Р.А. СИМАКОВ,
М.А. СМЯТКИН

Особенности архитектуры распределенной массивно-реляционной СУБД

УДК 004.65

Муромский институт (филиал) ФГБОУ ВПО «Владимирский государственный университет имени Александра Григорьевича и Николая Григорьевича Столетовых», г. Муром

В статье рассматривается архитектура распределенной СУБД, поддерживающей одновременно многомерную и реляционную модели данных. СУБД ориентирована на хранение, обработку и анализ больших объемов данных. В статье описываются некоторые архитектурные особенности и выделяются моменты, которым в дальнейшей работе необходимо уделить наиболее пристальное внимание.

In the paper architecture of distributed relational and multidimensional DBMS is considered. It is oriented to store, process and analyze big data sets. The paper describes some architectural features and points to most important of them that should be studied in detail.

Введение. Поддержка в распределенной СУБД одновременно многомерной и реляционной моделей делает её архитектуру уникальной. Актуальность, фундаментальные требования и области применения СУБД обозначены в работе [9]. В этой статье представлено формирование архитектурных решений, способствующих удовлетворению поставленных ранее требований.

Архитектура.

Допустимые операции над данными. Выделяют 3 операции, модифицирующие набор данных (записи в реляционной модели и ячейки в многомерной): добавление, обновление и удаление. Выполнение модифицирующих операций над одинаковыми данными

может привести к одной из следующих конкурентных ситуаций: добавление - добавление, обновление - обновление, обновление - удаление. Возможность возникновения конкурентного доступа к данным порождает необходимость использования журналов транзакций, двухфазовых фиксаций, наложения блокировок на данные и, возможно, некоторых других операций, снижающих масштабируемость и производительность системы. Частично, решение может быть найдено устранением из системы распределенных транзакций [1], однако, полное устранение зачастую не возможно. В таком случае, значительное упрощение проектирования системы, соответствующей поставленным требованиям, может быть осуществлено с отказом от некоторых операций над данными.

Операция добавления, так или иначе, необходима для формирования набора данных и её исключение не представляется возможным. Работа без операции удаления возможна, однако её исключение приведет лишь к ликвидации конкурентного доступа при обновлении-удалении. В то же время, целевой пользователь иногда хочет иметь возможность удаления данных, например, при очистке мусора, удалении производных данных (результаты анализа в OLAP и экспериментов в научных приложениях). Операция обновления в ряде приложений может быть естественным образом необходима (например, при изменении заработной платы сотрудника в OLTP системе). Однако в обозначенных выше областях применения обновления либо вообще не могут возникнуть, либо естественным образом ложатся на построенную модель данных, представляясь в виде добавления. Например, при фиксации датчиком изменения температуры в определенной точке – запись с соответствующей точкой в базе данных не обновляется, а добавляется новая с заданным временем измерения. В большинстве случаев, когда данные в OLAP или в научных приложениях обновляются, проблема разрешается подобным образом через добавление новой записи. Кроме того, данные в целевых приложениях обычно получаются в виде фактов: пользователь выполнил действие, датчик собрал информацию и т.д., которые сами по себе редко являются ошибочными, что подразумевает отсутствие необходимости в их обновлении с целью исправления.

Таким образом, исключение операции обновления позволяет увеличить масштабируемость и производительность системы, оказывая сравнительно небольшое влияние на функционал. Полученный набор операций – добавление и удаление – похож на набор операций append-only и принят за основу многими OLAP системами и распределенными СУБД. Принимая во внимание данную архитектуру, остаются два вопроса, требующие решения для полноценной работы системы: обработка конкурентного добавления данных и решение ситуаций, в которых нельзя обойтись без обновления. Обе проблемы решаются одинаковым образом – введением понятия контекста пользователя, способствующего реализации целостности в конечном счете (см. механизм транзакций). Существует довольно низкая вероятность одновременного добавления взаимоисключающих данных. Как правило, такие ситуации можно разделить на машинные (когда добавление данных осуществляется программно, исходя из показаний датчика, например) и человеческие (когда процесс инициируется и управляется человеком, например во время исследования). Контролировать проблемы первой группы сложно на уровне СУБД, и они должны быть разрешены в логике вышестоящего приложения. Проблемы же второго уровня не должны мешать пользователю. Предполагается, что исследователь, например, будет выполнять все операции, изменяющие данные в своем контексте, который не будет прерываться даже при возникновении конкурентных ситуаций, и, в худшем случае, при попытке фиксации ему будет предложена возможность сохранить результаты работы или повторить их автоматически на основе новых данных. Таким образом, появляется возможность обеспечения целостности в конечном счете.

Случаи, когда обновление данных необходимо, могут быть реализованы через удаление и добавление данных в контексте пользователя. Более того, из особенностей OLAP приложений результаты исследования или анализа должны храниться отдельно от исходных данных, т.е. исключая обновление последних, которые и являются общими в текущий момент времени. Такие данные могут обновляться в контексте пользователя, не снижая масштабируемость и производительность. Кроме прочих преимуществ, описанных в этом пункте, append-only модель позволяет использовать локальный кэш

в большинстве случаев, когда данные уже были до этого запрошены.

Разделяемая память. Из трёх наиболее общих структур разделения памяти между потоками выполнения (в данном случае, понятие рабочего потока приравнивается к одному вычислительному узлу системы) – shared-memory, shared-disk и shared-nothing [2] – в нашем случае выбрана последняя по следующим причинам:

- Распределение памяти в случаях shared-memory и shared-disk приводит к повышению нагрузки на сетевую подсистему, повышая количество коммуникаций между узлами, в то время как обмен сообщениями в распределенной системе является одной из самых дорогих операций.

- При наличии общей памяти возникает необходимость в создании центральных узлов, которые являются узким местом в пропускной способности системы, получая большое количество сетевых сообщений, и нарушают требование отказоустойчивости системы.

Безусловно, данные подходы имеют и своим преимущества, например, упрощение поддержки целостности данных и контролирования конкурентного доступа, которые, однако, при имеющихся требованиях являются менее критичными. В тоже время, shared-nothing подход позволяет создать систему, в которой каждый узел функционирует автономно, исключая ряд узких мест и приводя проект к более децентрализованному виду. Это способствует удовлетворению требований отсутствия централизованных узлов в системе и линейного роста производительности (требования 5 и 3 соответственно).

Однако, современные компьютеры и, вероятно, компьютеры ближайшего будущего нацелены на поддержку параллельной обработки информации и реализуют многоядерную архитектуру. Этот факт позволяет эффективно реализовать на локальном уровне распределение памяти по принципу shared-memory, как было реализовано в DORA [3]. Таким образом, архитектура системы принимает вид: shared-nothing на глобальном (сетевом) уровне и shared-memory на локальном (в пределах узла). Наиболее вероятно, что на каждом узле будет функционировать один процесс системы, который в зависимости от поддержки многоядерной архитектуры будет разбиваться на несколько потоков. Выбор потоков операционной

системы в качестве потоков выполнения основывается на возможности получения преимуществ от использования общей Кэш-памяти.

Распределение ролей между узлами системы. С целью соблюдения требований отказоустойчивости и прозрачности (требования 5 и 2), система будет проектироваться с целью возможности отведения любой роли любому узлу. То есть, каждый из имеющихся узлов системы может быть выбран клиентским приложением для посылки запросов на обработку, а системой для инициирования выполнения запроса (задача выбора лидера [3]), репликации, чтения, обработки данных и т.д. Подобная архитектура позволит обеспечить наиболее высокую пропускную способность и отказоустойчивость, а также предоставит системе широкие возможности по распределению нагрузки и перераспределению хранимых данных. В ряде случаев встает вопрос о выгоде использования центральных узлов, на которые накладывается определенная роль, вместо равномерного распределения ролей. Подобные случаи будут обсуждаться далее в статье.

Понятие транзакции. В классическом случае, согласно модели ACID [4], транзакционная система должна обеспечивать атомарность, целостность, изолированность и долговечность транзакций. В распределенных системах, при реализации ACID-транзакций возникает ряд сложностей: высокая вероятность возникновения распределенных транзакций, накладывание блокировок на распределенные данные, ведение журналов транзакций и т.д., которые могут значительно снизить эффективность системы. Многие современные распределенные СУБД, основываясь на «теореме» CAP [5], предпочли, с целью повышения масштабируемости, ACID-транзакциям BASE-транзакции [6], обеспечивающие согласованность в конечном счете. Существуют исследования, доказывающие возможность использования ACID-транзакций параллельно поддержке высокой масштабируемости системы [1]. Однако наиболее успешные из исследований базируются на заранее известном наборе запросов и оптимально подстроенном под него хранилище данных (для максимального исключения распределенных транзакций).

В пункте, посвященном выбору append-only модели, уже были освещены некоторые вопросы, касающиеся механизма транзакций.

В частности, описано применение контекста пользователя и соблюдение целостности в конечном счете. Первое понятие соответствует механизму snapshot-транзакций [7], которое обеспечивает работу пользователя со своими данными и с данными, которые имелись на момент начала работы. А понятие целостности в конечном счете характерно для BASE-транзакций. То есть, наиболее вероятно, что в системе будет реализована поддержка snapshot BASE-транзакций.

Каждая транзакция может быть завершена в контексте пользователя или завершена глобально, в зависимости от ее закрепления на репликах всех изменяемых данных. Во время фиксации изменений на узлах будет применяться механизм близкий к оптимистичным блокировкам, так как конкурентный доступ к данным маловероятен. В случае же возникновения конкурентной ситуации возможно предоставление таких вариантов как полный откат транзакции, сохранение определенных результатов или автоматическое повторение транзакции на новых данных. Транзакция, зафиксированная в контексте определенного пользователя, становится зафиксированной глобально, когда каждая из участвующих реплик зафиксирует соответствующие изменения. Для каждого запроса существует список узлов, участвующих в его исполнении и их реплик, так как каждая реплика является автономным узлом и может так же способствовать возникновению конкурентной ситуации. Как только какой-либо узел получает информацию о том, что все узлы, имеющие определенные данные зафиксировались на реплике (т.е. была наложена локальная блокировка), то транзакция помечается глобально зафиксированной и все узлы снимают блокировку.

Хранение метаданных. Под метаданными будут пониматься метаданные типичные для одноузловых систем: информация о таблицах, индексах, типах и т.д. Однако важный вопрос при проектировании системы хранения метаданных в распределенной СУБД состоит в выборе между централизованным и распределенным (локальным) хранением метаданных. В первом случае гарантируется целостность данных, однако увеличивается время выполнения запросов за счет обращения к хранилищу метаданных и появляется дополнительный центральный узел в системе с последствиями, описанными выше. Противоположный вариант является децентрализованным, но порождает избыточное хранение и гарантирует согласованность

только в конечном счете. Однако в этом случае достигается сохранение отказоустойчивости и равномерное распределение (а в случаях, описанных далее, снижение) нагрузки на сеть. В худшем случае, когда метаданные на узле устарели, они могут быть проверены и исправлены естественным способом во время распределения запроса по узлам. Стоит отметить, что подобные случаи в исследуемой предметной области с поддержкой архитектуры append-only должны встречаться достаточно редко. Также, избыточность при хранении метаданных не должна быть слишком велика и компенсируется уменьшением сетевых коммуникаций. Таким образом, для реализации системы будет выбрано локальное хранение метаданных. Логически хранение метаданных будет произведено в виде множеств, аналогично метаданным в реляционной модели.

Нахождение требуемого узла. Существует ещё одна разновидность метаданных, характерная именно для распределенных хранилищ – информация о физическом расположении данных. В отличие от метаданных, рассмотренных выше и описывающих логическую структуру базы данных, метаданные, описывающие физическую структуру, не предоставляются рядовому пользователю и способствуют прозрачной работе с распределенными данными (требование 2). Однако, предполагается предоставление подобных данных по запросу к серверу определенной группе пользователей в административных целях (визуализация схемы, показ статистики обращений к узлам и т.д.). Само распределение данных может осуществляться различными способами, часть которых описана в работе [2]:

- Размещение данных в сети в виде распределенной хэш-таблицы.

- Распределение по методу Round-Robin(циклическое кольцо), где информация циклически разбивается на набор узлов и в случае достижения последнего узла, продолжает сохранение от начала.

- Распределение на основе диапазонов значений (Range-based).

- Распределение на основе эвристических методов с учетом собранной статистики или статического анализа пользовательских запросов.

Большое значение при распределении данных имеют возможность расширяемости системы, балансировка нагрузки и снижение количества распределенных транзакций. Так, если значение последнего требования можно ослабить в случае реализации BASE-транзакций, то первые два в любом случае необходимо учитывать. Распределение данных по механизму Round-Robin или при использовании распределенной хэш-таблицы не поддерживают должным образом механизм балансировки нагрузки при наименьшей (для обеспечения отказоустойчивости) избыточности. В противоположность этим методам ставится создание таблицы маршрутизации (вероятно, Range-based) формирующейся на основе статистических данных и позволяющей распределять и находить данные в системе. Хранение таблицы маршрутизации (физических метаданных) по тем же причинам, что и хранение логических метаданных будет осуществляться локально на каждом узле.

Физическое хранение данных. В большинстве современных распределенных СУБД данные между узлами распределяются в виде кусков одинаковых или разных размеров, в зависимости от архитектурных особенностей. В данном случае куски данных будут формироваться на основе построения логических групп. Как правило, куски данных растут до определенного размера, а затем разбиваются на два меньших куска. На основе статистики можно получить приблизительное значение кардинальности и коэффициента сжатия ключа, по которому будут выстраиваться куски. И основываясь на максимальном размере куска рассчитать интервал значений по ключу, который будет храниться в куске соответствующего размера. Таким образом, размер куска будет ограничен физически определенной величиной на уровне конфигурации СУБД и логически на уровне каждой таблицы отдельно.

Хранение множеств картежей изучалось с момента возникновения реляционной модели хранения данных и достаточно подробно изучено на данный момент. Вероятно, как в большинстве реляционных систем хранение множеств записей будет произведено в страничном виде. В отличие от реляционной модели, модель хранения данных в виде массива является менее детально изученной и требует более внимательного подхода. Так как многомерная модель данных, как правило, не нормирована, и значения многих ячеек

(кортежей) могут отсутствовать, то необходимо предусмотреть возможность устранения резервирования избыточного места для отсутствующих ячеек. Решение этой проблемы планируется представлением данных по измерениям в виде разреженных массивов и их сжатием RLE-алгоритмом.

Для лучшего сжатия данных и снижения объема данных, передающихся по сети, выбрана вертикальная модель хранения данных [8]. Кроме того, на физическом уровне не будет поддерживаться разница между Empty и Null кортежами. Т.е. кортеж, в котором все поля хранят значение Null, является Empty кортежем.

Динамическое форматирование в базе данных будет отсутствовать, так как оно порождает дополнительные сложности при реализации и накладные расходы при вычислениях, являясь, при этом, необходимым лишь в документно-ориентированных СУБД.

Репликация. Основная роль репликации в распределенной системе – это повышение отказоустойчивости за счет увеличения избыточного хранения данных. Обычно пользователю достаточно иметь две копии одних и тех же данных, так как вероятность одновременного отказа двух узлов чрезвычайно мала. В связи с требованием 5 необходимо обеспечить отсутствие центральных элементов, которое не может быть реализовано при использовании ряда узлов только в роли хранилищ избыточных данных. Однако в архитектуре append-only с реализацией BASE-транзакций при высокой вероятности операций чтения и малой вероятности возникновения взаимоисключающих операций параллельно на нескольких копиях становится возможной реализация каждой реплики как автономного узла с наложением оптимистических блокировок. Таким образом, клиентский запрос будет выполняться на выбранной системой реплике так, как если бы существовала только одна копия данных. В данном случае необходимо уделить особое внимание соблюдению изолированности работы клиента – набор реплик, выделенный ему системой во время выполнения текущей транзакции должен сохраняться как минимум до момента фиксации изменений и, по возможности, даже дольше.

Один из важнейших вопросов при проектировании подсистемы репликации – это физическое разбиение данных на части и их последующее распределение между репликами. В первую очередь

необходимо решить насколько мелкими будут реплицируемые части:

- в случае репликации большим количеством небольших кусков размер таблицы маршрутизации возрастает, но увеличивается вероятность равномерного распределения нагрузки без применения статистических методов;

- в противном случае – при репликации большими кусками данных уменьшается размер таблиц маршрутизации, однако вероятность равномерного распределения нагрузки без применения статистических методов значительно уменьшается;

- одним из компромиссных вариантов является первоначальное распределение данных на основе одного из предыдущих вариантов или на основе параметров конфигурации, а затем перераспределение данных на основе статистического моделирования равномерного распределения нагрузки для уменьшения количества сетевых коммуникаций. Данный подход является сложно реализуемым и слабо изученным, однако предоставляет большие преимущества.

Кроме того, следует отметить, что первый вариант исключает преимущества от применения таблицы маршрутизации – увеличивая ее избыточность и уменьшая возможность реализации семантически равномерного распределения. Таким образом, при выборе первого варианта наиболее подходящими методами распределения данных по узлам являются распределенная хэш-таблица и Round-Robin, тогда как второй и третий метод позволяют использовать преимущества таблиц маршрутизации.

Выбор конечного алгоритма репликации на данный момент не является критически важным вопросом, так как это достаточно изолированная подсистема и может быть перепроектирована на любом этапе с нагрузочным тестированием на уже имеющейся системе.

Распределение нагрузки между узлами. Использование реплик в вычислениях аналогично обычным узлам системы создаёт проблему выбора подходящего узла для проведения вычислений. В любом случае, независимо от наличия статистических данных, распределение нагрузки будет производиться на основе вероятностных величин. В простейшем случае, когда имеется 2 копии данных и отсутствует статистика по узлам, на которых они находятся, существуют вероятности $p_1=0.5$ и $p_2=0.5$ отвечающие за выбор первой и

второй копии соответственно. Однако при наличии статистики может быть учтена загруженность узлов, их надежность и скорость обмена сообщениями с ними. Таким образом, вероятности могут быть равными p_1=0.4 и p_2=0.6 при одинаковой отказоустойчивости, скорости коммуникаций и загруженности второго узла в полтора раза превышающей загруженность первого.

Сборка статистики. Собираемую распределенными СУБД статистику можно разделить на две категории: статистика распределения нагрузки и статистика хранилища. Первая из них служит таким целям как обеспечение равномерной загрузки системы, уменьшение количества сетевых коммуникаций, выбор оптимального количества реплик для конкретных данных и т.д. На её основе зачастую строится физический план запроса с указанием конкретных узлов. Статистика хранящихся данных необходима для логической оптимизации запроса – создания оптимального дерева выполнения.

Сборка статистики загрузки узлов должна быть непрерывной, так как её необходимо вычислять параллельно с выполнением каждой сетевой или логической операцией. С этой целью предполагается ведение пассивного учета статистики: каждая операция, например выполнение запроса, записывает в статистику время своего выполнения. Статистика по хранящимся данным, напротив, собирается в определенные моменты времени и инициализируется, например, в фоновом потоке низкого приоритета или по какому-либо событию.

Сетевая статистика должна собираться на каждом узле по времени выполнения операций в нём самом и в других узлах, работу которых он координировал в момент запросов. Подобные данные в совокупности смогут быть использованы для вычисления: отказоустойчивости и загруженности конкретного узла, времени коммуникаций узлами и т.д.

Перераспределение структуры хранилища. Потенциально, имеется возможность проведения автоматического перераспределения данных с целью балансировки нагрузки в распределенном хранилище на основе собираемой статистики. Предполагается проведение предварительного моделирования оптимальной нагрузки системы по запросу администратора или периодически, аналогично сборке статистики. Вероятно, исследование и разработка методов и

алгоритмов автоматического перераспределения нагрузки по узлам будет находиться в центре проводимой научной работы и им будет уделено особое внимание.

Заключение.

При написании статьи ставились три основные цели – определить проблемы и требования, спроектировать архитектуру системы и выделить ряд проблем, решение которых требует особенно пристального внимания. Ниже кратко подытожены архитектурные особенности проектируемой СУБД:

- поддержка как многомерной, так и реляционной модели данных;
- предоставление только операций добавления и удаление данных, т.е. соответствие append-only модели;
- как можно большее исключение из системы центральных узлов, блокировок, журналов откатов;
- shared-nothingархитектура на сетевом уровне и shared-memoryна локальном;
- логическая и физическая гетерогенность системы: узлам могут назначаться любые роли, а сами узлы могут запускаться на различных платформах;
- поддержка snapshot BASE-транзакций, т.е. обеспечение целостности в конечном счете при проведении модифицирующих операций в контексте пользователя;
- локальное хранение метаданных и таблицы маршрутизации;
- отсутствие динамического форматирования данных, эквивалентность Empty записей и записей, заполненных NULL значениями; поддержка вертикальной схемы хранения;
- сборка статистики хранилища и оптимизатора запросов;
- использование статистики для распределения данных и нагрузки на узлах;
- поддержка репликации, притом, что каждая реплика – это полноценный узел.

Особое внимание при дальнейшем проектировании системы следует уделить:

- исследованию и разработке методов и алгоритмов распределения и перераспределения обработки данных по узлам;

- физической структуре хранилища. Необходимо спроектировать структуру хранилища, способную эффективным образом работать с append-only системой при поддержке как реляционной, так и многомерной моделей данных. Поддерживая при этом высокую производительность при работе с разделами данных;

- проработке механизма транзакций, обеспечивающего целостность в конечном счете при отсутствии единой точки отказа.

Литература

1. С. Д. Кузнецов. Транзакционные параллельные СУБД: новая волна. – Труды института системного программирования, т. 20, М., ИСП РАН, 2011, стр. 189 – 251.

2. J. Hellerstain, M. Stonebreaker. Anatomy of Database System. Red Book.

3. Rachid Guerraoui, Luis Rodrigues. Introduction to Distributed Algorithms. – New York: Springer-Verlag, 2004. – 238 p.

4. J. Gray and A. Reuter. Transaction Processing Concepts and Techniques. Morgan Kaufmann Publishers, 1993.

5. Eric Brewer. Towards Robust Distributed Systems, Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, July 2000, p. 7.

6. Dan Pritchett. BASE: An ACID Alternative. Queue – Object-Relational Mapping. – ACM, Volume 6 Issue 3, May/June 2008. – ACM New York, NY, USA, 2008. – pp. 48-55.

7. H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil. A critique of ANSI SQL isolation levels. In processing of SIGMOD, pp. 1-10, 1995.

8. S. Navarthe, S. Ceri, G. Wiederhold, J. Duo. Vertical partitioning algorithms for database design. ACM Transactions on Database Systems (TODS), Volume 9 Issue 4, Dec. 1984. – ACM New York, NY, USA, 1984, - pp. 680-710.

9. М.А. Смяткин, Р.А. Симаков. Предпосылки создания распределенной массивно-реляционной СУБД. Алгоритмы, методы и системы обработки данных. 2011. №18.

Р.А. СИМАКОВ

E-MAIL: ROMAN.SIMAKOV@GMAIL.COM

М. А. СМЯТКИН – ТЕЛ.: 8 (910) 171 82 76,

E-MAIL: SMYATKINMAXIM@GMAIL.COM