – выполняем make и make install последовательно в директориях, где распакованы исходники `/usr/src/scst-2.0.0.1` и `/usr/src/iscsi-scst-2.0.0`. Благодаря наличию директории `/lib/modules/{версия ядра}/build` путь к исходникам ядра и проверка имеющейся в наличии функциональности в процессе сборки проверяется автоматически. В итоге в директории `/lib/modules/$(uname -r)/extra/` мы должны получить два ядерных модуля iscsi-scst.ko и scst.ko, а также директорию `dev_handlers` с обработчиками тыловых (back-end) устройств, в которой нас будут интересовать модули `scst_vdisk.ko` и `scst_disk.ko`. Загрузим все эти модули последовательно:

```
modprobe scst
modprobe scst_disk
modprobe scst_vdisk
modprobe iscsi_scst
```

Если проблем с загрузкой не возникло, то таким образом всё готово к настройке сервера iSCSI. Сама по себе низкоуровневая конфигурация сервера представляет из себя запись параметров и команд в довольно разветвлённую директорию `/sys/kernel/scst_tgt`. Причём задача усложняется ещё и тем, что кое-где в этой директории попадаются специальные управляющие файлы под названием `mgmt`, в которые можно записывать команды вроде `clear` или `add_device` (полный набор команд можно прочитать в файлах README из пакетов SCST и iSCSI-SCST), в результате чего эта директория ветвится ещё больше, или же, наоборот, часть ветвей из неё пропадает. Я даже готов согласиться с разработчиками ядра Линукс, что интерфейс настройки SCST достаточно запутан, но, к счастью, можно воспользоваться программой `scstadmin`, которая использует один-единственный конфигурационный файл. Более того, этот файл и выглядит достаточно наглядно. Например, у меня он такой (`/etc/scst.conf`):

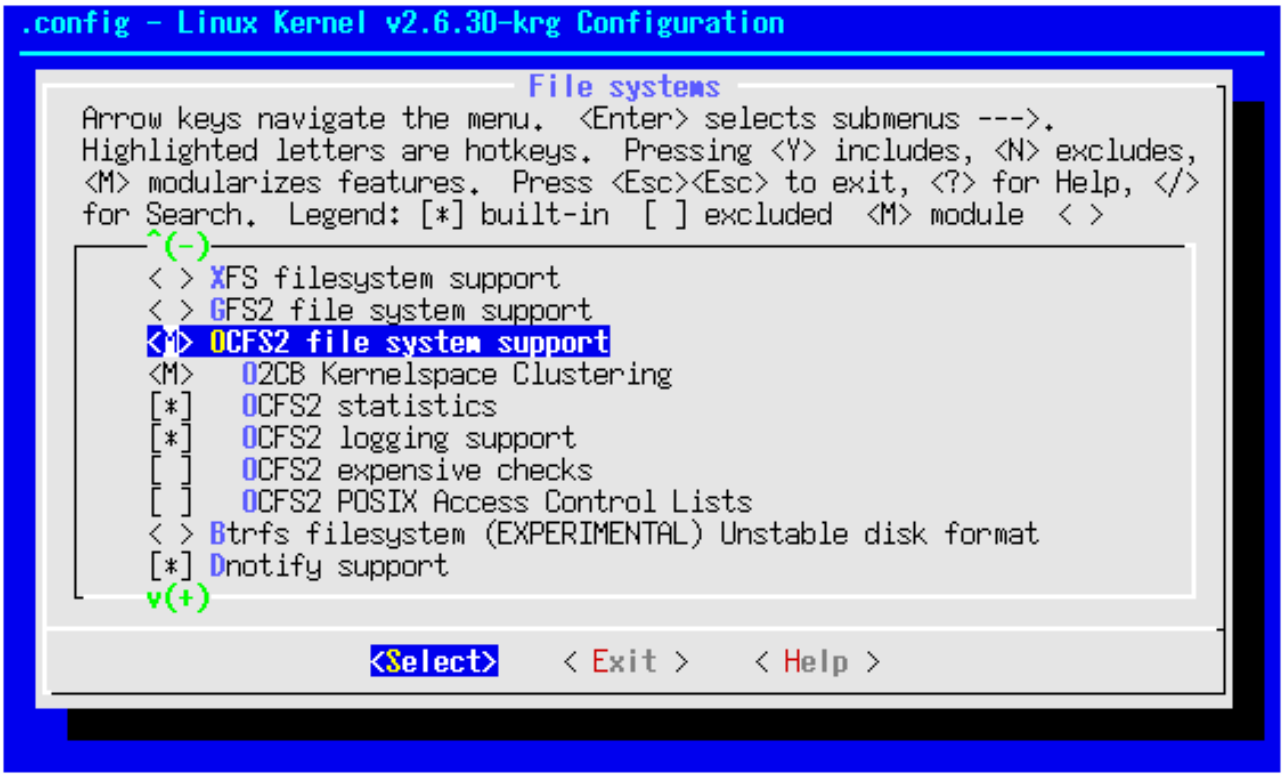

Рис. 2: Настройка кластерной файловой системы OCFS2 (меню File Systems)

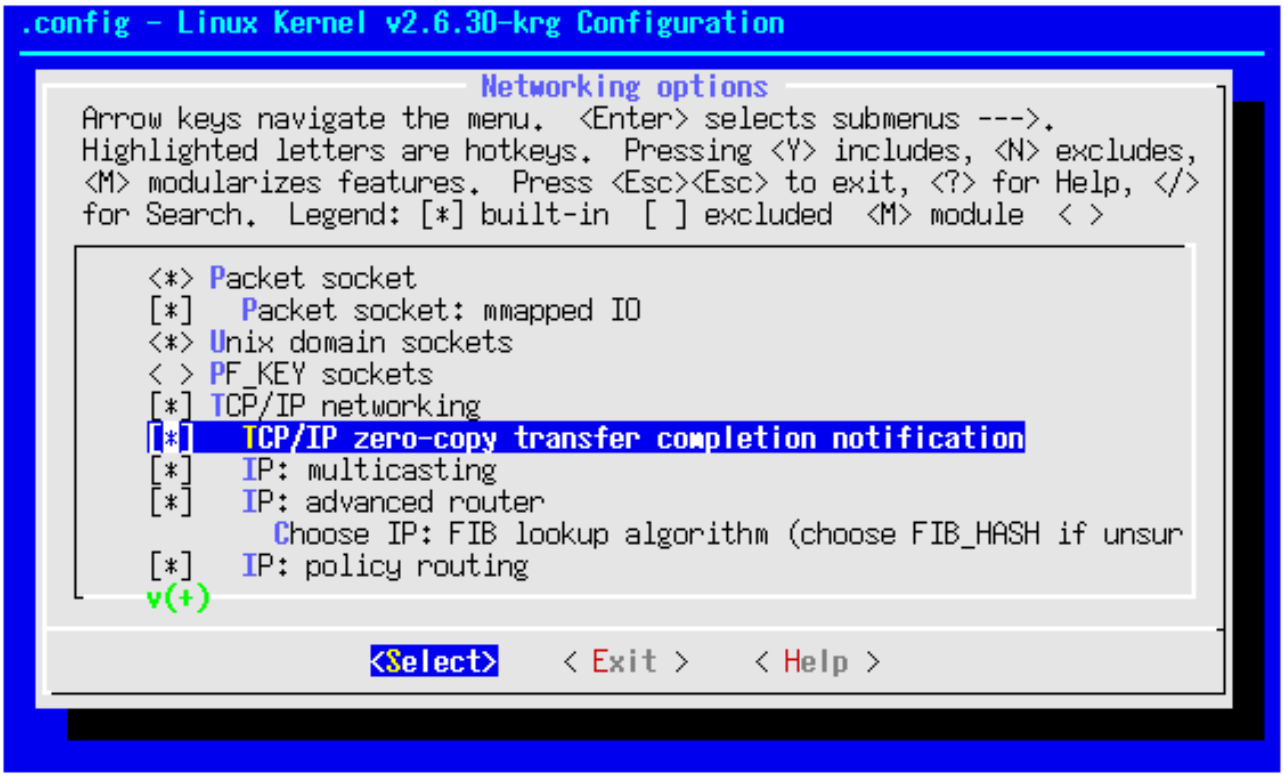

Рис. 3: Настройка TCP ZERO COPY для модуля iSCSI-SCST (меню Networking Support -> Neworking Options)

```
HANDLER vdisk_fileio {
        DEVICE disk01 {
               filename /dev/sda2
               blocksize 512
               threads_num 8
               threads_pool_type per_initiator
               nv_cache 1
        }
        DEVICE disk02 {
               filename /iscsi
               blocksize 512
               threads_num 8
               threads_pool_type per_initiator
               nv_cache 1
        }
}

TARGET_DRIVER iscsi {
              enabled 1
              TARGET iqn.2011-06.ru.example:dsk1 {
                     LUN 0 disk01
                     LUN 1 disk02
                     enabled 1
                     io_grouping_type never
              }

}
```

Здесь указано, что на сервере имеется всего одно целевое устройство (`iqn.2011-06.ru.example:dsk1`), управляемое драйвером `TARGET_DRIVER iscsi`, состоящее из двух дисков (с LUN 0 и 1). Имя целевого устройства с непривычки кажется довольно необычным, но на самом деле в случае iSCSI оно регламентируется двумя RFC3720-3721 [9, 10], согласно которым для обеспечения уникальности, имя должно начинаться со строки `iqn.` (расшифровывается как individual qualified name), затем идёт год и месяц создания устройства, доменное имя в обратной записи и после двоеточия – произвольное название целевого устройства. В качестве тыловых устройств я взял два различных дисковых устройства: в одном случае – это неотформатированный раздел (`/dev/sda2`), а во втором – обычный файл на корневом разделе, созданный командой `dd if=/dev/zero of=/iscsi bs=4096 count=100000`. В обеих случаях они обслуживаются одним и тем же модулем/драйвером `scst_vdisk` как файловые устройства (`HANDLER vdisk_fileio`). Это означает, что при доступе к ним со стороны инициаторов, целевым устройством будет использовано такая же процедура кэширования, как и при доступе к обычному файлу. Такое кэширование скорее полезно, поскольку ускоряет обмен данными, но и его можно отключить, если использовать тыловые устройства совместно с другими драйверами. Кроме того в настройках каждого из тыловых устройств указан параметр `nv_cache` (расшифровывается как non-volatile cache), который позволяет ещё более увеличить быстродействие в ущерб надёжности, игнорируя все команды по синхронизации кэша с дисковым устройством. Т.е. предполагается, что дисковый кэш по мере заполнения будет так или иначе сброшен на диск, и поскольку кэш у нас энерго-независим, то внезапное отключение питания на этот процесс никак не повлияет. Разумеется последнее предположение неверно, поэтому при использовании опции `nv_cache` нужно позаботиться о приобретении надёжного источника бесперебойного питания, который к тому же при отключении питания, взаимодействуя с NUT, может корректно завершить работу файлового сервера. Параметр `blocksize` устанавливает размер блока устройства видимый со стороны инициатора. Из общих соображений можно было бы сделать вывод, что для сетевых устройств желательно устанавливать размер блока побольше, скажем, 4096 или 8192 байт, но мне установить такой размер блока не удалось. Не знаю, связано ли это с каким-то конфликтом между размером блока устройства iscsi и тылового дискового накопителя и файловой системы, то ли это обусловлено собственными ограничением

OCFS2 или проблемами с железом, но при любом размере блока отличном от стандартного 512 байт, я наблюдал на инициаторе много чудес вплоть до падения системы. Если вы захотите поэкспериментировать с размером блока, то (следуя советам Болховитина из файла README) после изменения размеров блока сам файл или раздел устройства должен быть полностью затёрт нулями (например так: `cat /dev/zero > /dev/sda2`). Только после этого можно экспортировать его с новым размером блоков. Параметры `threads_num`, `threads_pool_type` и `io_grouping_type`, которые группируют I/O запросы от нескольких инициаторов, также установлены согласно рекомендациям из файла README, в частности значение `never` для последнего параметра предписано использовать в случае параллельного доступа нескольких инициаторов к одному разделу, что как раз подходит под типичное применение кластерных файловых систем.

Теперь всё готово для запуска файлового сервера. Полностью отказаться от демона в пользовательском пространстве Владиславу Болховитину не удалось, но этот демон играет вспомогательную роль, поскольку основная нагрузка всё же ложится на ядерный драйвер `iscsi_scst`. Так что в начале необходимо запустить этот вспомогательный демон `/usr/local/sbin/iscsi-scstd` и только после этого загрузить конфигурационный файл командой `scstadmin -noprompt -force -config /etc/scst.conf`. Параметры командной строки `-noprompt -force` означают, что изменение текущей конфигурации без дополнительного предупреждения пользователя следует проводить даже в том случае, если это приводит к переконфигурации уже существующего устройства (что может привести к потерям данных на нём, и поэтому по-умолчанию scstadmin такие команды игнорирует). Понятно, что эта команда не вполне безопасна, но вполне приемлема для тестирования системы. Если в итоге в директории `/sys/kernel/scst_tgt/targets/iscsi/` появится ветка `iqn.2011-06.ru.example:dsk1`, то можете себя поздравить – ваш самодельный SAN сервер заработал!

Быстродействие устройств во многом определяется размером и объёмом кэша. Как правило, увеличение производительности за счёт увеличения объёма кэша негативно влияет на надёжность, так что экспериментируя с размерами кэша всегда следует иметь в виду, каким объёмом данных вы готовы пожертвовать в случае, если что-то пойдёт не так.

Можно отключить использование кэша файловой системы, если вместо обработчика `vdisk_fileio` для неформатированного раздела использовать обработчик `HANDLER vdisk_blockio`, причём такую замену можно производить “наживую”, и она не должна приводить к повреждению данных на диске. Более того, драйвер SCST, позволяет использовать SCSI устройства напрямую, т.е. фактически непосредственно пересылать команды SCSI, полученные на сетевом интерфейсе, на тыловое SCSI устройство.

Например, вот как выглядит у меня на целевом устройстве листинг системной директории:

```
~# ls /sys/kernel/scst_tgt/devices/
0:0:0:0/  1:0:0:0/  disk01/  disk02/
```

`disk01` и `disk02` – это виртуальные scsi устройства созданные ранее. А вот два других вида `x:0:0:0` – это реальные SATA диски, и команда `lsscsi` подсказывает, что:

```
lsscsi
....
[0:0:0:0]    disk    ATA      ST3320613AS      SD22  -
[1:0:0:0]    cd/dvd  HL-DT-ST DVDRAM GH15F     EG00  -
```

где первое является системным жёстким диском (`/dev/sda`, где находится как системный корневой раздел, так и экспортированный под именем disk01 раздел /dev/sda2), а второе – SATA DVD-ROM, который я могу легко экспортировать через интерфейс iSCSI. Для этого мне потребуется только загрузить соответствующий драйвер

modprobe scst_cdrom и внести в конфигурационный файл /etc/scst.conf два исправления. Во-первых, добавить ещё один обработчик:

```
HANDLER dev_cdrom {
        DEVICE 1:0:0:0
}
```

и в разделе драйверов целевых устройств добавить ещё один раздел:

```
TARGET iqn.2011-06.ru.example:cdrom {
       LUN 0 1:0:0:0
       enabled 1
}
```

после чего перегрузить изменения с помощью команды scstadmin. Следует помнить, что нумерация LUN в каждом целевом устройстве должна начинаться с нуля!

Как только устройство iqn.2011-06.ru.example:cdrom импортировано в клиентскую систему, то там её можно будет использовать как обычный DVD-RW диск, к примеру такой последовательностью команд:

```
#cdrecord -scanbus
...
scsibus5:
        5,0,0   500) 'HL-DT-ST' 'DVDRAM GH15F    ' 'EG00' Removable CD-ROM
#cdrecord -eject dev=5,0,0
```

т.е. как локальный DVR-ROM. Таким же точно образом можно было бы предоставить доступ к SATA диску (драйвер scst_disk и обработчикHANDLER dev_disk), но у меня под рукой не нашлось свободного диска, чтобы поэкспериментировать с ним. Можно было бы ещё поспорить, будет ли скорость обмена с ним хуже или лучше, чем у виртуальных дисков, за счёт наличия или отсутствия кэширования, но в любом случае использовать программные RAID-устройства таким образом точно не получится. Но для SCSI-устройств такая возможность тоже существует.

3 Настройка клиентов

Экспортируемые iSCSI-диски могут быть размечены и использованы любыми операционными системами, но я ограничиваюсь рассмотрением только клиентов под управлением операционной системой Linux. Настройка Linux-клиентов существенно проще, чем настройка iSCSI-сервера, поскольку не требует обширной правки конфигурационных файлов и специального латания ядра. Тем не менее нам потребуются дополнительный модуль ядра вместе с несколькими user-space утилитами (типа, iscsiadm), которые обеспечивает доступ к накопителям через интерфейс iSCSI со стороны инициатора и распространяются в пакете Open-iSCSI [11]. Сам пакет нужно распаковать и собрать на клиентской системе традиционным способом make ; make install. Пакет совместим с ядром 2.6.30 даже с наложенными патчами от SCST и iSCSI-SCST, так что на инициаторе и файловом сервере можно использовать одну и ту же версию ядра.

Важно помнить, что в практически важных случаях, когда мы хотим смонтировать файловую систему установленную на целевом устройстве или же использовать это устройство в качестве файла подкачки, нельзя иметь одновременно на одной системе инициатор и целевое устройство. Это ограничение линуксового mmap(2), которое приводит к взаимоблокировкам (deadlocks) типа Out-of-Memory и проявляется не только в интерфейсе iSCSI, но и в других случаях, связанных с использованием сетевых носителей (взять, например, имеющийся в ядре Linux драйвер nbd). Кстати, нельзя также обойти эту проблему с помощью, так сказать, “циклического монтирования”, когда одна система экспортирует свой том файловой системы, при этом монтируя файловую систему, экспортированную другим

сервером, которые поступает зеркальным образом. Так что в любом случае для тестирования iSCSI, требуется как минимум две системы, расположенные либо на двух физически разных системах, либо, как уже говорилось, одну из них можно поместить в виртуальную машину.

После установки клиентского программного обеспечения, на каждом из клиентов подгрузим соответствующий ядерный модуль и запустим клиентский сетевой сервис:

```
modprobe iscsi_tcp
iscsid --initiatorname=/etc/iscsi/initiatorname.iscsi
```

предварительно создав конфигурационный файл с именем iSCSI-инициатора, например:

```
echo "InitiatorName=iqn.2011-06.ru.example:az" > /etc/iscsi/initiatorname.iscsi
```

Однако последняя команда необязательна, поскольку в противном случае все инициаторы всё равно будут видны на целевой системе под именами вида: `iqn.2005-03.org.open-iscsi:da43e54478a9` со случайной последовательностью после двоеточия.

Теперь всё готово для тестирования взаимодействия инициатора с целевой системой, для чего запросим с клиента у сервера набор имеющихся целевых устройств:

```
# iscsiadm -m discovery -t sendtargets  --portal 192.168.21.5
192.168.21.5:3260,1 iqn.2011-06.ru.example:cdrom
192.168.22.5:3260,1 iqn.2011-06.ru.example:cdrom
192.168.21.5:3260,1 iqn.2011-06.ru.example:dsk1
192.168.22.5:3260,1 iqn.2011-06.ru.example:dsk1
```

Поскольку на целевой системе у меня сконфигурированы два сетевых интерфейса (или портала/portal в терминологии iSCSI) с адресами 192.168.21.5 и 192.168.22.5, доступ к которым никак не ограничен в конфигурационном файле `scst.conf`, то со стороны клиента каждое из устройств присутствует в двух экземплярах. Любое из этих устройств может быть подключено к инициатору:

```
iscsiadm -m node --target "iqn.2011-06.ru.example:dsk1" --portal "192.168.21.5:3260" --
login
```

3260 – это TCP порт по-умолчанию для интерфейса iSCSI. Если подключение прошло успешно, то на инициаторе команда `dmesg` покажет сообщения наподобие вот этих:

```
[  460.893455] scsi 2:0:0:0: Direct-Access     SCST_FIO disk01            200 PQ: 0 ANSI: 5
[  460.893883] sd 2:0:0:0: Attached scsi generic sg2 type 0
[  460.894483] sd 2:0:0:0: [sdb] 195318270 512-byte hardware sectors: (100 GB/93.1 GiB)
[  460.894609] sd 2:0:0:0: [sdb] Write Protect is off
[  460.894614] sd 2:0:0:0: [sdb] Mode Sense: 83 00 10 08
[  460.894778] scsi 2:0:0:1: Direct-Access     SCST_FIO disk02            200 PQ: 0 ANSI: 5
[  460.895409] sd 2:0:0:0: [sdb] Write cache: disabled, read cache: enabled, supports DPO and FUA
[  460.895584] sd 2:0:0:1: Attached scsi generic sg3 type 0
[  460.896086] sd 2:0:0:1: [sdc] 2000000 512-byte hardware sectors: (1.02 GB/976 MiB)
[  460.896125]  sdb:<5>sd 2:0:0:1: [sdc] Write Protect is off
[  460.898482] sd 2:0:0:1: [sdc] Mode Sense: 83 00 10 08
[  460.898768] sd 2:0:0:1: [sdc] Write cache: disabled, read cache: enabled, supports DPO and FUA
```

т.е. условно говоря, в системе появилось два “локальных” SCSI диска – sdb и sdc. Нетрудно сообразить, что диск sdb соответствует устройству с LUN 0 и названием disk01, а sdc – LUN 1 и disk02. Команда lsscsi на инициаторе при этом должна обнаружить два новых scsi устройства:

```
# lsscsi -vvv
...
[2:0:0:0]    disk    SCST_FIO disk01            200  -
  dir: /sys/bus/scsi/devices/2:0:0:0  [/sys/devices/platform/host2/session1/target2:0:0/2:0:0:0]
[2:0:0:1]    disk    SCST_FIO disk02            200  -
  dir: /sys/bus/scsi/devices/2:0:0:1  [/sys/devices/platform/host2/session1/target2:0:0/2:0:0:1]
```

После этого с клиентской системы можно либо разбить каждый из дисков на логические разделы (например, с помощью команды fdisk), либо использовать диск целиком. Отключение сетевого устройства впоследствии осуществляется аналогичным образом:

```
iscsiadm -m node --target "iqn.2011-06.ru.example:dsk1" --portal "192.168.21.5:3260" --
login
```

но предварительно необходимо позаботиться о размонтировании файловой системы, расположенной на этом диске.

4 Создаём кластерную файловую систему.

Импортированные таким образом дисковое пространство можно отформатировать под любую файловую систему и затем смонтировать под любой операционной системой, если там имеются подходящие драйверы iSCSI. Но надо иметь в виду, что если один и тот же раздел используется несколькими операционными системами одновременно, то сам по себе протокол iSCSI вовсе не гарантирует сохранность и целостность данных. Для этой цели приходиться использовать специализированные файловые системы, так называемые shared-disk (с общим дисковым накопителем) кластерные файловые системы, которых к настоящему времени ванильное ядро Линукс насчитывает две разновидности – GFS2 от RedHat и OCFS2 от Oracle. Я решил использовать последнюю, поскольку она показалась мне более лёгкой в установке и настройке.

> Первую закрытую версию OCFS трудно было назвать файловой системой, поскольку предназначалась она главным образом для использовании совместно с кластерными продуктами Oracle, в частности для хранения файлов баз данных. Однако, следующая версия существенно отличалась от первой, поскольку за основу была на этот раз взята журналируемая файловая система Ext3, которая в системе Линукс демонстрирует строгое соответствие POSIX-семантике. Одновременно компания Oracle открыла код OCFS2 и передала его в ядро Linux, где этот модуль довольно быстро преодолел путь от экспериментального к проверенно-работоспособному. По-всей видимости к компании Oracle у разработчиков ядра Linux дополнительных вопросов не возникло.
>
> Как бы то ни было, OCFS2 в настоящее время можно использовать в кластерном окружении как файловую систему общего назначения.

Помимо собственно ядерного драйвера файловой системы (который, однако, состоит из несколько ядерных модулей), требуется также пакет ocfs2-tools, включающий в себя ещё несколько утилит, к примеру, команду `mount.ocfs2`, который можно скачать с сайта Oracle [12]. Конечному пользователю и администратору рекомендуется использовать именно ту версию драйвера, которая идёт с ядром Linux, поскольку процесс переустановки нигде как следует не задокументирован, и поэтому я, не задумываясь, последовал этому совету. В ядре 2.6.30 установлена какая-то из версий 1.4.x OCFS2 (более новый вариант 1.6 OCFS2 компания Oracle поставляет только со своим собственным дистрибутивом Linux), так что нам нужен последний релиз из ветки 1.4 утилит ocfs2-tools, каковым на момент написания статьи был 1.4.4. Можно было бы сказать, что этот пакет тоже устанавливается стандартной последовательностью команд:

```
./configure ;
make
make install
```

если бы только у меня в процессе сборки не вылезла ошибка с жалобами на синтаксис файла `tunefs.ocfs2/o2ne_err.et`. Оказалось, что этот файл содержал список стандартных сообщений об ошибках, причём строка 29 была слишком длиной. Укоротив её до стандартных 80-символов, я снова перезапустил процесс сборки и на этот раз он завершился без особых проблем.

Поскольку теперь в наших руках имеется команда `mkfs.ocfs2`, то теперь с её помощью можно отформатировать с любого из инициатора экспортированный диск, который планируется отвести под кластерную файловую систему. Эта команда имеет довольно большое число опций, но на самом деле практически все значения по умолчанию будут правильными, тем более, что в дальнейшем их можно будет подправить с помощью команды `tunefs.ocfs2`.

```
mkfs.ocfs2 --block-size 4K --cluster-size 4K --node-slots 8 --label "OCFS2" -T mail --
mount cluster /dev/sdb1
```

Отлична от значений по-умолчанию только опция метки тома (`--label`). Размер блока и кластера лучше выбрать из соображений, чтобы они плотно вписывались в размер MTU, так что, если у вас настроены jumbo-frames размером 9000, то лучше было бы выбрать размер блока и кластера размером 8 килобайт. Флажок `-T mail` означает, что этот том будет использован под файловую систему общего назначения. Другой способ `-T datafiles` форматирует систему наподобие первого варианта OCFS, оптимизированного под хранение файлов баз данных. Две опции `--node-slots 8 --mount cluster` работают согласованно, и означают, что эта файловая система будет работать в кластере с числом узлов до 8. Для OCFS2 есть жёсткий предел масштабирования – 255 узлов, т.е. в супер-компьютерах её использовать не получится. С помощью опции `--mount local` можно отформатировать файловую систему таким образом, что её можно будет подмонтировать локально без запуска кластерного окружения, о чём речь пойдёт ниже. Само собой, если у вас имеется всего одна система, где запущен инициатор iSCSI, то разницы между кластерным типом монтирования и локальным нет никакой. Так что, ещё раз повторюсь, приведённая ниже настройка должна быть применена в том случае, если вы планируете предоставить доступ к общему дисковому пространству нескольким операционным системам, которым, по крайней мере, это дисковое пространство физически доступно (т.е., скажем, `/dev/sdb1` указывает на один и тот же диск, импортированный через iSCSI каждой из систем с одной и той же целевой системы). Ещё раз напомню, что запуск целевого устройства и инициатора iSCSI на одной и той же операционной системе Linux в настоящее время не допускается.

Для формирования кластерного окружения в пакете ocfs2-tools есть несколько специализированных утилит, но мне они показались не слишком удобными, так что в итоге проще оказалось осуществить запуск вручную. Собственно само кластерное окружение состоит из нескольких ядерных процессов, запускаемых при подгрузке соответствующих модулей, которые контролируют доступ к общему дисковому пространству с помощью distributed lock manager (DLM), который тоже внедрён в код Linux усилиями Oracle. Для координирования работы этих процессов на всех узлах будущего кластера должны быть подготовлены точки монтирования для виртуальных файловых систем configfs (точка монтирования `/config`), dlmfs (`/dlm`) и, разумеется, точка монтирования под кластерную систему (предположим `/mnt/ocfs2`).

На каждом из инициаторов должен иметься конфигурационный файл `/etc/ocfs2/cluster.conf` с описанием кластера и всех входящих в него узлов. Например, такой:

```
cluster:
      node_count = 2
      name = azbuka
node:
      ip_port = 7777
      ip_address = 192.168.21.1
      number = 1
      name = az
      cluster = azbuka
node:
      ip_port = 7777
      ip_address = 192.168.21.2
      number = 2
      name = buki
      cluster = azbuka
```

Т.е. в секции `cluster` задано число узлов в кластере и его название, а в секциях `node` описан каждый из узлов, входящих в этот кластер. Фактически имеют значение только адрес `ip_address` и порядковый номер `number`, имя может быть произвольным. Этот файл без изменений должен быть скопирован на каждый из узлов. После этого настала пора запускать само кластерное окружение. Рекомендуемый способ состоит в запуске скрипта `/etc/init.d/o2cb start`, но если попытаться немного глубже разобраться в его внутреннем устройстве, то на самом деле процедура запуска сводится к следующей последовательности команд – загрузке необходимых модулей, монтировании вспомогательных файловых систем и собственно запуска нашего кластера “azbuka”:

```
modprobe ocfs2_stackglue
modprobe ocfs2_nodemanager
modprobe ocfs2
modprobe ocfs2_stack_o2cb
modprobe ocfs2_dlm
modprobe ocfs2_dlmfs

mount -t configfs none /config
mount -t ocfs2_dlmfs none /dlm

o2cb_ctl -H -n "azbuka" -t cluster -a online=yes
```

Эти команды должны быть вставлены в инициализационные скрипты на ваше усмотрение, но при этом запущены на каждой из клиентских систем. Синхронность старта при этом не требуется, т.е. узлы, описанные в файле `/etc/ocfs2/cluster.conf`, могут подключаться к кластеру в произвольной последовательности. По крайней мере свидетельством, что старт прошёл успешно, будет появление в директории `/config` ветки с названием кластера:

```
# ls -lA /config/cluster/azbuka/
total 0
drwxr-xr-x 3 root root    0 2011-06-20 21:15 heartbeat/
-rw-r--r-- 1 root root 4096 2011-06-27 14:14 idle_timeout_ms
-rw-r--r-- 1 root root 4096 2011-06-27 14:14 keepalive_delay_ms
drwxr-xr-x 6 root root    0 2011-06-20 21:15 node/
-rw-r--r-- 1 root root 4096 2011-06-27 14:14 reconnect_delay_ms
```

Директория heartbeat, вместе с файлами delay и timeout регламентируют через какой именно промежуток времени упавший или выключенный узел будет отключён от кластера (в OCFS2 для этого используется термин “огораживание”/fencing). Это определяется по промежутку времени в течении которых любой из узлов обновляет определённый блок на кластерной файловой системе, и по умолчанию равен одой минуте. Если в течении этого времени узел не сумел проделать эту операцию, то он считается выключенным, и может вновь подсоединиться к кластеру только после `reconnect_delay`. Менять значения по умолчанию у меня пока причин не было.

В итоге на каждом из узлов кластера можно будет смонтировать кластерную файловую систему командой:

```
mount -t ocfs2 /dev/sdb1 /mnt/ocfs2
```

или же внести запись в файл `/etc/fstab`, чтобы эта система монтировалась при старте узлов кластера:

```
/dev/sda1 /mnt/ocfs2  ocfs2 _netdev,defaults 0 0
```

Опция `_netdev` запускает старт монтирования файловой системы после настройки файловой системы, но разумеется, чтобы это опция сработала необходимо ещё иметь подключённый iSCSI диск.

5 Последние “штрихи напильником”.

Если просуммировать все настройки приведённые выше, то инициализационный скрипт на сервере для старта целевой системы должен выглядеть таким образом:

```
modprobe scst
modprobe scst_vdisk
modprobe scst_disk
modprobe scst_cdrom
modprobe iscsi_scst
/usr/local/sbin/iscsi-scstd
/usr/local/sbin/scstadmin -noprompt -force -config /etc/scst.conf
```

На стороне клиента инициализационные скрипты требуют небольшой модификации:

```
modprobe ocfs2_stackglue
modprobe ocfs2_nodemanager
modprobe ocfs2
modprobe ocfs2_stack_o2cb
modprobe ocfs2_dlm
modprobe ocfs2_dlmfs

mount -t configfs none /config
mount -t ocfs2_dlmfs none /dlm

o2cb_ctl -H -n "azbuka" -t cluster -a online=yes

modprobe iscsi_tcp
iscsid  --initiatorname=/etc/iscsi/initiatorname.iscsi
iscsiadm -m discovery -t st -p 192.168.21.5
iscsiadm -m node --target "iqn.2011-06.ru.example:dsk1" --portal "192.168.21.5:3260" --
login

sleep 2
echo deadline > /sys/block/sdb/queue/scheduler
echo 512 > /sys/block/sdb/queue/read_ahead_kb
echo 64 > /sys/block/sdb/queue/max_sectors_kb
echo 64 > /sys/block/sdb/queue/nr_requests

mount -t ocfs2 /dev/sdb1 /mnt/ocfs2
```

Хотя на целевой системе лучше всего использовать I/O планировщик CFQ (который кроме того является в Linux 2.6.30 планировщиком по-умолчанию), то на клиенте для файловой системы OCFS2 рекомендуется заменить его на deadline. Это связано с тем, что при интенсивных операциях ввода-вывода CFQ-планировщик может на долгое время лишить один из узлов доступа к файловой системе, так что тот не успеет обозначить своё присутствие в кластере и будет отключён от него. Обойти эту проблему можно использованием планировщика с гарантированным временем доступа (deadline), который обеспечивает это время за счёт меньшей в среднем отзывчивости файловой системы. Кроме того, поскольку дисковая подсистема в Linux по умолчанию рассчитана на локально подключённые диски, то для сетевых дисков желательно увеличить значение `read_ahead` и ограничить число запросов и количество данных, передаваемых за одну SCSI команду.

Важно также учитывать, каким образом будет останавливаться ваша кластерная система, в частности последовательность отключения инициаторов и целевой системы. Сервис `iscsi-scstd` будучи остановлен командой `kill -9` на самом деле останавливается достаточно корректно и успевает сбросить кэшированные данные на диск, т.е. обычная остановка сервера целостности данных не угрожает. Важно, чтобы это происходило после отмонтирования кластерной файловой системы на инициаторах и отключения от целевой файловой системы:

```
iscsiadm -m node --target "iqn.2011-06.ru.example:dsk1" --portal "192.168.21.5:3260" --
logout
```

По всей видимости, единственный способ осуществить такую синхронизацию, это отключение всех клиентов со стороны файлового сервера (которое в свою очередь может быть запущено источником бесперебойного питания при отключении электричества), а потом уже завершение работы самого сервера.

6 А теперь дискотека!

Хотя главное преимущество OCFS2 вовсе не быстродействие, а строгая POSIX-совместимость, но в заключение я приведу несколько тестов, которые продемонстрируют во сколько именно поддержка этой строгости нам обходится.

Во-первых, сравнение скорости чтения с дисковых устройств на стороне сервера и клиента через гигабитную switched сеть (что, собственно, измеряет пропускную способность iSCSI-подсистемы без учёта файловой системы). Локально:

```
#  hdparm -t /dev/sda

/dev/sda:
 Timing buffered disk reads:  362 MB in  3.01 seconds = 120.14 MB/sec
#  hdparm -T /dev/sda

/dev/sda:
 Timing cached reads:   2754 MB in  2.00 seconds = 1376.49 MB/sec
```

и удалённо (`/dev/sdb` – это iSCSI диск):

```
# hdparm -t /dev/sdb

/dev/sdb:
 Timing buffered disk reads:  112 MB in  3.03 seconds =  36.96 MB/sec
# hdparm -T /dev/sdb

/dev/sdb:
 Timing cached reads:   2264 MB in  2.00 seconds = 1132.55 MB/sec
```

Т.е. скорость непосредственного чтения с диска упала примерно в 3-4 раза, хотя за счёт кэширования эта разница не столь заметна. Теперь можно сравнить скорость OCFS2+iSCSI и NFS смонтированным тем же самым образом, т.е. NFS клиент является одновременно iSCSI инициатором, а NFS-сервер – целевой системой. Для тестирования использовалась программа `bonnie++`. Данные для OCFS2 с Рис. 4 демонстрируют примерно такую же скорость последовательного вывода, как и для непосредственным чтения с диска, но с более чем вдвое меньшей скоростью блочного чтения-записи 14 МБ/с. В этом отношении NFS лидирует с большим отрывом, хотя и за счёт более интенсивного использования циклов процессора.

В заключение можно сказать, что скорость OCFS2 по сравнению с NFS оказывается вполне приемлемой, при этом обеспечивая строгое соответствие POSIX-семантике. Кроме того сама по себе технология iSCSI открывает интересные перспективы использования общего дискового пространства и виртуальной памяти (хотя, в этом случае вопросы быстродействия и совместного доступа требуют дополнительного решения) в кластерном окружении.

Список литературы

[1] Сергей Довганюк. Отказоустойчивый кластер с минимальным бюджетом. *Системный администратор*, (5), 2006.

[2] Евгений Прокопьев. Кластеризация + виртуализация: Linux HA + OpenVZ. *Системный администратор*, (11), 2006.

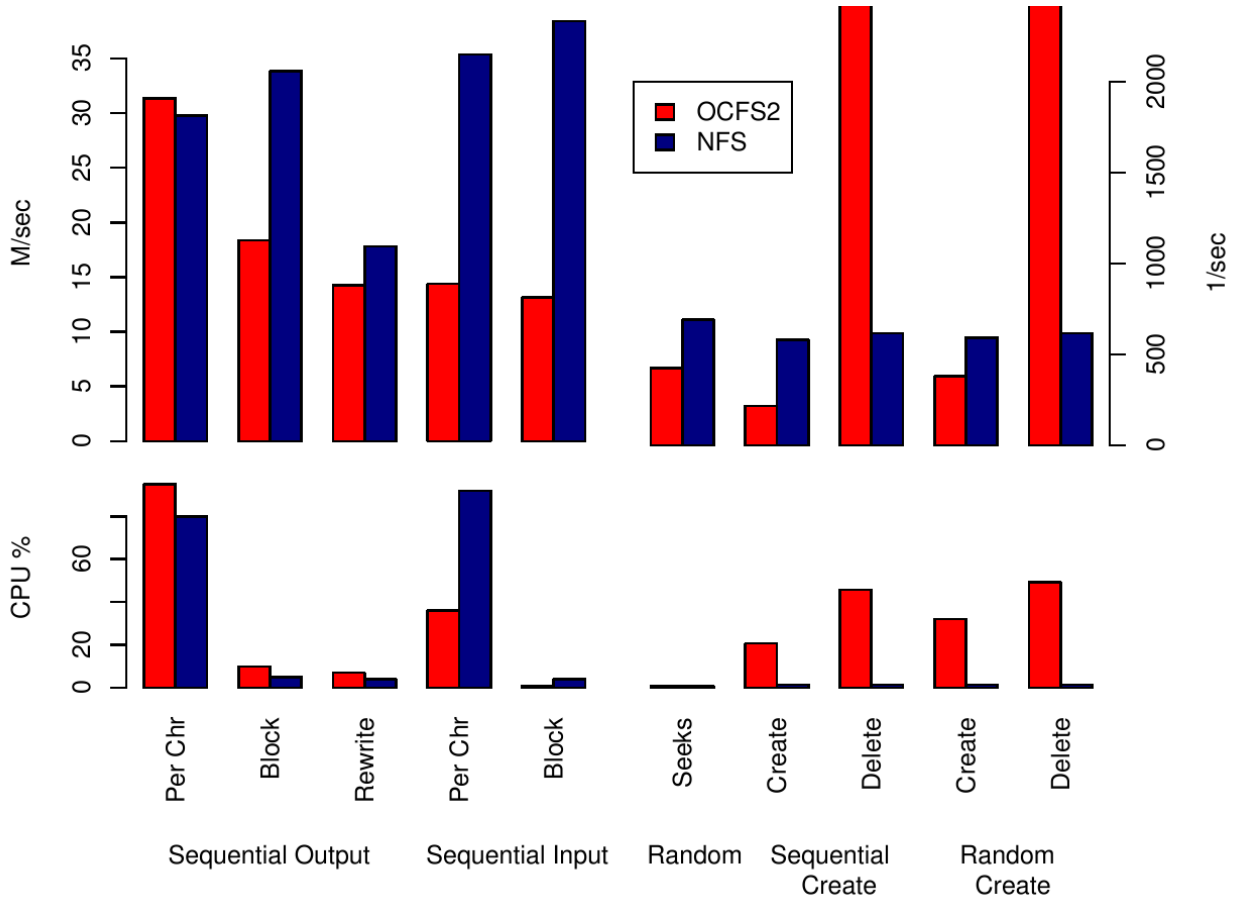

Рис. 4: Результат сравнительного тестирования OCFS2 и NFS. В верхнем ряду приведены скорости чтения-записи и файловых операций. В нижнем ряду показана загрузка процессора при выполнении этих тестов.

[3] Страница в Википедии.

[4] A tale of two SCSI targets.

[5] Сергей Яремчук. Openfiler - дистрибутив для организации NAS. *Системный Администратор*, (11), 2008.

[6] Домашняя страница проекта SCST.

[7] Домашняя страница интерфейсных модулей к SCST.

[8] Конфигурационный perl-скрипт.

[9] *RFC3720: Internet Small Computer Systems Interface (iSCSI).*

[10] *RFC3721: Internet Small Computer Systems Interface (iSCSI) Naming and Discovery*.

[11] Домашняя страница клиентского модуля iSCSI.

[12] ocfs-tools ver. 1.4 .