УДК 004.65
DOI: 10.15827/0236-235X.120.699-705
Дата подачи статьи: 30.03.17
2017. Т. 30. № 4. С. 699–705

ПРОТОКОЛ ДЛЯ ДЕЦЕНТРАЛИЗОВАННОЙ СИСТЕМЫ ХРАНЕНИЯ С ИЗБЫТОЧНЫМ КОДИРОВАНИЕМ

П.К. Карасюк, магистрант, p.karasyuk@gmail.com
(Новосибирский государственный университет, ул. Пирогова, 2, г. Новосибирск, 630090, Россия);
Д.С. Мигинский, к.ф.-м.н., научный сотрудник, dmiginsky@gmail.com
(Институт систем информатики им. А.П. Ершова СО РАН,
просп. Лаврентьева, 6, г. Новосибирск, 630090, Россия;
Новосибирский государственный университет, ул. Пирогова, 2, г. Новосибирск, 630090, Россия)

Репликация, используемая во многих распределенных системах хранения данных для обеспечения отказоустойчивости, приводит к многократному уменьшению эффективного дискового пространства. Чтобы решить эту проблему, для обеспечения сохранности данных вместо репликации можно использовать избыточное кодирование. Многие хранилища в силу CAP-теоремы для обеспечения высокой доступности и масштабируемости отказываются от строгой согласованности в пользу согласованности в конечном счете – гарантии, что согласованность данных будет достигнута через конечное время после внесения последнего внешнего изменения в систему. Однако при переходе от репликации к избыточному кодированию в парадигме согласованности в конечном счете возникают сложности при восстановлении данных, связанные с необходимостью поддерживать достаточное количество согласованных между собой фрагментов кодового слова.

В статье предложен основанный на Dynamo протокол для распределенного хранилища данных, который вычисляет контрольные суммы от хранимых объектов с помощью кодов Рида–Соломона и использует их в дальнейшем для возможного восстановления, что позволяет обеспечить тот же уровень отказоустойчивости, что и при репликации, но ценой меньшей избыточности. Протокол поддерживает возможность выполнения над объектом конкурентных операций записи и чтения, отслеживает сбои в сети и учитывает их в дальнейшей работе. Допускаются фиксированное число постоянных отказов узлов и любые кратковременные сбои без потери данных и отказа обслуживания клиентских запросов.

Протокол протестирован в системе имитации распределенной среды на заранее выбранных проверочных сценариях пользовательских сообщений и сбоев. В статье продемонстрирована работа протокола на некоторых сценариях.

Ключевые слова: *распределенное хранилище, p2p-хранилище, CAP-теорема, протокол, согласованность, согласованность в конечном счете, репликация, избыточное кодирование, коды Рида–Соломона, Dynamo.*

Многие распределенные хранилища ради высокой доступности и масштабируемости отказываются от строгой согласованности в пользу согласованности в конечном счете – гарантии, что система при отсутствии внешних факторов за некоторое время придет в согласованное состояние [1]. Взамен они получают возможность обрабатывать запросы без блокировки данных, которая не может выполняться в случае отказа узлов. Отказ некоторых узлов считается допустимым, при этом хранилище должно гарантировать сохранность данных. Наиболее распространенный способ – репликация данных на несколько машин [2]. Способ имеет ряд достоинств: нагрузка разделяется между серверами с копиями, и каждый клиент может выбрать наиболее удобную для него копию. Но есть и серьезный недостаток – избыточность составляет сотни процентов (по числу копий), а эффективное дисковое пространство линейно падает с уровнем репликации. Применение избыточного кодирования [3] вместо репликации для поддержания отказоустойчивости может значительно повысить эффективность использования дискового пространства [4]. Однако существующие протоколы без изменений не применимы для использования с избыточным кодированием. Основное усложнение протокола состоит в том, что для восстановления объекта по реплике вполне хватает любой из них, а для декодирования нужно, чтобы было доступно достаточное количество частей, размещаемых на разных узлах. В статье описан протокол – модификация Dynamo, использующий коды Рида–Соломона для обеспечения отказоустойчивости.

Существующие решения и протоколы

Рассмотрим примеры использования избыточного кодирования в распределенных системах хранения и Dynamo – набор техник, взятый за основу для предложенного протокола. В [5, 6] приведены примеры протоколов с использованием избыточного кодирования в модели строгой согласованности.

Windows Azure Storage (WAS) [7] использует избыточное кодирование для компактного распределенного хранения некоторых объектов, в частности, коды Рида–Соломона, которые сохраняют кодируемое слово как часть кодового. WAS демонстрирует применимость избыточного кодирования в распределенной среде, но не может служить самостоятельной моделью для разрабатываемого протокола, так как объекты в нем не изменяются после записи в хранилище и закодированные объекты только считываются из хранилища.

Quantcast File System [8] – распределенная файловая система, использующая коды Рида–Соло-

мона для хранения файлов. Тесты, проведенные разработчиками QFS, показали превосходство QFS над HDFS по скорости чтения и записи 20 Тб данных. Это позволяет сделать предположение, что основанный на избыточном кодировании протокол необязательно будет проигрывать по быстродействию репликационному. Недостаток QFS в том, что распределением и контролем данных управляет метасервер и при его отказе система не сможет работать.

Dynamo [9] – набор техник построения распределенного хранилища данных. На его основе построены такие системы, как Amazon DynamoDB и Apache Cassandra. За хранение данных отвечают виртуальные узлы (рис. 1). Они обозначаются уникальными целыми числами и образуют логическое кольцо. Отказоустойчивость достигается за счет репликации объекта на N последовательных в кольце виртуальных узлов, первый из которых определяется следующим образом: ищется виртуальный узел, ближайший от значения хеш-функции ключа объекта по направлению обхода на кольце. Виртуальные узлы распределены по физическим узлам сети таким образом, что все реплики объекта попадают на разные физические узлы. На каждом физическом узле хранится таблица соответствия виртуальных узлов физическим. Поскольку любой узел может определить местонахождение нужных данных, для их получения требуется минимальное число сетевых запросов, что обеспечивает высокую скорость отклика.

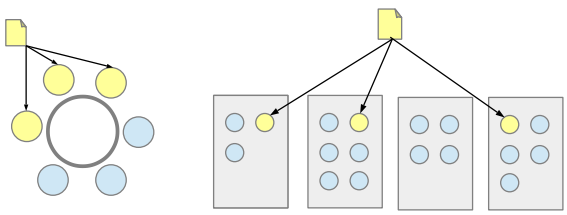

Рис. 1. Кольцо виртуальных узлов Dynamo

Fig. 1. Dynamo virtual node ring

Задача кодирования

Существуют разные модели ошибок, которые могут возникать в абстрактном канале связи. В рассматриваемых системах сообщением является объект хранения, а ошибкой – невозможность доступа к его фрагменту. Поэтому нас будет интересовать модель потери данных, в которой при ошибке часть сообщения теряется (рис. 2), при этом точно известно, какая конкретно. Принимающая сторона всегда знает об ошибке и о том, какая часть сообщения потеряна. В этой модели оптимальными являются коды Рида–Соломона [3], которым для исправления M ошибок достаточно M избыточных фрагментов. При этом код можно сконструировать таким образом, что кодируемое слово будет частью кодового. Тогда операция кодирования заключается в вычислении фрагментов контрольных сумм на основе фрагментов данных кодируемого слова и их конкатенации.

Требования к протоколу

Основные требования к протоколу – масштабируемость и отказоустойчивость. Поэтому в качестве парадигмы согласованности выбрана согласованность в конечном счете [2]: система должна быть полностью децентрализованной и обеспечивать корректную работу при одновременном постоянном отказе до M узлов и любых кратковременных отказах. Протокол должен поддерживать возможность одновременного выполнения любого числа операций записи и чтения над объектом. При этом одна из записанных версий в конечном счете должна получить приоритет во всей системе, а операции чтения вернуть любые, возможно различные, версии. Протокол должен отслеживать сбои и учитывать их в дальнейшей работе, а администратор иметь возможность вручную запустить восстановление любого узла сети.

Описание алгоритма

Основой для протокола выбран Dynamo. Вместо копий объектов он оперирует их фрагментами. Главные сценарии – запись и считывание объектов, мониторинг, восстановление узлов.

Описание модели данных (объекты и фрагменты). В системе хранятся объекты, каждому из которых сопоставлена метаинформация: уникальный строковый идентификатор и длина. Каждый объект делится на равные по длине фрагменты данных, к ним добавляются фрагменты контрольных сумм. Они вычисляются с помощью кода Рида–Соломона, с точки зрения которого фрагменты данных являются кодируемым словом. В целях кодирования недостающие данные заменяются нулями. Количество фрагментов контрольных сумм определяется схемой кодирования и соответствует количе-

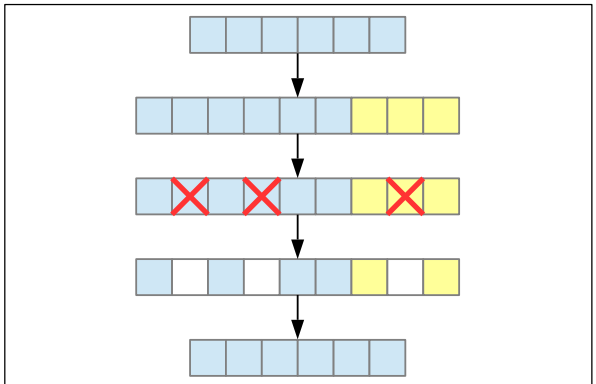

Рис. 2. Кодирование и декодирование

Fig. 2. Encoding and decoding

ству фрагментов, которые можно потерять без потери информации. Фрагменты идентифицируются объектом, к которому принадлежат, и порядковым номером в нем.

Распределение данных по узлам. За основу было принято кольцо виртуальных узлов, аналогичное Dynamo [9]. Узлы, ответственные за хранение фрагментов объекта, определяются по значению хеш-функции его идентификатора. Фрагменты упорядоченно распределяются по этим узлам, пропуская узлы, которым соответствуют нулевые фрагменты (рис. 3).

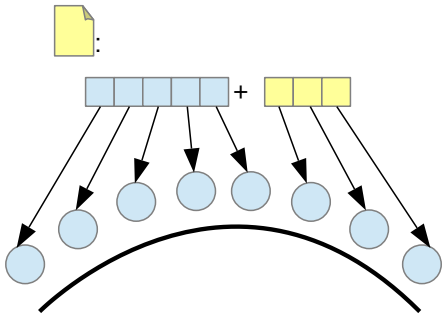

Рис. 3. Распределение фрагментов по виртуальным узлам

Fig. 3. Fragment distribution among virtual nodes

Версионность. Чтобы объекты не испортились во время конкурентных операций записи и чтения, им необходимо присвоить уникальные номера версий. На множестве номеров версий должно быть определено отношение порядка. Версии должны увеличиваться при каждом изменении объекта, и протокол должен удалять объекты старых версий при появлении новых. В качестве номеров версий используются векторные часы.

Векторные часы – вектор логических часов, сопоставленных каждому узлу, поддерживаемый на каждом узле [10]. Узел инкрементирует соответствующую себе запись в своих часах каждый раз непосредственно перед отправкой или приемом сообщения. К каждому сообщению прикрепляются часы отправителя на момент отправки. Узел, получивший такое сообщение, обновляет свои часы как поэлементный максимум с часами из сообщения. Часы, присоединенные к каждому сообщению, глобально уникальны и могут быть использованы в качестве версии объекта. Узел, записывающий объект в систему, инкрементирует соответствующую себе запись в часах и указывает получившиеся часы в качестве версии этого объекта. Отношение порядка векторных часов для возможности однозначно сравнивать версии произвольным образом дополнена от частичного порядка до полного, например, в виде лексикографического сравнения.

Легковесная блокирующая операция используется другими операциями, чтобы в некотором смысле гарантировать завершение и снизить вероятность избыточного одновременного выполнения одинаковых операций. Узел, инициирующий легковесную блокирующую операцию, рассылает всем узлам сети сообщение о блокировке, содержащее блокируемую операцию, случайное время ожидания (таймаут) и операцию, которую следует выполнить в случае истечения таймаута. При повторной блокировке таймаут обновляется на новый. После этого узел выполняет свою операцию и рассылает сообщение о снятии блокировки. Снять блокировку может любой узел. Таким образом, если случится отказ узла, выполняющего операцию, на другом узле истечет таймаут и выполнится операция, которая была указана при блокировке. Если выполнение этой операции приведет к тому, что узел инициирует такую же блокировку (например, если операция в случае истечения таймаута совпадает с блокируемой), на остальных узлах, на которых еще не истек таймаут, блокировка обновится и ответственность за выполнение операции перейдет к узлу с наименьшим таймаутом.

Сценарий записи объекта в систему проиллюстрирован на рисунке 4. Узел, получивший запрос о добавлении объекта в систему, делит его на фрагменты данных, вычисляет фрагменты контрольных сумм и отправляет получившиеся фрагменты на со-

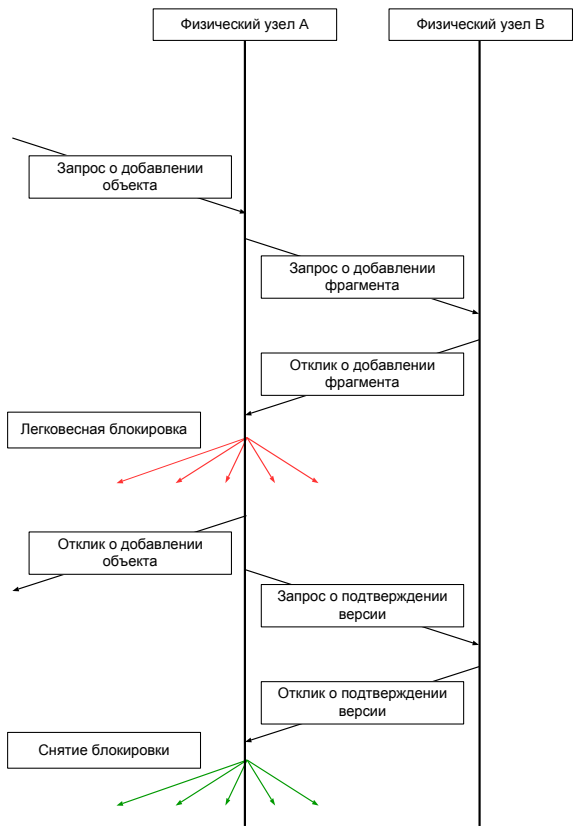

Рис. 4. Последовательность выполнения операции записи объекта

Fig. 4. Execution sequence of the object recording operation

ответствующие узлы. Узел, получивший запрос о записи фрагмента, добавляет его во временное хранилище и отправляет сообщение об успешном выполнении. Когда узел, ответственный за добавление объекта в хранилище, собирает достаточное количество откликов, он инициирует легковесную блокирующую операцию подтверждения версии и рассылает сообщение о подтверждении записи этого объекта. Если версия, для которой узел получил подтверждение записи, больше текущей версии фрагмента в основном хранилище, он заменяется на фрагмент новой версии из временного хранилища. Из временного хранилища удаляются все более старые версии.

Сценарий считывания объекта из системы. Узел, получивший запрос о считывании объекта, запрашивает фрагменты данных этого блока из основного хранилища соответствующих узлов. Если в течение заданного времени узел получит все отклики и все они будут одинаковой версии, он составит из них объект и ответит запросившему узлу. Если этого не произойдет, узел запросит фрагменты данных и фрагменты контрольных сумм всех версий. Как только фрагментов будет достаточно, узел отправит восстановленный объект запросившему узлу. Если по истечении таймаута фрагментов будет недостаточно, узел вернет сообщение об ошибке.

Мониторинг. В системах, гарантирующих согласованность в конечном счете, узлы должны постоянно отслеживать ошибки, возникшие по вине протокола, и сбои, произошедшие по внешним обстоятельствам.

Узлы периодически попарно обмениваются сообщениями мониторинга. Узел, не ответивший на сообщение мониторинга или любое другое, помечается как временно неактивный. Такой узел не может быть выбран другими узлами для выполнения операций. Узел, обнаруживший временную неактивность узла, оповещает остальные узлы. Временно неактивный узел, не ответивший некоторому узлу на фиксированное число запросов мониторинга, помечается как постоянно неактивный, остальные узлы оповещаются.

Сценарий восстановления физического узла. Если один из физических узлов вышел из строя, администратор может запросить его восстановление. При этом данные, находившиеся на поврежденном узле, будут перераспределены по остальным узлам сети. Сценарий проиллюстрирован на рисунке 5. Узел, получивший такой запрос, инициирует легковесную блокирующую операцию, а затем рассылает на произвольные узлы запросы о восстановлении виртуальных узлов, содержавшихся на восстанавливаемом физическом. Блокировка будет снята, когда он получит отклики о восстановлении всех виртуальных узлов. Узел, получивший запрос о восстановлении виртуального узла, сразу отправляет отклик вне зависимости от установленных блокировок. Если операция заблокирована, обработка завершается. Иначе узел инициирует легковесную блокирующую операцию восстановления указанного виртуального узла, при этом таймаут для себя выставляет меньше, чем для других узлов – таким образом, если операция не будет выполнена с первого раза не по вине этого узла, прогресс не будет потерян. Узлы, на которых лежат другие фрагменты блока, содержащего фрагмент, расположенный на восстанавливаемом виртуальном узле, зависят только от номера этого фрагмента. Узел, выполняющий операцию восстановления виртуального узла, рассылает запросы о восстановлении семейств фрагментов, определяемых виртуальным узлом и номером фрагмента. Узел, получивший такой запрос, собирает все версии фрагментов нужных номеров с соответствующих узлов, восстанавливает все возможные версии фрагментов и отправляет их запросившему узлу. Узел, получивший восстановленные фрагменты, добавляет их к себе в хранилище. При этом временное и основное хранилища восстанавливаемого узла отразятся в соответственные хранилища восстанавливающего.

Методика проверки протокола. Разрабатывать и тестировать протокол непосредственно в распределенной среде трудоемко и ненадежно, поэтому был разработан симулятор виртуальной среды. Основные требования: детерминированный порядок передачи управления между узлами, возможность отправки сообщений на узлы и возможность симулировать сбои в сети: деактивацию узлов и потерю сообщений. В симуляторе физические узлы поочередно получают контроль и выполняют итерацию протокола в парадигме кооперативной многозадачности.

Для тестирования система запускается с выбранными сценариями сообщений и сбоев. Корректность работы проверяется автоматическим анализом детализированного журнала, который записывается протоколом на каждом узле.

Примеры тестовых сценариев

Одновременные чтение и запись. Узел *A* записывает новую версию объекта, который считывает узел *B*. Если операция подтверждения версии еще не началась, значит, в основном хранилище всех узлов лежат фрагменты старой версии и можно восстановить старую версию. Если операция подтверждения версии началась, значит, на достаточном количестве узлов в основном или временном хранилище лежат фрагменты новой версии, и узел *B* сможет восстановить ее. Поскольку при неудаче оптимистической попытки сбора объекта узел запросит все фрагменты всех версий этого объекта, версия, фрагментов для которой достаточно, будет собрана.

Одновременная запись. К примеру, в каждом объекте 4 фрагмента данных и 2 фрагмента кон-

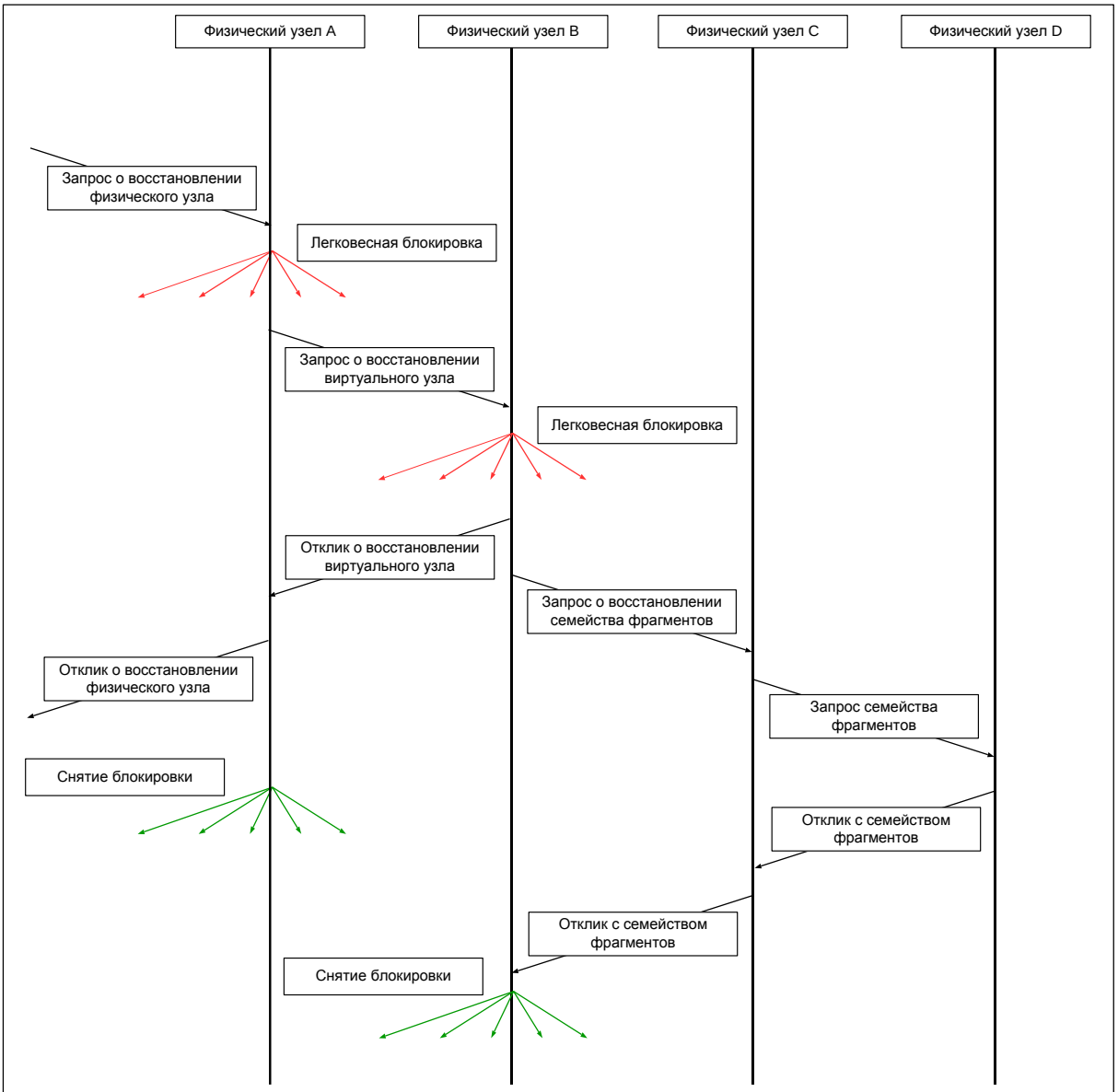

Рис. 5. Последовательность выполнения операции восстановления

Fig. 5. Execution sequence of the recovery operation

трольных сумм. Узлы *X* и *Y* записывают фрагменты разных версий (*x* и *y*, при этом *x* > *y*) некоторого объекта на узлы *A*, *B*, *C*, *D*, *E*, *F*. Если оба узла успешно отправят достаточное количество фрагментов (к примеру, все), во временном хранилище узлов *A*–*F* будут версии *x* и *y*. Возможна ситуация, в которой версия *x* сначала будет подтверждена на узлах *A*–*C*, а версия *y* на узлах *D*–*F*. Тогда в основном хранилище узлов *A*–*C* лежит версия *x*, а во временном пусто (так как при подтверждении *x* более старая версия *y* была удалена), а на узлах *D*–*F* в основном хранилище лежит *y*, а во временном *x*. В этот момент недостаточно фрагментов для восстановления версии *y*, но достаточно для восстановления *x*. Скажем, дальше подтверждение версии *y* пришлось на узлы *A*–*C*. У них в основном хранилище лежит более новая версия *x*, *A*–*C* ответят на эти запросы сообщениями об успехе, но не внесут никаких изменений в хранилище. Когда оставшиеся запросы о подтверждении версии *x* дойдут на узлы *D*–*F*, версия *y* в основном хранилище будет заменена более новой версией *x* из временного.

Потеря confirm-сообщения. Если в описанном сценарии откажет узел, ответственный за запись, до начала подтверждения версии, узел, запросивший запись объекта, не получит сообщение об успехе в течение таймаута. Если же он откажет после начала подтверждения версии, значит, была инициирована легковесная блокирующая операция, и операция подтверждения версии будет закончена некоторым другим узлом сети. При этом, если операцию продолжат несколько узлов, что допустимо при легковесной блокирующей операции, в ходе выполнения ничего не изменится, и она

успешно завершится на всех исполняющих ее узлах.

Запись на отказавший/восстанавливаемый узел. Пусть отказавший физический узел содержал виртуальный узел *X*. Фрагменты, которые должны быть записаны на *X* до начала операции восстановления, будут восстановлены во время операции восстановления *X*, если операция записи завершилась успешно. Первое действие узла, выполняющего восстановление *X*, после инициации легковесной блокирующей операции – оповещение других узлов о приеме себе *X*. Узлы, получившие это сообщение, при попытке записать фрагмент на *X* будут отправлять его на узел, выполняющий восстановление *X*. Узлы, не получившие это сообщение (если оно еще не успело дойти или было потеряно), не смогут записать фрагмент на *X*. Со временем все узлы будут оповещены о приеме *X* восстанавливающим узлом в результате фонового обмена списками виртуальных узлов.

Отказ узла, ответственного за восстановление физического узла. Восстановление физического узла – легковесная блокирующая операция, поэтому, если узел, ответственный за восстановление физического узла, откажет, операция будет продолжена другим узлом. Выполнение операции состоит в посылке запросов о восстановлении всех виртуальных узлов восстанавливаемого физического узла и сборе всех откликов. Отклики посылаются в любом случае сразу после приема сообщения и инициации легковесной блокирующей операции (или просто после приема, если блокировка уже стоит). Значит, на момент завершения операции восстановления физического узла последующее восстановление виртуальных узлов гарантируется легковесными блокирующими операциями.

Отказ узла, ответственного за восстановление виртуального узла. Восстановление виртуального узла – легковесная блокирующая операция, поэтому, если узел, ответственный за его восстановление, откажет, операция будет продолжена другим узлом. Если операция пройдет неуспешно не по вине узла, ответственного за ее выполнение, она будет продолжена тем же узлом с сохранением прогресса.

Заключение

В статье представлен основанный на Dynamo протокол одноранговой системы, обеспечивающий отказоустойчивость за счет избыточного кодирования объектов. Показана корректность работы протокола на проверочных сценариях в условиях системы имитации распределенной среды. Следующий шаг работы над протоколом – его оптимизация для работы в реальной среде.

В текущем виде протокол демонстрирует возможность применения избыточного кодирования при распределенном хранении данных, но это влечет делокализованное хранение объектов, что может оказаться существенным ограничением во многих приложениях.

Литература

1. Bailis P., Ghodsi A. Eventual consistency today: limitations, extensions, and beyond. Communications of the ACM, 2013, vol. 56, no. 5, pp. 55–63.

2. Saito Y., Shapiro M. Optimistic Replication. ACM Computing Surveys, 2005, no. 37, pp. 42–81.

3. Elwyn R. Berlekamp Algebraic Coding Theory. Revised Edition. World Scientific Publ., 2015.

4. Weatherspoon H., Kubiatowicz J. Erasure Coding vs. Replication: a quantitative comparison. Proc. of IPTPS 2002, 2002, pp. 328–338.

5. Zheng Zhang, Qiao Lian Reperasure: replication protocol using erasure-code in peer-to-peer storage network. Proc. 21st IEEE Sympos. Reliable Distributed Systems, 2002, pp. 330–335.

6. Frolund S., Merchant A., Saito Y., Spence S., and Veitch A. A decentralized algorithm for erasure-coded virtual disks. Dependable Systems and Networks, Intern. Conf. on. IEEE, 2004, pp. 125–134.

7. Calder B., Wang Ju, Ogus A., Nilakantan N., Skjolsvold A., McKelvie S., Xu Y., Srivastav Sh., Wu J., Simitci H., Haridas J., Uddaraju Ch., Khatri H., Edwards A., Bedekar V., Mainali Sh., Abbasi R., Agarwal A., Fahim ul Haq M., Ikram ul Haq M., Bhardwaj D., Dayanand S., Adusumilli A., McNett M., Sankaran S., Manivannan K., Rigas L. Windows Azure Storage: a highly available cloud storage service with strong consistency. Proc. 23rd ACM Sympos. on Operating Systems Principles, 2011, pp. 143–157.

8. Ovsiannikov M. The quantcast file system. Proc. VLDB Endowment, 2013, pp. 1092–1101.

9. Giuseppe DeCandia, et al. Dynamo: Amazon's highly available key-value store. Proc. 21st ACM SIGOPS Sympos. Operating Systems Principles, 2007, pp. 205–220.

10. Colin J. Fidge Timestamps in message-passing systems that preserve the partial ordering. Proc. of the 11th Australian Comp. Sc. Conf., 1988, pp. 56–66.

Software & Systems
DOI: 10.15827/0236-235X.120.699-705

Received 30.03.17
2017, vol. 30, no. 4, pp. 699–705

A PROTOCOL FOR A DECENTRALIZED STORAGE WITH REDUNDANT ENCODING

P.K. Karasyuk [1], *Graduate Student, p.karasyuk@gmail.com ()*
D.S. Miginsky [1,2], *Ph.D. (Physics and Mathematics), Research Associate, dmiginsky@gmail.com*

[1] *Novosibirsk State University, Pirogova St. 2, Novosibirsk, 630090, Russian Federation*
[2] *A.P. Ershov Institute of Informatics Systems (IIS), Siberian Branch of the Russian Federation Academy of Sciences, Lavrentev Av. 6, Novosibirsk, 630090, Russian Federation*

Abstract. Many distributed data storages use replication that leads to significant decreasing of the effective disk space. Applying redundant coding methods instead of replication for data safety can solve this problem. Due to CAP theorem, many storages abandon strong consistency in favor of eventual consistency, which is a guarantee that data will be consistent within a finite time after last external modification. The transition from replication to redundant encoding under the eventual consistency paradigm leads to complexity associated with the necessity to keep enough mutually consistent fragments of the code words for recovery.

The article proposes a Dynamo-based protocol for distributed data storage. It computes object checksums using Reed-Solomon codes and uses them later for recovery if necessary. It provides the same level of fault-tolerance with lower redundancy. The protocol supports concurrent execution of several read and write operations on the same object. It tracks node failures and considers them in further execution. The protocol allows a fixed number of permanent node failures and arbitrary transitional failures without data loss or denial of service. The protocol was tested in a distributed environment simulator with preselected scenarios of failures and user messages. The article demonstrates the protocol behavior in some of the scenarios.

Keywords: distributed storage, p2p storage, CAP theorem, protocol, consistency, eventual consistency, replication, erasure-correction codes, Reed-Solomon codes, Dynamo.

References

1. Bailis P., Ghodsi A. Eventual consistency today: limitations, extensions, and beyond. *Communications of the ACM*. 2013, vol. 56, no. 5, pp. 55–63.

2. Saito Ya., Shapiro M. Optimistic replication. *ACM Computing Surveys*. 2005, no. 37, pp. 42–81.

3. Elwyn R. *Berlekamp algebraic coding theory*. Revised ed. World Scientific Publ., 2015, 500 p.

4. Weatherspoon H., Kubiatowicz J. Erasure Coding vs. Replication: A Quantitative Comparison. *Proc. IPTPS 2002*. 2002, pp. 328–338.

5. Zheng Zhang, Qiao Lian. Reperasure: replication protocol using erasure-code in peer-to-peer storage network. *Proc. 21st IEEE Symp. on Reliable Distributed Systems*. 2002, pp. 330–335.

6. Frolund S., Merchant A., Saito Y., Spence S., Veitch A. A decentralized algorithm for erasure-coded virtual disks. *IEEE 2004 Int. Conf. on Dependable Systems and Networks*. 2004, pp. 125–134.

7. Calder B., Wang Ju, Ogus A., Nilakantan N., Skjolsvold A., McKelvie S., Xu Y., Srivastav Sh., Wu J., Simitci H., Haridas J., Uddaraju Ch., Khatri H., Edwards A., Bedekar V., Mainali Sh., Abbasi R., Agarwal A., Fahim ul Haq M., Ikram ul Haq M., Bhardwaj D., Dayanand S., Adusumilli A., McNett M., Sankaran S., Manivannan K., Rigas L. Windows Azure Storage: a highly available cloud storage service with strong consistency. *Proc. 23rd ACM Symp. on Operating Systems Principles*. 2011, pp. 143–157.

8. Ovsiannikov M. The quantcast file system. *Proc. VLDB Endowment*. 2013, pp. 1092–1101.

9. DeCandia G., Hastorun D., Jampani M., Kakulapati G., Lakshman A., Pilchin A., Sivasubramanian S., Vosshall P., Vogels W. Dynamo: Amazon's highly available key-value store. *Proc. 21st ACM SIGOPS Symp. on Operating Systems Principles*. 2007, pp. 205–220.

10. Colin J. Fidge Timestamps in Message-Passing Systems That Preserve the Partial Ordering. *Proc. 11th Australian Computer Science Conf*. 1988.

Примеры библиографического описания статьи

1. Карасюк П.К., Мигинский Д.С. Протокол для децентрализованной системы хранения с избыточным кодированием // Программные продукты и системы. 2017. Т. 30. № 4. С. 699–705. DOI: 10.15827/0236-235X.120.699-705.

2. Karasyuk P.K., Miginsky D.S. A protocol for a decentralized storage with redundant encoding. *Programmnye produkty i sistemy* [Software & Systems]. 2017, vol. 30, no. 4, pp. 699–705 (in Russ.). DOI: 10.15827/0236-235X.120.699-705.