Распределенные файловые системы

Кэширование и репликация

Кэширование

- В системах, состоящих из клиентов и серверов, потенциально имеется четыре различных места для хранения файлов и их частей: диск сервера, память сервера, диск клиента (если имеется) и память клиента.
- Наиболее подходящим местом для хранения всех файлов является **диск сервера**. Он обычно имеет большую емкость, и файлы становятся доступными всем клиентам. Кроме того, поскольку в этом случае существует только одна копия каждого файла, то не возникает проблемы согласования состояний копий.
- Проблемой при использовании диска сервера является **производительность**. Перед тем, как клиент сможет прочитать файл, файл должен быть переписан с диска сервера в его оперативную память, а затем передан по сети в память клиента. Обе передачи занимают время.

Кэширование в памяти сервера

- Значительное увеличение производительности может быть достигнуто за счет кэширования файлов **в памяти сервера**. Требуются алгоритмы для определения, какие файлы или их части следует хранить в кэш-памяти.
- При выборе алгоритма должны решаться две задачи.

  Во-первых, какими единицами оперирует кэш. Этими единицами могут быть или дисковые блоки, или целые файлы. Если это целые файлы, то они могут храниться на диске непрерывными областями (по крайней мере в виде больших участков), при этом уменьшается число обменов между памятью и диском а, следовательно, обеспечивается высокая производительность. Кэширование блоков диска позволяет более эффективно использовать память кэша и дисковое пространство.

Кэширование в памяти сервера

- Во-вторых, необходимо определить правило замены данных при заполнении кэш-памяти. Здесь можно использовать любой стандартный алгоритм кэширования, например, алгоритм LRU (least recently used), соответствии с которым вытесняется блок, к которому дольше всего не было обращения.
- Кэш-память на сервере легко реализуется и совершенно прозрачна для клиента. Так как сервер может синхронизировать работу памяти и диска, с точки зрения клиентов существует только одна копия каждого файла, так что проблема согласования не возникает.

Кэширование на стороне клиента

- Хотя кэширование на сервере исключает обмен с диском при каждом доступе, все еще остается обмен по сети. Существует только один путь избавиться от обмена по сети - это кэширование на стороне клиента, которое, однако, порождает много сложностей.
- Кэширование **на диске клиента** может не дать преимуществ перед кэшированием в памяти сервера, а сложность повышается значительно.
- Кэширование **в памяти клиента**, а не на его диске используется во многих системах. Существуют 3 способа: кэширование в каждом процессе, кэширование в ядре, кэш-менеджер в виде отдельного процесса.

Кэширование на стороне клиента

Самый простой состоит в кэшировании файлов непосредственно **внутри адресного пространства каждого пользовательского процесса**.

Обычно кэш управляется с помощью библиотеки системных вызов. По мере того, как файлы открываются, закрываются, читаются и пишутся, библиотека просто сохраняет наиболее часто используемые файлы. Когда процесс завершается, все модифицированные файлы записываются назад на сервер.

Хотя эта схема реализуется с чрезвычайно низкими издержками, она эффективна только тогда, когда отдельные процессы часто повторно открывают и закрывают файлы.

Таким является процесс менеджера базы данных, но обычные программы чаще всего читают каждый файл однократно, так что кэширование с помощью библиотеки в этом случае не дает выигрыша.

Кэширование на стороне клиента

Другим местом кэширования является **ядро**.

Недостатком этого варианта является то, что во всех случаях требуется выполнять системные вызовы, даже в случае успешного обращения к кэш-памяти (файл оказался в кэше).

Но преимуществом является то, что файлы остаются в кэше и после завершения процессов.

Например, предположим, что двухпроходный компилятор выполняется, как два процесса. Первый проход записывает промежуточный файл, который читается вторым проходом. После завершения процесса первого прохода промежуточный файл, вероятно, будет находиться в кэше, так что вызов сервера не потребуется.

Кэширование на стороне клиента

Третьим вариантом организации кэша является создание отдельного **процесса пользовательского уровня - кэш-менеджера**.

Преимущество этого подхода заключается в том, что ядро освобождается от кода файловой системы и тем самым реализуются все достоинства микроядер.

С другой стороны, когда ядро управляет кэшем, оно может динамически решить, сколько памяти выделить для программ, а сколько для кэша.

Когда же кэш-менеджер пользовательского уровня работает на машине с виртуальной памятью, то понятно, что ядро может решить выгрузить некоторые, или даже все страницы кэша на диск, так что для так называемого "попадания в кэш" требуется подкачка одной или более страниц.

Если в системе имеется возможность фиксировать некоторые страницы в памяти, то такая парадоксальная ситуация может быть исключена.

Консистентность кэшей

Кэширование в клиенте создает серьезную проблему - **сложность поддержания кэшей в согласованном состоянии**.

- ***Алгоритм со сквозной записью***.

  Этот алгоритм, при котором модифицируемые данные пишутся в кэш и сразу же посылаются серверу, не является решением проблемы. При его использовании в мультипроцессорах все кэши “подслушивали” шину, через которую там осуществляются все “сквозные” записи в память, и сразу же обновляли находящиеся в них данные. В распределенной системе такое “подслушивание” невозможно, а требуется перед использованием данных из кэша проверять, не устарела ли информация в кэше.

  Уменьшает интенсивность сетевого обмена только при чтении, при записи интенсивность сетевого обмена та же самая, что и без кэширования.

Консистентность кэшей

Алгоритм с отложенной записью. Через регулярные промежутки времени все модифицированные блоки пишутся в файл (так на традиционных ЭВМ работает ОС UNIX).

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 обычно более эффективна, чем много маленьких.

Эффективность выше, но семантика непонятна пользователю.

Алгоритм записи в файл при закрытии файла. Реализует семантику сессий. Такой алгоритм, на первый взгляд, кажется очень неудачным для ситуаций, когда несколько процессов одновременно открыли один файл и модифицировали его. Однако, аналогичная картина происходит и на традиционной ЭВМ, когда два процесса на одной ЭВМ открывают файл, читают его, модифицируют в своей памяти и пишут назад в файл.

Консистентность кэшей

Алгоритм централизованного управления.

Когда файл открыт, машина, открывшая его, посылает сообщение файловому серверу, чтобы оповестить его об этом факте.

Файл-сервер сохраняет информацию о том, кто открыл какой файл, и о том, открыт ли он 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, но открытие его для записи должно быть запрещено. Аналогично, если некоторый процесс открыл файл для записи, то все другие виды доступа должны быть предотвращены. При закрытии файла также необходимо оповестить файл-сервер для того, чтобы он обновил свои таблицы, содержащие данные об открытых файлах.

Модифицированный файл также может быть выгружен на сервер в такой момент.

Неэффективно, ненадежно, и плохо масштабируется.

Консистентность кэшей

Алгоритм со сквозной записью.

Этот метод эффективен частично, так как уменьшает интенсивность только операций чтения, а интенсивность операций записи остается неизменной

Алгоритм с отложенной записью.

Производительность лучше, но результат чтения кэшированного файла не всегда однозначен.

Алгоритм записи в файл при закрытии файла.

Удовлетворяет сессионной семантике.

Алгоритм централизованного управления.

Ненадежен вследствие своей централизованной природы.

Кэширование. Итоги

Кэширование на сервере несложно реализуется и почти всегда дает эффект, независимо от того, реализовано кэширование у клиента или нет.

Кэширование на сервере не влияет на семантику файловой системы, видимую клиентом.

Кэширование на стороне клиента дает увеличение производительности, но требует поддержания кэшей в согласованном состоянии, увеличивает сложность семантики.

Размножение файлов

Система может предоставлять такой сервис, как поддержание для указанных файлов нескольких копий на различных серверах.

Главные цели:

1) Повысить надежность.
2) Повысить доступность (крах одного сервера не вызывает недоступность размноженных файлов).
3) Распределить нагрузку на несколько серверов.

Ключевым вопросом, связанным с размножением файлов является **прозрачность**.

До какой степени пользователи должны быть в курсе того, что некоторые файлы размножаются? Должны ли они играть какую-либо роль в процессе репликации или размножение должно выполняться полностью автоматически?

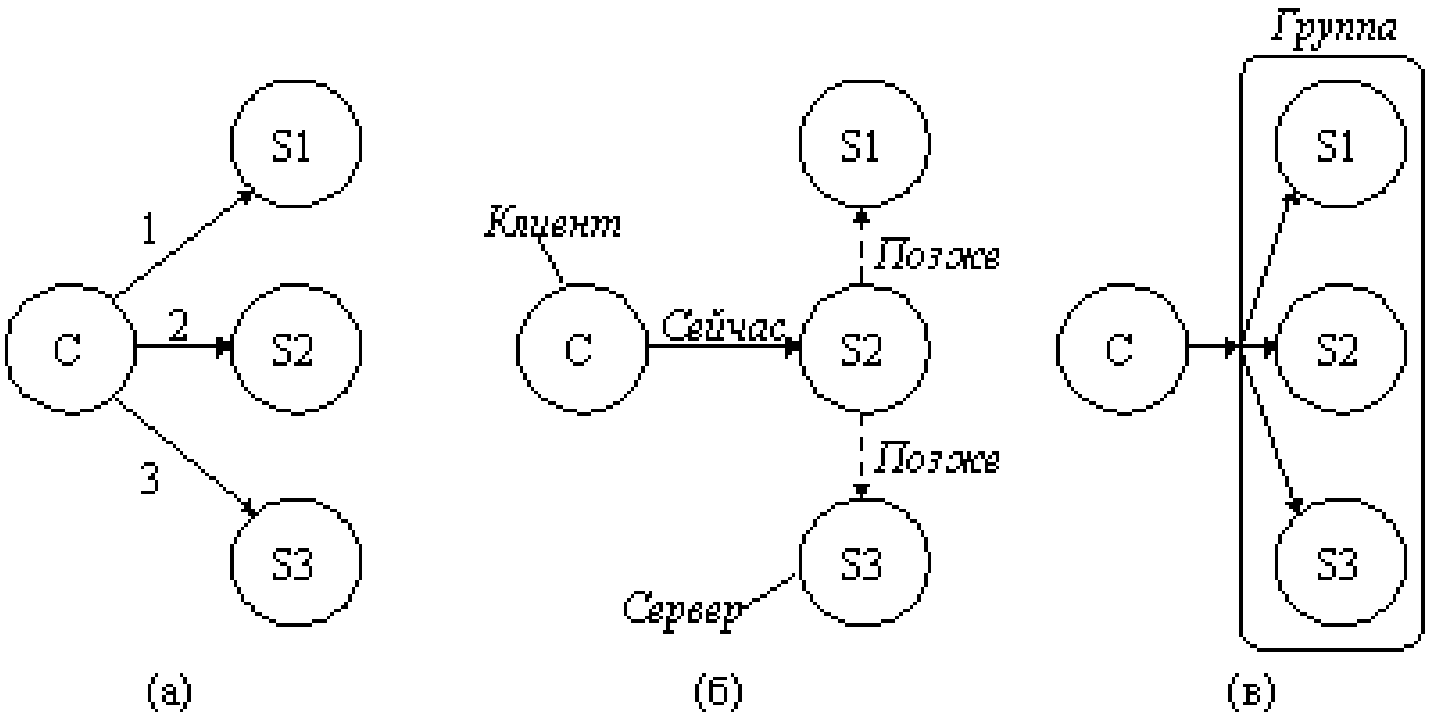
В одних системах пользователи полностью вовлечены в этот процесс, в других система все делает без их ведома.

Размножение файлов

Имеются три схемы реализации размножения:

- **Явное размножение** (непрозрачно). В ответ на открытие файла пользователю выдаются несколько двоичных имен, которые он должен использовать для явного дублирования операций с файлами.
- **«Ленивое» размножение**. Сначала копия создается на одном сервере, а затем он сам автоматически создает (в свободное время) дополнительные копии и обеспечивает их поддержание.
- **Симметричное размножение**. Все операции одновременно вызываются в нескольких серверах и одновременно выполняются.

Размножение файлов



Протоколы коррекции

Просто посылка сообщений с операцией коррекции каждой копии является не очень хорошим решением, поскольку в случае аварий некоторые копии могут остаться не скорректированными.

Имеются три алгоритма, которые решают эту проблему:

1) **Метод размножения главной копии.**

   Один сервер объявляется главным, а остальные - подчиненными. Все изменения файла посылаются главному серверу. Он сначала корректирует свою локальную копию, а затем рассылает подчиненным серверам указания о коррекции. Чтение файла может выполнять любой сервер. Для защиты от рассогласования копий в случае краха главного сервера до завершения им рассылки всех указаний о коррекции, главный сервер до выполнения коррекции своей копии запоминает в стабильной памяти задание на коррекцию.

   Слабость - выход из строя главного сервера не позволяет выполнять коррекции.

Протоколы коррекции

2) **Метод одновременной коррекции всех копий.**

Все изменения файла посылаются (используя надежные и неделимые широковещательные рассылки) всем серверам.

Чтение файла может выполнять любой сервер.

Протоколы коррекции

3) **Метод голосования.**

Идея - запрашивать чтение и запись файла у многих серверов (запись - у всех!).

Для успешного выполнения записи требуется, чтобы Nw серверов ее выполнили. При этом у всех этих серверов должно быть согласие относительно номера текущей версии файла. Этот номер увеличивается на единицу с каждой коррекцией файла.

Для выполнения чтения достаточно обратиться к Nr серверам и воспользоваться одним из тех, кто имеет последнюю версию файла. Значения для кворума чтения (Nr) и кворума записи (Nw) должны удовлетворять соотношению $Nr+Nw>N$.

Поскольку чтение является более частой операцией, то естественно взять $Nr=1$. Однако в этом случае для кворума записи потребуются все серверы.

Протоколы коррекции

3’) **Метод голосования с «приведениями»**

Выход из строя нескольких серверов приводит к отсутствию кворума для записи.

Голосование с приведениями решает эту проблему путем создания фиктивного сервера без дисков для каждого отказавшего или отключенного сервера.

Фиктивный сервер не участвует в кворуме чтения (прежде всего, у него нет файлов), но он может присоединиться к кворуму записи, причем он просто записывает в никуда передаваемый ему файл. Запись только тогда успешна, когда хотя бы один сервер настоящий.

Когда отказавший сервер перезапускается, то он должен обнаружить последнюю версию файла, которую он копирует к себе перед тем, как начать обычные операции.

Кэширование и репликация в NFS

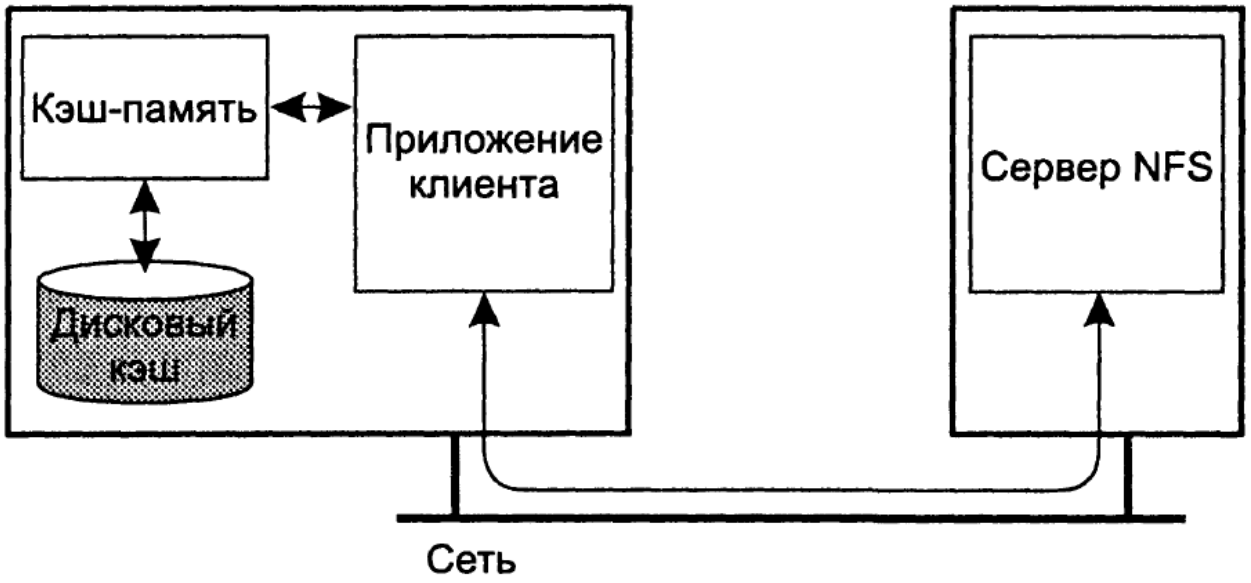
Кэширование в NFS версии 3 в основном не было определено в протоколах. Подобный подход приводил к созданию различных методик кэширования, большинство из которых никогда не гарантировали непротиворечивости.

В лучшем случае кэшированные данные пребывали в устаревшем состоянии в течение нескольких секунд, необходимых для сравнения их с данными, хранящимися на сервере.

Существовали и реализации, для которых вполне естественно было оставлять кэшированные данные устаревшими в течение 30 секунд без уведомления клиента.

Кэширование и репликация в NFS

В NFS версии 4 некоторые из этих проблем непротиворечивости были решены.



Клиенты могут кэшировать данные из файлов, атрибуты, дескрипторы файлов и каталогов.

Кэширование и репликация в NFS

NFS версии 4 поддерживает два способа кэширования данных.

Наиболее простой из них состоит в том, что клиент открывает файл и кэширует содержащиеся в нем данные, получаемые с сервера в результате различных операций чтения. Кроме того, над кэшированными данными можно осуществлять и операцию записи.

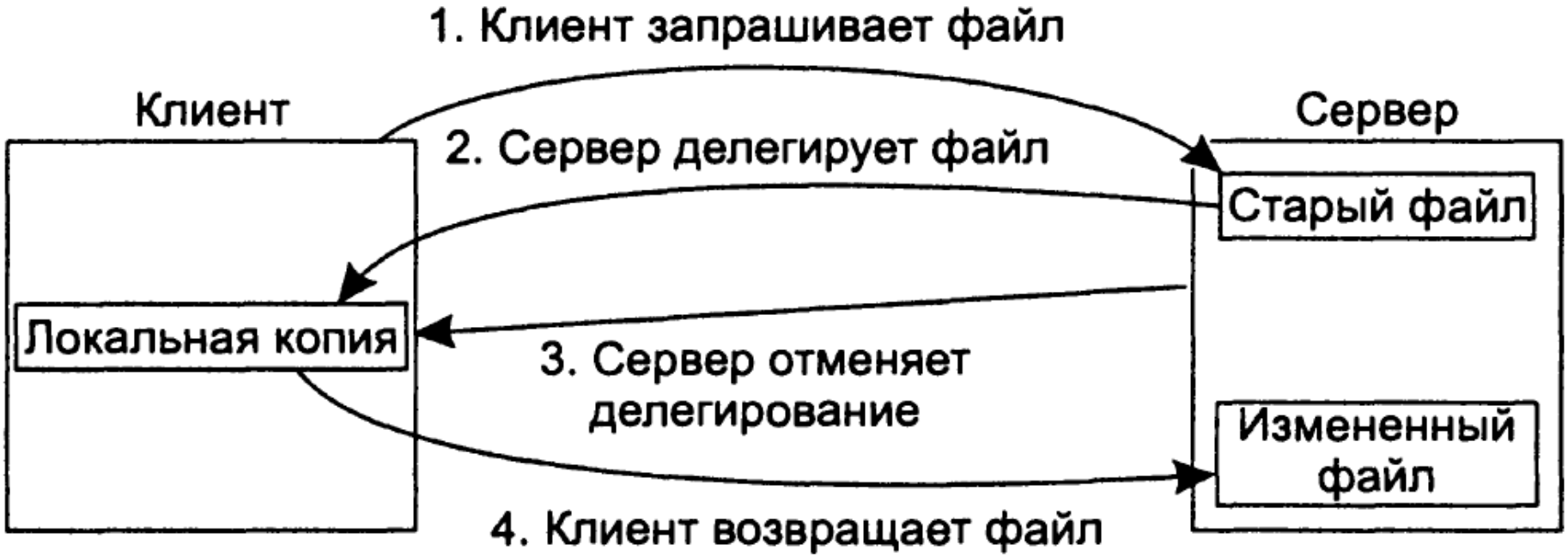
После того как клиент закрывает файл, NFS требует, чтобы в том случае, если в кэшированные данные были внесены изменения, они были переданы обратно на сервер. Подобный подход представляет собой не что иное, как реализацию семантики сеансов.

Как только хотя бы часть файла оказывается кэшированной, клиент может хранить соответствующие данные в кэше даже после закрытия файла. Кроме того, несколько клиентов на одной машине могут использовать один и тот же кэш совместно. NFS требует, чтобы когда бы клиент ни открыл ранее закрытый файл с даже частично кэшированными данными, он должен немедленно перепроверить эти данные.

Подобная проверка включает в себя проверку времени внесения в файл последнего изменения и обновление кэша, если он содержит устаревшие данные.

Кэширование и репликация в NFS

Новым в спецификации NFS версии 4 является то, что сервер после открытия файла может делегировать часть своих прав клиенту. В том случае, если машине клиента разрешается локально обрабатывать операции открытия и закрытия файлов других клиентов той же машины, имеет место ***делегирование открытия.***



Кэширование и репликация в NFS

Клиенты могут также кэшировать значения атрибутов. Модификации значений атрибутов немедленно передаются на сервер в соответствии с правилами согласованности кэша сквозной записи.

Таким же образом выполняется кэширование дескрипторов файлов (или, скорее,отображений имен файлов в дескрипторы) и каталогов.

Чтобы снизить отрицательный эффект от возможной в этом случае противоречивости, NFS использует аренду кэшируемых атрибутов, дескрипторов файлов и каталогов. После истечения определенного периода времени хранимая в кэше информация автоматически объявляется неверной. В результате для того, чтобы вновь использовать эти данные, их необходимо обновить с сервера.

Репликация в NFS

NFS версии 4 предоставляет минимальную поддержку репликации файлов. Реплицирована может быть только файловая система целиком (то есть все логическое устройство, включая файлы, атрибуты, каталоги и блоки данных). Поддержка предоставляется в виде атрибута FS_LOCATIONS, который для всех файлов является рекомендуемым.

Этот атрибут поддерживает список мест, в которых может находиться файловая система, содержащая данный файл. Каждое из этих мест указывается в форме DNS-имени или IP-адреса.

NFS версии 4 не определяет, каким образом должна происходить репликация.

Кэширование и репликация в Coda

Кэширование на клиенте выполняется для улучшения масштабируемости.

Кэширование повышает отказоустойчивость, поскольку клиент меньше зависит от доступности сервера.

Клиенты в Coda всегда кэшируют файлы целиком.

Когда файл открывается — на чтение или на запись, — клиенту передается полная копия файла, которая затем попадает в кэш.

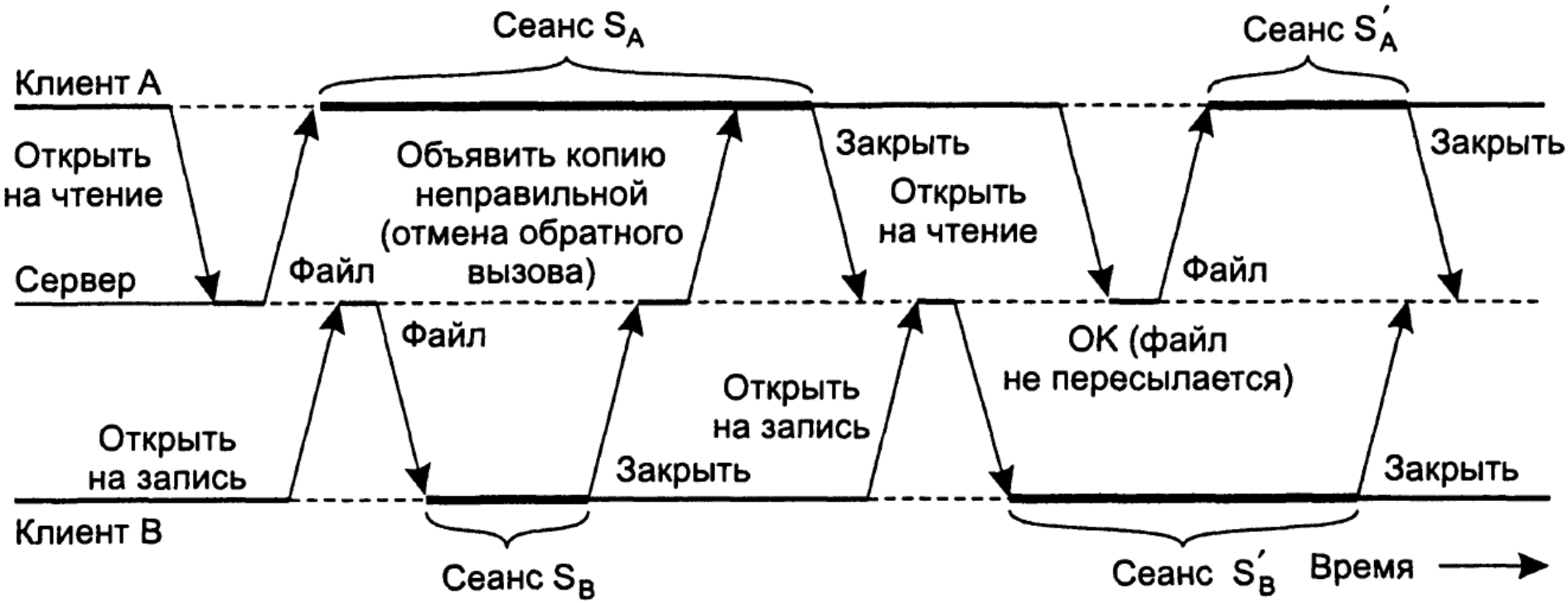
Согласованность кэша в Coda обеспечивается при помощи обратных вызовов. Для каждого файла сервер хранит информацию обо всех клиентах, которым была выслана копия этого файла. Сервер записывает для клиента ***обещание обратного вызова.***

После того как клиент изменит свою локальную копию файла в первый раз, он уведомляет об этом сервер, который, в свою очередь, рассылает остальным клиентам сообщения о некорректности их копий.

Таким образом, пока клиент знает, что он оставил серверу обещание обратного вызова, он благополучно работает с файлом локально.

Клиент может проверить на сервере, действительно ли еще это обещание. Если оно действительно, значит, у клиента нет необходимости снова получать файл у сервера.

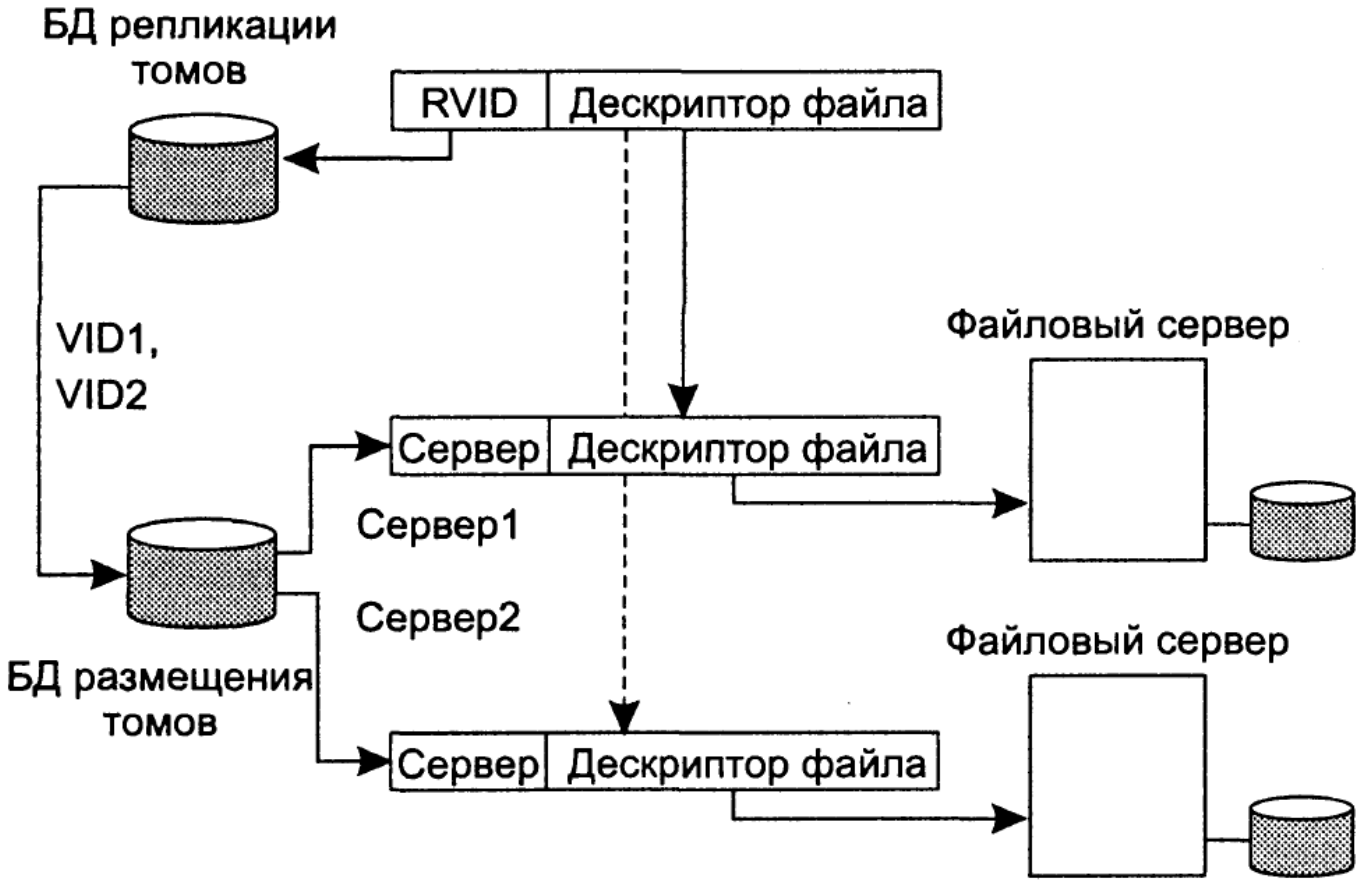
Кэширование и репликация в Coda



Кэширование и репликация в Coda

RVID -Replicated Volume Identifier, указывает на логический том

VID -Volume Identifier, указывает на физический том

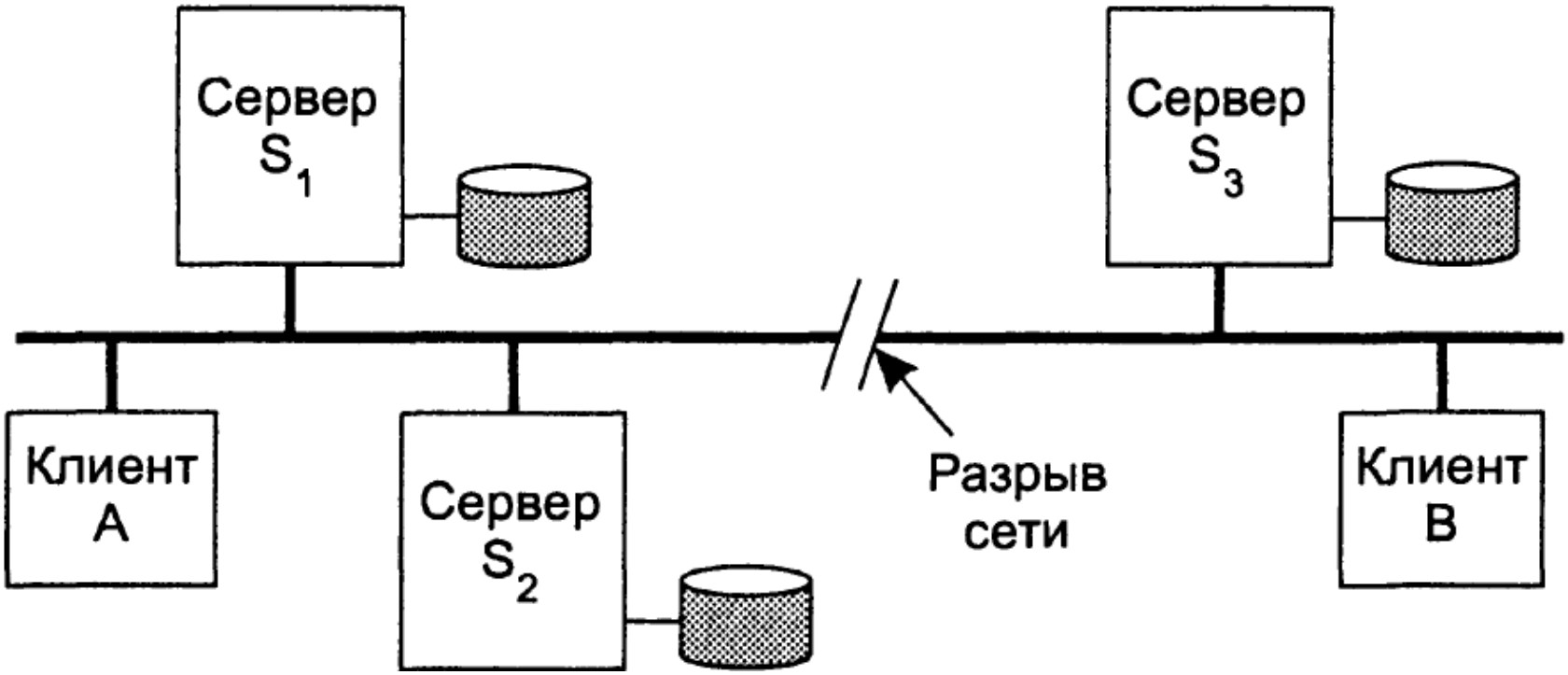


AVSG- Accessible Volume Storage Group, Read-One – Write-All (ROWA), MultiRPC

Кэширование и репликация в Coda

CVV - Coda version vector

CVV1(f) = CVV2(f) = [2,2,1]



CVV3(f) = [1,1,2]

Литература

Распределенные системы. Принципы и парадигмы / Э. Таненбаум, М. ван Стеен. — СПб.: Питер. — 877 с: ил. — (Серия «Классика computer science»). ISBN 5-272-00053-6