Системы управления данными категории NoSQL

С. Д. Кузнецов <kuzloc@ispras.ru>, А. В. Посконин <aposk@yandex.ru>

Факультет вычислительной математики и кибернетики
Московского государственного университета им. М. В. Ломоносова

Аннотация. В последнее десятилетие активно стали появляться и развиваться системы управления данными, получившие собирательное название «NoSQL». Основными особенностями таких систем являются отказ от реляционной модели данных и языка SQL, отсутствие полноценной поддержки ACID-транзакций, использование распределённой архитектуры (хотя существуют и нераспределённые NoSQL-системы). Благодаря этому в ряде задач удаётся добиться производительности, превосходящей производительность традиционных SQL-ориентированных СУБД, а также обеспечить хорошую масштабируемость при возрастающих нагрузках и огромных объемах данных, что является крайне важным, в частности, для Web-приложений. К сожалению, отсутствие транзакционной семантики накладывает некоторые ограничения на класс задач, которые можно эффективно решать с помощью NoSQL-систем, а выбор конкретной системы сильно зависит от решаемой задачи. В данной работе предлагается обзор основных классов систем управления данными, которые наиболее часто относят к категории NoSQL, рассматриваются примеры конкретных систем и задач, которые могут быть решены с их помощью.

Ключевые слова. NoSQL, нереляционные модели данных, масштабируемость, репликация, шардинг, согласованность данных.

1. Введение

Термин «NoSQL» был впервые применен в 1998 году Карло Строцци (Carlo Strozzi) в качестве названия для его небольшой реляционной СУБД, которая не использовала язык SQL для манипулирования данными [1]. С 2009 года термин «NoSQL» стал использоваться уже для обозначения растущего числа распределенных систем управления данными, которые отказывались от поддержки ACID-транзакций (Atomicity, Consistency, Isolation, Durability – Атомарность, Согласованность, Изолированность, Постоянство хранения) – одного из ключевых принципов работы с реляционными базами данных [2]. По мнению Строцци, современные системы категории NoSQL точнее было бы называть нереляционными («NoREL») [1]. В настоящее время термин «NoSQL» обычно расшифровывается как «Not Only SQL» , то есть «Не Только SQL» [3].

Главной предпосылкой к появлению систем, относимых к категории NoSQL, послужила растущая потребность в горизонтальной масштабируемости приложений, то есть в возможности наращивать производительность путём добавления новых вычислительных узлов к уже работающим. Таким образом, большинство NoSQL-систем изначально проектировались и создавались для работы в распределенной среде - кластере или облаке, где применение традиционных SQL-ориентированных систем связано с определёнными трудностями (см., например, [4], [5]). Основной причиной отказа от поддержки транзакционной семантики послужила сложность эффективной реализации транзакций в распределённой среде: в общем случае приходится использовать двухфазный протокол фиксации транзакций, который требует пересылки большого количества сообщений по сети (см., например, [6]). Хотя NoSQL-системы обычно не поддерживают ACID-транзакции в полном объеме, в ряде случаев поддерживаются, например, атомарные операции для чтения

и модификации, оптимистические блокировки и другие инструменты, помогающие упростить разработку приложений в условиях параллельного доступа к данным.

В данной работе будут рассматриваться системы, относимые в публикациях к категории NoSQL. К сожалению, рассмотреть все существующие NoSQL-решения не представляется возможным (на момент написания работы список [3] насчитывал порядка 150 систем), а многие системы достаточно сложны и обладают богатой функциональностью, поэтому в соответствующих разделах будут приведены только ключевые особенности конкретных решений, такие как модель данных, возможности масштабирования и т.д. Кроме того, NoSQL-движение является достаточно молодым, поэтому многие проекты находятся в стадии активной разработки и претерпевают значительные изменения от версии к версии. Наиболее актуальную и полную информацию о рассматриваемых системах читатель может получить, используя ссылки, приведённые в тексте.

2. Особенности NoSQL-систем

Как уже было отмечено, большинство NoSQL-систем являются распределенными. Распределённая архитектура позволяет достичь не только горизонтальной масштабируемости (в идеале – линейного роста производительности при добавлении новых узлов), но и увеличить надежность системы с помощью поддержания нескольких копий данных. В распределённых системах управления данными используются два основных приёма (практически всегда применяемых совместно):

Разделение данных (шардинг, sharding) – подход, при котором каждый узел системы содержит свою часть данных и выполняет операции над ними. Этот метод является основным средством обеспечения горизонтальной масштабируемости, однако теперь операции над несколькими объектами могут вовлечь в работу несколько узлов, что требует активной передачи данных по сети. Кроме того, при увеличении числа узлов, хранящих данные, увеличивается вероятность сбоев, что требует наличия избыточности – применения репликации (см. далее). Шардинг также порождает такие задачи, как распределение данных по узлам, балансировка нагрузки, оптимизация сетевых взаимодействий и т.д.

Репликация (replication) – подход, при котором одни и те же данные хранятся на нескольких узлах в сети. Этот метод помогает повысить надежность системы и справляться как со сбоями отдельных узлов, так и с потерей целого кластера. Кроме того, репликация позволяет масштабировать операции чтения (несколько реже - записи). Серьезной задачей является поддержка согласованного состояния копий данных (реплик): при синхронном обновлении реплик увеличивается время ответа системы, а при асинхронном - возникает промежуток времени, когда реплики находятся в несогласованном состоянии.

Существуют две основные схемы репликации (поддерживающие как синхронные, так и асинхронные варианты):

Ведущий-ведомый (master-slave) – при такой схеме репликации операции модификации данных обрабатывает только ведущий узел (master), а сделанные изменения синхронно или асинхронно передаются на ведомого (slave). Чтения могут осуществляться как с ведущего узла (гарантированно содержит последнюю версию данных), так и с ведомого (данные могут быть несколько устаревшими при асинхронной репликации и «отставать» от ведущего).

Ведущий-ведущий (master-master, multi-master) – при этой схеме все узлы могут обрабатывать операции записи и передавать обновления остальным. В этом случае реализовать синхронную репликацию достаточно сложно, к тому же резко возрастают задержки, связанные с сетевыми взаимодействиями. При асинхронном обновлении возникает другая проблема – могут появиться конфликтующие версии данных, которые требуют наличия

механизма для определения и разрешения конфликтов (автоматически или на уровне приложения).

2.1. Модели согласованности

Применение репликации в распределённой системе порождает задачу поддержания идентичного состояния копий данных на разных узлах (и, соответственно, видимых разными клиентами). Для обозначения гарантий согласованности данных, которые предоставляются системой, используют термин «модель согласованности» (consistency model). Следует отметить, что слово «согласованность» в этом контексте отличается от свойства согласованности из определения ACID-транзакций. Модель согласованности определяет физическую согласованность состояния данных на разных узлах системы, в то время как в случае ACID имеется в виду логическая согласованность (в рамках ограничений целостности, определённых для базы данных). Ниже приводятся примеры моделей, часто применяемых в NoSQL-системах (см., например, [7]):

Согласованность в «конечном счёте» (eventual consistency) гарантирует, что при отсутствии новых обновлений в течение некоторого времени все копии данных (реплики) станут согласованными.

Монотонные чтения (monotonic reads) являются усилением согласованности «в конечном счёте», гарантируя, что если какое-либо значение было прочитано клиентом, то последующие чтения никогда не вернут предыдущие значения.

Чтение своих записей (read your writes) также усиливает согласованность «в конечном счёте», давая гарантии того, что клиент всегда увидит данные, которые он до этого записал. Эта модель может также комбинироваться с монотонными чтениями.

Мгновенная согласованность (immediate consistency) означает, что как только операция модификации данных успешно завершена, все клиенты мгновенно увидят это изменение. Такую модель согласованности поддерживают, например, системы с синхронной репликацией.

Наиболее часто NoSQL-системы поддерживают согласованность «в конечном счёте» или её вариации. При этом возникает период времени, в течение которого данные находятся в несогласованном состоянии, называемый «окном несогласованности» (inconsistency window). Его величина зависит (при отсутствии сбоев) от скорости распространения обновлений, загруженности системы и количества узлов, содержащих реплики данных. Приложения, использующие системы управления данными, допускающие несогласованность, должны уметь справляться с появлением несколько «устаревших» данных (в [8] проверяется, насколько часто это может происходить на практике при использовании различных облачных NoSQL-систем). Кроме того, при одновременных обновлениях реплик данных возникают конфликтующие версии. Для обнаружения конфликтов применяются такие подходы, как временные метки и векторные часы (см., например, [9]). Для разрешения конфликтов существуют различные стратегии, но обычно наиболее точное решение о том, какую версию выбрать, может сделать только само приложение (о различных стратегиях разрешения конфликтов см., например, [10]).

В некоторых системах уровень поддержки согласованности можно варьировать, используя различные конфигурации, настройки и/или операции. Рассмотрим, как этого можно добиться на практике с помощью кворума. Пусть данные реплицируются по N-узлам; обозначим через R – число узлов, к которым обращается клиент при чтении данных, а через W – число узлов, от которых ожидается подтверждение успешной записи. Изменяя эти параметры, можно добиться различного поведения распределённой системы:

При R + W > N множества реплик для записи и чтения пересекаются, а значит, чтение всегда возвратит результат успешно завершившейся операции записи (мгновенная согласованность).

При R + W ≤ N невозможно гарантировать возврат последней версии данных, гарантируется лишь согласованность «в конечном счёте».

При R = N, W = 1 поддерживается мгновенная согласованность, а операции записи работают быстро за счёт более медленных и дорогих чтений (с N реплик). При W = N, R = 1 получается обратная ситуация.

Следует отметить, что чем ближе значения R и W к общему числу реплик N, тем больше вероятность, что операция может завершиться ошибкой из-за выхода каких-либо узлов из строя.

Вариации согласованности «в конечном счёте» (например, наличие монотонных чтений и чтений своих записей) во многом зависят от реализации взаимодействия клиентов с системой (привязывается ли клиент к конкретному серверу, используется ли узел-маршрутизатор, поддерживаются ли версии объектов и т.д.). Более подробно о нюансах моделей согласованности в NoSQL-системах читатель может узнать, например, из [7] и [11].

2.2. Теорема CAP

Для обоснования компромиссов, выбираемых NoSQL-системами, часто приводится эмпирическое утверждение, известное как теорема CAP, или теорема Брюэра (Eric Brewer) [12]. Это утверждение гласит, что распределенная система не может гарантировать одновременного выполнения следующих трёх свойств:

1) *Согласованность (Consistency)* - все узлы в каждый момент времени имеют согласованные данные (все пользователи в любой момент времени видят одинаковые данные).

2) *Доступность (Availability)* - при выходе из строя каких-либо узлов оставшиеся узлы должны продолжать функционировать.

3) *Устойчивость к разделению (Partition Tolerance)* - если из-за сбоя сети система распадается на группы узлов, не связанные между собой, то каждая группа должна продолжать функционировать.

Несмотря на то, что данное утверждение само по себе недостаточно формализовано, оно было уточнено и доказано для некоторых частных случаев [13]. Однако желание обеспечить устойчивость к разделению сети и высокую доступность системы – не единственные причины для ослабления согласованности данных. Как отмечает Дэниэл Абади (Daniel Abadi) в [14], формулировка теоремы CAP не учитывает такого важного свойства системы, как величина задержки при ответе на запрос пользователя. Усиление модели согласованности ведёт в общем случае к более высоким задержкам (достаточно вспомнить, например, случай синхронной репликации, когда обновления должны распространиться по большинству узлов или даже по всем узлам для того, чтобы операция изменения данных считалась завершенной). Особенно важно минимизировать передачу данных по сети между узлами в случае большой географической распределённости кластеров.

2.3. Модели данных и классификация

Еще одним важным отличием систем категории NoSQL от реляционных баз данных являются их нереляционные модели данных и способы осуществления запросов. В целом

модели данных, лежащие в основе NoSQL, значительно проще, чем классическая реляционная модель, что в ряде случаев облегчает работу с ними. Обычно (хотя и с некоторыми вариациями) NoSQL-системы делят (см., например, [15], [16], [17]), исходя из модели данных, на следующие основные классы:

1) Системы «ключ-значение» (Key-Value Stores)

2) Документные СУБД (Document Stores)

3) Системы типа Google BigTable (Extensible Record Stores / Wide Column Stores / Column Families)

Иногда под термином «NoSQL» понимают также вообще все системы, не являющиеся реляционными (SQL-ориентированными), однако чаще всего имеются в виду именно представители приведённых трёх классов систем. Это деление является достаточно общим, так как внутри каждой группы системы значительно различаются и в плане поддержки согласованности данных, и в нюансах работы с ними (например, атомарность операций, использование блокировок или мультиверсионного доступа (MVCC, Multi-Version Concurrency Control) [18]), однако оно отражает основные аспекты и область применения соответствующих NoSQL-систем. Кроме того, многие системы имеют черты более чем одного класса, и иногда их трудно классифицировать по такому принципу. Далее будет дана общая характеристика каждого из классов и рассмотрены примеры некоторых конкретных систем, которые могут быть к ним отнесены.

3. Системы «ключ-значение»

NoSQL-системы типа «ключ-значение» хранят данные (неструктурированные или структурированные) и позволяют иметь доступ к ним при помощи единственного уникального ключа. Работа с данными обычно осуществляется с помощью простых операций вставки, удаления и поиска по ключу. Вторичные ключи и индексы в таких системах не поддерживаются. При этом может поддерживаться некоторая структура данных, позволяющая менять отдельные поля объекта, но не позволяющая строить по ним запросы (в этом заключается основное отличие систем типа «ключ-значение» от документных СУБД [15]). В этом отношении системы «ключ-значение» похожи на популярную распределённую систему кэширования в оперативной памяти Memcached [19], но предоставляют постоянное хранение данных и ряд дополнительных возможностей. Существуют NoSQL-системы, обладающие обратной совместимостью с Memcached, что позволяет использовать тот же интерфейс и те же клиентские библиотеки, облегчая переход с Memcached (например, MemcacheDB [20], Couchbase Server [21] и др.). Далее будут подробнее рассмотрены некоторые системы класса «ключ-значение».

3.1. Project Voldemort

Project Voldemort [22] – система управления данными типа «ключ-значение» с открытым исходным кодом, реализованная на Java и активно используемая в социальной сети LinkedIn [23]. Помимо скалярных значений, Project Voldemort допускает также списки значений и записи с именованными полями, ассоциируемые с одним ключом. Самим значением и ключом может являться любая сущность, для которой определена сериализация (правила преобразования объекта в последовательность байтов и обратно). Вся работа с данными осуществляется с использованием трёх операций: put, get и delete. Project Voldemort использует механизм MVCC при модификации данных.

Project Voldemort поддерживает шардинг и репликацию, а также несколько способов физической организации системы. Для распределения ключей по логическому кольцу узлов

используется метод консистентного хэширования (consistent hashing, см., например, [24]). Шардинг осуществляется прозрачно для приложения, узлы могут быть добавлены или удалены в процессе работы, восстановление при сбоях также происходит автоматически. Project Voldemort использует асинхронную репликацию и поддерживает согласованность «в конечном счёте», конфликты разрешаются при чтении (read-repair). Также используется механизм «hinted handoff», который позволяет сохранить данные даже при выходе хранящих их узлов из строя, используя соседние узлы. Project Voldemort может хранить данные в оперативной памяти и обеспечивать постоянство хранения с помощью одного из поддерживаемых механизмов, например, Berkeley DB [25]. Более подробно архитектура Project Voldemort описывается в [26].

3.2. DynamoDB

DynamoDB [27] – облачный сервис, представленный компанией Amazon в начале 2012 года [28]. Эта система является последователем таких технологий Amazon, как Dynamo [29] и SimpleDB. DynamoDB предоставляет пользователям быструю и масштабируемую систему управления данными, репликация и шардинг в которой осуществляются автоматически. Высокая производительность системы достигается за счёт использования твердотельных накопителей (SSD) и ряда других оптимизаций. Использование облачного сервиса снимает с пользователей необходимость в установке и администрировании серверов, позволяя оплачивать лишь потребляемые ресурсы, такие как трафик и объем хранимой информации.

Модель данных DynamoDB является достаточно гибкой и богатой для систем типа «ключ-значение». Данные хранятся в так называемых таблицах, обладающих первичным ключом (простым или составным) и набором атрибутов (заранее зафиксированной схемы нет, поддерживаются скалярные типы данных и множества). Для работы с данными используются операции поиска, вставки и удаления по первичному ключу, условные операции (например, обновить, если выполнено условие), атомарные модификации (например, увеличение значения атрибута на единицу) и поиск по неключевым атрибутам путём полного сканирования таблицы. Последняя операция делает эту систему еще ближе к документным СУБД, однако эффективного способа запрашивать данные по неключевым атрибутам DynamoDB не имеет. Желаемый уровень согласованности может быть указан при чтении данных («eventually consistent reads» или «strongly consistent reads»). Полностью согласованные чтения гарантированно возвращают последнюю версию данных, однако это сказывается на производительности. Клиентские библиотеки для работы с DynamoDB доступны для большого числа языков программирования, включая Java, .NET, PHP, Perl, Python, Ruby и др.

3.3. Redis

Redis [30] – система управления данными типа «ключ-значение» с открытым исходным кодом, написанная на C и также поддерживающая достаточно богатую для таких систем модель данных. Значения могут содержать не только строки, но и множества, списки и другие структуры данных. Помимо обычных операций получения, сохранения и удаления данных по ключу, Redis поддерживает атомарные операции, такие как увеличение числа на единицу, добавление элемента в список и т. д. Кроме того, поддерживаются транзакции, содержащие группы операций и обладающие свойствами изолированности и атомарности, а также оптимистические блокировки.

Redis работает в оперативной памяти, за счёт чего достигается высокая производительность. Для обеспечения долговременного хранения могут применяться снимки данных в

определённые моменты времени или постоянная запись операций модификации данных на диск; также возможна работа вообще без использования дисков.

Горизонтальная масштабируемость может быть достигнута с помощью разделения данных (логика реализуется на стороне клиента). Также поддерживается асинхронная репликация со схемой «ведущий-ведомый».

Клиентские библиотеки для работы с Redis доступны для большинства языков программирования, а сама эта система используется в таких крупных проектах, как Twitter, Instagram, Digg, Github, StackOverflow, Flickr и других [31].

3.4. Riak

Riak [32] – мощная система управления данными типа «ключ-значение» с открытым исходным кодом, написанная на Erlang. На архитектуру Riak оказала значительное влияние система Amazon Dynamo [29].

Организация кластера Riak похожа на Project Voldemort: также используется консистентное хэширование и hinted handoff. Такая архитектура обеспечивает надежность, децентрализацию и легкое добавление новых физических узлов. На каждом физическом узле (node) может работать несколько виртуальных узлов (vnode), что помогает при балансировке нагрузки; репликация выполняется асинхронно, количество реплик может быть гибко настроено (N). Riak позволяет указывать число реплик для чтения (R) и записи (W) при соответствующих операциях и, таким образом, варьировать уровень согласованности, однако атомарных операций не предусмотрено. Riak использует вариант MVCC для реализации параллельного доступа. Конфликты обновлений могут разрешаться либо по принципу «последний выигрывает», либо на уровне приложения (в этом случае приложению возвращаются конфликтующие версии объектов). Поддерживаются триггеры (называемые «commit hooks»), позволяющие запускать функции на JavaScript или Erlang перед или после модификации объекта.

Ключи в Riak организуются в корзины (bucket»), что позволяет задавать такие параметры, как число реплик, список триггеров и метод разрешения конфликтов, на уровне корзины. Таким образом, объект однозначно идентифицируется парой (корзина, ключ). Объекты в Riak представляются в виде JSON [33] и могут иметь несколько полей данных. Кроме того, объект содержит метаданные, в том числе поддерживаются ссылки на другие объекты. Riak поддерживает вторичные индексы (возможность приписать объекту пару атрибут-значение для последующих запросов по ним) и MapReduce [34] для более сложных запросов. Функциональность Riak делает эту систему близкой к документным СУБД, однако в Riak не поддерживаются запросы к полям объектов (единственный способ – MapReduce), поэтому обычно эту систему относят к классу систем «ключ-значение».

Подсистема хранения данных в Riak является подключаемым модулем, что позволяет использовать различные реализации, подходящие для разных задач. Взаимодействовать с Riak пользователь может с помощью REST-интерфейса или с помощью программного интерфейса, доступного для большинства языков программирования. Riak используется в достаточно большом количестве проектов [35].

3.5. Aerospike

Aerospike (ранее эта система называлась Citrusleaf) [36] интересна тем, что, являясь NoSQL-системой типа «ключ-значение», поддерживает оптимистические блокировки, атомарные операции, синхронную репликацию и мгновенную согласованность. При сбоях кластер Aerospike может работать в режиме устойчивости к разделению («partition tolerant mode»,

система продолжает работу в разделенном состоянии) или в режиме согласованности («high consistency mode», часть кластера может быть отключена, чтобы не допустить несогласованности данных) [37].

Система Aerospike оптимизирована для работы в оперативной памяти и на твердотельных носителях (Flash, SSD), при этом ключи всегда находятся в оперативной памяти. Для оптимизации сетевых взаимодействий применяются клиентские библиотеки, использующие сведения о состоянии кластера. Модель данных в Aerospike на самом верхнем уровне содержит пространства имён (namespaces), внутри которых содержатся множества (sets), которые хранят записи (records), обладающие уникальным ключом и типизированными атрибутами (bins). Подробно архитектура Aerospike описывается в [38].

3.6. Резюме

На момент написания данной работы список NoSQL-систем класса «ключ-значение» в [3] насчитывал более 30 систем, включая такие системы, как Scalaris, Tokyo Cabinet, GenieDB, LevelDB и др. Все эти системы обладают своими особенностями и нюансами и подходят для различных задач. При выборе конкретной системы приходится принимать во внимание множество факторов, таких как желаемый уровень согласованности данных, наличие атомарных операций, легкость масштабирования и администрирования, надежность, наличие клиентских библиотек для используемого языка программирования и т.д. В целом, достоинствами систем типа «ключ-значение» являются хорошая горизонтальная масштабируемость, простота и производительность, однако часто их моделей данных оказывается недостаточно для построения серьезных приложений, где возникает потребность, например, в поиске по сочетанию атрибутов, поддержка вложенных объектов, индексы на полях объектов и т.д. В этом случае нужно обратить внимание на системы с более сложной моделью данных, например, документные СУБД.

4. Документные СУБД

Документные СУБД предоставляют больше возможностей, чем системы типа «ключ-значение». Единицей хранения данных в таких системах является документ – некоторый объект, обладающий произвольным набором атрибутов (полей), который может быть представлен, например, в JSON [33]. Документные системы поддерживают поиск по полям документов, индексы, часто допускаются вложенные документы и массивы, а заранее предопределённой схемы данных, как правило, нет. В отличие от систем типа «ключ-значение», документные СУБД позволяют запрашивать коллекции документов на основании нескольких ограничений на атрибуты, могут осуществлять агрегатные запросы, сортировку результатов, поддерживают индексы на полях документов и т.д. Документные системы обычно не поддерживают семантику ACID, однако между собой они значительно различаются в плане поддержки согласованности данных, наличии атомарных операций, а также в способах контроля параллельного доступа к документам и во многих других аспектах.

4.1. MongoDB

MongoDB [39] – документная СУБД с открытым исходным кодом, написанная на C++ и разрабатываемая компанией 10gen. MongoDB обладает достаточно богатой функциональностью и является одной из самых популярных NoSQL-систем на данный момент [40].

MongoDB позволяет оперировать JSON-документами (хранимыми и передаваемыми в виде BSON – более компактного двоичного представления JSON), объединяемыми в коллекции, которые, в свою очередь, объединяются в базы данных. Каждый документ в коллекции должен содержать уникальный идентификатор (сгенерированный автоматически или пользователем), который не может изменяться после создания документа. Кроме идентификатора, документ также может содержать произвольный набор полей, которые могут содержать массивы и вложенные документы. Заранее предопределенной схемы данных нет: документы в одной коллекции могут содержать разные наборы полей.

Для работы с документами предусмотрены операции поиска, вставки, удаления и обновления документов. Для поиска документов в коллекции используется метод запросов по образцу, поддерживаются сортировка, проекция, просмотр результатов запроса с помощью курсора. Кроме того, поддерживается MapReduce, а также Aggregation Framework – способ формирования запроса из последовательных шагов, таких как агрегация, проекция, сортировка и т.д., что позволяет выполнять достаточно сложные аналитические запросы. Обновление документа может производиться либо полной заменой документа (с сохранением идентификатора), либо изменением полей существующего документа (в том числе добавление элемента в массив, увеличение числа и т.д.); при этом операция модификации одного документа всегда является атомарной (если обновление затрагивает несколько документов, то это уже не так). Кроме того, поддерживается атомарная операция «findAndModify», которая находит и изменяет документ, возвращая старую или новую версию. MongoDB использует блокировки для синхронизации параллельного доступа в пределах одного узла.

Для ускорения поиска документов поддерживается создание индексов на одном или нескольких полях документов в коллекции (реализованные с помощью B-деревьев), а также двумерные пространственные индексы. Существует возможность «подсказать» оптимизатору запросов, какой индекс использовать (hint), и проанализировать план выполнения (explain).

Масштабируемость в MongoDB достигается за счёт разделения документов из коллекции по узлам на основании выбранного ключа (shard key). Поддерживается асинхронная репликация в режиме «главный-подчиненный»: операции записи обрабатываются только главным узлом, а чтения могут осуществляться как с главного узла, так и с одного из подчиненных. Клиент может работать в разных режимах: неблокирующем (не дожидаясь подтверждения) или блокирующем (ожидая подтверждения от заданного количества узлов). Таким образом, MongoDB поддерживает различные модели согласованности в зависимости от того, разрешены ли чтения с вторичных узлов и от скольких узлов ожидаются подтверждения при записи. MongoDB может быть использована также и в качестве распределённой файловой системы благодаря функционалу GridFS.

Клиентские библиотеки для работы с MongoDB доступны для большого числа языков программирования, кроме того, поддерживается REST-интерфейс. Эта система используется в большом числе крупных компаний и проектов, среди которых SourceForge, Foursquare, The Guardian, Forbes, The New York Times и другие [41].

4.2. CouchDB

CouchDB [42] – проект Apache Software Foundation с открытым исходным кодом, реализованный на Erlang. CouchDB является распределённой документной СУБД, также оперирующей JSON-документами.

Заранее предопределённой схемы данных в CouchDB также не предусмотрено – документы могут содержать различные наборы полей (поддерживаются скалярные поля, массивы,

вложенные документы и т.д.) и имеют уникальный идентификатор, а также номер ревизии (revision); документы организуются в базы данных. Отчёты и запросы к базам данных строятся с использованием MapReduce-представлений (views) – специальных функций на JavaScript, позволяющих задавать вид возвращаемых данных, а также выполнять агрегацию; эти функции помещаются в специальные документы (design documents). Запросы к представлениям позволяют задавать ограничения на возвращаемые данные, осуществлять сортировку, ограничивать количество возвращаемых результатов и т.д. Для представлений строятся индексы на основе B-деревьев, обновляемые при модификациях данных. Операции модификации на уровне документа обладают свойствами ACID, а читатели никогда не блокируются благодаря использованию MVCC; CouchDB поддерживает оптимистические блокировки при работе с документами. После каждой операции, изменяющей данные, происходит немедленный сброс данных на диск. Данные дописываются в конец файла, старые ревизии документов сохраняются, поэтому требуется периодически проводить сжатие (compaction) базы данных (в процессе сжатия система остается доступной для чтения и записи). Также поддерживаются JavaScript-функции для валидации данных и проверки прав доступа к ним при обновлениях.

CouchDB может масштабироваться только за счет репликации, выполняемой асинхронно. Поддерживается как схема репликации «ведущий-ведомый», так и «ведущий-ведущий». Кроме того, существует механизм условной (filtered) репликации, когда реплицируются только определённые документы. Каждый клиент видит согласованное состояние базы данных, однако эти состояния могут различаться для разных клиентов (усиленный вариант согласованности «в конечном счёте»). Когда один и тот же документ изменяется на разных узлах, возникает конфликт. При обнаружении конфликта одна из версий документа автоматически становится «победителем», а «проигравшая» версия сохраняется и может быть использована для разрешения конфликта. Существует также проект CouchDB Lounge [43], предоставляющий возможности шардинга для CouchDB.

Работа с CouchDB осуществляется через REST-интерфейс, клиентские библиотеки доступны для большого числа различных языков, в том числе Java, .NET, Python, PHP, Ruby и др. CouchDB используется в достаточно большом количестве проектов [44].

4.3. Couchbase Server

Couchbase Server [21] – проект, являющийся слиянием проектов Membase (система типа «ключ-значение», совместимая с Memcached) и системы CouchDB, рассмотренной ранее.

Couchbase Server может быть использован как в качестве системы управления данными типа «ключ-значение», совместимой с протоколом Memcached, так и в качестве документной СУБД, работающей с JSON-документами через REST-интерфейс. Документы могут содержать произвольный набор полей, имеют уникальный идентификатор и хранятся в «корзинах» (data bucket). Запросы осуществляют с помощью MapReduce-представлений (views) на JavaScript аналогично CouchDB. Представления строятся инкрементально и асинхронно, поэтому по умолчанию моделью согласованности является согласованность «в конечном счёте», однако на уровне операции можно указать, чтобы данные индексировались сразу же. Подсистема хранения данных также функционирует аналогично CouchDB – данные записываются в конец файла, требуется периодическое сжатие базы данных.

Важной особенностью Couchbase Server по сравнению с CouchDB является поддержка автоматического и прозрачного для приложения шардинга. Кроме того, Couchbase поддерживает два различных вида репликации – внутри кластера (intra-cluster) и межкластерную (inter-cluster, XDCR – Cross Datacenter Replication). Первый вид репликации осуществляется в пределах кластера, где узлы содержат как свои собственные данные, так и

реплики других узлов, и поддерживает мгновенную согласованность на уровне документа – репликация в стиле Membase. Второй вид репликации предназначен для географически распределённых кластеров, соединённых с помощью WAN, и выполняется асинхронно, обеспечивая согласованность в «конечном счёте» между кластерами; разрешение конфликтов в этом случае осуществляется аналогично CouchDB – в кластерах выбирается один и тот же «победитель».

Couchbase Server является новой и активно развивающейся системой с богатыми возможностями. Наиболее актуальную и подробную документацию по Couchbase Server читатель может найти в [45].

4.4. Резюме

Документные СУБД имеют гибкую модель данных, которая в ряде случаев является более удобной, чем фиксированная схема, и лучше сочетается с объектно-ориентированным программированием, сокращая прослойку между языком программирования и СУБД. Системы этого класса могут достаточно легко масштабироваться, хотя делают это немного по-разному. Способы построения запросов также различаются, но в целом поддерживаются достаточно сложные выборки, включающие ограничения на значения полей, агрегацию, сортировку и т.д. К вопросу согласованности данных документные системы также подходят по-разному, обычно позволяя в определенной степени варьировать их в зависимости от конфигурации и потребностей приложения. Кроме того, могут быть реализованы дополнительные механизмы, например, оптимистические блокировки, атомарные операции и т.д., что позволяет усилить гарантии согласованности данных для тех приложений, где это необходимо. Документные СУБД обычно поддерживают постоянство хранения данных, используя запись на жесткие диски или SSD-накопители, а надежность обеспечивается с помощью журналирования и репликации; индексы и часто используемые документы при этом обычно хранятся в оперативной памяти для быстрого доступа. Транзакционная семантика на уровне нескольких документов в таких системах обычно не поддерживается, единственной возможностью является реализация на уровне приложения. Тем не менее документные СУБД по функциональности постепенно приближаются к традиционным SQL-ориентированным СУБД.

На момент написания работы список документных СУБД в [3] насчитывал порядка 20 систем, включая такие системы, как Terrastore, RethinkDB, RavenDB и др.

5. Системы типа Google BigTable

Разработка Google BigTable [46] была начата в 2004 году для поддержки различных сервисов Google, таких как Google Earth, Google Maps, Google Analytic и др. BigTable базируется на Google File System (GFS, используется для хранения данных и журнала), Chubby (используется для координации и хранения некоторых метаданных) и других разработках компании и не распространяется за пределами Google, но возможность её использования предоставляется в рамках Google App Engine. BigTable проектировалась таким образом, чтобы легко масштабироваться на сотни и тысячи узлов и работать с петабайтами данных.

Таблица BigTable представляет собой отображение ключа ряда (row key), ключа столбца (column key) и временной метки (timestamp) в значение в виде строки. Ключ ряда и ключ столбца также являются обычными строками. Ключи рядов упорядочены в лексикографическом порядке, а столбцы объединены в семейства столбцов (column family), которые должны быть определены до использования, после чего в каждое семейство столбцы могут быть добавлены динамически. Семейства столбцов обычно хранят однотипные данные, и их число невелико (не более сотни), в то время как столбцов в

семействе может быть неограниченное количество. Каждая ячейка таблицы может содержать несколько версий данных, помеченных временными метками и упорядоченными по ним, текущее значение имеет наибольшую временную метку, поддерживается автоматическое удаление старых версий; ячейки также могут вообще не содержать данных. Таким образом, каждая строка таблицы (ряд) может содержать произвольное число атрибутов (столбцов), входящих в заранее определённые семейства.

Ряды таблицы разделяются по диапазонам ключей, формируя относительно небольшие по размеру сегменты («tablets» в терминологии BigTable), являющиеся единицами распределения при балансировке нагрузки. Кластер BigTable содержит один главный сервер (master) и сервера, непосредственно хранящие сегменты. Главный сервер отвечает за распределение сегментов по узлам, балансировку нагрузки, операции со схемой и т.д. Семейства столбцов являются единицами контроля прав доступа и параметров хранения. Данные хранятся по столбцам, а семейства столбцов, доступ к которым обычно осуществляется вместе, могут быть выделены в группы локальности (locality groups), что позволяет оптимизировать чтения. На уровне группы можно указать, например, чтобы данные её семейств столбцов постоянно находились в оперативной памяти и читались из неё, а также настроить сжатие данных. BigTable поддерживает асинхронную репликацию между кластерами, гарантируя при этом согласованность «в конечном счёте».

В BigTable предусмотрены операции для создания и удаления таблиц и семейств столбцов, изменения метаданных (например, прав доступа), записи и удаления значений, чтения определенных строк, просмотра подмножеств данных (например, столбцов из определённого семейства). Также поддерживаются атомарные операции над строками таблицы и исполнение скриптов для обработки данных. Клиентские библиотеки кэшируют метаданные о расположении сегментов и большую часть времени обращаются непосредственно к узлам, хранящим данные. Также BigTable может быть использована с MapReduce.

Подробности реализации BigTable читатель может найти в [46]. Успех BigTable положил начало новому семейству систем, применяющих схожие подходы для обеспечения масштабируемости и высокой производительности.

5.1. HBase

HBase [47] – проект с открытым исходным кодом на Java, разрабатываемый Apache Software Foundation. HBase следует принципам BigTable и использует Apache Hadoop [48] – набор библиотек и инструментов для разработки и выполнения распределённых вычислений.

Вместо GFS в HBase используется HDFS (Hadoop Distributed File System) – распределённая файловая система, являющаяся частью Apache Hadoop и предназначенная для надежного хранения больших файлов, распределённых по блокам между узлами. Кроме того, могут быть использованы и другие файловые системы. Также HBase поддерживает работу с Hadoop MapReduce. Роль сервиса Chubby в HBase выполняет Apache ZooKeeper.

Архитектура и функциональность HBase во многом соответствует BigTable (описанной в [46]), хотя имеются и некоторые отличия. Например, HBase поддерживает несколько главных (master) серверов, чтобы повысить надежность системы. В HBase не поддерживается концепция групп локальности (locality groups), вся конфигурация выполняется на уровне семейств столбцов. Как и BigTable, HBase не поддерживает семантику ACID в полном объеме, однако определённые свойства, усиливающие гарантии согласованности, обеспечиваются (см. [49]). HBase не поддерживает вторичные индексы: записи могут быть запрошены только с помощью первичного ключа или сканирования таблицы. Индексы, тем не менее, могут быть построены вручную с помощью дополнительных таблиц.

Работа с HBase может осуществляться через Java API, REST-интерфейс, а также с помощью Avro и Thrift. HBase используется в крупных и высоконагруженных приложениях и проектах, таких как Facebook (сервис Facebook Messages) и Twitter (для поддержки MapReduce, поиска по людям и других задач).

5.2. Cassandra

Система Cassandra [50] была разработана и использовалась в Facebook. В её основе лежат идеи Google BigTable [46] и Amazon Dynamo [29]. В настоящее время Cassandra является проектом с открытым исходным кодом (на Java), поддерживаемым Apache Software Foundation.

По организации модели данных Cassandra схожа с BigTable и HBase, однако терминология и детали несколько отличаются. База данных в Cassandra называется «пространством ключей» (keyspace) и содержит семейства столбцов (column family), которые являются аналогом таблиц и служат контейнерами для строк (рядов, rows), идентифицируемых уникальными ключами (row key). Строки состоят из столбцов (column) или супер-столбцов (super column). Столбец является минимальной единицей данных в Cassandra и состоит из имени, значения и временной метки (все эти поля предоставляются клиентом), хранится только последняя версия данных (в противоположность BigTable и HBase). Супер-столбцы, в свою очередь, содержат внутри себя столбцы, добавляя тем самым еще один уровень вложенности. Кроме того, поддерживаются специальные столбцы, такие как счётчики или столбцы с указанным временем жизни (TTL). Разным строкам необязательно должен соответствовать один и тот же набор столбцов или супер-столбцов. Семейства столбцов хранятся в отдельных файлах с сортировкой по ключам строк и должны содержать столбцы, доступ к которым в запросах предполагается осуществлять вместе.

Для работы с данными Cassandra поддерживает SQL-подобный язык CQL (Cassandra Query Language), кроме того, есть поддержка Hadoop MapReduce. Для ускорения запросов поддерживается создание вторичных индексов. Операции модификации данных являются атомарными на уровне одной строки таблицы, постоянство хранения обеспечивается с помощью записи в журнал, поддерживается сжатие данных. Cassandra позволяет гибко варьировать уровень согласованности данных на уровне операций. Конфликты разрешаются на основании временных меток (выигрывает последняя версия).

Cassandra проектировалась так, чтобы обеспечить хорошую масштабируемость и надежность на большом количестве недорогих (и ненадежным) машин. В отличие от BigTable и HBase, в кластере Cassandra нет выделенных узлов, все они равноправны и выполняют одни и те же функции. Для распределения данных по узлам применяется консистентное хэширование и hinted handoff, новые узлы могут быть легко добавлены в кластер, а обнаружение сбоев и восстановление происходят автоматически. Разделение строк по узлам может осуществляться как случайным образом, так и с сохранением порядка. Репликация поддерживается как в пределах кластера, так и между географически распределёнными кластерами.

Клиентские библиотеки для работы с Cassandra доступны для большинства языков программирования (основаны на Thrift). Эта система используется во многих проектах с высокой нагрузкой [51].

5.3. Резюме

Рассмотренные системы во многом следуют архитектуре и подходам, применённым в BigTable. Эти системы созданы для поддержки высоконагруженных приложений и работы на

больших кластерах недорогих машин, что достигается за счёт несколько более сложной модели данных, чем документная модель: требуется внимательное проектирование семейств столбцов и выбор ключей строк, а реализация некоторых функций (например, вторичные индексы в HBase) перекладывается на разработчика.

6. Другие системы

Как было отмечено, иногда под термином «NoSQL» понимают также вообще все системы управления данными, не являющиеся реляционными (SQL-ориентированными). Многие из этих систем появились еще до зарождения и популяризации NoSQL-движения, а также часто поддерживают ACID-транзакции и не всегда являются распределёнными, что нетипично для новых систем. Тем не менее перечислим некоторые классы систем, не вошедшие в данный обзор, но иногда относимые к NoSQL: объектно-ориентированные СУБД, графовые системы, XML-ориентированные СУБД, многомерные системы и др.

7. Заключение

В данной работе была дана общая характеристика достаточно нового направления в области управления данными, получившего название «NoSQL». Мотивацией к созданию этих систем послужило активное развитие Web, что привело к появлению приложений с гигантской нагрузкой и огромными объемами данных. Таким образом, NoSQL-системы фокусируются в основном на том, чтобы обеспечить требуемую масштабируемость (и отказоустойчивость) даже за счёт снижения гарантий согласованности данных и отказа от привычной транзакционной семантики. Модели данных, поддерживаемые NoSQL-системами в целом проще, чем реляционная модель, а жестко определённой схемы данных и ограничений целостности, как правило, нет. При использовании NoSQL-систем разработка приложений часто упрощается за счёт более простых и гибких моделей данных (например, документной) и меньшего «impedance mismatch», то есть несоответствия объектно-ориентированной модели языка программирования и модели данных используемой СУБД (см., например, [52]). К сожалению, в целом NoSQL-системы плохо подходят для задач, где требуется транзакционная семантика.

Большое разнообразие систем класса NoSQL обусловлено общей тенденцией к специализации в области СУБД (см., например, [53]): каждая NoSQL-система приспособлена для решения определённого класса задач. Таким образом, выбор конкретных решений обусловлен спецификой решаемой задачи: предполагаемая нагрузка, соотношение интенсивности чтений и записи, вид хранимых данных и типы запросов к ним, желаемый уровень согласованности, требования к надежности, наличие клиентских библиотек для выбранного языка и т.д. В этом отношении традиционные SQL-ориентированные СУБД претендуют на некоторую универсальность, хотя их масштабируемость ограничена. Кроме того, NoSQL-системы являются достаточно молодыми по сравнению с SQL-ориентированными СУБД, поэтому значительного опыта их применения еще не накоплено. В настоящее время также появляются новые распределённые SQL-ориентированные системы, обладающие лучшей масштабируемостью и сохраняющие поддержку SQL и ACID-транзакций (например, VoltDB [54] и H-Store [55]).

8. Библиография

[1] C. Strozzi, «NoSQL: A Relational Database Management System,» [В Интернете]. URL: http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/nosql/Home%20Page. [Дата обращения: 14 марта 2013].

[2] J. Gray, «The Transaction Concept: Virtues and Limitations,» в *Seventh International Conference on Very Large Databases*, 1981.

[3] «NOSQL Databases,» [В Интернете]. URL: http://nosql-database.org/. [Дата обращения: 14 марта 2013].

[4] A. Wiggins, «SQL Databases Don't Scale,» 2009. [В Интернете]. URL: http://adam.heroku.com/past/2009/7/6/sql_databases_dont_scale/. [Дата обращения: 14 марта 2013].

[5] D. Obasanjo, «Building scalable databases: Denormalization, the NoSQL movement and Digg,» 2009. [В Интернете]. URL: http://www.25hoursaday.com/weblog/2009/09/10/BuildingScalableDatabasesDenormalizationTheNoSQLMovementAndDigg.aspx. [Дата обращения: 14 марта 2013].

[6] B. A. Philip, V. Hadzilacos и N. Goodman, «Distributed Recovery,» в *Concurrency Control and Recovery in Database Systems*, Addison Wesley Publishing Company, 1987, pp. 240-264.

[7] W. Vogels, «Eventually Consistent,» *ACM Queue*, т. 6, № 6, 2008.

[8] H. Wada, A. Fekete, L. Zhaoy, K. Lee и A. Liu, «Data Consistency Properties and the Tradeoffs in Commercial Cloud Storages: the Consumers' Perspective,» в *Conference on Innovative Data Systems Research*, 2011.

[9] R. Baldoni и M. Raynal, «Fundamentals of Distributed Computing - A Practical Tour of Vector Clock Systems,» 2002. [В Интернете]. URL: http://net.pku.edu.cn/~course/cs501/2008/reading/a_tour_vc.html. [Дата обращения: 14 марта 2013].

[10] T. B. Douglas, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer и C. H. Hauser, «Managing update conflicts in Bayou, a weakly connected replicated storage system,» в *SOSP '95 Proceedings of the fifteenth ACM symposium on Operating systems principles*, New York, NY, USA, 1995.

[11] D. Merriman, «On Distributed Consistency,» 26 марта 2010. [В Интернете]. URL: http://blog.mongodb.org/post/475279604/on-distributed-consistency-part-1. [Дата обращения: 14 марта 2013].

[12] E. Brewer, «Towards Robust Distributed Systems,» в *ACM Symposium on the Principles of Distributed Computing*, Portland, Oregon, 2000.

[13] S. Gilbert и N. Linch, «Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services,» *ACM SIGACT News*, т. 33, № 2, pp. 51-59, 2002.

[14] D. Abadi, «Problems with CAP, and Yahoo's little known NoSQL system,» 2010. [В Интернете]. URL: http://dbmsmusings.blogspot.com/2010/04/problems-with-cap-and-yahoos-little.html. [Дата обращения: 14 марта 2013].

[15] R. Cattell, «Scalable SQL and NoSQL Data Stores,» 2011. [В Интернете]. URL: http://www.cattell.net/datastores/Datastores.pdf. [Дата обращения: 14 марта 2013].

[16] A. Lith и J. Mattsson, Investigating storage solutions for large data: A comparison of well performing and scalable data storage, Goteborg, Sweden: Chalmers University Of Technology, Department of Computer Science and Engineering, 2010.

[17] C. Strauch, «NoSQL Databases,» 2011. [В Интернете]. URL: http://www.christof-strauch.de/nosqldbs.pdf. [Дата обращения: 14 марта 2013].

[18] P. A. Bernstein и N. Goodman, «Concurrency Control in Distributed Database Systems,» *ACM Computing Surveys*, т. 13, № 2, pp. 185-221, 1981.

[19] «memcached - a distributed memory object caching system,» [В Интернете]. URL: http://memcached.org/. [Дата обращения: 14 марта 2013].

[20] «memcachedb - A distributed key-value storage system designed for persistent,» [В Интернете]. URL: http://memcachedb.org/. [Дата обращения: 14 марта 2013].

[21] «Couchbase Server,» [В Интернете]. URL: http://www.couchbase.com/. [Дата обращения: 14 марта 2013].

[22] «Project Voldemort,» [В Интернете]. URL: http://www.project-voldemort.com/. [Дата обращения: 14 марта 2013].

[23] J. Kreps, «Project Voldemort: Scaling Simple Storage at LinkedIn,» 20 мая 2009. [В Интернете]. URL: http://blog.linkedin.com/2009/03/20/project-voldemort-scaling-simple-storage-at-linkedin/. [Дата обращения: 14 марта 2013].

[24] D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins и Y. Yerushalmi, «Web Caching with Consistent Hashing,» MIT Laboratory for Computer Science, 1999. [В Интернете]. URL: http://www8.org/w8-papers/2a-webserver/caching/paper2.html. [Дата обращения: 14 марта 2013].

[25] «Oracle Berkeley DB,» Oracle, [В Интернете]. URL: http://www.oracle.com/technetwork/products/berkeleydb/overview/index.html. [Дата обращения: 14 марта 2013].

[26] «Project Voldemort Design,» [В Интернете]. URL: http://www.project-voldemort.com/voldemort/design.html. [Дата обращения: 14 марта 2013].

[27] «Amazon DynamoDB,» Amazon, [В Интернете]. URL: http://aws.amazon.com/dynamodb/. [Дата обращения: 14 марта 2013].

[28] W. Vogels, «Amazon DynamoDB – a Fast and Scalable NoSQL Database Service Designed for Internet Scale Applications,» 18 января 2012. [В Интернете]. URL: http://www.allthingsdistributed.com/2012/01/amazon-dynamodb.html. [Дата обращения: 14 марта 2013].

[29] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall и W. Vogels, «Dynamo: Amazon’s Highly Available Key-value Store,» в *21st ACM Symposium on Operating Systems Principles*, Stevenson, WA, 2007.

[30] «Redis,» [В Интернете]. URL: http://redis.io/. [Дата обращения: 14 марта 2013].

[31] «Who's using Redis?,» [В Интернете]. URL: http://redis.io/topics/whos-using-redis. [Дата обращения: 14 марта 2013].

[32] «Riak,» Basho, [В Интернете]. URL: http://basho.com/riak/. [Дата обращения: 14 марта 2013].

[33] «Introducing JSON,» [В Интернете]. URL: http://json.org/. [Дата обращения: 14 марта 2013].

[34] J. Dean и S. Ghemawat, «MapReduce: Simplified Data Processing on Large Clusters,» в *OSDI'04: Sixth Symposium on Operating System Design and Implementation*, San Francisco, CA, 2004.

[35] «Riak Users,» [В Интернете]. URL: http://basho.com/riak-users/. [Дата обращения: 14 марта 2013].

[36] «NoSQL Database, In-Memory or Flash Optimized and Web Scale - Aerospike,» [В Интернете]. URL: http://www.aerospike.com/. [Дата обращения: 14 марта 2013].

[37] «Aerospike - Acid Compliant Database for Mission-Critical Applications,» [В Интернете]. URL: http://www.aerospike.com/performance/acid-compliance/. [Дата обращения: 14 марта 2013].

[38] V. Srinivasan и B. Bulkowski, «Citrusleaf: A Real-Time NoSQL DB which Preserves ACID,» в *Very Large Databases (VLDB)*, 2010.

[39] «MongoDB,» 10gen, [В Интернете]. URL: http://www.mongodb.org/. [Дата обращения: 14 марта 2013].

[40] «DB-Engines Ranking,» [В Интернете]. URL: http://db-engines.com/en/ranking. [Дата обращения: 14 марта 2013].

[41] «MongoDB Production Deployments,» [В Интернете]. URL: http://www.mongodb.org/about/production-deployments/. [Дата обращения: 14 марта 2013].

[42] «Apache CouchDB,» [В Интернете]. URL: http://couchdb.apache.org/. [Дата обращения: 14 марта 2013].

[43] «CouchDB Lounge,» [В Интернете]. URL: http://tilgovi.github.com/couchdb-lounge/. [Дата обращения: 14 марта 2013].

[44] «CouchDB in the Wild,» [В Интернете]. URL: http://wiki.apache.org/couchdb/CouchDB_in_the_wild. [Дата обращения: 14 марта 2013].

[45] Couchbase, «Learn about Couchbase Server,» [В Интернете]. URL: http://www.couchbase.com/learn. [Дата обращения: 14 марта 2013].

[46] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra и A. Fikes, «Bigtable: A Distributed Storage System for Structured Data,» в *OSDI'06: Seventh Symposium on Operating System Design and Implementation*, Seattle, WA, 2006.

[47] «Apache HBase,» [В Интернете]. URL: http://hbase.apache.org/. [Дата обращения: 14 марта 2013].

[48] «Apache Hadoop,» [В Интернете]. URL: http://hadoop.apache.org/. [Дата обращения: 14 марта 2013].

[49] «Apache HBase ACID Properties,» [В Интернете]. URL: http://hbase.apache.org/acid-semantics.html. [Дата обращения: 14 марта 2013].

[50] «Apache Cassandra,» [В Интернете]. URL: http://cassandra.apache.org/. [Дата обращения: 14 марта 2013].

[51] «Cassandra Users,» [В Интернете]. URL: http://www.datastax.com/cassandrausers. [Дата обращения: 14 марта 2013].

[52] T. Neward, «The Vietnam of Computer Science,» 26 июня 2006. [В Интернете]. URL: http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx. [Дата обращения: 14 марта 2013].

[53] M. Stonebraker и U. Çetintemel, «"One Size Fits All": An Idea Whose Time Has Come and Gone,» в *ICDE '05: Proceedings of the 21st International Conference on Data Engineering*, Washington, 2005.

[54] «VoltDB,» [В Интернете]. URL: http://voltdb.com. [Дата обращения: 14 марта 2013].

[55] «H-Store,» [В Интернете]. URL: http://hstore.cs.brown.edu/. [Дата обращения: 14 марта 2013].

NoSQL Data Management Systems

S. D. Kuznetsov <kuzloc@ispras.ru>, A. V. Poskonin <aposk@yandex.ru>

Faculty of Computational Mathematics and Cybernetics
Lomonosov Moscow State University

Abstract. A number of new data management systems have emerged over the last decade. These systems are now referred to as "NoSQL" systems and abandon relational data model and SQL as well as ACID transactions. Most of NoSQL systems are distributed and can scale horizontally to

accommodate data and throughput growth. Scalability is extremely important for modern Web applications with millions of users generating enormous amounts of data. Unfortunately NoSQL systems typically lack transaction support and thus can't be used for some applications. Each NoSQL system in most cases is designed to solve specific problems and offers a certain trade-off between various features and guarantees. This paper reviews several classes of NoSQL systems and key problems they solve.

Keywords. NoSQL, non-relational data models, scalability, replication, sharding, data consistency.