Большие данные: современные подходы к хранению и обработке
Шлюйкова Д. П.

Шлюйкова Дарья Петровна
бакалавр, студент магистратуры,
Российский экономический университет имени Г. В. Плеханова, г. Москва

Аннотация: *большие данные поставили перед традиционными системами хранения и обработки новые сложные задачи. В данной статье анализируются возможные способы их решения, ограничения, которые не позволяют сделать это эффективно, а также приводится обзор трех современных подходов к работе с большими данными.*
Ключевые слова: *большие данные, обзор технологий работы с большими данными.*

В различных сферах производства, торговли и ведения бизнеса возникает необходимость работы с данными, которых, зачастую, становится очень много. Но что такое «много»? Где тот порог, преодолев который данные становятся «большими»? Часто используется характеристика, данная исследовательской компанией Gartner: «Большие данные» характеризуются объемом, разнообразием и скоростью, с которой структурированные и неструктурированные данные поступают по сетям передачи в процессоры и хранилища, наряду с процессами преобразования этих данных в ценную для бизнеса информацию» [4].

Как видно из этого определения, большие данные имеют четыре основные характеристики: объем, разнообразие, скорость и ценность.

Рассмотрим их подробнее: 1. Объем. Нарастающее количество данных, создаваемых как людьми, так и машинами, предъявляет к IT инфраструктуре новые требования в отношении хранения, обработки и предоставления доступа. 2. Разнообразие. Данные содержат разнообразную информацию, представленную разными структурами. Со всем этим, от логов доступа к веб-серверу до операций по кредитным картам, от результатов научных экспериментов до фотографий и видео необходимо уметь работать. 3. Скорость. Важно осознавать, что под скоростью понимается не только скорость, с которой данные поступают в хранилище, но и скорость, с которой важная информация из этих данных извлекается. 4. Ценность. Большие объемы данных - это ценный ресурс. Но еще ценнее он становится, если позволяет отвечать на насущные вопросы или вопросы, которые могут возникнуть в будущем.

До определенного момента практически единственным ответом на вопрос «как хранить и обрабатывать данные?» являлась какого-либо рода реляционная СУБД.

Но с увеличением объемов появились проблемы, с которыми классическая реляционная архитектура не справлялась, поэтому инженерам пришлось придумывать новые решения.

Попробуем представить те шаги, которые можно предпринять, если СУБД прекращает справляться с объемом выполняемых операций: 1. Естественным первым шагом является попробовать наименее затратные способы. Самый простой, при наличии финансов способ - это ничего не предпринимать, а просто купить более мощное оборудование (вертикальное масштабирование). Однако бесконечно мощного сервера не существует, а значит, вертикальный рост конечен. 2. Более затратный способ - это оптимизировать запросы, проанализировав планы их исполнения, и создать дополнительные индексы. Такой метод может принести временное увеличение скорости работы с данными, но дополнительные индексы порождают дополнительные операции, а с ростом объемов обрабатываемых данных эти дополнительные операции приводят к деградации. 3. Следующим шагом может быть внедрение кэша на чтение. При правильной организации такого решения, можно избавить СУБД от существенной части операций чтения, но нанести ущерб строгой консистентности данных. К тому же, этот подход приводит к усложнению клиентского ПО. 4. Выстраивание операций вставки/обновления в очередь – также способно повысить производительность работы с данными, но размер очереди ограничен. К тому же, для обеспечения строгой консистентности необходимо организовать персистентность самой очереди, а это непростая задача. 5. Наконец, когда все прочие способы перестают работать, наступает момент пересмотреть способ организации самих данных. В первую очередь - произвести денормализацию схемы, чтобы уменьшить число нелокальных обращений. 6. Ну а когда и это не работает, то остается только масштабировать горизонтально, т. е. разносить вычисления на разные узлы. Здесь приходится окончательно попрощаться с нормализацией и внешними ключами, к тому же нужно ответить на вопросы: «по каким признакам распределять новые кортежи по узлам?» и «как произвести миграцию существующей схемы?».

Подводя итоги, можно заключить, что попытки приспособить реляционную СУБД к работе с большими данными приводят к следующему: 1. Отказу от строгой консистентности. 2. Уходу от нормализации и внедрению избыточности. 3. Потере выразительности языка SQL и необходимости моделировать часть его функций программно. 4. Существенному усложнению клиентского

программного обеспечения. 5. Сложности поддержания работоспособности и отказоустойчивости получившегося решения.

Необходимо, правда, отметить, что производители реляционных СУБД осознают все эти проблемы и уже начали предлагать масштабируемые кластерные решения. Однако стоимость внедрения и сопровождения подобных решений зачастую не окупается.

При взгляде на выводы из предыдущего раздела, в голове сразу рождается довольно очевидная мысль - а почему бы не спроектировать архитектуру, способную адаптироваться к возрастающим объемам данных и эффективно их обрабатывать? Подобные мысли привели к появлению движения NoSQL.

Популярность NoSQL стала набирать силу в 2009 г, в связи с появлением большого количества веб-стартапов, для которых важнейшей задачей является поддержание постоянной высокой пропускной способности хранилища при неограниченном увеличении объема данных.

Рассмотрим основные особенности NoSQL подхода [7]: 1. Исключение излишнего усложнения. Реляционные базы данных выполняют огромное количество различных функций и обеспечивают строгую консистентность данных. Однако для многих приложений подобный набор функций, а также удовлетворение требованиям ACID являются излишними. 2. Высокая пропускная способность. Многие NoSQL решения обеспечивают гораздо более высокую пропускную способность данных, нежели традиционные СУБД. 3. Неограниченное горизонтальное масштабирование. В противовес реляционным СУБД, NoSQL решения проектируются для неограниченного горизонтального масштабирования. При этом добавление и удаление узлов в кластере никак не сказывается на работоспособности системы.

Дополнительным преимуществом подобной архитектуры является то, что NoSQL кластер может быть развернут на обычном аппаратном обеспечении, существенно снижая стоимость всей системы. 4. Консистентность в жертву производительности. При описании подхода NoSQL нельзя не упомянуть теорему CAP. Следуя этой теореме, многие NoSQL базы данных реализуют доступность данных (availability) и устойчивость к разделению (partition tolerance), жертвуя консистентностью в угоду высокой производительности. И, действительно, для многих классов приложений строгая консистентность данных - это то, от чего вполне можно отказаться.

Пионером в области больших данных можно считать компанию Google, которая в 2003 г. описала распределенную файловую системы GFS [6], а в 2004 г. представила миру вычислительную модель MapReduce [1, с. 10]. Именно эти публикации помогли разработчикам свободного поискового движка ApacheNutch создать проект Hadoop, который сегодня фактически стал синонимом термина «большие данные».

Перед тем как рассмотреть Hadoop внимательней, следует ответить на вопрос: «а зачем нужен еще один продукт, если многие NoSQL базы данных предоставляют интерфейсы для MapReduce вычислений?» [5]. Ответ можно получить, рассмотрев производительность современных жестких дисков. Средняя производительность жесткого диска сегодня ~100 МБ/с, что означает возможность прочитать 1 ТБ информации примерно за 2.5 часа. Улучшить такие удручающие показатели можно параллельным чтением с нескольких дисков. Например, тот же самый 1 ТБ можно прочесть со 100 дисков за 2 минуты. Но ведь почти все NoSQL решения поддерживают горизонтальное масштабирование, а, следовательно, и параллельные дисковые операции?

И тут ключевым фактором становится время позиционирования головки. Для того чтобы операции обновления и чтения были эффективными, NoSQL базам (CouchDB, MongoDB) приходится использовать структуры с произвольным доступом, например, B-деревья. А значит, если отказаться от произвольного обновления данных и обрабатывать весь набор последовательно, можно добиться серьезного прироста производительности. Именно этот принцип и положен в основу архитектуры Hadoop. За хранение и организацию данных в Hadoop кластере отвечает распределенная файловая система HDFS [3]. При проектировании которой использовались следующие принципы: 1. Аппаратные сбои неизбежны. Поэтому HDFS реализует надежные алгоритмы репликации данных, а для метаданных файловой системы поддерживается журнал, позволяющий восстановить требуемое состояние. 2. Поточная обработка и большие объемы. HDFS устроены таким образом, чтобы обеспечить максимальную производительность поточного доступа к данным. К тому же структуры файловой системы оптимизированы для работы с большими файлами. 3. Локальность данных. Намного эффективней выполнять вычисления рядом с данными. HDFS предоставляет приложениям программный интерфейс, который позволяет выполнять вычисления ближе к необходимым данным, сокращая пересылки между узлами кластера. Вычисления в Hadoop представляются в виде последовательности map и reduce задач. В начале вычислений входное множество данных разбивается на несколько подмножеств. Каждое подмножество обрабатывается на отдельном узле кластера. Map задача на каждом узле получает на вход множество парключ-значений и возвращает другое множество пар. Далее все пары группируются по ключу, сортируются и подаются на вход reduce задаче, которая формирует финальный результат или вход для другой map задачи.

Управление процессами в распределенной системе - сложная задача. А сложнее всего справиться с частичными отказами, когда ошибки в отдельных процессах не должны влиять на вычисления в общем. MapReduce избавляет разработчика от необходимости думать об ошибках, требуется только лишь обеспечить код двух функций: map и reduce. Об остальном позаботится реализация, автоматически определив неудачно завершившиеся задачи и перезапустив их. Такая возможность возникает, потому что задачи независимы, ибо не имеют разделяемых ресурсов. Подобная особенность делает модель акторов идеальной для реализации MapReduce фреймворка. Этим и воспользовались разработчики исследовательского центра Nokia, создав проект Disco.

Особенностью Dicso является то, что ядро системы разработано на функциональном языке Erlang, снискавшем славу инструмента, отлично подходящего для программирования распределенных вычислений на основе модели акторов.

Компания Ericsson, использующая язык для программирования коммутационных узлов своей телефонной сети, даже заявила о достижении показателя отказоустойчивости оборудования в 99.9999999 % [2].

Hadoop, Disco и подобные им проекты отлично выполняют задачу распределенной пакетной обработки больших объемов данных. Фокус на пакетной обработке, в частности, приводит к тому, что вычисления происходят с большой задержкой. Подобные задержки могут быть неприемлемы для целого класса задач, где ответы на вопросы нужно получать незамедлительно.

В этой работе были рассмотрены проблемы, которые поставили перед реляционными СУБД большие данные. Проанализировав возможные пути решения этих проблем, мы указали на те концептуальные ограничения, которые не позволяют классической реляционной архитектуре справляться со стремительно возрастающим объемом информации. Далее были рассмотрены три подхода к работе с большими данными: NoSQL, MapReduce и обработка потоков событий в реальном времени. Мы обратили внимание на те архитектурные особенности, которые позволяют каждому из них эффективно решать поставленную задачу. Важно отметить, что ни один из представленных подходов не предлагает решения всех возможных задач, которые возникли в контексте больших данных. Каждый из них эффективно решает свой класс задач, позволяя пользователю выбрать наиболее подходящий для него инструмент.

Литература

1. *Jeffrey Dean, Sanjay Ghemawat*. MapReduce: simplified data processing on large clusters. Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation, vol. 6, p. 10-10, USENIX Association Berkeley, CA, USA, 2004.
2. *Joe Armstrong*. [Электронный ресурс]. Concurrency Oriented Programming in Erlang. URL: http://ll2.ai.mit.edu/talks/armstrong.pdf (дата обращения: 21.10.2015).
3. *Konstantin Shvachko, Hairong Kuang, Sanjai Radia, Robert Chansler*. The Hadoop Distributed File System. MSST '10 Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2010. pp. 1-10.
4. *Mark A. Beyer, Douglas Laney*. [Электронный ресурс]. The Importance of «Big Data»: A Definition. URL: http://www.gartner.com/DisplayDocument?id=2057415 (дата обращения: 21.10.2015).
5. [Электронный ресурс]. Riak. URL: http://basho.com/products/riak-overview/ (дата обращения: 11.12.2015).
6. *Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung*. The Google File System. 19th ACM Symposium on Operating Systems Principles, Lake George, NY, October, 2003.
7. *Кузнецов Сергей* К свободе от проблемы Больших Данных. «Открытые системы». № 02. 2012.