УДК 004

Е.В. Назаренко

Сравнение архитектуры распределенных файловых систем

Дан обзор популярных распределенных файловых систем разных типов для операционных систем *Linux*. Показано, как их архитектура адаптирована к решению специфических задач. Проведено сравнение рассмотренных файловых систем.

This report contains the survey of actual distributed file systems of different types for the Linux. It is shown how their architecture is adapted to the solution of specific issues. The comparison of the surveyed file systems is made.

Дано огляд популярних розподілених файлових систем різних типів для операційних систем *Linux*. Показано, як їх архітектуру адаптовано до розв'язання специфічних задач. Проведено порівняння розглянутих файлових систем.

Введение. Локальные файловые системы – неотъемлемая часть операционной системы. Они предоставляют пользователю высокоуровневый интерфейс к локальным запоминающим устройствам. Распределенные файловые системы обеспечивают интерфейс к запоминающим устройствам удаленных компьютеров. Кроме того, они позволяют разделять ресурсы одного компьютера между несколькими пользователями. Для пользователя *UNIX*-подобных систем распределенная файловая система, как правило, неотличима от локальной: имена файлов локальной ФС и удаленной ФС начинаются с общего корня и имеют одинаковую структуру, а приложения, работающие с файлами локальной ФС, могут также работать и с файлами удаленной ФС. Это стало возможно благодаря внедрению в операционную систему уровня виртуальной файловой системы, изначально разработанного для поддержки файловой системы *NFS* в *UNIX*.

Программы, запускаемые на вычислительных кластерах, состоят из множества процессов, выполняемых параллельно на разных узлах кластера. При этом часто программы оперируют данными очень большого объема, которые не помещаются в оперативной памяти вычислительных узлов. Во время расчета их процессы интенсивно обращаются к дисковой памяти кластера. Такие задачи выдвигают дополнительные требования к распределенным файловым системам. Во-первых, они должны обеспечивать высокую суммарную пропускную способность ввода–вывода. Это достигается путем объединения ресурсов множества узлов ввода–вывода. Во-вторых, файловые системы должны позволять одновременный доступ к файлу большого количества параллельных процессов с высокой пропускной способностью. Для этого данные файла распределяются между набором узлов ввода–вывода. Стратегии распределения могут быть разными. Например, файловые системы *PVFS* и *Lustre* могут разделять файл между узлами ввода–вывода на непересекающиеся подмножества данных (порции), а файловая система *GFS* вдобавок поддерживает несколько копий каждой порции, позволяя считывать данные параллельно.

Распределенные файловые системы используются также для долгосрочного хранения больших объемов данных. Системы, выполняющие эту функцию, обычно состоят из большого количества элементов. Как следствие, вероятность выхода из строя отдельного элемента системы достаточно высока. Поэтому такие системы обладают отказоустойчивостью и обеспечивают высокую доступность данных. Они также обеспечивают одновременный доступ множества клиентов к хранимым данным. Примером такой файловой системы служит *GFS*. *Lustre* также используется в этих целях. Важная характеристика для систем такого типа – масштабируемость.

Существует большое количество распределенных файловых систем. При выборе распределенной файловой системы для компьютерной сети пользователь руководствуется тем, какие задачи будут решаться на его оборудовании. Цель статьи – показать, как архитектура распределенных файловых систем адаптирована к решению специфических задач. Далее дан обзор популярных распределенных файловых систем разных типов для ОС *Linux*. Такие об-

зоры по системам одного класса делались неоднократно. Например, в [1] приведено сравнение *PVFS* и *Lustre* – файловых систем для вычислительных кластеров, в [2] рассмотрены системы для хранения больших объемов данных.

Файловая система *NFS*

Файловая система *NFS* (*Network Filesystem* – сетевая файловая система) впервые разработана корпорацией *Sun* для предоставления удаленного доступа к файловым системам [3]. *NFS* состоит из протокола, серверной части и клиентской части.

Протокол *NFS* определяется набором процедур для доступа к файлам и каталогам. Вызовы процедур синхронны, т.е. клиент блокируется до тех пор, пока сервер не завершит обработку вызова. Протокол *NFS* есть протоколом без состояния. Это значит, что параметры каждого вызова процедуры содержат всю необходимую информацию для того, чтобы завершить вызов, сервер не содержит информации о предыдущих вызовах. Например, процедура *read* чтения данных из файла, в отличие от соответствующей процедуры некоторой локальной файловой системы, требует указания позиции в файле, с которой следует начать чтение. Использование протокола без состояния упрощает восстановление после сбоев: при возникновении сбоя на сервере клиент повторно посылает запросы до тех пор, пока не получит ответ, что позволяет серверу вообще не проводить восстановление, при этом данные не будут потеряны. При сбое клиента ни серверу, ни клиенту проводить восстановление не требуется.

Для выполнения файловых операций используется предоставленный сервером *дескриптор файла*, непрозрачный для клиента. Он используется для указания файла на сервере при вызове процедур. В качестве дескриптора файла *UNIX*-сервер может использовать совокупность, состоящую из номеров *i*-узла, его поколения и идентификатора файловой системы. Номер поколения *i*-узла – это число, содержащееся в *i*-узле и увеличиваемое на единицу всякий раз, когда *i*-узел освобождается. Таким образом, если сервер выдал дескриптор файла с номером *i*-узла файла, который затем был удалён, а *i*-узел использован повторно, при получении первоначального дескриптора файла от клиента сервер может определить, что номер *i*-узла ссылается на другой файл и сообщить об этом клиенту. Новый дескриптор файла возвращают процедуры *lookup* (просмотреть, процедура возвращает новый дескриптор файла и атрибуты по имени файла в каталоге), *create* и *mkdir*, которые в свою очередь принимают дескриптор файла (описывающий каталог) как аргумент.

Для получения дескриптора файла корневого каталога по его имени используется протокол *MOUNT*. *Монтирование* связывает каталог клиента с каталогом сервера. В результате образуется единое дерево каталогов и имена удалённых файлов имеют ту же структуру, что и имена локальных файлов. Монтирование проводится вызовом программы *mount*. Любой узел дерева файловой системы может быть точкой монтирования другой файловой системы. Сервер *MOUNT* – это процесс (демон), обслуживающий запросы на монтирование по мере их поступления. При поступлении запроса от клиента проводится проверка списка экспортируемых файловых систем и клиентов, которые могут их импортировать. Если клиент имеет разрешение импортировать запрошенную файловую систему, он получает дескриптор файла экспортируемого сервером каталога.

Для того чтобы скрыть от программ пользователя различия между удаленной и локальной файловыми системами, в операционную систему *UNIX* был добавлен уровень *VFS* (*Virtual File System* – виртуальная файловая система). Место *VFS* в операционной системе иллюстрирует рис. 1 [4]. С каждым файлом или каталогом связывается *v*-узел – структура, содержащая операции, которые могут быть выполнены над файлом или каталогом. Операции реализуются конкретной файловой системой, а ядро взаимодействует не с файловой системой непосредственно, а с *VFS*. Такая реализация позволяет операционной системе обращаться с файловыми системами разных типов одинаково, независимо от конкретной реализации. Для получения *v*-узла файла по *v*-узлу каталога и по

имени файла в каталоге служит операция *lookup*. При разборе пути операционная система разбивает путь на составляющие, вызывая *lookup* для получения *v*-узла каждой следующей составляющей. Путь не передается целиком, поскольку реализации *v*-узла недоступна информация о точках монтирования других файловых систем. В свою очередь, передача полных путей в файловой системе *NFS* привела бы к тому, что сервер был бы вынужден хранить информацию о точках монтирования клиента и уже не был бы сервером без состояния.

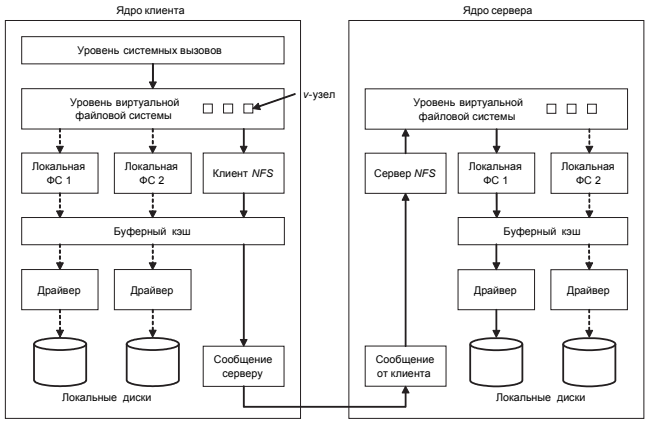

Рис. 1. Архитектура *NFS*

Сервер *NFS* не поддерживает блокировку файла между запросами. Поскольку одна операция записи может потребовать нескольких обращений к серверу, при одновременной записи файла двумя клиентами данные могут смешиваться. *UNIX* позволяет удалить запись каталога для открытого процессом файла, после чего файл не будет иметь имени в файловой системе, а процесс будет продолжать читать и записывать файл. Эта возможность используется некоторыми программами для реализации временных файлов. Эта особенность файловой системы *UNIX* была поддержана в *NFS*. При выполнении операции *VFS remove* клиент проверяет, открыт ли файл, и если да, переименовывает его. Когда *v*-узел становится неактивным, файл удаляется.

Для повышения производительности файловой системы *NFS* предпринят ряд улучшений. Среди них отметим следующие. Для уменьшения количества запросов к серверу на стороне клиента применяется буферное кэширование и кэширование атрибутов файлов. Обмен информацией между клиентом и сервером выполняется большими порциями. Для ускорения последовательного чтения применяется опережающее чтение.

Параллельная файловая система *PVFS*

Параллельная виртуальная файловая система *Parallel Virtual File System* (*PVFS*) создана с целью обеспечить *Linux* кластеры высокопроизводительной файловой системой [5]. Ее архитектура изображена на рис. 2. *PVFS* организована как клиент-серверная система с более чем одним процессом ввода–вывода (сервером). Процессы ввода–вывода выполняются на узлах ввода–вывода. Прикладные процессы выполняют операции ввода–вывода с помощью клиентской библиотеки. Единое дерево каталогов для прикладных процессов поддерживает процесс менеджер *PVFS* (мастер на рис. 2). Клиенты, процессы ввода–вывода и менеджер *PVFS* не обязательно должны выполняться на различных узлах. Однако выполнение их на разных узлах может привести к повышению производительности файловой системы.

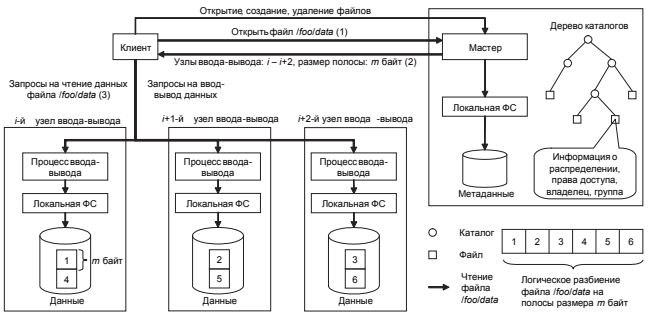

Рис. 2. Архитектура *PVFS* и *Lustre*

PVFS распределяет данные каждого файла между упорядоченным набором узлов ввода–вывода. Информация о распределении файлов, как и прочие метаданные (права доступа, владелец, группа и др.), хранится в файлах локальной файловой системы и поддерживается менеджером *PVFS*. За распределение файла отвечают три параметра: номер базового узла ввода–вывода (*base*), количество узлов ввода–вывода (*pcount*) и размер полосы (*ssize*). Распределяемый файл логически разбивается на по-

лосы размера *ssize* байт. Узел ввода–вывода под номером *base+i* ($0 \le i < pcount$) содержит полосы *i*, *pcount+i* и т.д. Эти параметры можно установить при создании файла, в противном случае *PVFS* использует значения по умолчанию. Каждый узел хранит свою порцию файла *PVFS* в файле локальной файловой системы. Имя этого файла основано на номере *i*-узла, который менеджер присвоил файлу *PVFS*. Такой способ распределения облегчает параллельный доступ к данным файла.

Когда прикладной процесс открывает файл *PVFS*, менеджер *PVFS* возвращает ему расположение тех узлов ввода–вывода, которые содержат данные файла. При выполнении дальнейших операций чтения-записи прикладной процесс взаимодействует напрямую с узлами ввода–вывода без участия менеджера. При чтении данных файла клиентская библиотека посылает дескриптор запрашиваемой области файла процессам ввода–вывода. Область может не быть непрерывной. Область с регулярным шагом описывают параметры *offset*, *gsize* и *stride*. При вызове функции чтения по такой области будут прочитаны *gsize* байт файла с позиций *offset, offset+stride, offset+2*stride, ….* Процессы ввода–вывода определяют, какая часть области содержится в обслуживаемых ими данных, и выполняют необходимый ввод-вывод.

Клиентские библиотеки, работающие в пространстве пользователя, содержат реализацию процедур ввода–вывода *UNIX*/*POSIX API*. Эти процедуры заменяют функции библиотеки C (*libc*), осуществляющие системные вызовы (*libc syscall wrappers*). На каждом вычислительном узле с *PVFS* связывается каталог. Файл, путь к которому содержит этот каталог, трактуется клиентской библиотекой как файл *PVFS*. Таким образом существующие приложения, использующие *UNIX API*, могут работать с файлами *PVFS* без перекомпиляции. Кроме того, с файлами *PVFS* могут работать общепринятые команды оболочки *UNIX*, такие как *ls*, *cp* и *rm*. *PVFS* также имеет свой *API*. Его функции позволяют считывать несмежные зоны файла одним вызовом функции. *PVFS* также поддерживает интерфейс *MPI–IO*.

Lustre – файловая система для *Linux* кластеров

Распределённая параллельная файловая система *Lustre* (сочетание терминов «*Linux*» и «*Clusters*») предназначена для работы в больших кластерах под управлением *Linux* [6]. Её используют в кластерах с десятками тысяч клиентских систем, а многие центры HPC используют *Lustre* как глобальную файловую систему, обслуживающую десятки кластеров вычислительного центра [7]. *Lustre* совместима с *POSIX*. Как и прочие распределённые файловые системы, *Lustre* разделяет вычислительные ресурсы и ресурсы хранения данных. Это освобождает вычислительные узлы от чтения, передачи и записи данных, а файловые серверы – от вычислительной работы.

Устройство *Lustre* представлено на рис. 2. Полезный ввод-вывод файловой системы выполняют серверы хранения объектов *Object Storage Servers* (*OSS*). Интерфейс к запоминающим устройствам сервера предоставляют *OST* (*Object Storage Targets*). *OST* рассматривает свой накопитель как объектно-ориентированный диск (*OBD – Object-Based Disk*), доступ к которому обеспечивает модуль *Lustre*, отвечающий за взаимодействие с локальными журналируемыми файловыми системами *Linux*, среди которых разработанная для *Lustre ldiskfs* (*Lustre Disk Filesystem*) [8]. Один сервер может обслуживать несколько *OST*. Метаданные файловой системы хранятся на сервере метаданных (*MDS – MetaData Server*). Сервер метаданных обслуживает все операции пространства имён файловой системы, такие как поиск файлов, создание файла, управление атрибутами файлов и каталогов. Для *MDS* и *OSS* характерны разные формы доступа к данным. *MDS* выполняет большое количество операций чтения и записи малых объёмов данных. Напротив, *OSS* как правило выполняют операции чтения и записи больших объёмов данных. На производительность *MDS* влияет время доступа, для *OSS* важна пропускная способность диска. Прикладные процессы взаимодействуют с *Lustre* через *VFS*.

На уровне файловой системы *Lustre* рассматривает файлы как объекты (а не как набор

блоков данных), местонахождение которых устанавливается через *MDS*. С каждым обычным файлом, каталогом, символической ссылкой и специальным файлом *Lustre* связывает *i*-узел. Однако *i*-узлы обычных файлов вместо указателей на блоки данных файла содержат ссылки на объекты из *OSTs*, которые хранят данные файла. Данные файла на одном *OST* хранятся в одном объекте. Файлу может отвечать несколько объектов. В этом случае данные файла будут распределены между ними по той же схеме, по которой файл *PVFS* распределяется между узлами ввода–вывода. Как и в *PVFS*, заданные по умолчанию параметры распределения файла можно настроить при создании файла. *Lustre* также позволяет определить эти параметры для всех файлов каталога (и рекурсивно для всех файлов подкаталогов этого каталога). Распределение файла между *OST* позволяет повысить пропускную способность ввода–вывода при доступе к файлу. Кроме того, это позволяет создавать файлы размером больше одного *OST* (*Lustre* допускает файлы больше 1 ПБ).

При создании файла клиент обращается к серверу метаданных. *MDS* создаёт *i*-узел файла, после чего обращается к *OSTs*, чтобы они создали объекты для хранения данных файла. Аналогично при доступе к существующему файлу клиент обращается к *MDS*, чтобы получить информацию о расположении файла. Последующий ввод-вывод файла выполняется напрямую между клиентом и *OSTs*.

Lustre спроектирована так, что отказ или перезагрузка одного сервера не выводит из строя всю систему. Так, если для активного сервера имеется резервный сервер, то при отказе активного сервера его функции возьмёт на себя резервный сервер, при этом приложения продолжат выполняться без сбоя, только испытают задержку в выполнении системных вызовов, обращающихся к файловой системе. Кроме того, при создании новых файлов отслеживаются неисправные *OST*. Дополнительную надёжность файловой системе обеспечивает использование журналируемых файловых систем для хранения метаданных и объектов из *OSTs*. *Lustre* позволяет добавлять новые *OSS* не прерывая работу файловой системы.

The Google File System

Файловая система *Google* (*GFS*) была спроектирована с учетом рабочих нагрузок, характерных для внутренних приложений *Google* [9]. Так, система состоит из сравнительно небольшого (несколько миллионов) количества файлов очень большого размера – в среднем 100 МБ, многогигабайтные файлы общеприняты. Файлы, как правило, изменяются путем добавления новых данных, а не перезаписи существующих, произвольные записи крайне редки. Например, распределенные приложения *Google* интенсивно используют файлы для реализации очереди много производителей – один потребитель, в которую сотни производителей пишут одновременно. При чтении возможны две формы доступа: большие потоковые чтения (порциями около 1 МБ, как и для записей) и малые произвольные чтения (по нескольку КБ). Кроме того, изначально предполагалось эксплуатировать файловую систему на недорогом оборудовании. Поскольку система состоит из большого количества элементов, отказ отдельных составляющих системы – скорее норма, чем исключение.

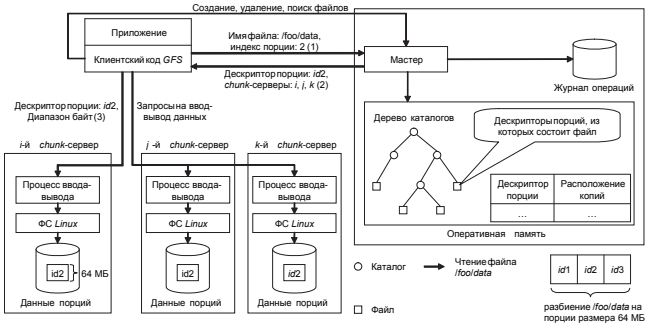

Рис. 3. Архитектура *GFS*

Файл в *GFS* состоит из *порций* (*chunk*) размера 64 МБ. Каждая порция имеет неизменный 64-х битный идентификатор – *дескриптор порции*, уникальный в пределах системы. Пространства имен файлов и порций, а также информацию о том, из каких порций состоит файл, содержит единственный в системе *мастер*. Сами порции хранятся на *chunk-серверах* как *Linux*-файлы, по умолчанию в трех экземплярах. Фай-

лы растут по мере необходимости. Расположение копий каждой порции хранится на мастере. Приложения, выполняемые на машинах под управлением *Linux*, взаимодействуют с *GFS* через *API* файловой системы, реализуемый клиентским кодом, с которым они компонуются. *GFS* не имеет *POSIX*-интерфейса. Допускается выполнение chunk-сервера и клиента на одной машине. Клиенты, *chunk*-серверы и мастер выполняются в пространстве пользователя. Элементы файловой системы показаны на рис. 3.

При чтении данных клиент сначала вычисляет индекс порции в файле по заданному в байтах смещению. Затем он посылает мастеру запрос, содержащий имя файла и индекс порции. В ответ клиент получает дескриптор порции и расположение копий порции. После этого клиент читает данные с одного из *chunk*-серверов, содержащего копию порции, сообщая ему дескриптор порции и диапазон байт. Хотя система и имеет единственного мастера, к которому обращаются все клиенты, он не является узким местом системы. При чтении и записи, клиент обменивается с мастером лишь небольшим количеством метаданных, выполняя обмен данными напрямую с *chunk*-серверами. Кроме того, клиенты кэшируют результаты запросов к мастеру, причем, читая данные, клиент обычно запрашивает информацию о нескольких порциях в одном запросе, а мастер может дополнить свои ответы информацией о следующих за запрошенными порциях.

Запись данных в *GFS* требует более сложного взаимодействия между элементами системы, чем чтение. Во-первых, каждая порция имеет несколько копий, и запись должна быть сделана в каждую из них. Во-вторых, одна порция может изменяться многими клиентами одновременно. Для того чтобы одинаково влиять на разные копии, изменения во все копии вносятся в одном порядке. Упорядочением записей в порцию занимается одна из ее копий – *первичная* копия. При записи данных в порцию клиент сначала запрашивает у мастера расположение ее первичной и остальных (*вторичных*) копий, кэшируя полученную информацию. Далее, клиент посылает данные во все копии. Получив от всех копий подтверждение о получении данных, клиент направляет первичной копии запрос на запись. Первичная копия, взаимодействуя с вторичными копиями, удовлетворяет этот запрос в установленном порядке.

Особой формой записи в *GFS* есть атомарная операция добавления *record append*. При одновременной записи файла многими клиентами, смещение в файле, которое содержит отдельный клиент, может не совпадать с настоящим концом файла. Операция *record append* записывает данные в конец файла независимо от текущего смещения в клиенте.

Мастер держит все метаданные файловой системы в оперативной памяти, что возможно благодаря большому размеру порции и сравнительно небольшому количеству файлов в файловой системе. Пространства имен и отображение из файлов в порции также хранятся на локальном диске мастера, в отличие от расположения копий порций, которое может быть получено у *chunk*-серверов. Мастер периодически сканирует свое состояние в фоновом режиме, выполняя сборку мусора в файловой системе, повторное копирование при наличии отказа *chunk*-серверов, а также перемещая порции для балансировки нагрузки и использования дискового пространства между *chunk*-серверами. Мастер обслуживает запросы на создание, удаление, поиск файлов.

Сравнение файловых систем

Характеристики рассмотренных файловых систем сведены в таблицу.

Заключение. Рассмотренные распределенные файловые системы спроектированы в соответствии с нагрузками, для которых они предназначены. Файловые системы *PVFS*, *Lustre* и *GFS* решают задачу параллельного доступа к файлу множества клиентов, распределяя файл между узлами ввода–вывода. Разработчики *PVFS* отказались от кэширования данных на клиенте, чтобы избежать накладных расходов на поддержание клиентских кэшей в непротиворечивом состоянии (это необходимо для научных приложений, для которых предназначена файловая система) [10]. В файловой системе *NFS* кэш разных клиентов может быть несо-

Таблица.

	NFS	*PVFS*	*Lustre*	*GFS*
Область применения (назначение)	Предоставляет удаленный доступ к файловым системам	Высокопроизводительный ввод–вывод на вычислительных кластерах, *Lustre* также используют в информационных центрах как внутреннюю файловую систему общего назначения		Обслуживает внутренние распределенные приложения *Google*, которые требуют интенсивной обработки данных
Способ распределения одного файла между узлами ввода–вывода	Не распределяет файл	Файл расчленяется между узлами ввода–вывода по схеме *RAID* 0, количество узлов ввода–вывода и размер полос, по которым распределяется файл, устанавливает пользователь		Файл разбивается на порции фиксированного размера, каждая из которых дублируется, количество копий настраивается
Кэширование данных на клиенте	Да	Нет	Да	Нет
Способ доступа к файловой системе	*VFS* модуль	Клиентские библиотеки или *VFS* модуль	*VFS* модуль	Киентские библиотеки
Поддерживаемые *API*	*POSIX*	Свой *API*, *POSIX*, *MPI–IO*	*POSIX*, *MPI–IO*	Свой *API*

гласован [4]. В *GFS*, как и в *PVFS*, клиенты также не кэшируют данные, поскольку кэширование не оправдывает себя при типичной форме доступа приложений к файлам *GFS* и больших размерах файлов, характерных для файловой системы. Специфическое назначение распределенной файловой системы отражается и в ее *API*. *API PVFS* позволяет описать область с регулярным шагом для доступа к файлу, *GFS* имеет атомарную операцию добавления. Файловая система *GFS* обеспечивает высокую доступность данных, поддерживая множественные копии данных.

1. *A comparative* experimental study of parallel file systems for large-scale data processing / Z. Sebepou, K. Magoutis, Marazakis M. et al. // Institute of computer science, 2008-06-06. – http://www.usenix.org/event/lasco08/tech/full_papers/sebepou/sebepou_html/
2. *Черняк Л.* Файловые системы для Больших Данных // Открытые системы. – 2011. – № 5. – С. 68. – http://www.osp.ru/os/2011/05/13009415/
3. *Design* and implementation of the Sun Network Filesystem / R. Sandberg, D. Goldberg, S. Kleiman et al. // Proc. of the Summer, Portland. June 1985 USENIX Conf. – P. 119–130.
4. *Таненбаум Э.* Современные операционные системы. – СПб.: Питер, 2010. – 1120 с.
5. *PVFS*: a parallel file system for Linux Clusters / P.H. Carns, W.B. Ligon III, R.B. Ross et.al. // Proc. of the 4th Annual Linux Showcase and Conf., Atlanta, GA, Oct. 2000. – P. 317–327. – http://www. parl.clemson.edu/pvfs/el2000/extreme2000.ps
6. *Braam P.J.* Lustre: a scalable, high-performance file system. Nov. 11th, 2002.
7. *Lustre* file system. High performance storage architecture and scalable cluster file system. Dec. 2007.
8. *Understanding* Lustre filesystem internals / F. Wang, S. Oral, G. Shipman et al. // April 2009. – http://wiki.lustre.org/images/d/da/Understanding_Lustre_Filesystem_ Internals.pdf
9. *Ghemawat S.*, *Gobioff H.*, Shun-Tak Leung. The Google File System // SOSP'03, Oct. 19–22, 2003, New York: Bolton Landing, USA.
10. *Frequently* Asked Questions about PVFS. – *http*://www.pvfs.org

Поступила 03.06.2012

Тел. для справок: +380 44 526-3603 (Киев)
E-mail: *eugn@ukr.net*

●