# К свободе от проблемы Больших Данных

Сергей Кузнецов

Пока исследователи и разработчики умудряются справиться с вчерашними «большими данными», появляются новые Большие Данные новых видов, с которыми совладать по-прежнему невозможно. В этой связи сегодняшний всплеск ажиотажа вокруг Больших Данных во многом является искусственным. Вечность и призрачность проблемы вряд ли позволяют рассчитывать на ее полное и окончательное решение. Это плохо для пользователей и разработчиков приложений, но гарантирует постоянную занятость исследователей и разработчиков СУБД. Конечно, их деятельность напоминает попытки моряков доплыть до миража, навеянного фата-морганой, но сами эти попытки увлекательны и полезны, поскольку поддерживают само развитие человечества.

Большие Данные и перспективные СУБД

Принято считать, что традиционные РСУБД позволяют эффективно управлять транзакционными и аналитическими базами данных. Транзакционные предназначены для поддержки оперативных транзакционных приложений (системы резервирования, управления логистикой, торговые системы и т. д.), работающих с данными, отражающими текущее состояние той или иной области деятельности, причем быстро и часто обновляемыми. Аналитические базы содержат исторические данные, поступающие из разных источников, одним из которых являются транзакционные базы данных.

Проблема Больших Данных знакома обеим этим категориям. Объемы транзакционных баз растут из-за развития оперативных потребностей пользователей, бизнеса или науки. Например, транзакционные базы интернет-магазинов сильно увеличиваются в объеме при персонализации услуг. Объемы аналитических баз увеличиваются в силу своей природы — данные в них всегда только накапливаются и не уничтожаются. Другой серьезной причиной роста объема аналитических баз является потребность бизнес-аналитиков в привлечении новых источников данных, например черпаемых из открытых ресурсов Сети.

Для транзакционных баз частный случай проблемы Больших Данных можно сформулировать следующим образом: нужно обеспечить технологию относительно недорогого масштабирования СУБД и транзакционных приложений, позволяющую поддерживать требуемую скорость обработки транзакций при росте объема данных и увеличении числа одновременно выполняемых транзакций. Для аналитических баз частный случай проблемы звучит так: требуется обеспечить технологию относительно недорогого масштабирования СУБД и аналитических приложений, позволяющую аналитикам расширять возможности СУБД по выполнению аналитических запросов и обеспечивать эффективную оперативную аналитическую обработку данных при росте их объема.

В первом десятилетии нового века исследователям во главе с Майклом Стоунбрейкером удалось нащупать пути решений обоих частных случаев, взяв за основу следующие общие принципы:

перенос вычислений как можно ближе к данным;

использование архитектуры без совместно используемых ресурсов (sharing nothing);

эффективное разделение данных по узлам системы с возможностью их репликации в нескольких узлах.

Первый принцип означает, что СУБД и приложения организуются таким образом, чтобы минимизировались пересылки данных между узлами системы. Очевидно, что важность этого принципа растет при росте объема данных. Следствием принципа является потребность в переносе приложений баз данных на сторону сервера.

Второй принцип говорит о возможности реального распараллеливания работы СУБД и приложений, поскольку при отсутствии общих ресурсов между узлами вычислительной системы (фактически при использовании кластерной архитектуры) уменьшается вероятность конфликтов.

Третий принцип обеспечивает эффективную параллельную обработку транзакций или эффективную поддержку оперативной аналитической обработки данных.

Все три принципа далеко не новы и датированы 80-ми годами прошлого века. Например, на принципах sharing nothing основывается популярная и эффективная параллельная СУБД Teradata, успешно используемая уже на протяжении нескольких десятков лет. Однако именно сегодня все три принципа удалось успешно применить при создании реально масштабируемых параллельных транзакционных и аналитических СУБД.

Применение этих принципов является необходимым, но не достаточным для реализации обеих категорий систем, и в каждом случае приходится прибегать к дополнительным идеям. В частности, транзакционные параллельные СУБД оказывается выгодно основывать на давно известных идеях обработки в основной памяти, а для обеспечения надежности данных применять развитую репликацию. В аналитических же системах более эффективна технология хранения табличных данных во внешней памяти по столбцам в совокупности с поддержкой разнообразных избыточных структур данных (идеи хранения данных по столбцам не новы и использовались, например, в аналитической СУБД Sybase IQ).

Пути решения проблемы больших транзакционных и аналитических данных намечены, но это не означает, что решены даже частные виды проблем. Например, транзакционные параллельные СУБД эффективно работают только при таком разделении данных, которое минимизирует число распределенных транзакций в имеющейся рабочей нагрузке, а при изменении рабочей нагрузки требуется перераспределять данные. Аналитические параллельные СУБД справляются со сложными аналитическими запросами только в тех случаях, когда разделение данных соответствует специфике запросов. Другими словами, время от времени приходится перераспределять данные громадного объема (в том числе и при горизонтальном масштабировании систем).

Масштабируемая параллельная серверная бизнес-аналитика

Итак, сообщество разработчиков баз данных сравнительно хорошо научилось строить горизонтально масштабируемые параллельные аналитические СУБД, способные поддерживать эффективное выполнение стандартных аналитических запросов. Но как быть с принципом приближения вычислений к данным? Если на сервере СУБД работают лишь базовые средства аналитики, то в любом мало-мальски серьезном аналитическом приложении придется перетягивать крупные объемы аналитических данных на рабочие станции или в лучшем случае — на промежуточные аналитические серверы. Единственным способом устранения этого дефекта является расширение серверных аналитических средств новыми аналитическими функциями, поставляемыми бизнес-аналитиками. На первый взгляд, соответствующие возможности предоставляют средства SQL, позволяющие пользователям определять собственные функции, процедуры и типы данных, но SQL не обеспечивает распараллеливания программ. Другими словами, аналитик вынужден писать параллельные программы для выполнения на стороне сервера при обработке будущих аналитических запросов, причем кусочки этих программ должны будут выполняться поблизости от соответствующих кусочков данных. Как это сделать?

Единственным распространенным методом параллельного программирования в кластерной среде является MPI — интерфейс передачи сообщений, позволяющий программисту указать расположение параллельно выполняемых частей программы в узлах кластеров и обеспечить их взаимодействие для формирования окончательного результата. Программирование с использованием MPI представляет собой сложность даже для профессиональных программистов. Можно ли сделать профессиональными программистами бизнес-аналитиков, которым ближе математическая статистика, а не методы параллельного программирования? Скорее всего, типичный аналитик, которому будет предложено приступить к решению такой задачи, предпочтет пользоваться старыми пакетами на своей рабочей станции, что загубит всю идею горизонтальной масштабируемости. На первый взгляд, проблема кажется неразрешимой, равно как неразрешимой кажется и более общая проблема обеспечения удобных и эффективных средств параллельного программирования для программистов широкого профиля. Недавно удалось найти подход и к решению этой проблемы, по крайней мере первой ее части (здесь следует отметить заслуги разработчиков параллельных СУБД Greenplum и Asterdata), — это MapReduce.

Технология MapReduce появилась в недрах Google как замена аналитическим параллельным СУБД для решения собственных аналитических задач компании. Технология быстро обрела популярность в среде практиков, но поначалу вызвала глубокое возмущение в сообществе разработчиков баз данных, авторитетные специалисты которого утверждали, что MapReduce — это возврат к доисторическому времени, когда для решения проблем управления данными требовалось явное программирование, и упрекали сторонников MapReduce в невежестве и неразумном отрицании результатов прошлых лет. Скорее всего, эти доводы правильны — MapReduce не может и не должна заменить технологию баз данных. Но оказалось, что эта технология может быть чрезвычайно полезной, если применить ее внутри самой параллельной аналитической СУБД для поддержки параллельного программирования и выполнения аналитических функций, поставляемых пользователями.

MapReduce концептуально несравненно проще, чем MPI, — программисту нужно понимать только одну идею того, что данные надо сначала распределить по узлам кластера, а потом обработать. Результат обработки можно снова распределить по узлам кластера и снова обработать и т. д. От программистов приложений требуется лишь обеспечить код двух функций: map — разделение данных по узлам кластера и reduce — обработка данных в полученных разделах. Такая парадигма программирования гораздо проще, чем MPI, но, что важно, она понятийно близка аналитикам.

Традиционная аналитика имеет дело с данными, явно или неявно представляемыми в виде многомерного куба. Первый и, по-видимому, революционный шаг на пути к обеспечению аналитической обработки табличных SQL-ориентированных данных в свое время обеспечивала работа Джима Грея, предложившего оператор roll up динамического построения многомерного куба в реляционной базе данных. Именно Грей, во-первых, понял, что традиционный оператор group by позволяет получить грань многомерного куба, а общая процедура построения многомерного куба сводится к повторному выполнению оператора group by до тех пор, пока не будет получено нужное число измерений. Во-вторых, Грей первым понял, что для выполнения операции roll up требуется столько же усилий, сколько и для выполнения простого group by. Что же такое тогда аналитика в контексте SQL-ориентированной СУБД? Грубо говоря, она сводится к применению различных агрегатных аналитических функций к граням куба.

Операцию group by можно было бы с успехом считать операцией «партиционирования» таблицы в соответствии со значениями заданного столбца группировки, а после этого можно было бы легко обрабатывать полученные группы параллельно. Можно считать, что функция map в программе MapReduce — это обобщенный оператор group by, обеспечивающий разделение исходных данных по некоторым явно запрограммированным пользователем правилам. Функцию reduce можно считать обобщением агрегатной функции в SQL. Получается, что аналитик при написании новой аналитической функции не обязан выходить за пределы того набора понятий, к которому он привык, работая с традиционной аналитической СУБД.

Создается впечатление, что поддержка MapReduce внутри параллельной аналитической СУБД должна полностью удовлетворить потребности аналитиков, а будущие аналитические приложения станут серверными, выполняемыми параллельно поблизости от своих данных. Все это означает, что может быть обеспечена горизонтальная масштабируемость будущих аналитических систем и тем самым решена и проблема Больших Данных. Однако здесь хочется сделать крамольное замечание. Фактически в новом поколении аналитических параллельных СУБД обеспечивается возможность параллельного программирования аналитических приложений, которое проще и понятней для неподготовленных специалистов. То есть частично решается более общая проблема — проблема параллельного программирования для суперкомпьютеров. Возникает вопрос, а не заслуживает ли этот подход более широкого применения, нежели расширение аналитического параллельного сервера баз данных? Не стоит ли попытаться применять MapReduce для программирования параллельных задач обработки больших объемов данных? Действительно, Большие Данные бессмысленны без больших вычислений, а большие вычисления чаще всего невозможны без поддержки эффективного доступа к Большим Данным. Не стоит ли об этом задуматься?