Оглавление

1 Архитектура сетевой файловой системы

1.1 Модели удалённого доступа

- удалённый доступ, основанный на **множестве документов**, связанных гиперссылками.
- удалённый доступ, основанный на **сетевых файловых системах**.

1.2 Компоненты сетевой файловой системы

Как и в централизованных системах, в сетевых системах функцией файловой системы является хранение файлов и предоставление доступа к ним.

Сетевая файловая система поддерживается одним или более компьютерами.

Компьютеры, которые позволяют пользователям сети получать доступ к своим файлам, обычно называют **файловыми серверами**.

Файловые серверы отрабатывают запросы на чтение или запись файлов, поступающие от других компьютеров сети, которые в этом случае являются **клиентами** файловой службы.

Во многих сетевых файловых системах клиентский компьютер может **монтировать** сетевые файловые системы к своим локальным файловым

системам, обеспечивая пользователю удобный доступ к удаленным каталогам и файлам. **При этом данные монтируемых файловых систем физически никуда не перемещаются, оставаясь на серверах.**

С программной точки зрения сетевая файловая система - это **сетевая служба.**

Файловая служба включает

- **программы-серверы**
- **программы-клиенты,**

взаимодействующие с помощью **определенного протокола** по сети между собой.

Т.о., файловым сервером называют **компьютер**, на котором хранятся предоставляемые в совместный доступ файлы, и **программу** (или процесс, в рамках которого выполняется данная программа), которая работает на этом компьютере и обеспечивает совокупность услуг по доступу к файлам и каталогам на удаленном компьютере.

Соответственно программу, работающую на клиентском компьютере и обращающуюся к файловому серверу с запросами, называют клиентом файловой системы, как и компьютер, на котором она работает.

В сети может одновременно работать несколько программных файловых серверов, каждый из которых предлагает различные файловые услуги.

Кроме того, один компьютер может в одно и то же время предоставлять пользователям сети услуги различных файловых служб, для этого нужно, чтобы на этом компьютере работало несколько процессов, каждый из которых реализовывал бы файловую службу определенного типа.

В хорошо организованной распределенной системе пользователи не должны знать, как реализована файловая система.

В частности, они не должны знать количество файловых серверов, их месторасположение и функции.

Они только знают, что если процедура определена в файловой службе, то требуемая работа каким-то образом выполняется, возвращая им результаты.

Более того, пользователи даже не должны знать, что файловая система является распределенной. В идеале для пользователя она должна выглядеть так же, как и централизованная файловая система.

Современные сетевые файловые системы пока еще не полностью соответствуют идеалу. В большинстве ОС пользователь должен явно указать имя файлового сервера при доступе к его ресурсам.

2 Модель сетевой файловой системы

Сетевая файловая система в общем случае включает следующие элементы

На стороне сервера:

- *локальную файловую система сервера;*
- *интерфейс локальной файловой системы сервера;*
- *сервер сетевой файловой системы;*

На стороне клиента:

- *клиента сетевой файловой системы;*
- *интерфейс сетевой файловой системы;*

2.1 . Алгоритм взаимодействия клиентов и серверов сетевой файловой системы.

Клиенты обслуживают запросы приложений на доступ к файлам, хранящимся на удаленном компьютере

- Клиент сетевой ФС передает по сети запросы серверу сетевой ФС, работающему на удаленном компьютере.
- Сервер, получив запрос, может выполнить его либо самостоятельно, либо, что является более распространенным вариантом, передать запрос локальной файловой системе для отработки.
- После получения ответа от локальной файловой системы сервер передает его по сети клиенту, а тот, в свою очередь, - приложению, обратившемуся с запросом.

Приложения обращаются к клиенту сетевой ФС, используя определенный программный интерфейс, который в данном случае является интерфейсом

сетевой файловой системы. **Этот интерфейс стараются сделать как можно более похожим на интерфейс локальной файловой системы, чтобы соблюсти принцип прозрачности.**

При полном совпадении интерфейсов приложение может обращаться к локальным и удаленным файлам и каталогам с помощью одних и тех же системных вызовов, совершенно не принимая во внимание места хранения данных.

2.2 Особенности протокола взаимодействия клиента и сервера файловой системы.

2.2.1 Особенность хранения таблицы открытых файлов

Локальная файловая система запоминает состояние последовательных операций, которые приложение выполняет с одним и тем же файлом, за счет ведения внутренней системной таблицы открытых файлов (системные вызовы open, read, write изменяют состояние этой таблицы).

Если таблица открытых файлов хранится на серверном компьютере, то после его перезагрузки, вызванной крахом системы, содержимое этой таблицы теряется, так что приложение, работающее на клиентском компьютере, не может продолжить нормальную работу с открытыми до краха файлами.

Одно из решений этой проблемы основано **на передаче функции ведения и хранения таблицы открытых файлов от сервера клиенту**. Файловый сервер в этом варианте получил название «**stateless**», то есть «**не запоминающий состояния**».

Протокол клиент-сервер при такой организации упрощается, так как перезагрузка сервера приводит только к паузе в обслуживании, но работа с файлами может быть после этого продолжена безболезненно для клиента.

2.2.2 Кэширование.

В случае больших задержек обслуживания из-за заторов в сети и перегрузки файлового сервера при подключении большого числа клиентов. Протокол может для решения этой проблемы организовывать кэширование файлов целиком или частично на стороне клиента. При этом протокол должен

учитывать то обстоятельство, что в сети при этом может образоваться одновременно большое количество копий одного и того же файла, которые независимо могут модифицироваться разными пользователями. То есть протокол должен каким-то образом обеспечивать согласованность копий файлов, имеющихся на разных компьютерах.

2.2.3 Репликация.

Для повышения отказоустойчивости файловой системы в сети можно хранить несколько копий каждого файла (или целиком локальной файловой системы), причем каждую копию - на отдельном компьютере (**реплики (replica) файлов**).

Протокол сетевого доступа к файлам должен учитывать такую организацию файловой службы, например, обращаясь в случае отказа одного файлового сервера к другому, работоспособному и поддерживающему реплику требуемого файла.

Репликация файлов не только повышает отказоустойчивость, но решает также и проблему перегрузки файловых серверов, так как запросы к файлам распределяются между несколькими серверами и повышают производительность сетевой файловой системы.

Репликация в некоторых аспектах похожа на кэширование — в том и другом случаях в сети создается несколько копий одного и того же файла, при этом повышается скорость доступа к данным.

Основным отличием репликации от кэширования является то, что реплики хранятся на файловых серверах, а кэшированные файлы - на клиентах.

Перечисленные проблемы решаются обычно комплексно, в том числе за счет соответствующей организации файловых серверов и клиентов, а также создания специальных служб, таких как служба централизованной репликации.

2.3 Реализация множества протоколов

Все функции файловой службы обязательно находят свое отражение в протоколе взаимодействия клиентов и серверов, в результате чего создаются различные протоколы, поддерживающие тот или иной набор дополнительных функций и в общем случае по-своему решающие проблемы эффективного взаимодействия (иногда эту роль возлагают на отдельные протоколы, которые работают наряду с основным).

Поэтому **для одной и той же локальной файловой системы сервера могут существовать различные протоколы сетевой файловой системы**.

Так, к файловой системе NTFS можно получить доступ с помощью различных протоколов.

С другой стороны, с **помощью одного и того же протокола может реализовываться удаленный доступ к локальным файловым системам разного типа.**

Например, протокол SMB используется для доступа не только к файловой системе FAT, но и к файловым системам NTFS и HPFS. Эти файловые системы могут располагаться как на разных, так и на одном компьютере.

3 Принципы построения сетевых файловых систем

3.1 Модель переноса данных

Вопрос заключается в выборе между двумя моделями:

- модель считывания/записи
- модель удаленного доступа.

В модели считывания/записи (рис. 12), чтобы получить доступ к файлу, процесс сначала считывает его с удаленного сервера, на котором хранится этот файл. Когда процесс заканчивает работу с файлом, обновленный файл отправляется обратно на сервер.

В модели удаленного доступа файл остается на сервере, а клиент посылает серверу команды для выполнения работы на месте.

Преимущество модели считывания/записи заключается в ее простоте и том факте, что перенос файла целиком эффективнее, чем перенос его по частям.

К недостаткам данной модели относится необходимость наличия достаточно большого объема памяти для хранения файла целиком локально, к тому же перенос файла целиком, когда требуется только его часть, представляет собой излишние расходы.

Наконец, при наличии нескольких конкурирующих пользователей возникает проблема непротиворечивости файлов.

3.2 Алгоритмы разделения файлов

Когда два или более пользователей разделяют один файл, необходимо точно определить семантику чтения и записи, чтобы избежать проблем с интерпретацией результирующих данных файла.

3.2.1 Семантика UNIX.

В соответствии с данной моделью система придерживается абсолютного временного упорядочивания всех операций и всегда возвращает самое последнее значение данных.

Если запись осуществляется в открытый многими пользователями файл, то все эти пользователи немедленно видят результат изменения данных файла.

Обычно такая модель называется семантикой UNIX. В централизованной системе, где файлы хранятся в единственном экземпляре, ее легко и понять, и реализовать.

Семантика UNIX может быть обеспечена и в распределенных системах, но только если в ней имеется лишь один файловый сервер и клиенты не кэшируют файлы. Для этого все операции чтения и записи направляются на файловый сервер, который обрабатывает их строго последовательно.

На практике, однако, производительность распределенной системы, в которой все запросы к файлам идут на один сервер, часто становится неудовлетворительной.

Эта проблема иногда решается за счет разрешения клиентам обрабатывать локальные копии часто используемых файлов в своих личных кэшах.

Если клиент сделает локальную копию файла в своем локальном кэше и начнет ее модифицировать, а вскоре после этого другой клиент прочитает этот файл с сервера, то он получит неверную копию файла.

Одним из способов устранения этого недостатка является немедленный возврат всех изменений в кэшируемом файле на сервер.

Такой подход хотя и концептуально прост, но не эффективен. Распределенные файловые системы обычно используют более свободную семантику разделения файлов.

3.2.2 Сеансовая семантика.

В соответствии с этой моделью изменения в открытом файле сначала видны только процессу, который модифицирует файл, и только после закрытия файла эти изменения могут видеть другие процессы.

При использовании сеансовой семантики возникает проблема одновременного использования одного и того же файла двумя или более клиентами.

Одним из решений этой проблемы является принятие правила, в соответствии с которым окончательным является тот вариант, который был закрыт последним. Однако из-за задержек в сети часто оказывается трудно определить, какая из копий файла была закрыта последней. Менее эффективным, но гораздо более простым в реализации является вариант, при котором окончательным результирующим файлом на сервере считается любой из этих файлов, то есть результат операций над файлом не является детерминированным.

3.2.3 Семантика неизменяемых файлов.

Следующий подход к разделению файлов заключается в том, чтобы сделать все файлы неизменяемыми.

Тогда файл нельзя открыть для записи, а можно выполнять только операции

create (создать) и read (читать).

Тогда для изменения файла остается только возможность создать полностью новый файл и поместить его в каталог под новым именем.

Следовательно, хотя файл и нельзя модифицировать, его можно заменить (автоматически) новым файлом. Таким образом, проблема, связанная с одновременным использованием файла, для файловой системы просто исчезнет, но с ней столкнутся пользователи, которые будут вынуждены вести учет имен своих копий модифицированного файла.

3.2.4 Транзакционная семантика.

Четвертый способ работы с разделяемыми файлами в распределенных системах — это использование механизма неделимых транзакций.

3.2.5 Контроль доступа

С каждым разделяемым файлом обычно связан список управления доступом (Access Control List, ACL), обеспечивающий защиту данных от несанкционированного доступа.

В том случае, когда локальная файловая система поддерживает механизм ACL для файлов и каталогов при локальном доступе, сетевая файловая система использует этот механизм и при доступе по сети.

Если же механизм защиты в локальной файловой системе отсутствует, то сетевой файловой системе приходится поддерживать его самостоятельно, иногда — упрощенным способом, защищая разделяемый каталог и входящие в него файлы и подкаталоги как единое целое.

В Windows существуют два механизма защиты - на уровне разделяемых каталогов и на уровне локальных каталогов и файлов. Последний работает только в том случае, когда в качестве локальной файловой системы используется система NTFS, поддерживающая механизм ACL. Механизм защиты разделяемых каталогов нужен для того, чтобы защищать данные, хранящиеся в локальной файловой системе FAT, не имеющей механизмов защиты.

3.3 Варианты распределения серверной и клиентской частей между узлами сети.

Первый вариант.

На всех компьютерах сети работает **одно и то же базовое программное обеспечение**, включающее как клиентскую, так и серверную части, так что любой компьютер, который захочет предложить услуги файловой службы, может это сделать.

Для этого администратору ОС достаточно объявить имена **выбранных каталогов разделяемыми** (экспортируемыми в терминах NFS), чтобы другие машины могли иметь к ним доступ.

В некоторых случаях выпускается так называемая **серверная версия ОС** (например, Windows Server), которая использует то же программное обеспечение файловой службы, но только позволяющее за счет выделения файловому серверу большего количества ресурсов (в основном оперативной памяти) обслуживать одновременно большее число пользователей, чем версии файлового сервера для клиентских компьютеров.

Второй вариант

Файловый сервер — это специализированный компонент серверной ОС, отсутствующий в клиентских компьютерах.

3.4 Взаимодействие клиентов и серверов с операционной системой

Для повышения эффективности работы **файловый сервер и клиент** обычно являются **модулями ядра ОС**, работающими в привилегированном режиме.

В современных ОС эти компоненты чаще всего оформляются как **высокоуровневые драйверы**, работающие в составе подсистемы ввода-вывода.

Эффективность работы при этом повышается за счет прямого доступа ко всем внутренним модулям ОС, в том числе и к дисковому кэшу, без выполнения дополнительных операций и смены пользовательского режима на привилегированный (без переключения контекста).

Прямой доступ к содержимому дискового кэша существенно повышает скорость доступа к данным файлов по сети, поэтому для повышения

производительности файлового сервера ОС должна быть сконфигурирована для поддержки дискового кэша большого объема.

Файловый сервер и клиенты могут в некоторых случаях оформляться и как модули, работающие в **пользовательском режиме**.

Используется такой режим работы в файловых серверах ОС, основанных на **микроядерной архитектуре**. Перенесение сервера в пользовательский режим обусловлено общим подходом к архитектуре ОС, преследующим такие цели, как повышение устойчивости и модифицируемости ОС. Однако работа файлового сервера в пользовательском режиме снижает его производительность, из-за чего на практике такая архитектура применяется пока редко.

3.5 Файловые серверы типа stateful и stateless

Файловый сервер может быть реализован по одной из двух схем:

- с *запоминанием данных о последовательности файловых операций клиента - по схеме* ***stateful***.
- *без запоминания данных о последовательности файловых операций клиента - по схеме* ***stateless***.

Серверы **stateful** работают по схеме, обычной для любой локальной файловой службы. Такой сервер поддерживает тот же набор вызовов, что и локальная система, то есть вызовы **open, read, write, seek и close.**

Открывая файлы по вызову open, переданному по сети клиентом, сервер **stateful** должен запоминать в своей внутренней системной таблице, какие файлы открыл каждый пользователь

Обычно при открытии файла клиентскому приложению возвращается по сети дескриптор файла **fd** или другое число, которое используется при последующих вызовах для идентификации файла.

При поступлении вызова **read, write** или **seek** сервер использует дескриптор файла для определения, какой файл нужен. В этой таблице хранится также значение указателя на текущую позицию в файле, относительно которой выполняется операция чтения или записи.

Сервер **stateless** после получения запроса от клиента, выполняет его, отсылает ответ, а затем удаляет из своих внутренних таблиц всю информацию о запросе.

Между запросами на сервере не хранится никакой текущей информации о состоянии клиента. Для сервера stateless каждый запрос должен содержать исчерпывающую информацию (полное имя файла, смещение в файле и т. п.), необходимую серверу для выполнения требуемой операции. Очевидно, что эта информация увеличивает длину сообщения и время, которое тратит сервер на локальное открытие файла каждый раз, когда над ним производится очередная операция чтения или записи.

Серверы, работающее по схеме stateless, не поддерживают в протоколе обмена с клиентами таких операций, как открытие (**open**) и закрытие (**close**) файлов.

Принципиально набор команд, предоставляемый клиенту сервером stateless, может состоять только из двух команд: **read** и **write** . Эти команды должны передавать в своих аргументах всю необходимую для сервера информацию — имя файла, смещение от начала файла и количество читаемых или записываемых байт данных.

Для того чтобы обеспечить приложениям, работающим на клиентских машинах, привычный файловый интерфейс, включающий вызовы открытия и закрытия файлов, клиент файловой службы должен самостоятельно поддерживать таблицы открытых его приложениями файлов.

Основным последствием переноса таблиц открытых файлов на клиентов при применении серверов stateless является реакция файловой службы на отказ сервера.

При отказе сервера **stateful** теряются все его таблицы, и после перезагрузки неизвестно, какие файлы открыл каждый пользователь. Последовательные попытки провести операции чтения или записи с открытыми файлами будут безуспешными.

Серверы **stateless** в этом плане являются более отказоустойчивыми, позволяя клиентам продолжать операции с открытыми файлами, и это является основным аргументом в пользу их применения.

Платой за отказоустойчивость может быть скорость работы сервера, так как ему приходится выполнять больше операций с файлами.

Кроме того, применение серверов stateless затрудняет реализацию блокировок файлов, так как информацию о блокировке файла одним из пользователей необходимо запоминать на всех клиентах файлового сервера.

Преимущества каждого из подходов можно обобщить следующим образом.

Серверы **stateless**:

- отказоустойчивы;
- не нужны вызовы open/close;
- меньше памяти сервера расходуется на таблицы;
- нет ограничений на число открытых файлов;
- отказ клиента не создает проблем для сервера.

Серверы **stateful:**

- более короткие сообщения при запросах;
- лучше производительность;
- возможно опережающее чтение;
- возможна блокировка файлов.

3.6 Кэширование в сетевых файловых системах

Кэширование широко используется в сетевых файловых системах. Оно позволяет не только повысить скорость доступа к удаленным данным (это по-прежнему является основной целью кэширования), но и улучшить **масштабируемость** и надежность файловой системы.

Схемы кэширования, применяемые в сетевых файловых системах, отличаются решениями по трем ключевым вопросам:

месту расположения кэша;

способу согласования копий;

проверке достоверности кэша.

Кроме того, на схему кэширования влияет выбранная в файловой системе модель переноса файлов между сервером **и** клиентами:

модель **загрузки-выгрузки**, переносящая файл целиком,

или **модель удаленного доступа**, позволяющая переносить файл по частям.

Соответственно в первом случае файл кэшируется целиком, а во втором кэшируются только те части файла, к которым выполняется обращение.

Место расположения кэша

В системах, состоящих из клиентов и серверов, потенциально имеются три различных места для хранения кэшируемых файлов и их частей:

память сервера,

диск клиента (если имеется)

и память клиента.

Память сервера практически всегда используется для кэширования файлов, к которым обращаются по сети пользователи и приложения.

Кэширование в памяти сервера уменьшает время доступа по сети за счет исключения времени обмена с диском сервера. При этом файловый сервер может использовать для кэширования своих файлов существующий механизм локального кэша операционной системы, не применяя никакого дополнительного программного кода.

Однако кэширование только в памяти сервера не решает всех проблем — скорость доступа по-прежнему снижают задержки, вносимые сетью, и перегруженность процессора сервера при одновременном обслуживании большого потока сетевых запросов от многочисленных клиентов.

Кэширование на стороне клиента. Это способ кэширования файлов, который подразделяется на кэширование на диске клиента и в памяти клиента,

исключает обмен по сети и освобождает сервер от работы после переноса файла на клиентский компьютер.

Использование **диска** на стороне клиента позволяет кэшировать большие файлы, что особенно важно при применении модели загрузки-выгрузки, переносящей файлы целиком. Диск также является более надежным устройством хранения информации по сравнению с оперативной памятью. Однако такой способ кэширования вносит дополнительные задержки доступа, связанные с чтением данных с клиентского диска, а также неприменим на бездисковых компьютерах.

Кэширование в **оперативной памяти** клиента позволяет ускорить доступ к данным, но ограничивает размер кэшируемых данных объемом памяти, выделяемой под кэш, что может стать существенным ограничением при применении модели загрузки-выгрузки.

Согласования копий файлов.

Существование в одно и то же время в сети нескольких копий одного и того же файла, хранящихся в кэшах клиентов, порождает проблему согласования копий.

Для решения этой проблемы необходимо, чтобы модификации данных, выполненные над одной из копий, были своевременно распространены на все остальные копии. Существует несколько вариантов распространения модификаций. От выбранного варианта в значительной степени зависит **семантика разделения файлов.**

Алгоритма сквозной записи.

Когда кэшируемый элемент (файл или блок) модифицируется, новое значение записывается в кэш и одновременно посылается на сервер для обновления главной копии файла. Теперь другой процесс, читающий этот файл, получает самую последнюю версию данных.

Один из недостатков алгоритма сквозной записи состоит в том, что он уменьшает интенсивность сетевого обмена только при чтении, при записи интенсивность сетевого обмена та же самая, что и без кэширования.

Алгоритм отложенной записи:

Вместо того чтобы выполнять запись на сервер, клиент просто помечает, что файл изменен. Примерно каждые 30 секунд все изменения в файлах собираются вместе и отсылаются на сервер за один прием. Одна большая запись для сетевого обмена обычно более эффективна, чем множество коротких.

Алгоритм «запись по закрытию»

Запись файла на сервер производится только после закрытия файла. Этот и приводит к тому, что если две копии одного файла кэшируются на разных машинах и последовательно записываются на сервер, то второй записывается поверх первого. Данная схема не может снизить сетевой трафик, если объем изменений за сеанс редактирования файла невелик.

Для всех схем, связанных с задержкой записи, характерна низкая надежность, так как модификации, не отправленные на сервер на момент краха системы, теряются. Кроме того, задержка делает семантику совместного использования файлов не очень ясной, поскольку данные, которые считывает процесс из файла, зависят от соотношения момента чтения с моментом очередной записи модификаций.

Проверка достоверности кэша

Распространение модификаций решает только проблему согласования главной копии файла, хранящейся на сервере, с клиентскими копиями.

В то же время этот прием не дает никакой информации о том, когда должны обновляться данные**,** находящиеся в кэшах клиентов: данные в кэше одного клиента становятся недостоверными, когда данные, модифицированные другим клиентом, переносятся в главную копию файла.

Следовательно, необходимо каким-то образом проверять, являются ли данные в кэше клиента достоверными. В противном случае данные кэша должны быть повторно считаны с сервера.

Существуют два подхода к решению этой проблемы:

инициирование проверки клиентом

инициирование проверки сервером.

В первом случае клиент связывается с сервером и проверяет, соответствуют ли данные в его кэше данным главной копии файла на сервере.

Клиент может выполнять такую проверку одним из трех способов:

- Перед каждым доступом к файлу. Этот способ дискредитирует саму идею кэширования, так как каждое обращение к файлу вызывает обмен по сети с сервером.
- Периодические проверки. Улучшают производительность, но делают семантику разделения неясной, зависящей от временных соотношений.
- Проверка при открытии файла

Проверки достоверности кэша при инициировании проверки сервером(метод централизованного управления)

Когда файл открывается, то клиент, выполняющий это действие, посылает соответствующее сообщение файловому серверу, в котором указывает режим открытия файла - чтение или запись.

Файловый сервер сохраняет информацию о том, кто и какой файл открыл, а также о том, открыт файл для чтения, для записи или для того и другого.

Если файл открыт для чтения, то нет никаких препятствий для разрешения другим процессам открыть его для чтения, по открытие его для записи должно быть запрещено. Аналогично, если некоторый процесс открыл файл для записи, то все другие виды доступа должны быть запрещены.

При закрытии файла также необходимо оповестить файловый сервер для того, чтобы он обновил СБОИ таблицы, содержащие данные об открытых файлах. Модифицированный файл также может быть выгружен на сервер в такой момент.

Когда новый клиент делает запрос на открытие уже открытого файла и сервер обнаруживает, что режим нового открытия входит в противоречие с режимом текущего открытия, то сервер может ответить на такой запрос следующими способами;

- отвергнуть запрос;
- поместить запрос в очередь;

- запретить кэширование для данного конкретного файла,

потребовав от всех клиентов, открывших этот файл, удалить его из кэша.

Подход, основанный на централизованном управлении, весьма эффективен, обеспечивает семантику разделения UNIX, но обладает следующими недостатками:

- Он отклоняется от традиционной модели взаимодействия клиента и сервера, при которой сервер только отвечает на запросы, инициированные клиентами. Это делает код сервера нестандартным и достаточно сложным.
- Сервер обязательно должен хранить информацию о состоянии сеансов клиентов, то есть иметь тип **stateful.**
- По-прежнему должен использоваться механизм инициирования проверки достоверности клиентами при открытии файлов.

3.7 Репликация

Сетевая файловая система может поддерживать репликацию (тиражирование) файлов в качестве одной из услуг, предоставляемых клиентам.

Иногда репликацией занимается отдельная служба ОС.

Репликация {replication) подразумевает существование нескольких копий одного и того же файла, каждая из которых хранится на отдельном файловом сервере, при этом обеспечивается автоматическое согласование данных в копиях файла.

Имеется несколько причин для применения репликации, главными из которых являются две:

- **Увеличение надежности** за счет наличия независимых копий каждого файла, хранящихся на разных файловых серверах. При отказе одного из них файл остается доступным.
- **Распределение нагрузки** между несколькими серверами. Клиенты могут обращаться к данным реплицированного файла на ближайший файловый сервер, хранящий копию этого файла. Ближайшим

может считаться сервер, находящийся в той же подсети, что и клиент, или же первый откликнувшийся, или же выбранный некоторым сетевым устройством, балансирующим нагрузки серверов.

Репликация похожа на кэширование файлов на стороне клиентов тем, что в системе создается несколько копий одного файла. Однако существуют и принципиальные отличия репликации от кэширования, прежде всего в преследуемых целях.

Если кэширование предназначено для обеспечения локального доступа к файлу одному клиенту и повышения за счет этого скорости работы этого клиента, то репликация нужна для повышения надежности хранения файлов и снижения нагрузки на файловые серверы. Реплики файла доступны всем клиентам, так как хранятся на файловых серверах, а не на клиентских компьютерах, и о существовании реплик известно всем компьютерам сети. Файловая система обеспечивает достоверность данных реплики и защиту ее данных.

Прозрачность репликации

Ключевым вопросом, связанным с репликацией, является прозрачность.

До какой степени пользователи должны быть в курсе того, что некоторые файлы реплицируются?

Должны они играть какую-либо роль в процессе **репликации** или репликация должна выполняться полностью автоматически?

В одних системах пользователи полностью вовлечены в этот процесс, в других **система** все делает без их ведома.

В последнем случае говорят, что **система репликационно прозрачна**.

Прозрачность репликации зависит от двух факторов:

используемой схемы именования реплик

степени вовлеченности пользователя в управление репликацией.

Именование реплик. Прозрачность доступа к файлу, существующему в виде нескольких реплик, может поддерживаться системой именования, которая отображает **имя файла на его сетевой идентификатор**, однозначно определяющий место хранения файла.

Если в сети используется централизованная справочная служба, которая позволяет хранить отображения имен объектов на их сетевые идентификаторы (например, IP-адреса серверов), то для реплицированных файлов допустима прозрачная схема именования.

В этом случае файлу присваивается имя, не содержащее старшей части, соответствующей имени компьютера.

В справочной службе этому имени соответствует несколько идентификаторов, указывающих на серверы, хранящие реплики файла. При обращении к такому файлу приложение использует имя, а справочная система возвращает ему один из идентификаторов, указывающий на сервер, хранящий реплику.

Наиболее просто такую схему реализовать для неизменяемых файлов, все реплики которых всегда (или почти всегда, если файлы редко, но все же изменяются) идентичны. В случае, когда реплицируются изменяемые файлы, полностью прозрачный доступ требует ведения некоторой базы, хранящей сведения о том, какие из реплик содержат последнюю версию данных, а какие еще не обновлены. Для поддержки полностью прозрачной системы репликации необходимо также постоянное использование в файловом интерфейсе имен, не зависящих от местоположения, то есть не содержащих старшей части с указанием имени сервера. В более распространенной на сегодня схеме именования требуется явное указание имени сервера при обращении к файлу, что приводит к не полностью прозрачной системе репликации.

Управление репликацией. Этот процесс подразумевает определение количества реплик и выбор серверов для хранения каждой реплики. В прозрачной системе репликации такие решения принимаются автоматически при создании файла на основе правил стратегии репликации, определенных заранее администратором системы. В непрозрачной системе репликации решения о количестве реплик и их размещении принимаются с участием пользователя, который создает файлы, или разработчика приложения, если файлы создаются в

автоматическом режиме. В результате существуют два режима управления репликацией:

- При ***явной репликации (explicit replication)*** пользователю (или прикладному программисту) предоставляется возможность управления процессом репликации. При создании файла сначала создается первая реплика с явным указанием сервера, на котором размещается файл, а затем создается несколько реплик, причем для каждой реплики сервер также указывается явно. Пользователь при желании может впоследствии удалить одну или несколько реплик. Явная репликация не исключает автоматического режима поддержания согласованности реплик, которые создал и разместил на серверах пользователь.
- При ***неявной репликации (implicit replication)*** выбор количества и места размещения реплик производится в автоматическом режиме, без участия пользователя. Приложение не должно указывать место размещения файла при запросе на его создание. Файловая система самостоятельно выбирает сервер, на который помещает первую реплику файла. Затем в фоновом режиме система создает несколько реплик этого файла, выбирая их количество и серверы для их размещения. Если надобность в некоторых репликах исчезает (это также определяется автоматически), то система их удаляет.

Согласование реплик

Согласование реплик, обеспечивающее хранение в каждой реплике последней версии данных файла, является одним из наиболее важных вопросов при разработке системы репликации.

Так как изменяемые файлы являются наиболее распространенным типом файлов, то в какой-то момент времени данные в одной из реплик модифицируются и требуется предпринять усилия для распространения модификаций по всем остальным репликам.

Существует несколько способов обеспечения согласованности реплик.

Чтение любой — запись во все (Read-Any — Write-All). При необходимости записи в файл все реплики файла блокируются, затем выполняется запись в каждую копию, после чего блокировка снимается и файл становится доступным для чтения. Чтение может выполняться из любой копии. Недостатком является то, что запись не всегда можно осуществить, так как некоторые серверы, хранящие реплики файла, могут в момент записи быть неработоспособными.

Запись в доступные (Available-Copies). Этот метод снимает ограничение предыдущего, так как запись выполняется только в те копии, серверы которых доступны на момент записи. Чтение осуществляется из любой реплики файла, как и в предыдущем методе. Любой сервер, хранящий реплику файла, после перезагрузки должен соединиться с другим сервером и получить от него обновленную копию файла и только потом начать обслуживать запросы на чтение из файла. Для обнаружения отказавших серверов в системе должен работать специальный процесс, постоянно опрашивающий состояние серверов. Недостатком метода является возможность появления несогласованных копий файла из-за коммуникационных проблем в сети, когда невозможно выявить отказавший сервер.

Первичная реплика (Primary-Copy). В этом методе запись разрешается только в одну реплику файла, называемую первичной (primary). Все остальные реплики файла являются вторичными (secondary), и из них можно только читать данные. После модификации первичной реплики все серверы, хранящие вторичные реплики, должны связаться с сервером, поддерживающим первичную реплику, и получить от него обновления. Этот процесс может инициироваться как первичным сервером, так и вторичными (периодически проверяющими состоянии первичной реплики). Это метод является одной из реализаций метода «чтение любой — запись во все», в которой процесс записи реализован иерархическим способом. Для аккумулирования нескольких модификаций и сокращения сетевого трафика распространение модификаций может быть выполнено с запаздыванием, но в этом случае ухудшается согласованность

копий. Недостатком метода является его низкая надежность — при отказе первичного сервера модификации файов невозможны (для решения этой проблемы необходимо повысить статус некоторого вторичного сервера до первичного).

Кворум (Quorum). Этот метод обобщает подходы, реализованные в предыдущих методах.

Пусть в сети существует ***n*** реплик некоторого файла.

Алгоритм основывается та том. что при модификации файла изменения записываются в ***w*** реплик, а при чтении файла клиент обязательно производит обращение к ***r*** репликам.

Значения ***w*** и ***r*** выбираются так, что ***w+r>n***.

При чтении клиент должен иметь возможность сначала проверить версию каждой реплики, а затем выбрать старшую и читать данные уже из нее.

Очевидно, что при модификации файла номер версии должен наращиваться, а если при записи в ***w*** реплик они имеют разные версии, то выбирается максимальное значение версии для наращивания и присваивания всем репликам.

При выполнении условия ***w+r> n*** ,среди любых выбранных произвольным образом ***r*** реплик всегда найдется хотя бы одна из ***w*** реплик, в которую были записаны последние обновления.