NOSQL – АЛЬТЕРНАТИВА ДЛЯ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

САРАНЧУК Дориан, ЛАЗАРЕВ Виталий

Universitatea Tehnică a Moldovei

Аннотация: *В статье рассматривается понятие NoSQL. Показаны основные проблемы при работе с БД. Представлено толкование термина NoSQL, а так же перечислены основные концепции NoSQL. Приведены примеры распределенных систем БД на основе NoSQL. В заключении показана взаимосвязь традиционных SQL и NoSQL систем, так же случаи наилучшего использования NoSQL.*

Ключевые слова: *база данных, реляционная модель, NoSQL, распределенные БД.*

1.Введение

Как известно, задача разработки СУБД в ИТ-индустрии изначально не стояла, но были актуальны прикладные задачи, требующие обработки массивов структурированной информации. Постепенно выяснилось, что во многих случаях алгоритмы накопления, поиска и анализа данных одинаковы, а следовательно, их можно сделать универсальными и выделить в виде отдельных инструментальных средств — систем управления базами данных. К прикладному ПО обработки больших объемов информации всегда предъявляются одни и те же требования — скорость работы и стоимость разработки, которые трансформируются в требования к используемой СУБД. Чем гибче и богаче возможности СУБД, тем проще разработчику прикладной системы приспосабливать их для своих нужд, а значит, тем меньше оказывается стоимость создаваемого ПО.

Таким образом, можно сформулировать наиболее общие требования к СУБД — гибкость и скорость работы. Другие часто предъявляемые и не менее важные требования, такие как поддержка широко распространенных языков запросов или возможность параллельной обработки данных, изначально не были ключевыми. Изменение требований к прикладному ПО влечет за собой изменение требований к СУБД, откуда следует, что развитие технологий СУБД определяется потребностями разработчиков прикладного ПО. Именно поэтому всевозможные экспериментальные СУБД часто остаются невостребованными даже в том случае, если превосходят свои аналоги по каким-либо характеристикам.

2.От SQL к NoSQL и обратно

Как известно, задача разработки СУБД в ИТ-индустрии изначально не стояла, но были актуальны прикладные задачи, требующие обработки массивов структурированной информации. Постепенно выяснилось, что во многих случаях алгоритмы накопления, поиска и анализа данных одинаковы, а следовательно, их можно сделать универсальными и выделить в виде отдельных инструментальных средств — систем управления базами данных. К прикладному ПО обработки больших объемов информации всегда предъявляются одни и те же требования — скорость работы и стоимость разработки, которые трансформируются в требования к используемой СУБД. Чем гибче и богаче возможности СУБД, тем проще разработчику прикладной системы приспосабливать их для своих нужд, а значит, тем меньше оказывается стоимость создаваемого ПО.

Таким образом, можно сформулировать наиболее общие требования к СУБД — гибкость и скорость работы. Другие часто предъявляемые и не менее важные требования, такие как поддержка широко распространенных языков запросов или возможность параллельной обработки данных, изначально не были ключевыми. Изменение требований к прикладному ПО влечет за собой изменение требований к СУБД, откуда следует, что развитие технологий СУБД определяется потребностями разработчиков прикладного ПО. Именно поэтому всевозможные экспериментальные СУБД часто остаются невостребованными даже в том случае, если превосходят свои аналоги по каким-либо характеристикам. За многолетнюю историю развития СУБД было выработано несколько моделей представления структурированной информации: иерархическая, сетевая, реляционная, объектно-реляционная и т. д. Наибольшее распространение получила реляционная модель, предложенная Эдгаром Коддом в 1970 году, которая надолго стала стандартом представления

структурированной информации. Причины этого вытекают из сформулированных требований к СУБД. Во-первых, реляционная модель оказалась достаточно простой, с точки зрения прикладного программиста. Во-вторых, она обладает достаточной гибкостью и позволяет представлять информацию из самых разных предметных областей. В-третьих, в рамках этой модели используется мощный и удобный подход к манипулированию данными (реляционная алгебра), впоследствии оформленный в виде языка SQL. В-четвертых, простота и естественность реляционной алгебры позволили создать высокопроизводительные и универсальные алгоритмы выполнения запросов, удовлетворяющие большую часть потребностей разработчиков прикладных систем. Наконец, к сегодняшнему дню создано множество инструментальных средств, ориентированных на обработку реляционных данных, что дает еще одно преимущество реляционных СУБД.

Итак, при разработке прикладного ПО из исходных требований следует выбор модели представления информации, а уже из нее следует выбор конкретной СУБД. Эта схема может показаться абстрактной в силу того, что на текущий момент доминирующей является реляционная модель, а выбор СУБД производится из достаточно ограниченного списка, который заранее продиктован другими факторами, такими как интеграция, простота поддержки, стоимость и т. д.

Реляционная модель предполагает оперирование только атомарными данными и исключает обработку какой-либо неструктурированной информации, а это приводит к тому, что хранение графических, аудио- или видеоданных становится невозможным. Более того, невозможно даже хранение текстовых документов произвольной длины. Поэтому, первым отступлением от классической реляционной модели в сторону удобства разработки прикладного ПО можно считать введение типа BLOB (Binary Large Object — «бинарные данные большого объема»), который сегодня поддерживается большинством современных СУБД и закреплен в стандартах языка SQL. Каждая СУБД, помимо BLOB, может иметь свои собственные «дополнительные» типы данных, которые обычно требуют введения дополнительных поисковых операций (например, полнотекстовый поиск по BLOB). Это еще дальше уводит от классической реляционной модели.

По мере роста объемов хранимой информации и сложности ее внутренних взаимосвязей стали возникать новые проблемы. Увеличение сложности SQL-запросов привело к тому, что СУБД уже далеко не всегда способны выработать оптимальный план выполнения поступившего запроса. На практике это производится двумя способами. Первый — в тексте SQL указываются «подсказки», помогающие оптимизатору запросов выработать наиболее эффективный план. Второй — вместо запроса используется хранимая процедура. Причина хорошей применимости обоих способов кроется в том, что встроенные в СУБД оптимизаторы запросов располагают только статистической информацией о содержимом базы и строят планы выполнения запросов только на ее основе, а разработчик прикладного ПО всегда обладает гораздо большими сведениями.

Оба отступления от базовых принципов реляционной модели (введение неатомарных типов данных и тонкое управление алгоритмами обработки) напрямую следуют из особенностей самой модели, провозглашающей универсальность способов хранения информации и алгоритмов ее обработки. За этой универсальностью скрывается невозможность учета специфики данных, позволяющей существенно оптимизировать работу алгоритмов.

3.Основные концепции NoSQL

Важной вехой для высоконагруженных систем стало развитие Интернета, ряд сервисов которого (DNS-серверы, поисковые машины, социальные сети и т.д.) изначально должны обрабатывать большие массивы информации и отвечать на огромное число запросов. Это требует не только максимального учета любой специфики обрабатываемой информации, но и перехода на распределенные вычисления. Никакой сколь угодно мощный сервер в принципе не способен в одиночку обеспечить нужную производительность. В итоге основными принципами NoSQL стали отказ от реляционной модели для учета специфики обрабатываемых данных, а также хорошая горизонтальная масштабируемость до сотен и тысяч серверов для обеспечения скорости работы. Однако это выявило еще одну проблему, сформулированную в виде теоремы CAP (Consistence, Availability, Partition tolerance — «согласованность, доступность, устойчивость к разделению»): в распределенной вычислительной системе невозможно одновременно выполнить требования по согласованности данных, доступности системы и устойчивости к разделению. Под последним

требованием понимается то, что система не распадается на несколько изолированных секций, внутри которых выполняются требования по согласованности и доступности.

Нестрогое доказательство теоремы CAP основано на простых рассуждениях. Пусть распределенная система состоит из N серверов, каждый из которых обрабатывает запросы некоторого числа клиентских приложений . При обработке запроса сервер должен гарантировать актуальность информации, содержащейся в отсылаемом ответе на запрос, для чего предварительно нужно выполнить синхронизацию содержимого его собственной базы с другими серверами. Таким образом, серверу необходимо ждать полной синхронизации либо генерировать ответ на основе не синхронизированных данных. Возможен и третий вариант, когда по каким-либо причинам синхронизация производится только с частью серверов системы. В первом случае оказывается не выполненным требование по доступности, во втором — по согласованности, в третьем — по устойчивости к разделению.

4.Системы NoSQL

Существует множество различных NoSQL-систем обработки данных, в котором можно выделить следующие основные классы: хранение XML-документов, хранение пар «ключ – значение», хранение графов, хранение кортежей произвольной длины, Triple Storages, многомерные данные и т. д.

Хранилища XML-документов (BaseX, eXist) представляют собой средства для работы с большим количеством XML-документов или с документами большого размера. Информация содержится не в текстовом виде, а в некотором внутреннем формате, позволяющем быстро выполнять операции поиска (запросы XPath и XQuery) и изменения документов (XSLT, eXtensible Stylesheet Language Transformations). При загрузке документа проводится его синтаксический анализ, результаты которого помещаются в базу, а при извлечении документа его текст восстанавливается на основе содержимого внутренних структур базы.

Формат XML — не единственный способ текстового представления структурированной информации: по аналогии с XML-хранилищами существуют СУБД (Apache Couch DB и MongoDB) для работы с данными, представленными в виде JSON (Java Script Object Notation) или BSON (Binary JSON).

Хранение пар «ключ – значение» заключается в реализации двух операций: запись информации по ключу и чтение по ключу — при этом одному ключу может соответствовать сразу несколько значений. Востребованность данной функциональности при построении высоконагруженных систем привела к тому, что появилось целое множество соответствующих СУБД, которые, в зависимости от способа реализации, можно разделить на постоянные (CDB), редко изменяющиеся (Appache Cassandra, membase, MemcacheDB) и часто изменяющиеся (memcached).

Постоянные СУБД не поддерживают изменения информации «на лету» — база данных создается один раз и затем используется продолжительное время в режиме чтения. Это, во-первых, позволяет провести глубокий анализ содержимого базы на этапе ее создания и обеспечить высокую степень компрессии данных, что впоследствии позволит минимизировать ввод/вывод и тем самым ускорить операции поиска. Во-вторых, если база не изменяется, то упрощается и ускоряется многопользовательская работа, поскольку не требуется выполнять никаких блокировок.

Системы для работы с часто изменяющимися данными всю информацию содержат только в оперативной памяти и вообще не используют ввода/вывода. Такой подход популярен при реализации механизмов кэширования, но принципиально не подходит при построении основного хранилища, поскольку любая авария питания приведет к потере данных. Наконец, системы с редко изменяющимися данными являются некоторым промежуточным звеном, обеспечивая возможность корректировки базы, достаточно высокую скорость поиска и сохранение информации на диске.

Часть хранилищ пар «ключ – значение» основаны на хэш-таблицах, а часть — на B-деревьях (BerkleyDB и MemcacheDB), что помимо выполнения операции поиска дает возможность упорядочивать ключи по возрастанию или убыванию.

В общем случае системы рассматривают ключ и значение как массив байтов, причем длина ключа может достигать нескольких килобайтов, а значения — нескольких мегабайтов. Однако часто некоторые СУБД предоставляют дополнительные возможности для структурирования ключей и значений. Так, Apache Cassandra позволяет разбивать значения на несколько колонок и работать с каждой из них отдельно, а BigTable, напротив, использует трехкомпонентные ключи (ключ столбца, ключ строки и временная метка), но значения рассматривает как массив байтов. Такое устройство

ключа позволяет использовать BigTable как средство работы с двумерными таблицами большого размера. Примерно по такой же схеме работают хранилища кортежей произвольной длины.

Системы хранения графов и Triple Storages ориентированы на работы с семантическими данными, представленными в виде узлов и дуг. Отличительной чертой Triple Storages является то, что они ориентированы на поддержку стандартов SemanticWeb.

Заключение

Движение NoSQL резко возрастает с 2009 года благодаря увлечению большого количества компаний связанных с использованием больших объёмов данных. Появляется все больше систем позволяющих организовывать и поддерживать огромные массивы данных, обрабатывать и контролировать эти данные.

Два направления развития СУБД — SQL и NoSQL — во многом противопоставляются, хотя и подчиняются единым законам и движутся навстречу друг другу. Пока это движение почти незаметно, но оно активизируется по мере появления необходимости в средствах построения высоконагруженных систем, в результате чего произойдет слияние этих направлений. Получившиеся таким образом СУБД могут оказаться слишком громоздкими для использования на практике, поэтому, возможно, произойдет отказ от текущей архитектурной концепции СУБД в сторону появления СУБД, представляющих собой набор «кубиков» для построения целевой системы, обладающей характеристиками, необходимыми для конкретных высоконагруженных приложений.

NoSQL лучшее решение когда :

- данные докмуент-ориентированы,
- вы работаете с объект-ориентированной системой,
- вы работаете с распределенной системой, требовательной к параметрам горизонтальной масштабируемости и небольшому времени отклика,
- вам необходимо дешевое решение для хранения данных и есть возможность интегрировать это решение с хостинг-провайдером вашего приложения.

Библиография

1. *NoSQL. Your Ultimate Guide to the Non-Relational Universe!* URL: http://nosql-database.org (дата обращения 19.11.2013).
2. *Oracle NoSQL Database*. URL: http://www.oracle.com/technetwork/database/database-technologies/nosqldb/overview/index.html (дата обращения 19.11.2013).
3. *Конец реляционных баз данных? Что такое NoSQL?* URL: http://www.programming-workshop.ru/data/databases/chto-takoe-nosql.html (дата обращения 19.11.2013).
4. *Обзор NoSQL систем.* URL: http://habrahabr.ru/blogs/nosql/77909/ (дата обращения 20.11.2013).
5. *NoSQL и SQL*. URL: http://ru.wikipedia.org/wiki/NoSQL (дата обращения 20.11.2013).