Об организации кэша распределённой графовой базы данных♣

© А.А. Демидов

Исследовательский центр искусственного интеллекта ИПС им. А.К. Айламазяна РАН
alex@dem.botik.ru

Аннотация

Статья посвящена проблеме организации клиентского кэша распределённой базы данных, обеспечивающего непротиворечивое чтение и изменение данных. При этом ставится задача минимизации синхронизации между кэшем и серверами распределённой системы, для чего применяется пассивный метод синхронизации данных кэша по возникновению конфликта версий. Предлагаемое решение позволяет хранить в кэше данные различной степени актуальности и непротиворечиво обрабатывать их, избегая излишней синхронизации.

1. Введение

При построении распределённой системы возникает проблема синхронизации данных между удалёнными серверами – узлами системы, каждый из которых в определённый момент времени обрабатывает некоторую часть общих данных. Выбор стратегии синхронизации узлов влияет на оперативность обработки информации и устойчивость системы к отказам серверов или коммуникаций [1]. Излишняя степень синхронизации влечёт существенный рост накладных расходов: частое согласование данных между узлами требует применения более мощных серверов и большей пропускной способности каналов связи, а также приводит к излишнему усложнению алгоритмов работы системы и соответствующему снижению её надёжности. Поэтому в распределённой среде уменьшение синхронизационных издержек является важной задачей.

Наиболее радикальным шагом в направлении ослабления требований к синхронизации конкурирующих процессов является отказ от строгой непротиворечивости ACID транзакций и переход на принципы BASE, где синхронизация копий в распределённой системе достигается лишь по прошествии некоторого времени после изменения данных – ярким представителем систем подобного рода является сервис Amazon Dynamo. Процедуру разрешения конфликтов в Amazon Dynamo должно обеспечить приложение-клиент при чтении данных, что не всегда реализуемо на практике, хотя в рамках принципов BASE существуют различные варианты организации такой процедуры обеспечения непротиворечивости, в том числе и при записи объектов, как и различные вариации самой модели непротиворечивости [2, с.335] – здесь важен сам принцип отложенного проявления изменений в системе.

Необходимая степень синхронизации конкурирующих процессов напрямую зависит от структуры решаемой задачи: хранение и обработка слабосвязанных данных позволяет существенно ослабить требования к синхронизации: каждый элемент данных обрабатывается атомарно, а его изменение не затрагивает никакие другие элементы в силу отсутствия связей между ними. Таковы документ-ориентированные базы данных и хранилища пар ключ-значение: Google BigTable, Apache Hadoop, Apache CouchDB и другие. Тем не менее, остаётся большой класс задач, требующих для своего представления структур данных значительной степени связности. При построении распределённых систем для таких задач можно использовать традиционные реляционные или графовые СУБД с поддержкой полноценных ACID транзакций – Oracle Coherence, HyperGraphDB или подобные системы.

Между тем, использование распределённых ACID транзакций не всегда является необходимым условием обеспечения непротиворечивости. Если сервис не является системой реального времени и допускает некоторую задержку между моментом изменения информации и моментом проявления этих изменений в системе, то в таком случае можно использовать подходы BASE – кроме этой задержки, обусловленной характерным временем распространения изменений, такой сервис с точки зрения клиента будет неотличим от аналогичной системы, построенной на принципах ACID. При этом за счёт снижения синхронизационных издержек операционные возможности системы существенно возрастут. В данной работе исследуется один из вариантов построения распределённой системы на принципах BASE, в которой универсальная процедура непротиворечивого разрешении конфликтов в значительной мере возлагается на клиентский кэш.

Труды 13[ii] Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» – RCDL'2011, Воронеж, Россия, 2011

2. Определения

Обобщённое отношение обусловленности является основным понятием всей работы.

Определение 1. Введём понятие пространства байт или *атомарного пространства* – абстрактного адресного пространства M неделимых элементов, доступного для хранения информации базы данных. Как правило, пространство байт естественным образом сопоставляется реальному дисковому пространству серверов распределённой системы.

Определение 2. Определим *непосредственное отношение обусловленности* элементов данных следующим образом: если при изменении одного элемента данных a значение второго b теряет смысл в рамках принятой модели данных, то будем говорить, что имеется непосредственная направленная зависимость элемента a от элемента b, и записывать: $a \to b$, или $a \leftrightarrow b$ в случае обоюдной зависимости. Множество всех пар зависимых элементов базы данных и является определяемым отношением обусловленности элементов. Функциональная зависимость элементов, хорошо известная в теории реляционных баз данных, является частным случаем непосредственного отношения обусловленности: если b функционально зависит от a, то $a \to b$, обратное неверно.

Определение 3. Также введём более широкое нетранзитивное или *обобщённое отношение обусловленности*: если $(a \to b) \wedge (b \to c)$, то будем говорить, что элемент c обобщённо зависит от элемента a и записывать $a \dashrightarrow c$, даже в случае отсутствия непосредственной зависимости $a \to c$ между ними. По нетранзитивным связям становится возможным появление циклических зависимостей элементов: $a \dashleftrightarrow c$. Определяемое обобщённое отношение обусловленности является множеством: $\{(x, y): x \dashrightarrow y\} = \{(x, y): (\exists z \in M)\ (x \to z) \wedge (z \to y)\}$.

Определение 4. На основе отношения обусловленности введём поле или иначе – *предтопологическое пространство зависимостей* (R), определив систему предокрестностей с помощью оператора предзамыкания (preclosure operator); в отличие от обычного оператора замыкания он не является идемпотентным и не требует совпадения замкнутого множества со своим замыканием [3, с.191]. При этом расстоянием $\mathrm{d}(a, c)$ между двумя элементами a и c будем считать длину минимальной последовательности $a \ldots c$ в направленности[1] $a \dashrightarrow c$. Тогда пространство зависимостей R будет задано предбазой P замкнутых предокрестностей $O(a, r)$ элементов атомарного пространства M:

$O(a, r) = \{x \in M: \mathrm{d}(a, x) \le r\}$,

$P = \{O(a, r): a \in M, r \ge 0\}$.

Требования к оператору замыкания допустимо ещё несколько ослабить, применив оператор псевдозамыкания со свойствами, указанными в работе [4], который определяется как отображение $CO: F(M) \to F(M)$ семейства всех подмножеств множества M в себя, удовлетворяющее только первым двум свойствам из четырёх:

(CO1): $[\varnothing] = \varnothing$

(CO2): $\forall A \in F(M)$, $A \subset [A]$

(CO3): $[A \cup B] = [A] \cup [B]$

(CO4): $[[A]] = [A]$

– оператор предзамыкания дополнительно удовлетворяет свойству CO3, а обычный оператор замыкания – ещё и CO4. Введённый таким образом оператор псевдозамыкания вместо CO3 обладает только более слабым свойством: $\forall A \forall B \in F(M)$, $A \subset B \Rightarrow [A] \subset [B]$, следующим из включения $C_A \subset C_B$ семейств всех замкнутых множеств, содержащих A или B соответственно [6, с.35]. Введённое таким образом «V-пространство»[2] оказывается более общим, чем предтопологическое «D-пространство», построенное с использованием CO3, и поскольку свойство CO3 нам не требуется, далее можно рассматривать R как V-пространство.

Определение 5. Будем говорить, что две *направленности сходятся*, если начиная с некоторого элемента все их последующие элементы совпадают (здесь допускается наличие циклов).

Определение 6. Назовём классическим или просто *объектом* псевдозамыкание в R произвольного подпространства пространства R. Псевдозамыкание позволяет лаконичней учесть межобъектные связи, включая их в состав объекта; сами объекты при этом становятся пересекающимися замкнутыми псевдоокрестностями в пространстве R. На уровне реляционных баз данных им отвечают каскадные операции: при изменении ключа главной записи зависимые записи должны быть также изменены, либо операция должна быть прервана. В предлагаемой терминологии это действие находит более естественное выражение – меняется один объект, но он пересекается с другими и занимает нелокальную область пространства M.

Определение 7. Назовём *замкнутым объектом* такой классический объект, для которого в R выполняется свойство идемпотентности CO4. То есть такой объект, включающий в себя все области зависимостей данных, от которого в остальной части пространства R уже ничего не зависит.

Определение 8. Будем говорить, что объект a *перекрывает* объект b, если $a \to b$. Перекрытие объектов ослабляет понятие пересечения псевдоокрестностей объектов, поскольку зависимость в общем случае носит направленный характер. Если направление зависимости не имеет значения или $a \leftrightarrow b$, будем говорить, что объекты a и b *пересекаются*.

Выделение объектов здесь в достаточной мере произвольно – в качестве объектов можно выбирать подграфы любого размера в зависимости от нужд приложения, хотя минимальный размер замкнутых объектов ограничен требованием соблюдения свойства CO4. Далее с целью упрощения выкладок будем предполагать, что каждой выявленной зависимости отвечает реальное ребро между соответствующими вершинами в графе базы данных – в противном случае все зависимости для алгоритмов работы с транзакциями должны быть заданы в метаданных.

3. Пример выделения объектов

Пусть необходимо организовать ведение картотеки служащих, которые занимают или занимали в прошлом определённые государственные посты. Вся информация будет храниться в базе данных, которую можно рассматривать в качестве атомарного пространства данной задачи. Как всегда, объекты удобно сгруппировать в классы и далее вести проектирование на уровне схемы данных.

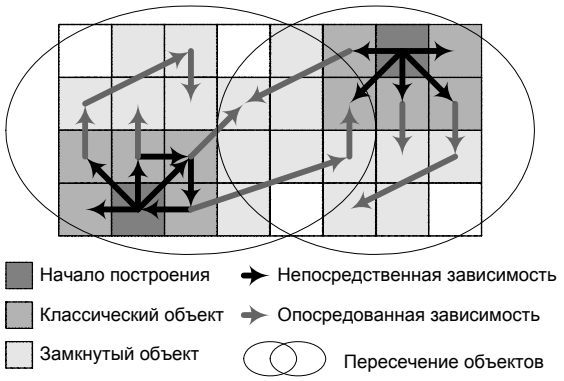

Рис. 1 Объекты в атомарном пространстве

Рассмотрим в данном пространстве байты, представляющие некоторое имя. При произвольном изменении любого из них вся совокупность теряет актуальность: «Алексей» может превратиться в «Олексей», что, по-видимому, не соответствует никакому имени в справочнике. Следовательно, эти байты взаимозависимы и образуют некоторый атрибут базы данных. Конечно, столь детальное рассмотрение не является необходимым в общем случае – элементами атомарного пространства можно сразу взять интуитивно выделяемые атрибуты. Итак, в результате первичного объединения по отношению $a \leftrightarrow b$ образуется набор связанных атрибутов: эти связи наследуются от первоначальных зависимостей байт в атомарном пространстве, а переход к атрибутам вызывает лишь свёртку наиболее тесных из них. Пусть картотека имеет набор атрибутов, указанный на Рис. 2.

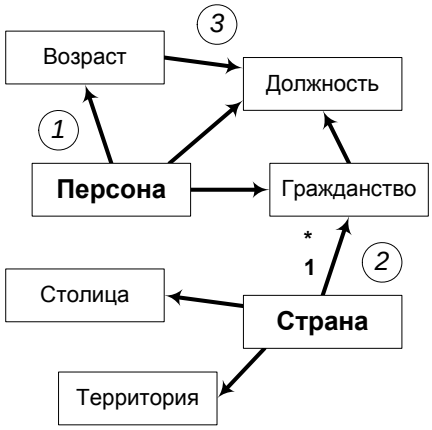

Рис. 2 Анализ зависимостей между атрибутами

Вообще говоря, сами по себе атрибуты (и даже типы байт атомарного пространства) уже являются классами по данному нами определению, хотя это, как правило, – не замкнутые классы. Работа с такими мелкими сущностями в любом случае неудобна, поэтому необходимо продолжить укрупнение, переходя к кортежам. Для этого классифицируем зависимости между атрибутами (Рис. 2): 1 – зависимости полей от первичного ключа, 2 – зависимости между потенциальными объектами и 3 – внутренние зависимости потенциальных объектов. Анализ зависимостей производится следующим образом.

Сперва внимание обращается на непосредственные зависимости и выделяются те элементы данных, от которых зависит множество других: на рисунке это «Персона» и «Страна». Все такие связи, как правило, соответствуют первому типу – зависимость от первичного ключа. Далее выделяются межобъектные связи – такова направленная зависимость полей «Страна» и «Гражданство», соединяющая два подграфа большей степени связности в один граф и более того – имеющая мощность 1:N. Имеется и третий тип зависимостей – это внутриобъектные связи, на рисунке таким образом связаны атрибуты «Возраст» и «Должность» а также «Гражданство» и «Должность». Эти связи здесь возникают на основе закона о госслужбе РФ, в соответствии с которым государственную должность не имеют права занимать иностранные граждане или лица старше 65 лет. От атрибутов теперь легко можно перейти к более крупным классам – на уровень кортежей, либо к классическим, либо к замкнутым объектам, как показано на Рис. 3.

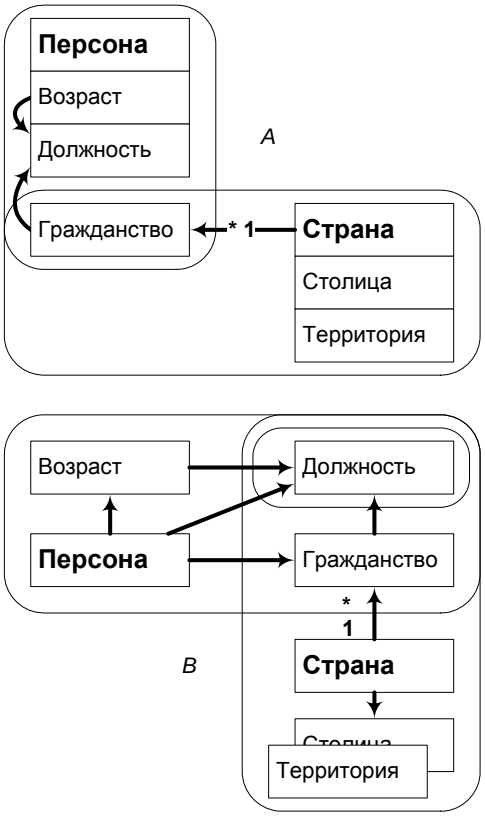

Рис. 3 (A) Классические объекты со связями – 2 шт. (B) Замкнутые объекты – 5 шт.

4. Обеспечение непротиворечивости

В основе метода обеспечения непротиворечивости лежит концепция замкнутых объектов: если обеспечить атомарность операций с замкнутым объектом, то никаких других средств обеспечения непротиворечивости не потребуется. Действительно, по определению замкнутого объекта от него не зависит никакая другая область данных, поэтому его можно рассматривать совершенно изолированно от остальной части базы данных. Однако, использование замкнутых объектов имеет существенное ограничение: при значительной степени связности данных каждый такой объект стремится занять весь объём базы целиком, при этом транзакции вырождаются в последовательные сессии изменения базы данных – параллелизм в системе попросту исчезает.

Эта проблема имеет изящное решение с использованием системы предокрестностей пространства зависимостей R. Пусть имеются две зависимости $a \dashrightarrow c$ и $k \dashrightarrow m$, по определению пространства R они задают некоторые системы замкнутых псевдоокрестностей $U_a = \{O(a, r)\colon \mathrm{d}(a, x) \le r\}$ и $U_k = \{O(k, r)\colon \mathrm{d}(k, x) \le r\}$. Имеет место следующая

Теорема 1. В непротиворечивой базе данных любые две транзакции обрабатывают либо сходящиеся, либо не пересекающиеся направленности зависимостей данных.

Доказательство. Допустим, после одной из общих точек j две транзакции t_1 и t_2 разошлись, то есть в обрабатываемых данных появились несовпадающие точки. Это означает, что замкнутая псевдоокрестность точки j для этих двух транзакций выглядит по-разному: $O_{t1}(j, 1) \ne O_{t2}(j, 1)$. По определению пространства пересечение замкнутых множеств также является замкнутым множеством: $O_{t1}(j, 1) \cap O_{t2}(j, 1) = O_{sub}(j, 1)$. Поскольку $O(j, 1)$ по определению – минимальная замкнутая псевдоокрестность точки j, не равная ей, то $O_{sub}(j, 1) = j$. Следовательно, направленности должны расходиться абсолютно – области непосредственных зависимостей от точки j для разных транзакций не могут пересекаться. Тогда замкнутая псевдоокрестность $O_{t1}(j, 1)$ одной транзакции не может являться замкнутой для другой при любой конфигурации последующих зависимостей в направленностях, поскольку содержит саму точку j, а по предположению – $O_{t1}(j, 1)$ не является замкнутой псевдоокрестностью j с точки зрения t_2. Поэтому транзакции работают в разных пространствах зависимостей R_1 и R_2, а значит – меняют данные противоречивым образом. Ч.т.д.

Следствие. Невозможны наличие петель и дивергенция зависимостей: если один элемент имеет несколько зависимых, то все они входят в любую направленность, включающую данный элемент.

С использованием псевдоокрестностей пространства R достаточно просто предусмотреть средства синхронизации при усечении замкнутых объектов до обычных. Рассмотрим поднаправленность $j \dashrightarrow m$ направленности $a \dashrightarrow m$ зависимостей данных, которую меняет некоторая транзакция t_1. Новое значение элемента j может быть обусловлено только двумя факторами: внешними входными данными и текущими значениями элементов, от которых j зависит непосредственно, включая значение самого элемента j: $\{x_i\} \subset (a \dashrightarrow j)$. Перейдём к объектам. Направленность элементов $a \dashrightarrow m$ может быть преобразована в направленность объектов путём объединения некоторых элементов между собой и с другими независимыми элементами R. Для простоты в обозначениях не будем учитывать выпадения элементов и сохраним запись: $\{x_i\} \subset (a \dashrightarrow j)$, подразумевая объекты в качестве элементов. Объекты, входящие в такую направленность будут перекрываться – по крайней мере, все главные будут перекрывать непосредственно зависимые объекты.

Теперь проблему непротиворечивости можно свести к задаче обеспечения целостности состояния объектов – никакой не разделяющийся на независимые части объект не должен быть изменён в транзакции лишь частично; транзакция должна гарантировать целостное состояние каждого из объектов, что при наличии перекрытий обеспечит непротиворечивое изменение всей направленности. Действительно, если объекты $\{x_i\}$ зависят в свою очередь от множества объектов $\{p_i\}$ и некоторые из них были изменены транзакцией t_2, то транзакция t_2 обязана проверить непосредственно зависимые объекты на предмет потери адекватности данных – просто по определению отношения обусловленности, в противном случае такая транзакция меняла бы базу данных противоречивым образом. Следовательно, если новые значения $\{p_i\}$ не допускают старых значений $\{x_i\}$, то транзакция t_2 обязана изменить $\{x_i\}$, чтобы привести базу данных в согласованное состояние. И так далее – пока объекты не придут в согласованное состояние, либо пока транзакция не дойдёт до последних элементов направленности, от которых уже ничего не зависит. А поскольку дивергенция направленностей невозможна, то такие изменения не могут обойти ни один из зависимых объектов направленности – коль скоро меняется объект j, будут проверены и все зависимые от него объекты.

Теорема 2. Для обеспечения непротиворечивости базы данных необходимо и достаточно установить блокировку на непосредственно участвующие в транзакции объекты.

Доказательство. Необходимость доказывается просто – если не установить блокировку на объекте, то другая транзакция t_2 сможет его поменять несовместным образом и при подтверждении t_1 объект будет содержать противоречивые данные.

Докажем достаточность. Пусть объект j был изменён t_1 и на нём была установлена блокировка. Тогда, если значения подчинённых объектов $\{y_i\}$ не согласуются с текущими изменениями объекта j, то транзакция t_1 должна была рекуррентно поменять значения и далее по иерархии. И если где-то ниже встретится блокировка транзакции t_2, то операция не сможет завершиться и транзакция t_1 откатится или будет вынуждена ждать. Если же значения $\{y_i\}$

согласуются с текущими изменениями объекта j, то тогда подчинённые им значения $\{s_i\}$ транзакция ни проверять ни менять не обязана и блокировки ниже по направленности обнаружены не будут. Однако, объекты $\{s_i\}$ не зависят от j непосредственно, поэтому при его изменении не могут потерять адекватность, если адекватными являются промежуточные объекты $\{y_i\}$. Аналогично – если транзакция t_3 производит изменения выше по направленности, она должна обнаружить блокировку объекта j транзакцией t_1 или же не должна повлиять на элемент j вовсе. Те же самые рассуждения справедливы и для случая чтения данных – транзакция должна установить блокировку на объекты, использованные ею в своей работе. Ч.т.д.

Следствие. Для того, чтобы избежать ситуации взаимной блокировки (deadlock), достаточно обеспечить одну точку входа в циклы $x \rightarrow y$ из любой направленности, включающей данные циклы. Это можно обеспечить, например, если до обращения к «неправильной» точке входа искусственно установить блокировки на все элементы цикла между «правильной» и «неправильной» точками входа, а затем по мере прохождения цикла снимать те из них, что не соответствуют изменению данных. Процесс согласования тогда нельзя прерывать до прохождения всего цикла, даже если текущий элемент согласован с предыдущим.

В ряде случаев применение коммутативных операций может помочь избежать блокировок. Например, пусть в базе данных учёта платежей имеется агрегирующий атрибут *total*, содержащий сумму всех платежей *amount*, проведённых на настоящий момент. Если производить агрегацию на клиенте по формуле *new_total* = *old_total* + *amount* и использовать некоммутативный оператор обновления атрибута *total* = *new_total*, то блокировка будет необходима, если же вместо этого использовать коммутативный оператор *total* += *amount* агрегации на сервере, то блокировка не потребуется – достаточно будет кратковременно захватить атрибут *total* на время подтверждении распределённой транзакции. Коммутативные операции нечувствительны к порядку их выполнения, поэтому значение *total* не может быть испорчено.

5. Пассивный кэш клиента

Проблема синхронизации кэша возникает при последовательной работе транзакций с данными на компьютере клиента или на промежуточных серверах приложений, когда размер замкнутых объектов предметной области достаточно велик. В таком случае имеется дилемма: либо загружать замкнутые объекты в кэш целиком, что может потребовать неприемлемо больших объёмов памяти, либо подгружать только нужные данные из моментального снимка базы данных на момент старта транзакции, но это, в свою очередь, требует сохранения транзакции открытой всё время работы с объектом, приводя к большому количеству «длинных» транзакций в системе. В большинстве случаев ни один из этих двух вариантов не является хорошим решением. Можно предложить ещё один способ работы с кэшем, допускающий хранение в нём данных различной степени актуальности. Будем исходить из следующей модели функционирования кэша:

- Клиентский кэш представляет собой прослойку между приложением и сервером базы данных; каждый клиент имеет свой собственный кэш.
- Между объектами отсутствуют неизвестные системе неявные зависимости: если зависимость существует, ей должна соответствовать связь в самом графе или в метаданных.
- Объекты подгружаются из базы данных в кэш по требованию приложения-клиента в коротких транзакциях с уровнем изоляции «моментальный снимок» (snapshot), по завершении загрузки порции данных транзакция завершается.
- Если при загрузке объекта выясняется, что версии главных или зависимых элементов устарели, то система выполняет рекуррентное обновление данных с генераций событий для приложения-клиента.
- Изменённый в кэше объект не может быть перезагружен в автоматической операции обновления данных до момента его сохранения в базе данных.
- Любой объект хранится в кэше только в одном экземпляре; многоверсионность или блокировки не обеспечивается, поэтому конкурирующие транзакции на одном клиенте невозможны (это не отменяет конкуренцию между клиентами).

Последнее условие вводится для упрощения доказательства и будет отменено – *параллельный кэш оказывается возможным* (что открывает возможность его использования на серверах приложений). Генерация событий необходима для того, чтобы приложение-клиент смогло обновить внутренние переменные, связанные с обновляемыми объектами или являющиеся агрегациями значений этих объектов. Альтернативой генерации событий является откат транзакции с обновлением или удалением из кэша устаревших данных, вызвавших конфликт. Для предлагаемой модели имеет место следующая

Теорема 3. В кэше графовой базы данных, хранящем модифицируемые, устаревающие данные с явными связями, при корректной работе конкурирующих транзакций с уровнем изоляции snapshot в количестве не более одной на клиента невозможно возникновение противоречий данных или deadlock-подобной блокировки изменёнными объектами процесса автоматического обновления данных.

Доказательство. Условие последовательного выполнения транзакций на каждом клиенте гарантирует, что в то же самое время в одном кэше не могут существовать изменения более чем одной транзакции, поскольку изменения предыдущих транзакций данного клиента либо подтверждены, либо откачены. Но само по себе это не обеспечивает отсутствие тупиков обновления и непротиворечивость данных, поскольку кэш содержит данные различной степени

актуальности, которые могут менять другие клиенты системы.

Рассмотрим некоторую направленность $a \dashrightarrow m$, допустим текущая транзакция t_1 изменила в ней объект j. При этом, в зависимости от структуры данных, транзакции могло понадобиться просмотреть часть предыдущих объектов направленности: $g \dashrightarrow j$. Если часть этих объектов уже находилась в кэше на момент старта t_1, то при чтении недостающих элементов версии объектов могут не совпасть, что можно обнаружить при наличии метки транзакции, поместившей объекты в кэш – если считываемые элементы были изменены позже. При несовпадении версий область устаревших данных рекуррентно обновляется, при этом для всех устаревших объектов генерируются соответствующие события.

Для доказательства теперь достаточно показать, что среди устаревших данных не может встретиться изменённая запись, такая что не приводила бы к противоречию. Вернёмся к поднаправленности $g \dashrightarrow j$. Поскольку по условию теоремы между объектами отсутствуют неявные зависимости, то для того, чтобы добраться от объекта g к j или обратно, транзакция должна прочитать все элементы какой-либо из всевозможных подпоследовательностей $g \ldots j$, входящих в поднаправленность $g \dashrightarrow j$. То есть, область чтения $g \dashrightarrow j$ является непрерывной в пространстве зависимостей R. Однако, может существовать множество различных путей из g в j, поэтому при загрузке какой-то одной подпоследовательности $g \ldots j$ не все объекты поднаправленности $g \dashrightarrow j$ могут оказаться загруженными в кэш.

Допустим теперь, что при первом изменении объекта j использовались устаревшие данные кэша, а теперь та же самая транзакция t_1 собирается изменить объект k, доступаясь к нему по пути $p \ldots k$ поднаправленности $p \dashrightarrow k$ направленности $s \dashrightarrow m$, который ещё не был загружен в кэш. Пусть в этом пути встретился объект, который был обновлён позже помещения в кэш элементов первой последовательности $g \ldots j$. Тогда, если направленности $p \dashrightarrow k$ и $g \dashrightarrow j$ не имеют общих точек, то соответствующие области пространства R не пересекаются и не зависят друг от друга, поэтому их совместное использование не может вызвать тупиков или противоречий. При этом полные направленности $a \dashrightarrow m$ и $s \dashrightarrow m$ могут иметь общие точки, тогда при каскадном изменении связанных объектов данный вариант сведётся к следующему.

Если направленности $p \dashrightarrow k$ и $g \dashrightarrow j$ имеют общие точки, то в процессе чтения последовательности $p \ldots k$ может потребоваться обновление объектов, одновременно входящих в первую последовательность устаревших элементов $g \ldots j$. Если при этом потребуется обновление изменённого объекта j, это будет означать, что транзакция t_1 изменила значение j в кэше без учёта его последних изменений в базе данных – поскольку об изменениях кэша базе данных ничего не известно, невозможно предотвратить изменение объекта j конкурирующими транзакциями. Следовательно, изменённое значение j следует считать противоречивым; необходимо выполнить откат t_1 и рекуррентно удалить или обновить в кэше элементы, вызвавшие конфликт, включая сам объект j. Связность областей чтения транзакции в пространстве R при этом гарантирует, что необходимость обновления элементов кэша всегда будет обнаружена – в отсутствие непрерывности последовательностей необходимость совпадения версий соседних элементов ниоткуда не следует.

Допустим теперь обратную ситуацию, пусть первая последовательность $g \ldots j$ является более свежей, чем вторая последовательность $p \ldots k$. Такая ситуация возможна только в том случае, если эти последовательности не пересекаются, иначе их общий элемент должен содержать свежие данные, а по свойству непрерывности – и вся вторая последовательность. Остаётся только упомянуть, что более свежие данные кэша не обязаны содержать самые последние версии объектов базы данных, но это и не требуется по условиям теоремы – механизмы принудительного обновления данных реализуются отдельно и не входят в конфликт с условиями теоремы. На этом мы рассмотрели все возможные варианты конфликтов обновления кэша. Ч.т.д.

Как видно из доказательства теоремы, условие последовательного выполнения транзакций может быть снято, если кэш обеспечивает хранение множества версий одного объекта и элементарные средства синхронизации доступа к кэшу конкурирующих процессов. Видоизменённый принцип работы параллельного кэша может быть следующим. Когда транзакция t_1 считывает объекты из кэша и проверяет согласованность их версий, она игнорирует значения элементов кэша, модифицированных в базе данных позже старта самой t_1 (они могут оказаться в кэше из-за более поздних транзакций). Также она игнорирует все элементы, которые изменены локально в самом кэше и пока не подтверждены конкурирующими транзакциями. Когда значение элемента меняется транзакцией t_1, она должна создать его версию в кэше с такой меткой, чтобы никакие транзакции кроме неё не видели этих изменений до момента подтверждения t_1.

Параллельный многоверсионный кэш может быть применён не только на клиенте, но и на промежуточных серверах распределённой системы.

6. Архитектура распределённой системы

Пассивный принцип функционирования кэша в отсутствие событийных механизмов синхронизации его состояния с изменениями базы данных хорошо согласуется с принятым в [7] подходом к организации BASE транзакций на основе моментального снимка многоверсионной базы данных на некоторый прошлый момент времени. Там была предложена архитектура системы на основе многоверсионной модели данных и распределённых досок объявлений, используемых для синхронизации конкурирующих транзакций в распределённой среде: на

базовом уровне – это совокупность накопителей для хранения информации и досок объявлений для синхронизации процессов. Доски объявлений распределялись по узлам системы, и каждый объект логически приписывался к одной или нескольким доскам. Сами объекты при этом могли иметь произвольное число копий и свободно путешествовать по узлам распределённой системы. При изменении объекта по доске объявлений производилась проверка конфликта и одновременно делалась пометка о совершённом изменении. Чтение объектов реализовывалось посредством организации моментального снимка, или среза многоверсионной базы данных (snapshot) с использованием меток времени (multiversion timestamp ordering) [8]. При этом архитектура позволяла реализовать как слабую, BASE, так и строгую, ACID непротиворечивость – в последнем случае синхронизация по доскам объявлений была необходима также и при чтении объектов.

Однако, в распределённой среде организация моментального снимка сама требует поддержания актуальности данных по всем серверам через механизмы репликации – если срез делается на момент времени после репликации, транзакция не увидит часть подтверждённых на тот момент изменений и система упорядочения по временным меткам даст сбой, в базе данных при этом может возникнуть противоречие. Но если обязать всех клиентов работать с базой данных только посредством кэша, то требования к репликации можно существенно ослабить: момент старта транзакции уже не обязательно ограничивать временем последней репликации данных, и даже сама репликация отчасти может осуществляться в ходе выполнения транзакций.

При таком подходе отдельный узел распределённой системы на чтение становится простым хранилищем данных пассивного кэша в его параллельной модификации: отдельный узел хранит только те многоверсионные данные и на тот момент времени, которые были запрошены кэшами клиентов. Синхронизация записи данных по доскам объявлений при этом сохраняется в неизменном виде, а репликация становится полезным, но не обязательным инструментом обновления устаревших данных. Непротиворечивость данных соблюдается во всех рассматриваемых здесь случаях в обязательном порядке, хотя в режимах работы на принципах BASE изменения данных становятся доступны по всей системе не сразу, а по прошествии некоторого времени. Непротиворечивость обеспечивается здесь не за счёт сильной степени синхронизации, а с помощью специальных методов организации транзакций (при работе без кэша) или применения пассивного кэша.

Таким образом, архитектура системы на основе распределённых досок объявлений позволяет реализовать три разных модели непротиворечивой работы с данными, которые мы сейчас и рассмотрим.

6.1 ACID непротиворечивость

Приложение-клиент, имеющее кэш, соединяется к ближайшим к нему сервером распределённой системы, этот сервер будем называть локальным для данного клиента, а все остальные серверы системы – удалёнными.

Все запросы клиента к данным поступают на локальный сервер, который сперва пробует найти информацию на своём носителе, а в случае неудачи – обращается к серверу приписки объекта, который всегда имеет любую необходимую его версию. Если копия берётся локально, то всё равно происходит обращение к серверу приписки объекта с целью проверки актуальности локальной версии по доске объявлений, и если локальная копия устарела, она обновляется и передаётся клиенту.

При записи объекта по соответствующей доске объявлений производится проверка отсутствия конфликта с конкурирующими транзакциями, после чего данные передаются на узел приписки объекта, а также сохраняются локально. Вставка и удаление объекта производятся подобным образом, причём удаление происходит по правилам многоверсионных баз данных – ранее стартовавшие транзакции должны получать свою копию объекта, существовавшую до удаления.

Процесс асинхронной репликации сводится к удалению устаревших локальных копий данных для предотвращения разбухания базы данных. Альтернативой асинхронной репликации является синхронное распространение изменений объекта в распределённой системе при завершении транзакции. Тогда проверка версий при чтении объектов становится ненужной, но при этом существенно страдает надёжность системы – в случае отказа любого из серверов системы изменение данных становится невозможным.

Кэш на приложении-клиенте работает в синхронном режиме: перед началом транзакции он полностью очищается, в процессе работы используется режим изоляции транзакции snapshot, гарантирующий неизменность состояния базы данных с точки зрения текущей транзакции. При изменении объекта возможны два режима работы: либо изменения передаются на сервер немедленно при их совершении, либо передача откладывается до момента завершения транзакции и тогда все изменения передаются одним пакетом. Первый способ имеет некоторое преимущество, поскольку при пакетном изменении не производится регистрация изменений объектов на досках объявлений в процессе выполнения транзакции, из-за чего короткие транзакции, проведённые за время работы длинных, могут помешать подтверждению длинных транзакций и вызвать их откат.

6.2 BASE непротиворечивость

Компоненты распределённой системы остаются теми же, что и в случае ACID, видоизменяются только принципы работы подсистем.

Непротиворечивость работы с данными обеспечивается за счёт введения задержки между изменением информации и проявлением изменений в системе, необходимой для того, чтобы произошёл про-

цесс репликации объекта по всем серверам распределённой системы, содержащим его версии. Этот механизм гарантирует, что подтверждённые данные будут видны всем клиентам распределённой системы сразу или не видны никому, что и позволяет обеспечить непротиворечивость работы с базой данных. При этом могут возникать дополнительные конфликты обновления объектов, когда конкурирующая транзакция пытается изменить объект, который уже был изменён другой завершившейся транзакцией, но изменения ещё не проявились – конкурирующая транзакция просто не имеет права получить последнюю подтверждённую версию объекта до истечения времени проведения репликации. Однако, эти конфликты мало чем отличаются от случая одновременного изменения объекта конкурирующими транзакциями, когда одна из них вынуждена откатиться или ждать завершения другой, поэтому они не представляют особой проблемы.

Таким образом, при такой модели доступа к данным, транзакция не может стартовать с меткой времени позже последней репликации текущего состояния распределённой базы данных. То есть, метка времени старта транзакции должна содержать более раннее значение, отстоящее от текущего на интервал, необходимый для того, чтобы любое изменение объекта достигло всех заинтересованных серверов системы. Например, если репликация данных производится каждые 5 минут, то метка времени старта транзакции должна содержать текущее время минус эти 5 минут.

Алгоритмы работы подсистем тогда видоизменяются следующим образом. После поступления запроса клиента на локальный сервер, тот ищет информацию на своём носителе, а при отсутствии локальной копии – берёт необходимую версию с удалённого узла приписки объекта. При выдаче локальной копии обращение к доске объявлений на удалённом узле приписки объекта уже не производится – непротиворечивость обеспечивается за счёт сдвига момента старта транзакции и репликации данных за этот интервал времени. Запись объекта, его создание и удаление производятся по тем же правилам, что и в предыдущей модели ACID непротиворечивости.

К процессу асинхронной репликации предъявляются жёсткие требования по максимальному времени проведения изменений – если данные не будут синхронизированы за предусмотренный интервал времени, то транзакции могут считать противоречивое состояние базы данных. Этого можно избежать, если организовать процесс репликации с передачей маркера по кольцу серверов распределённой системы – до прихода маркера старт новых транзакций необходимо запретить или, по крайней мере, ограничить метку времени старта транзакций моментом отправления данного маркера в кольцо.

Кэш на приложении-клиенте работает в синхронном режиме, точно так же, что и в модели ACID: перед началом транзакции он полностью очищается, в процессе работы используется режим изоляции транзакции snapshot, изменения также могут отправляться на сервер немедленно либо пакетом по завершении транзакции.

6.3 BASE с пассивным кэшем

Компоненты распределённой системы остаются теми же, что и в двух предыдущих случаях, видоизменяются только перечисленные ниже принципы работы подсистем.

Кэш на приложении-клиенте перестаёт очищаться перед стартом транзакций и работает в асинхронном режиме в рамках предложенной модели: теперь он содержит данные различной степени актуальности. В случае наличия конфликта кэш клиента запрашивает более новые версии данных, которые берутся с узлов приписки объектов, сохраняются локально и передаются кэшу. Здесь можно применить дополнительную оптимизацию, добавив в запрос идентификатор необходимой версии: если данная версия была обновлена локально по требованию параллельных транзакций того же сервера, запрос к удалённому узлу можно не производить.

В данной модели транзакция может стартовать с произвольной меткой времени, вплоть до текущего времени системы. Изменения данных, тем не менее, могут быть не видны транзакции до тех пор, пока не образуется конфликта несовпадения версий объектов или пока не пройдёт процесс репликации. В этом отношении модель с пассивным кэшем похожа на предыдущую модель BASE непротиворечивости.

Чтение объектов из базы данных в кэш происходит по требованию приложения-клиента в коротких транзакциях с уровнем изоляции snapshot, по завершении загрузки порции данных транзакция завершается. Запись объектов осуществляется в отдельной транзакции точно по тем же правилам, что и в модели ACID непротиворечивости.

Процесс репликации в системе полезен, но не необходим и может быть сведён к удалению устаревших локальных копий данных. При этом периодическая чистка устаревающих данных полезна также и в кэше клиента – при этом сокращается объём используемой памяти и, что более важно, уменьшается количество конфликтов при обновлении информации, вызванных использованием неактуальных копий объектов.

7. Алгоритмы и оценка эффективности

Можно предложить следующие алгоритмы работы пассивного кэша, эффективность работы которых и будет оцениваться далее (для простоты будем рассматривать только вариант кэша без многоверсионности).

7.1 Алгоритм выборки данных из кэша

1. Клиент в рамках распределённой транзакции запрашивает у кэша необходимую ему часть данных (актуальных на время старта распределённой транзакции t_{ds}).

2. Если необходимые данные находятся в кэше, то они немедленно возвращаются клиенту – КОНЕЦ.
3. Производится старт технической транзакции с сервером в режиме snapshot и обращение к серверу на выборку классического объекта *a* (обычно – в состоянии на момент времени t_{ds}).
4. Полученный объект *a* помещается в кэш, при этом у него заполняются два служебных поля: метка времени изменения данных t_{ch} (приходит с сервера) и метка времени начала транзакции, поместившей их в кэш – t_{tr} (это текущая техническая транзакция).
5. Выполняется проверка зависимых объектов (отношение $a \to b$ или $a \leftrightarrow b$): если $t_{ch}(a) \le t_{tr}(b)$, то техническая транзакция завершается и данные возвращаются клиенту (здесь возможна оптимизация: если помечать объекты, прочитанные или изменённые текущей распределённой транзакцией, то остальные можно будет не проверять) – КОНЕЦ.
6. Если устаревший объект *b* помечен как изменённый текущей распределённой транзакцией, то техническая транзакция откатывается, а клиенту возвращается флаг *ошибки* распределённой транзакции – КОНЕЦ.
7. Если зарегистрирован обработчик события устаревших данных, то объект *b* подгружается с сервера заново и производится вызов этого обработчика с параметрами *b_old* и *b_new*, после чего происходит переход к шагу 4 с объектом *b*.
8. Если обработчик события не зарегистрирован, то производится удаление устаревшего объекта *b* из кэша, техническая транзакция откатывается, а клиенту возвращается флаг *ошибки* распределённой транзакции – КОНЕЦ.
9. Техническая транзакция завершается и данные возвращаются клиенту – КОНЕЦ.

7.2 Алгоритм изменения данных кэша

1. Клиент в рамках распределённой транзакции информирует кэш о добавлении, изменении или удалении данных.
2. Если при добавлении нового объекта ему требуется идентификатор, кэш запрашивает значение соответствующего счётчика с сервера и обновляет ключевое поле объекта.
3. Кэш добавляет или обновляет локальную копию объекта и помечает её как добавленную, изменённую или удалённую текущей распределённой транзакцией t_{ds}.

7.3 Алгоритм отката транзакции

1. Клиент уведомляет кэш об откате распределённой транзакции.
2. Кэш удаляет локальные копии всех объектов, изменённых в рамках данной распределенной транзакции – КОНЕЦ.

7.4 Алгоритм подтверждения транзакции

1. Клиент уведомляет кэш о подтверждении распределённой транзакции.
2. Производится старт технической транзакции с сервером, все изменённые в текущей распределённой транзакции объекты передаются на сервер и производится попытка завершения технической транзакции.
3. В случае ошибки подтверждения технической транзакции она откатывается, а клиенту возвращается флаг *ошибки* распределённой транзакции – КОНЕЦ.
4. Клиенту возвращается флаг успешного завершения распределённой транзакции – КОНЕЦ.

7.5 Оценка эффективности алгоритмов

Пусть общее число объектов базы данных равно *N*, кэш вмещает *M* объектов, а количество связанных объектов в типичном кластере, вовлекаемом в транзакцию, равно *K*. Пусть каждый период времени *T* происходит изменение половины из имеющихся объектов базы данных (аналог периода полураспада). Вероятность встретить в кэше устаревший кластер объектов будет пропорциональна *K* и обратно пропорциональна *T*. Влияние параметра *M* двояко – с одной стороны, небольшой кэш с необходимостью обновляется часто, а с другой – вместительный кэш может содержать практически всю базу данных и не обращаться к ней в течение длительного времени, пока не возникнет конфликт обновления информации. В любом случае, вероятность зависит от предыстории использования кэша, поэтому можно дать лишь приближённую оценку. По формуле вероятности появления хотя бы одного из независимых событий вероятность изменения произвольного кластера объектов:

$$P(t) \sim 1 - \prod q_i \text{ (произведение } 1 \ldots K),$$

где q_i – вероятность неизменности объекта:

$$q_i(t) = 1 - p_i(t),$$

где p_i – вероятность по закону полураспада:

$$p_i(t) = (N - N_t) / N = 1 - 2^{-t/T},$$

окончательно получаем для кластера объектов:

$$P(t) \sim 1 - 2^{-tK/T}.$$

Эта функция растёт довольно быстро – на начальном этапе с достаточной точностью её можно аппроксимировать линейной функцией времени, причём скорость роста существенно зависит от сложности обрабатываемых кластеров. Так, при $K = 10$ и $T = 1$ час вероятность встретить устаревший кластер спустя 1 минуту после полного обновления кэша составляет 11%, а при усложнении кластеров в 2 раза ($K = 20$) – почти 21%. Хотя характерное время *T* реальных баз обычно значительно выше.

Полученная вероятность *P(t)* встретить устаревший кластер позволяет оценить синхронизационные издержки кэша – затраты на передачу данных с сервера и их повторную обработку в текущей распределённой транзакции. Если характерное время выполнения распределённых транзакций существенно меньше интервала *T*, такое что *P(t)* за время

транзакции возрастает незначительно, то кэш будет работать эффективно – однажды загруженные данные смогут использоваться повторно другими транзакциями.

Традиционные кэши в режиме ACID не допускают смешение данных различной степени актуальности, поэтому они должны либо полностью очищаться по завершении каждой транзакции, либо иметь средства синхронизации данных с актуальным состоянием базы данных. В первом случае применение кэша теряет всякий смысл – данные передаются заново для каждой новой транзакции, а во втором – распределённая система теряет в масштабируемости из-за возникновения синхронизационных издержек с оценкой *O(n)* по количеству клиентов *n*. Эти издержки обусловлены необходимостью оперативной передачи изменений сразу всем кэшам всех клиентов, работающих с теми же данными. Системы на основе пассивного кэша, очевидно, лишены этого недостатка, поскольку не требуют немедленной синхронизации данных. Поэтому более корректно сравнивать их с аналогами, работающими на принципах BASE.

Наиболее удобной для сравнения является архитектура сервиса Amazon Dynamo, процедуру разрешения конфликтов в котором должно обеспечить приложение-клиент при чтении данных. Этот подход в общем допускает хранение в клиентском кэше данных различной степени актуальности, поскольку клиент имеет набор формальных правил для проверки непротиворечивости используемой информации (опустим вопрос, используется ли эта возможность реально – всё же сам сервис ориентирован на работу со слабо связанными данными). Принцип работы такого кэша тоже является пассивным – необходимость обновления данных выявляется в процессе проверки их непротиворечивости, поэтому динамические характеристики системы на его основе будут аналогичны. Основное отличие предлагаемого подхода состоит в том, что наличие заданных априори формальных правил валидации здесь не требуется. Это упрощает алгоритмы работы клиента и весьма важно при использовании кэша на промежуточных серверах распределённой системы, где универсального набора правил, справедливых одновременно для всех возможных клиентов, может не существовать.

Заключение

Применение пассивного кэша на стороне клиента приводит к существенному ослаблению требований к синхронизации процессов в распределённой системе. Непротиворечивость работы с данными при этом гарантируется, а выигрыш происходит за счёт другого ресурса – изменения становятся видны в системе не сразу, а по истечении некоторого времени. При использовании пассивного кэша эта задержка не является фиксированной, а определяется исключительно интенсивностью работы с кэшем и скоростью его обновления данными сервера. За это распределённая система практически избавляется от необходимости синхронизации данных своих серверов – этот процесс происходит автоматически при запросе кэшами необходимых им данных, что существенно повышает надёжность и масштабируемость системы в целом.

Полученные результаты используются в проекте глобального ресурса знаний для системы извлечения информации из текстов ИСИДА-Т.

Литература

[1] Brewer E.A. Towards robust distributed systems (abstract) // Proc. of the 19th Annual ACM Symposium on Principles of Distributed Computing, Portland, Oregon, United States, 2000, July 16 – 19. – P. 7.

[2] Таненбаум Э., ван Стеен М. Распределённые системы. Принципы и парадигмы. – СПб.: Питер, 2003. – ISBN: 5-272-00053-6.

[3] Кутателадзе С.С. Основы функционального анализа. – 3-е изд., испр. – Новосибирск: Изд-во Ин-та математики, 2000. – xii+336 с. (Современная математика – студентам и аспирантам).

[4] Largeron C., Bonnevay S. A pretopological approach for structural analysis // Information Sciences. – 2002. – V. 144, July. – P. 169-185.

[5] Келли Дж. Общая топология: Пер. с англ. – М.: Наука, 1968. – 385 с.

[6] Энгелькинг Р. Общая топология: Пер. с англ. – М.: Мир, 1986. – 752 с.

[7] Демидов А.А. Проектирование распределённых систем обработки объектных структур данных. // Труды XII Всероссийской научной конференции RCDL'2010. – Казань. Казанский университет, 2010. – С. 441-447.

[8] Чардин П. Многоверсионность данных и управление параллельными транзакциями // «Открытые системы». – 2005. – № 1. – С. 64-69.

Towards Structure of Cache for Distributed Graph Database

© A.A. Demidov

This paper is devoted to the problem of choosing structure of cache for distributed graph database, which would provide consistent reading and writing data and would minimize synchronization overhead at the same time. To meet the requirements a method of passive data synchronization on version conflict is applied. The given solution allows to mix data of various actuality within one cache and to process them in a consistent way without synchronization overhead.

♣ Работа выполнена по программе фундаментальных исследований Президиума РАН №3, проект «Высокопроизводительные масштабируемые средства работы с фактографическими базами большого объёма».

[1] Направленность есть пара *(S, ≥)*, где *S* – функция и *≥* – направление на её области определения [5, с.95].

[2] В терминологии работы [4].