ПРОГРАММИРОВАНИЕ, 2020, № 5, с. 60–71

БАЗЫ ДАННЫХ, ХРАНИЛИЩА ДАННЫХ

УДК 681.3

РЕПЛИКАЦИЯ В РАСПРЕДЕЛЕННЫХ СИСТЕМАХ: МОДЕЛИ, МЕТОДЫ И ПРОТОКОЛЫ

© 2020 г. А. Р. Насибуллин[a],*, Б. А. Новиков[b],**

[a] *Санкт-Петербургский государственный университет*
199034 Санкт-Петербург, Университетская набережная, д. 7/9, Россия

[b] *Национальный исследовательский университет "Высшая школа экономики"*
190121 Санкт-Петербург, ул. Союза Печатников, д. 16, Россия

**E-mail: nevskyarseny@yandex.ru*

***E-mail: bnovikov@hseu.ru*

Поступила в редакцию 10.03.2019 г.
После доработки 20.09.2019 г.
Принята к публикации 25.10.2019 г.

Репликация данных применяется для повышения надежности, доступности и пропускной способности систем хранения данных за счет увеличения сложности и стоимости обновления данных. Во многих случаях в системах хранения данных с репликацией применяются ослабленные критерии согласованности. В обзоре рассматриваются различные схемы репликации данных; обсуждаются разнообразные модели и протоколы, обеспечивающие согласованность.

DOI: 10.31857/S0132347420050064

1. ВВЕДЕНИЕ

Статья посвящена обсуждению различных моделей поддержки согласованности и протоколов репликации данных. *Репликация данных* – это способ организации хранения данных, в котором каждый элемент данных хранится в нескольких копиях, размещенных на разных серверах. Каждая из таких копий называется *репликой*. Задача репликации – защита от потери данных. Репликация данных обладает следующими свойствами:

• *Доступность*. Обеспечивается возможность доступа к данным в случае, если часть реплик недоступна.

• *Надежность*. Предотвращается потеря данных при частичном разрушении серверов или потере части реплик.

• *Пропускная способность*. Увеличивается пропускная способность чтения данных.

В статье будут использованы следующие количественные характеристики баз данных: пропускная способность, масштабируемость. *Пропускная способность* системы определяется как количество выполняемых действий разной сложности за единицу времени. *Масштабируемость* определяется следующей формулой:

$$S = \frac{M}{N}, \qquad (1)$$

где M означает пропускную способность базы данных, состоящей из множества вычислителей, с определенным объемом данных и определенной нагрузки. Под N подразумевается пропускная способность системы, состоящей из одного вычислителя.

Введем определение распределенной базы данных. Под *распределенной базой данных* понимается система хранения данных, состоящая из локальных баз данных, размещенных в различных узлах компьютерной сети. Каждая из локальных баз данных может иметь реплику.

При использовании репликации может возникать проблема поддержки всех реплик в согласованном состоянии [1–5]. Другими словами, результат выполнения запроса может отличаться в зависимости от того, с какой репликой работает клиент. Клиентский запрос, выполняемый на одной из реплик, может обрабатывать неакутальные данные. При этом, поддержка согласованности на всех репликах может привести к ухудшению производительности распределенной базы данных и сокращению количества обрабатываемых клиентских запросов, а также возникновению проблем масштабирования.

Традиционно согласованность рассматривается в контексте СУБД как инструмент для соотнесения значений разных элементов данных и реализуется средствами управления транзакциями. Неформально транзакции принято характеризо-

вать свойствами ACID, а степень согласованности описывается в терминах уровней изоляции. Такое понимание согласованности абстрагировано от архитектуры системы (централизованной или распределенной) и основано на предположении о *единой логической копии* каждого элемента данных.

В распределенных системах с репликацией такое понимание согласованности оказывается недостаточным и дополняется рассмотрением согласованности нескольких копий одного элемента данных и согласованности реплик. В литературе по распределенным системам (см., например, [6]) обычно понятие транзакций явно не упоминается, все операции обновления считаются атомарными и предполагается, что локальная согласованность каждой реплики сохраняется при обновлениях, а для достижения глобальной согласованности данных достаточно обеспечить согласованность (то есть идентичность) копий.

Однако буквальная реализация концепции единой логической копии зачастую оказывается слишком дорогостоящей, поэтому в большинстве моделей, обсуждаемых далее в разделе 3, применяются ослабленные критерии согласованности. В нашей работе мы рассматриваем модели и протоколы согласованности копий необязательно использующие механизмы транзакций. Далее в этой работе рассматривается согласованность реплик, если явно не оговорено другое.

В последнее время исследователи предложили различные решения задач масштабирования и поддержки согласованности данных. Например, в работах [7, 8] предлагается ослабленная модель согласованности, обеспечивающая высокую пропускную способность системы и упрощающая задачу масштабирования. В работах [5, 9] предлагается более усиленная модель согласованности, чем упомянутая выше.

Оставшаяся часть статьи построена следующим образом. В разделе 2 проведен анализ различных схем репликации. В разделе 3 рассматриваются модели согласованности. Одной из важных задач, обсуждаемых в разделе 4, является задача по распространению данных от одной реплики к другим. Также, в разделе 4 представлено краткое сравнение различных подходов доставки данных к серверам. В разделе 5 затронуты различные протоколы согласованности. Краткий обзор алгоритмов репликации изложен в разделе 6. В разделе 7 подведены итоги данной статьи.

2. СХЕМЫ РЕПЛИКАЦИИ

В этом разделе мы обсудим два типа схем репликации – статические и динамические [10, 11].

Схема называется статической, если задано фиксированное количество реплик в распределенной базе данных. Число реплик в такой схеме может быть изменено путем изменения конфигурации и рестарта системы. Недостатком данного типа схемы репликации является большая трудоемкость по расширению распределенной базы данных под растущие нужды бизнеса.

В отличие от статических схем, в динамической схеме новые реплики добавляются или удаляются динамически без рестарта системы в зависимости от количества клиентских запросов. Благодаря увеличению количества работающих серверов удается повысить пропускную способность. Администратор баз данных может изменять количество реплик в кластере, позаботившись о синхронизации добавленной реплики с остальными. Динамические схемы особенно полезны в географически распределенных системах, где в каждой локальной зоне может быть создано сколько угодно новых реплик. Но несмотря на это, динамические схемы имеют недостаток – необходимость выполнять дополнительную работу по определению физических адресов других серверов, хранящих реплики, что увеличивает количество сообщений, пересылаемых между серверами и вспомогательными сервисами. Статические схемы лишены данного недостатка [12, 13].

Для статической и динамической схем репликации известны два подхода по распространению обновлений: централизованный и децентрализованный [10, 22]. Достоинством децентрализованного подхода является отсутствие фиксированного сервера, принимающего все запросы от клиентов. Недостатком децентрализованного подхода является необходимость решать проблему синхронизации данных между репликами.

В централизованном подходе выделяется роль основного сервера-координатора, обрабатывающего запросы клиентов. Отказ работы основного сервера приведет к неработоспособности всего кластера. Данная проблема может быть решена с помощью кворумных протоколов [23, 24]: если координатор становится недоступным, оставшиеся доступные сервера выбирают координатора путем голосования. В таблице 1 представлены примеры централизованных и децентрализованных решений на основе статических и динамических схем.

Как в статической, так и в динамической схемах рассматривают два класса стратегий репликации данных, влияющие на организацию копирования данных по всем репликам: полная и частичная [22].

Под полной репликацией подразумевается копирование всех данных, хранящихся в основном сервере, на все сервера. При этом, полная репликация позволяет достичь надежности и высокой

Таблица 1. Примеры централизованных и децентрализованных решений

	Статическая	Динамическая
Централизованная	CitusDB [14], PostgreSQL [15] MySQL [16]	Google Spanner [5] MongoDB [17], Cockroach DB [18]
Децентрализованная	Bigtable [19]	Cassandra [20], Dynamo [21]

доступности распределенной базы данных. Применение подобного подхода приводит к большим затратам, так как каждый сервер должен обладать достаточно большим объемом свободного места на носителях для хранения данных, пересылаемых между серверами по сети.

При частичной репликации, в отличие от полной репликации, каждый сервер кластера базы данных хранит лишь некоторое подмножество данных. Другими словами, для каждого элемента данных основного сервера хранится реплика на нескольких серверах, но не на всех. При отказе большого количества серверов часть данных может стать недоступной. Тем не менее, стоимость хранения данных и коммуникация между серверами кластера значительно меньше, нежели в стратегии полной репликации данных.

3. МОДЕЛИ СОГЛАСОВАННОСТИ

В данном разделе будут рассмотрены модели согласованности, используемые в схемах репликации.

Модель согласованности определяется как контракт между базой данных и клиентами. Если для потока клиентских запросов определен порядок выполнения, то произойдет следующее: данные будут согласованны, клиент получит корректные данные. Различают два семейства моделей согласованности [1]: *клиент-ориентированные модели согласованности* и *модели согласованности ориентированные на данные*.

Для удобства определения моделей согласованности введем следующие обозначения. Предположим, что имеются два клиентских процесса P_1, P_2, и хранилище данных *DS*. Здесь и далее в статье клиентские процессы для базы данных представляют собой транзакции. Будем предполагать, что P_i процесс выполняется на сервере *i*. Под $W(x)a$ мы будем подразумевать операцию записи процессов в область памяти x значения a. $R(x)a$ будет означать операцию чтения процессом значения области памяти x, которое возвращает значение a. Будем подразумевать, что результат выполнения операции записи процессом распространяется на другие сервера в рамках другого процесса. Здесь и далее процесс распространения данных от одного сервера к другим обозначен кривой линией на рисунках.

3.1. Модели согласованности ориентированные на данные

В *моделях согласованности ориентированных на данные* акцент делается на том, что результат выполнения запроса на обновление одного сервера немедленно перенаправляется на другие сервера. В этом классе моделей согласованности предполагается несколько одновременно работающих процессов обновления данных.

Известны несколько разновидностей моделей согласованности ориентированных на данные. Далее они перечислены в убывающем порядке по степени строгости согласованности данных.

3.1.1. Внешняя согласованность. Модель внешней согласованности является самой строгой моделью согласованности. Более детально модель внешней согласованности рассматривается в работе [25]. В данной модели выполняется следующее правило: *любая операция чтения области памяти x получает значение последней успешной операции записи в x*. На рисунке 1 процесс P_2 читает значение в области памяти x успешно записанное процессом P_1. Рассматриваемая модель согласованности используется в распределенной базе данных Google Spanner [5].

3.1.2. Последовательная согласованность. Последовательная модель согласованности основана на следующей идее: операции процессов, запущенных на разных серверах, могут чередоваться. При этом операции каждого отдельного процесса

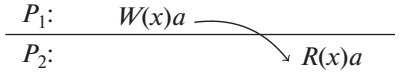

Рис. 1. Внешняя согласованность.

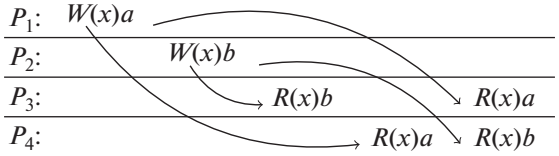

Рис. 2. Последовательная согласованность.

должны быть видны в этой последовательности в порядке, в каком они были выполнены в рамках процесса. Все процессы должны видеть измененные значения данных в одном и том же порядке. В отличие от внешней согласованности, в последовательной модели согласованности не требуется синхронизация глобального времени между всеми процессами, что ослабляет согласованность данных. Как показано на рисунке 2, для P_3 сначала были получены данные от выполнения операции процесса P_2, а потом P_1. Для P_4 были выполнены операции процесса P_1, а затем P_2, что нарушает последовательную согласованность.

3.1.3. Ситуационно-зависимая согласованность. Согласованность данной модели определяется соблюдением следующего условия частичного порядка выполнения операции процессов: выполнение операций процессов P_1, P_2 видны всем процессам в одинаковом порядке в том случае, если операция записи области памяти x процессом P_1 произошла до выполнения операции чтения/записи области памяти x процессом P_2, и выполнение операции чтения/записи процессом P_2 зависит от выполнения операции записи процессом P_1. Такой частичный порядок выполнения операций разных процессов может быть достигнут, например, с помощью часов Лампорта [6]. Если операции чтения/записи процессов P_1, P_2 работают с разными областями памяти, то такие процессы не являются взаимосвязанными и могут быть выполнены на разных серверах в разном порядке, что не допустимо во внешней модели согласованности. Один из примеров реализации ситуационно-зависимой согласованности представлен в работе [26]. На рисунке 3 изображен пример, иллюстрирующий рассматриваемую модель согласованности: операция $W(x)b$ процесса P_2 зависит от выполнения операции $W(x)a$ процесса P_1. Процессы P_3 и P_4 видят одинаковый порядок операции процессов. Операции $W(x)c$ и $W(x)b$ процессов P_1, P_2 являются параллельными.

3.1.4. PRAM(FIFO)-согласованность. PRAM (FIFO)-согласованность определяет следующее правило: операции записи, выполненные процессом на одном сервере, должны быть видны всем другим процессам в том порядке, в котором они были выполнены. При этом операции записи разных процессов могут быть видны в различном порядке на разных серверах.

3.2. Клиент-ориентированные модели согласованности

Если в *моделях согласованности ориентированных на данные* делается акцент на согласованность данных в хранилище между всеми репликами, то в клиент-ориентированных моделях не требуется, чтобы все реплики находились в актуальном состоянии.

Класс клиент-ориентированных моделей согласованности допускает работу клиентских процессов с неактуальными данными на серверах в силу того, что актуальные данные не были распространены по всем серверам. Такое послабление согласованности данных позволяет повысить доступность серверов и увеличить пропускную способность. В статье [27] формально определены модели данного класса.

Для определения разновидностей клиент-ориентированных моделей согласованности потребуются следующие обозначения. Предполагаем, что клиентская согласованность рассматривается в рамках одного процесса P_i. Под L_1, L_2 будем подразумевать

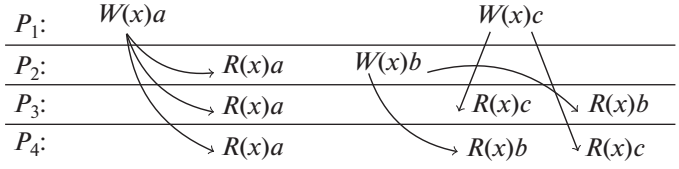

Рис. 3. Ситуационно-зависимая согласованность.

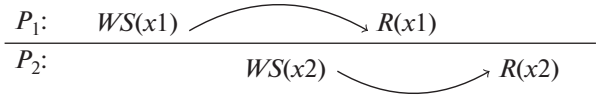

Рис. 4. Монотонное чтение.

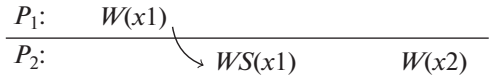

Рис. 5. Монотонная запись.

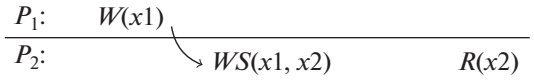

Рис. 6. Читай то, что записал.

реплики, размещенные на разных физических серверах, одного и того же хранилища. Процесс P_i выполняется на реплике L_i. Запись $R(x_j)$ означает операцию чтения данных в области памяти x_j. Операция $WS(x_1, x_2, .., x_n)$ означает множество локальных операций записи некоторого сервера и операций записи, полученных от других серверов в ходе распространения результатов операций записи.

3.2.1. Согласованность в конечном счете. Как утверждается в [28], реплицированные данные, обновленные на одном сервере, в конечном счете обновятся на остальных серверах. Если нет новых обновлений со стороны клиентов, то реплики перейдут в согласованное состояние. При частом обновлении состояние данных в серверах может различаться, что может привести к чтению несогласованных данных из разных серверов [7]. Модель согласованности в конечном счете допускает возникновение аномалии потерянного обновления. Данная модель реализована в отказоустойчивых системах с ленивой моделью репликации [20, 21, 29].

3.2.2. Монотонные чтения. Хранилище реализует модель согласованности монотонного чтения [30] при выполнении следующего условия: если процесс P_i прочитал значение $R(x)$, то, при последующих обращениях к этим данным, процесс получит это же или более новое значение.

На рисунке 4 процесс P_1 пишет в одну реплику значение x_1 и через некоторое время читает его. Спустя некоторое время на вторую реплику распространяется значение x_1 и после чего процесс P_2 пишет во вторую реплику другое значение x_2, а затем в результате чтения на этой реплике клиент гарантированно получит более свежее значение x_2.

3.2.3. Монотонные записи. Идея данной модели [30, 31] заключается в следующем: все операции записи упорядочены в рамках процесса и в таком же порядке распространяются на другие сервера. Как показано на рисунке 5, клиент выполняет на второй реплике операцию записи значения в область памяти x_2 в том случае, если значение в области памяти x_1 успешно распространилось с первой реплики на вторую. В случае, если значение в x_1 не распространилось на вторую реплику и при этом клиент пишет во вторую реплику значение в x_2, согласованность данной модели нарушается.

В отличие от модели *монотонного чтения*, в которой делается акцент на операции чтения в рамках одного процесса, в модели согласованности *монотонной записи* делается акцент на операции записи того же самого процесса.

3.2.4. Читай то, что записал. Как утверждается в [30, 31], это модель, в которой процесс P_i читает на реплике результат последней успешно завершенной операции записи, выполненной на другой реплике тем же процессом. Данная модель согласованности является частным случаем ситуационно-зависимой согласованности. Согласно [27], модель согласованности *читай то, что записал* является более сильной моделью по сравнению с моделью согласованности *монотонного чтения*. Это обусловлено тем, что в модели *читай то, что записал* результат операции записи процессом немедленно распространяется на сервера, на которых этот же процесс выполняет операцию чтения. Данное условие опущено для модели согласованности *монотонного чтения*. Как показано на рисунке 6, процесс P_1 пишет значение, а затем процесс P_2 сначала выполняет операцию $WS(x1, x2)$ и следом опе-

Таблица 2. Пример различных решений, реализующих асинхронную и синхронную модели

	Основной сервер	Любой сервер
Синхронная модель	CitusDB [14], PostgreSQL [15]	Dynamo [21]
Асинхронная модель	MongoDB [17]	Cassandra [20]

рацию чтения, возвращающую актуальное значение x_2.

3.2.5. Запись следует за чтением. Согласно [27, 30, 31], это модель согласованности, в которой процесс P_i производит запись нового значения в область памяти x некоторого сервера только после операции чтения значения x. Данная модель является более сильной моделью согласованности по сравнению с моделью *монотонная запись*, согласно [27]. Это обусловлено тем, что в модели *запись следует за чтением* для новой операции записи в рамках процесса P_i на другом сервере важен результат записи операции в область памяти x, выполненный процессом на другом сервере.

Достоинством таких моделей согласованности, как *монотонная запись*, *монотонное чтение* и *читай то, что записал*, является относительная простота реализации [30, 31]. Если клиент работает с одним сервером, данные модели согласованности гарантируют согласованность данных. В случае, если изначальная сессия на одном сервере оборвалась и клиент подключился на другой сервер, то возникает проблема синхронизации данных между серверами и, как результат, чтение несогласованных данных.

Все перечисленные модели согласованности клиент-ориентированного класса в конечном счете распространяют значения от одного сервера к другим. Когда говорят о согласованности в конечном счете, обычно подразумевают использование связи двух моделей данного класса – *монотонная запись* и *читай то, что записал* [27].

4. ПРОТОКОЛЫ СИНХРОНИЗАЦИИ СЕРВЕРОВ

Важной деталью при реализации алгоритма репликации базы данных является выбор способа копирования данных на все сервера. Согласно [32, 33], существует две модели распространения данных по всем серверам: синхронная модель и асинхронная.

4.1. Синхронная модель

В рамках одной транзакции результат работы, выполненный на одном сервере, фиксируется на других серверах. При этом основной сервер ожидает завершения всех транзакций на остальных серверах. В синхронной модели на всех серверах определена одна и та же модель согласованности, как и на основном сервере. Недостатком синхронной модели является откат глобальной транзакции в случае, если откатывается одна из локальных транзакций. Другим недостатком синхронной модели является высокое время отклика серверов в следствие синхронизации всех реплик в рамках глобальной транзакции при использовании алгоритма двухфазной фиксации [34].

Достоинством данной модели является гарантия согласованности данных на всех серверах. В случае, если основной сервер становится недоступным, клиенты могут быть уверены, что их данные не потеряны, а реплику возможно восстановить из других серверов. Например, синхронная модель реализована в расширении CitusDB [14] для СУБД PostgreSQL. В расширении CitusDB назначается специальный сервер с ролью координатора, который принимает все клиентские запросы и координирует синхронизацию реплик посредством алгоритма двухфазной фиксации [34].

4.2. Асинхронная модель

В асинхронной модели данные фиксируются на основном сервере. После чего выполняются независимые транзакции для обновления данных на других репликах. В отличие от синхронной модели, основной сервер не ожидает выполнения транзакций на остальных серверах и это позволяет увеличить пропускную способность, так как отсутствуют блокирующие операции, в отличие от синхронной репликации. В конечном счете через некоторое время данные на других серверах будут обновлены. Недостатком данной модели является несогласованность данных на разных репликах в некоторый момент времени. Асинхронная модель обеспечивает более высокую доступность системы, чем синхронная. В системах, где производительность системы является важным фактором,

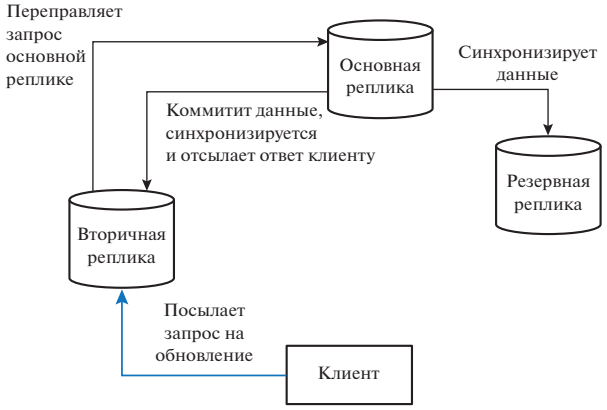

Рис. 7. Протокол удаленной записи.

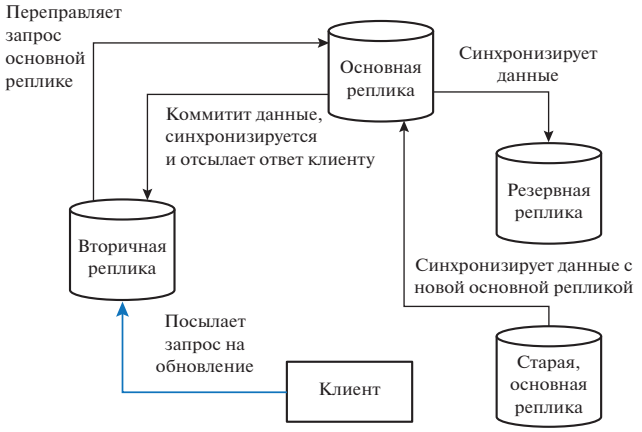

Рис. 8. Протокол локальной записи.

данная модель синхронизации данных остается предпочтительной. Асинхронная модель реализована в таких системах, как [17, 20, 21].

4.3. Сопоставление протоколов синхронизации серверов

В работах [33, 35] рассматриваются распределенные версии асинхронной и синхронной моделей репликации. В таблице 2 представлены различные централизованные и децентрализованные решения, использующие асинхронную и синхронные модели синхронизации данных. В случае с синхронной моделью, обновления могут распространяться от основного сервера к вторичным, где основной сервер играет роль мастер-сервера, обрабатывая все запросы клиентов, блокируя вторичные сервера для синхронизации данных [33]. В качестве альтернативного варианта возможна стратегия, когда клиент посылает запрос всем серверам для обновления данных. В таком случае все транзакции выполняются атомарно [33].

5. ПРОТОКОЛЫ СОГЛАСОВАННОСТИ

В главе 3 мы обсудили различные модели согласованности. В данном разделе будут рассмотренны реализации клиент-ориентированных моделей согласованности и моделей согласованности ориентированных на данные.

5.1. Протоколы на основе первичной копии

Семейство протоколов на основе первичной копии с использованием последовательной модели согласованности работает с упорядоченной последовательностью клиентских запросов. Класс протоколов на основе первичной копии разделен на следующие типы: *протокол удаленной записи*, *протокол локальной записи*, *протокол трансляции журнала транзакций* и *протокол логической репликации*.

В протоколах на основе первичной копии используются следующие принципы работы. Все клиентские запросы обрабатываются специально выделенным сервером называемым *основным* или *ведущим*. Реплика на ведущем сервере называется основной. Сервер, который отслеживает изменения на ведущем сервере, называется *ведомым* или *резервным*. Резервный сервер, к которому нельзя подключаться до тех пор, пока он не будет повышен до ведущего сервера, называется сервером *теплого резерва*. Сервер, который может принимать соединения и обрабатывать клиентские запросы только на чтение, называется сервером *горячего резерва*.

5.1.1. Протокол удаленной записи. Алгоритм протокола удаленной записи [36], изображенный на рисунке 7, состоит из следующих шагов:

1. Основной сервер обрабатывает клиентский запрос и обновляет локально данные.

2. Затем создается запрос на обновление реплики на резервном сервере.

3. Дождавшись ответа от резервного сервера, основной сервер отправляет ответ клиенту.

Если запрос отправляется на один из резервных серверов, то он перенаправляется на основной сервер и запрос проходит все вышеперечисленные шаги протокола.

Недостатком данного подхода является блокирующая операция обновления основного и резервного серверов. Это порождает проблемы с производительностью и доступностью системы. В качестве решения могут быть введены неблокирующие операции [36]. На 1 фазе алгоритма основной сервер отправляет запрос резервному серверу и не дожидается от него ответа. Тем самым увеличивается пропускная способность базы данных.

5.1.2. Протокол локальной записи. В отличие от протокола удаленной записи, в протоколе локальной записи за основу взят подход локального выполнения операции чтения и записи. Такой способ работы характерен для устройств, которые могут работать в оффлайн режиме, например, не подключенные к сети Интернет ноутбуки. В данном алгоритме, устройство может работать со своей копией данных как с основной репликой в оффлайн режиме. Как только устройство становится доступным в сети, происходит процесс синхронизации локальной реплики с основной, как показано на рисунке 8.

5.1.3. Протокол трансляции журнала транзакций. Протокол, реализованный в таких базах данных как PostgreSQL [15] и MySQL [16], довольно прост: ведущий сервер принимает клиентские запросы и заносит запись о результате работы транзакций в журнал транзакций, в то время как каждый резервный сервер работает в режиме приема записей журнала транзакций в виде файлов от ведущего сервера. Если основной сервер отказывает, резервный сервер, содержащий почти все данные с основного сервера, может быть быстро преобразован в новый ведущий сервер. Помимо трансляции журнала транзакций в виде файлов, существует *потоковая репликация*, которая отличается от протокола трансляции журнала транзакций в виде файлов тем, что происходит непрерывная передача записей журнала транзакций резервным серверам в потоковом режиме. Такой подход позволяет работать резервным серверам с меньшей задержкой нежели при передаче файлов журнала транзакций [37].

5.1.4. Протокол логической репликации. Протокол логической репликации появился сравнительно недавно. Реализации протокола встречаются в таких популярных базах данных, как PostgreSQL [15], MySQL [16], OracleDB [38]. В протоколе используется концепция модели читатель-писатель. Основной сервер является писателем, а ведомые – читателями. На стороне писателя определяются данные, которые должны быть распространены на ведомые сервера. Такие данные называются публикациями. Изменения, производимые над публикациями, последовательно распространяются на читателей, тем самым гарантируется согласованность данных.

Недостатком протокола логической репликации является возникновение нарушения согласованности данных в случае, если на стороне читателя выполняются клиентские операции записи. Преимуществом протокола является его относительная простота реализации и возможность выбирать, какие данные могут быть реплицированы.

5.2. Кворумные протоколы

Основная идея кворумных протоколов заключается в следующем: клиент должен получить подтверждение большинства серверов на операции чтения или записи [1–3, 39, 40]. Под кворумом подразумевается количество $\frac{N}{2}+1$ серверов, где N – общее количество серверов. Протоколы на основе кворума широко используются во многих реализациях [5, 41, 42].

Существует две группы кворумных протоколов. Первая группа основана на семействе алгоритмов Paxos. Вторая группа протоколов основана на базе алгоритма Raft. Каждая из групп имеет свои преимущества и недостатки, но их объединяет свойство доступности большинства серверов. Недостатком обеих групп протоколов данного семейства является следующее: в случае, если клиент запрашивает на чтение данные у одного из серверов, то клиент может получить неактуальные данные. Это возможно в силу того, что сервера не успели получить данные от серверов, обладающих свежими данными. Преимуществом является доступность серверов не участвующих в обработке клиентского запроса.

Группа кворумных протоколов имеет следующие ограничения:

- $N_R + N_W > N$
- $N_W > \frac{N}{2}$,

где N – общее количество серверов, N_R – кворум серверов готовых принять запрос на чтение, и N_W – кворум серверов на операцию обновления. Первое ограничение означает правило, при котором при чтении актуальных данных необходим как минимум один сервер из кворума записи, имеющий актуальные данные. Второе ограничение означает правило, при котором для записи обновленных данных необходим кворум из более, чем половины серверов.

5.2.1. Протоколы на базе алгоритма Paxos. Алгоритмы семейства Paxos призваны решить задачу консенсуса в сети ненадежных компонент [24]. Л. Лампорт предложил [43] базовый алгоритм Paxos, работающий с множеством входных данных и гарантирующий одно и только одно выходное значение. В алгоритме Paxos выделяются следующие роли:

- *Клиент*. Клиент отсылает запрос заявителю на обработку.
- *Заявитель*. Получает запросы от клиентов на обработку. Заявитель пытается получить кворум выборщиков для обработки данных, отправленных в запросе клиентом.

• *Выборщик*. Получает запросы от заявителя, одобряет или отклоняет их. Хранит в себе данные клиентских запросов. Помимо этого, содержит специальный целочисленный идентификатор, увеличивающийся каждый раз при одобрении обработать клиентский запрос.

В общих чертах, базовый алгоритм Paxos состоит из двух фаз [23]:

1. *Подготовительная фаза*. На этой фазе заявитель инициирует голосование, в котором получает либо одобрение от выборщиков на обработку клиентского запроса, либо отказ. Заявитель отправляет всем выборщикам сообщение с некоторым целочисленным идентификатором. Каждый из выборщиков принимает сообщение и, если идентификатор сообщения более свежий, чем хранящийся у выборщика, выборщик обещает не принимать сообщения с идентификатором меньше, чем у принятого сообщения. После чего выборщик посылает положительный ответ заявителю.

2. *Фаза принятия*. Как только заявитель получает одобрение выборщиков, он отправляет данные всем выборщикам, согласным принять клиентский запрос.

Один из примеров системы, в которой используется Paxos, рассмотрен в работе [41]. Система организована как множество подмножеств серверов баз данных. В каждом из множеств выбирается локальный лидер. С помощью алгоритма двухфазной фиксации результат работы одного подмножества может быть распространен другим подмножествам серверов [34].

Еще одим примером использования алгоритма Paxos является система, разработанная Дж. Бэкером с коллегами под названием Megastore [42]. Данная система поддерживает ACID семантику, хорошо масштабируется. Paxos используется для синхронизации данных по всем серверам.

Помимо базового алгоритма Paxos, существуют его различные расширения. Одно из них – Multi-Paxos. В случае базового алгоритма Paxos запросы поступают от одного клиента. В реализации Multi-Paxos клиентов может быть более одного. В таком случае возможны ситуации, когда одновременно проходят несколько стадий голосования по разным клиентским запросам. Выполнение параллельных фаз голосования в распределенной базе данных может привести к тому, что порядок получившихся результатов голосования может привести все сервера в системе к несогласованному состоянию. Такая проблема может быть решена путем использования журнала серверов, описанного в работе [24]. В работе [5] представлен пример реализации алгоритма Multi-Paxos в связке с внешней моделью согласованности.

Стоит заметить, что в системе из $2F + 1$ серверов одновременно не могут отказать F серверов. Например, в системе из 100 серверов могут отказать 10. В таком случае для принятия решения необходим кворум из 21 сервера.

Немалый интерес представляют другие алгоритмы из семейства Paxos. Один из них – Cheap Paxos [44]. Алгоритм отличается от базового алгоритма Paxos количеством $F + 1$ работающих серверов. Другие F вспомогательных серверов необходимы для переконфигурации системы в случае отказа менее половины серверов. В базовой версии алгоритма требуется для работы $2F + 1$ серверов.

В отличие от систем, где используется любой консенсус-алгоритм, в системах, где используется 2PC для фиксации обновления на всех серверах, отказ хотя бы одного сервера останавливает работу всей системы. При обновлении данных на всех серверах через алгоритм двухфазной фиксации пропускная способность системы ниже, чем в системах на базе алгоритма консенсуса.

5.2.2. Протокол на базе алгоритма Raft. Алгоритм Raft рассматривается как альтернатива семейству алгоритмов Paxos. В отличие от Paxos, алгоритм Raft более понятен в ходе изучения и прост в реализации. Разработчики алгоритма утверждают, что по характеристикам отказоустойчивости и пропускной способности он ничем не уступает алгоритмам семейства Paxos [45, 46]. Исследователи формально доказали корректность данного алгоритма [45, 47].

Алгоритм Raft определяет следующие роли: *лидер*, *ведомый*, *кандидат*. Каждая из перечисленных ролей ответственна за определенные задачи. Raft разделяет две основные задачи:

• *Выборы лидера*. Изначально, все сервера в кластере выступают в роли кандидатов. Алгоритм выбора лидера начинается с фазы голосования. Каждый сервер рассылает сообщения, что готова стать лидером и голосует за других. В конечном счете один и только один сервер становится лидером. Дабы поддерживать статус лидера, сервер рассылает небольшие сообщения ведомым, подтверждая факт, что он все еще доступен и способен принимать клиентские запросы. В случае, если ведомые не принимают сообщения от лидера, они начинают заново выбирать лидера.

• *Репликация лога*. Лидер принимает запросы от клиента, подготавливает лог-сообщение, содержащее упорядоченное множество команд чтения и записи, которые должны быть выполнены на ведомых серверах, и отправляет сообщения ведомым. В случае, если лидер получает от большинства серверов готовность принять сообщение, лидер фиксирует результат локально, а затем рассылает сооб-

щения ведомым серверам из кворума. Каждая ведомая реплика фиксирует лог-сообщение локально.

Задача алгоритма Raft, как и алгоритма Paxos, – принятие решения о готовности обработать клиентский запрос. Основное отличие алгоритма Raft от Paxos состоит в том, что в Raft используется сервер-координатор, принимающий клиентские запросы. В Paxos обработать клиентский запрос может любой сервер. Другим отличием алгоритма Raft от Paxos является фаза голосования. Если в Raft отсылается сообщение о готовности принять новые данные, то в алгоритме Paxos, как уже было сказано, сервер с ролью выборщика сравнивает специальное целочисленное значение, принятое от заявителя с идентификатором, хранимым в выборщике. От сравнения идентификаторов зависит сможет ли сервер обработать клиентский запрос. Недостатком алгоритма Raft является задержка при передаче сообщений между серверами.

На данный момент известны десятки реализации алгоритма Raft [46]. Алгоритм Raft используется в распределенной базе данных CockroachDB [18].

5.3. Активная репликация

Согласно [33], основная идея протокола активной репликации заключается в том, что все сервера принимают и обрабатывают одни и те же клиентские процессы. Это существенное отличие от алгоритмов на основе консенсуса, в которых подмножество серверов обрабатывают клиентский процесс. Согласованность данных гарантируется тем, что все сервера получают на вход один и тот же порядок клиентских процессов и гарантировано возвращают одинаковые ответы на запросы. Клиентский процесс взаимодействует не с одним сервером, а со всеми. Достоинством данного подхода является его простота. Недостатком протокола активной репликации является сложность обработки ситуации, в которой необходимо восстановить данные одного из поврежденных серверов.

Активная репликация используется крайне редко. Это обусловлено тем, что операция обновления всех серверов достаточно дорогостоящая. Алгоритмы на основе кворумных протоколов выглядят предпочтительнее.

6. ДРУГИЕ ПОДХОДЫ К ЗАДАЧЕ РЕПЛИКАЦИИ

В данном разделе рассмотрены работы, использующие различные модели согласованности.

В работе [22] представлен обзор схем репликации и алгоритмов распространения данных по всем серверам. Исследователи не провели обзор классов моделей согласованности и их реализаций.

В динамической, распределенной, отказоустойчивой, масштабируемой NoSQL базе данных MongoDB [17] используется согласованность в конечном счете. Данная система предоставляет возможность выбора одного из двух режимов синхронизации реплик. Идея первого режима состоит в выделении основного сервера, принимающего запросы клиентов, и ведомых. Во втором режиме любой сервер может принимать клиентские запросы.

В работе [8] представлен новый репликационный протокол TAPIR, основаный на слабой модели согласованности. TAPIR предоставляет все свойства ACID семантики. Разработчики в [8] смогли добиться фиксации транзакции в распределенной системе за одну фазу. Также была реализована возможность децентрализованной коммуникации между серверами.

Google предложила высокопроизводительную систему BigTable [19]. Динамическая система использует согласованность в конечном счете. BigTable предоставляет возможность через настройки выбрать модель согласованности *читай то, что записал*.

Amazon разработала децентрализованную систему баз данных Dynamo [21]. Система поддерживает согласованность в конечном счете в связке с кворум-алгоритмом Paxos. Представленная база данных хорошо масштабируется; отказоустойчивость гарантируется выбором нового лидера-координатора в случае, если предыдущий лидер недоступен.

В статье [26] представлено распределенное решение, основанное на ситуационно-зависимой модели согласованности в распределенной базе данных. Используя ситуационно-зависимую модель согласованности и выполнения неблокирующих операции чтения ко всем серверам, COPS гарантированно возвращает клиенту согласованные данные.

В статье [48] представлен фреймворк для NoSQL решений. Решение реализует модель строгой согласованности и предоставляет возможность динамически масштабировать распределенную базу данных. Фреймворк использует алгоритм четырехфазной фиксации4PC, гарантирующий свойства ACID семантики. Недостатком системы является высокое время отклика вследствие обмена сообщениями между серверами.

7. ЗАКЛЮЧЕНИЕ

В работе рассматриваются классы моделей согласованности и протоколы синхронизации серверов. Репликация данных решает проблему отказоустойчиво-

сти в распределенной базе данных, повышает пропускную способность чтения данных. Также, были проанализированы различные работы, использующие описанные в статье модели согласованности.

СПИСОК ЛИТЕРАТУРЫ

1. *Andrew S.* Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006. ISBN 0132392275.
2. *Cheung S.Y., Casavant T.L., Singhal M., Ahamad M., Ammar M.H.* Replicated data management in distributed systems. 1994.
3. *Jiménez-Peris R., Pati no-Martínez M., Kemme B., Alonso G.* How to select a replication protocol according to scalability, availability, and communication overhead. In *SRDS*, 2001.
4. *Anderson T., Breitbart Yu., Korth H.F., Wool A.* Replication, consistency, and practicality: Are these mutually exclusive? *SIGMOD Rec.* 1998. V. 270 (2). P. 484–495. ISSN 0163-5808.
5. *Corbett J.C., Dean J., Epstein M., Fikes A., Frost Ch., Furman J.J., Ghemawat S., Gubarev A., Heiser Ch., Hochschild P., Hsieh W., Kanthak S., Kogan E., Li H., Lloyd A., Melnik S., Mwaura D., Nagle D., Quinlan S., Rao R., Rolig L., Yasushi Saito, Szymaniak M., Taylor Ch., Wang R., Woodford D.* Spanner: Googles globally distributed database. *ACM Trans. Comput. Syst.* 2013. V. 310 (3). P. 8:1–8:22. ISSN 0734-2071.
6. *Mani Chandy K. and Lamport L.* Distributed snapshots: Determining global states of distributed systems. *ACM Trans. Comput. Syst.* 1985. V. 30 (1). P. 63–75. ISSN 0734-2071. https://doi.org/10.1145/214451.214456. URL http://doi.acm.org/10.1145/214451.214456.
7. *Bailis P., Ghodsi A.* Eventual consistency today: Limitations, extensions, and beyond. *Queue*. 2013. V. 110 (3). P. 20:20–20:32. ISSN 1542-7730.
8. *Zhang I., Sharma N.Kr., Szekeres A., Krishnamurthy A., Ports D.R.K.* Building consistent transactions with inconsistent replication. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP'15. New York, NY, USA. 2015. P. 263–278 ACM. ISBN 978-1-4503-3834-9.
9. *Oki B.M., Liskov B.H.* Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing*, PODC'88, pages 8–17, New York, NY, USA, 1988. ACM. ISBN 0-89791-277-2.
10. *Mansouri N., Dastghaibyfard G.H., Mansouri E.* Combination of data replication and scheduling algorithm for improving data availability in data grids. *Journal of Network and Computer Applications*. 2013. V. 360 (2). P. 711–722. ISSN 1084-8045.
11. *Elmirghani J.M.H., El-Gorashi T.E.H.* Data replication schemes for a distributed storage scenario. Munich, Germany, 07 2010. IEEE Computer Society. ISBN 978-1-4244-7798-2.
12. *UroEЎ ДЊibej, BoEЎtjan Slivnik, and Borut RobiДЌ.* The complexity of static data replication in data grids. *Parallel Computing*, 2005. V. 310 (8). P. 900–912, 2005. ISSN 0167-8191.
13. *Chervenak A., Deelman E., Foster I., Guy L., Hoschek W., Iamnitchi A., Kesselman C., Kunszt P., Ripeanu M., Schwartzkopf V. et al.* Giggle: A framework for constructing scalable replica location services. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, page 1–17. IEEE Computer Society Press, IEEE Computer Society Press, 2002.
14. *CitusDB official website*, 2019. URL www.citusdata.com.
15. *PostgreSQL official website*, 2019. URL www.postgresql.org.
16. *MySQL official website*, 2019. URL www.mysql.com.
17. *Banker K. Mongo DB in Action*. Manning Publications Co., Greenwich, CT, USA, 2011. ISBN 1935182870, 9781935182870.
18. *Cockroach DB official website*, 2019. URL www.cockroachlabs.com.
19. *Chang F., Dean J., Ghemawat S., Hsieh W.C., Wallach D.A., Burrows M., Chandra T., Fikes A., Gruber R.E.* Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.* 2008. V. 260 (2). P. 4:1–4:26. ISSN 0734-2071.
20. Lakshman A., Malik P. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.* 2010. V. 440 (2). P. 35–40. ISSN 0163-5980.
21. *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. *SIGOPS Oper. Syst. Rev.* 2007. V. 410 (6). P. 205–220. ISSN 0163-5980.
22. *Martins V., Pacitti E., Valduriez P.* Survey of data replication in P2P systems. Research Report RR-6083, INRIA, 2006.
23. *Lamport L.* Paxos made simple. *ACM SIGACT News 32.*
24. *Van Renesse R., Altinbuken D.* Paxos made moderately complex. *ACM Comput. Surv.* 2015. V. 470 (3). P. 42:1–42:36. ISSN 0360-0300.
25. *Kenneth Gifford D. Information Storage in a Decentralized Computer System*. PhD thesis, Stanford, CA, USA, 1981. AAI8124072.
26. *Lloyd W., Freedman M.J., Kaminsky M., Andersen D.G.* Don't settle for eventual: Scalable causal consistency for wide-area storage with cops. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP'11. New York, NY, USA, 2011. P. 401–416. ACM. ISBN 978-1-4503-0977-6.
27. *Zhu Y., Wang J.* Client-centric consistency formalization and verification for system with large-scale distrib-

uted data storage. *Future Generation Computer Systems*. 2010. V. 260 (8). P. 1180–1188, 2010. ISSN 0167-739X.

28. *Lamport L.* Readings in computer architecture. chapter How to Make a Mulitprocessor Computer That Correctly Executes Multiprocess Programs, pages 574–575. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000. ISBN 1-55860-539-8.
29. *Petersen K., Spreitzer M.J., Terry D.B., Theimer M.M., Demers A.J.* Flexible update propagation for weakly consistent replication. *SIGOPS Oper. Syst. Rev.* 1997. V. 310 (5). P. 288–301. ISSN 0163-5980.
30. *Petersen K., Spreitzer M.J., Theimer M.M., Welch B.B., Terry D.B., Demers A.J.* Session guarantees for weakly consistent replicated data. In *Proceedings of 3rd International Conference on Parallel and Distributed Information Systems*, Austin, TX, USA, 1994. IEEE. ISBN 0-8186-6400-2.
31. *Bernstein P.A., Das S.* Rethinking eventual consistency. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD'13, pages 923–928, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2037-5. URL http://dl.acm.org/10.1145/2463676.2465339.
32. *Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems.* O'Reilly, 2016. ISBN 978-1-4493-7332-0.
33. *Wiesmann M., Pedone F., Schiper A., Kemme B., Alonso G.* Understanding replication in databases and distributed systems. In *Proceedings of the The 20th International Conference on Distributed Computing Systems (ICDCS 2000)*, ICDCS'00. Washington, DC, USA, 200. P. 464. IEEE Computer Society. ISBN 0-7695-0601-1.
34. *Daniel A. and Tatuo Nakanishi*. Performance evaluation of a two-phase commit based protocol for ddbs. ACM, 1982. P. 247–255. ISBN 0-89791-070-2.
35. *Gray J., Helland P., O'Neil P., Shasha D.* The dangers of replication and a solution. *SIGMOD Rec.*, 250 (2):0 173–182, June 1996. ISSN 0163-5808.
36. *Budhiraja N., Marzullo K., Schneider F.B., Toueg S.* Distributed systems (2nd ed.). chapter The Primary-backup Approach, pages 199–216. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1993. ISBN 0-201-62427-3.
37. *Streaming replication*, 2019. URL postgrespro.ru/docs/postgrespro/11/warm-standby.
38. *Oracle official website*, 2019. URL www.oracle.com.
39. *Rane D. and Mahendra Dhore*. Overview of data replication strategies in various mobile environment. 04 2016.
40. *Thomas R.H.* A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Syst.* 1979. V. 40 (2). P. 180–209.
41. *Glendenning L., Beschastnikh I., Krishnamurthy A., Anderson T.* Scalable consistency in scatter. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP'11. New York, NY, USA, 2011. P. 15–28. ACM. ISBN 978-1-4503-0977-6.
42. *Baker J., Bond C., Corbett J.C., Furman J.J., Khorlin S., Larson J., Leon J.-M., Li Y., Lloyd A., Yushprakh V.* Megastore: Providing scalable, highly available storage for interactive services. In *CIDR*, 2011.
43. *Lamport L.* The part-time parliament. *ACM Trans. Comput. Syst.* 1998. V. 160 (2). P. 133–169. ISSN 0734-2071.
44. *Lamport L., Massa M.* Cheap paxos. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, DSN'04. Washington, DC, USA, 2004. P. 307. IEEE Computer Society. ISBN 0-7695-2052-9.
45. *Ongaro D., Ousterhout J.* In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14. Berkeley, CA, USA, 2014. P. 305–320. USENIX Association. ISBN 978-1-931971-10-2.
46. *Raft consensus algorithm website*, 2019. URL https://raft.github.io/.
47. *Woos D., Wilcox J.R., Anton S., Tatlock Z., Ernst M.D., Anderson T.* Planning for change in a formal verification of the raft consensus protocol. In *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2016. P. 154–165, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4127-1.
48. *Lotfy A.E., Saleh A.I., El-Ghareeb H.A., Ali H.A.* A middle layer solution to support acid properties for nosql databases. *Journal of King Saud University – Computer and Information Sciences*. 2016. V. 280 (1). P. 133–145, 2016. ISSN 1319-1578.