Системы управления и моделирование

ACI-транзакции в имитационном моделировании сложных систем*

А. А. Демидов

Аннотация. Предложен способ согласования версий разделяемых данных в распределённой системе, не требующий синхронизации конкурирующих процессов. В отличие от существующих подходов ACI-транзакции обеспечивают масштабируемость системы за счёт ослабления гарантий сохранности подтверждённых изменений (свойство Durability) при сохранении всех остальных свойств ACID-транзакций, в том числе — гарантий непротиворечивости.

Ключевые слова: *кластер, имитационное моделирование, транзакции, синхронизация, согласованность, непротиворечивость.*

Введение

Пусть конкуренция в системе устроена так, что каждый процесс считывает все разделяемые данные в локальную память, вносит необходимые изменения, и затем записывает эти данные целиком обратно в общую память. Если операции чтения и записи общей памяти атомарны, то есть мгновенны для конкурирующих процессов, то система всё время будет оставаться в непротиворечивом состоянии. Но изменение этого состояния будет почти последовательным, из многих совместных историй будет выбран один вариант, остальные результаты будут отброшены по принципу «последний забирает всё»: если конкурирующая транзакция завершилась после того, как текущая транзакция прочитала общие данные, то все её изменения будут затёрты результатами текущей транзакции, но если до завершения текущей транзакции результаты той транзакции успеет прочитать третья транзакция, завершающаяся после текущей, то развитие получит конкурирующая версия данных, где результаты текущей транзакции не отражены.

Этот подход лежит в основе ACI-транзакций, не гарантирующих ACID-свойство Durability — долговечность произведённых изменений. Оказывается, атомарность операций с общей памятью можно реализовать с помощью версий без привлечения синхронизации, а распараллеливаемость зависит от самих данных и не является принципиальным ограничением метода — с этой целью вводятся так называемые «открытые объекты», позволяющие определить области данных, которые могут изменяться параллельно и независимо друг от друга. Отбрасывание части результатов позволяет обойтись без синхронизации процессов, но оно же ограничивает применение ACI-транзакций, которые не могут использоваться в задачах, требующих соответствия состояния системы некоторому реальному положению дел.

* Работа выполнена в рамках проекта Президиума РАН № 16 по программе фундаментальных исследований, проекта РФФИ 12–07–00533-а и НИР регистрационный № 01201354592.

Пример 1. Пусть в интернет-магазине два покупателя A и B одновременно пытаются оплатить последнюю единицу некоторого товара. В реальной ситуации товар достанется кому-то одному из них, а второй абсолютно точно останется ни с чем. Если же мы только тестируем эту систему, нам совершенно не важно, кому конкретно, A или B, достанется этот товар. Один процесс временно может считать, что товар достался покупателю A, а второй — что покупателю B. ACI-транзакции гарантируют, что когда данные различных процессов будут объединяться в общее состояние, то часть конфликтующих данных будет отброшена таким образом, что останется только один из двух вариантов: «товар достался покупателю A» или «товар достался покупателю B».

В класс задач, где ACID-транзакции могут быть заменены ACI-транзакциями, входят многие задачи имитационного моделирования, хотя не везде применение ACI-транзакций будет эффективным. Клеточные автоматы, сети Петри являются естественными сферами применения ACI-транзакций. В общем случае любая модель с дискретными событиями состоит из сети взаимосвязанных очередей [1, § 18.4.1], а сети Петри [2] являются формальным языком описания таких структур.

1. Постановка задачи

Пусть разделяемые данные системы распределены между процессами, возможно — с пересечениями, образуя их локальные данные. Чтобы получить доступ к данным друг друга, процессы обмениваются сообщениями. Акт посылки или получения сообщения называется событием. Следующие друг за другом события одного процесса вполне упорядочены, события разных процессов могут быть упорядочены лишь частично: посылка сообщения предшествует его получению. Все события, частично упорядоченные посредством сообщений, образуют пространство событий системы, или её историю. Состояние разделяемых данных описывается с помощью понятия глобального состояния, которое формируется динамически и может быть представлено в виде сечения — подмножества пространства событий, включающего наряду с каждым своим элементом и все предшествующие ему [3].

Глобальное состояние считается непротиворечивым, если удовлетворяет всем известным ограничениям целостности [4, § 14.1]. Если каждый процесс в отдельности меняет глобальное состояние непротиворечивым образом, то совокупность конкурирующих процессов будет оставлять это состояние непротиворечивым при условии, что результат совместного выполнения процессов эквивалентен результату некоторого последовательного их выполнения [4, § 15.6]. Формально непротиворечивость определяется как свойство глобального состояния, состоящее в сохранении причинности: если событие включено в сечение, то любое влияющее на него событие также должно быть включено в это сечение [5]. Противоречивое глобальное состояние может содержать несовместные данные, такие, совокупность которых не отвечает никакой осмысленной модели предметной области.

Через каждую точку пространства событий системы можно провести множество различных сечений, поэтому непротиворечивое глобальное состояние, вообще говоря, не единственно. Это создаёт проблему, когда необходимо получить значение глобального свойства (Global Predicate Evaluation, GPE), например: завершились ли вычисления, не возникла ли ситуация взаимной блокировки и т. п. Значение динамично изменяющегося глобального свойства зависит от сечения, для которого оно получено, поэтому если всем процессам необходимо получить одинаковое значение, то следует назначить одно, одинаковое для всех процессов выделенное непротиворечивое глобальное состояние. Обычно для этой цели выбирается сечение, соответствующее времени идеализированного внешнего наблюдателя, а соответствующее выделенное непротиворечивое глобальное состояние называется просто — состоянием системы.

С ростом числа процессов возрастает количество сообщений, которыми они вынуждены обмениваться для получения различных частей разделяемых данных. В результате производительность системы упирается в свой потолок и почти перестаёт увеличиваться при добавлении новых вычислительных мощностей. Это составляет суть проблемы масштабируемости.

Кэширование, то есть размещение нескольких копий данных ближе к использующим их процессам, позволяет решить эту проблему: если необходимые данные уже содержатся в локальном кэше процесса, то нет необходимости создавать сообщения для получения их у соседей. При этом возникает новая проблема поддержания кэшей в когерентном состоянии, то есть обеспечения согласованности копий одних и тех же данных, тождественности их между собой. При условии, что ни один из процессов не производит изменений, все они могут использовать разделяемые данные совместно, не рискуя привести систему в противоречивое состояние. Но как только хотя бы один процесс изменит локальные данные в своём кэше, копии тех же данных в других кэшах *мгновенно* окажутся устаревшими. Одновременное использование свежих и устаревших данных с большой вероятностью приведёт к противоречию. Масштабируемость системы напрямую зависит от

того, каким образом обеспечивается эта согласованность: если количество сообщений для поддержания согласованности сравнимо с тем, которое было сэкономлено в результате кэширования, то эффект от этого кэширования будет нулевым.

В общем случае это представляет серьёзную и, скорее всего, неразрешимую проблему: по всей видимости, имеются объективные ограничения масштабируемости систем — одной из формулировок этих ограничений является CAP-тезис Брюера [6]. Однако можно попытаться отыскать эффективные частные решения для ограниченного класса задач. Парадигма ACI-транзакций как раз и позволяет эффективно обойти эту проблему в рамках класса задач имитационного моделирования, не требующих наличия в системе выделенного непротиворечивого глобального состояния (редко меняющиеся свойства, такие как признак завершения вычислений, не нуждаются в выделенном состоянии, поскольку оказываются одинаковыми для всех сечений после своего изменения).

2. Состояние исследований

Ограничения масштабируемости распределённых систем часто связывают с CAP-тезисом Брюера, утверждающим недостижимость одновременно трёх свойств: постоянной готовности, согласованности данных и устойчивости к отказам [7]. Из-за изначально нестрогой формулировки CAP-тезиса возникло множество его интерпретаций, часто небесспорных. Так, часто путают понятия согласованности и непротиворечивости, отождествляя свойство Consistency CAP-тезиса и ACID-транзакций. Такое смешение понятий приводит к тому, что для лучшей масштабируемости пытаются ослабить гарантии непротиворечивости. Обычно это производится одним из двух способов, условно объединяемых в рамках парадигмы NoSQL [8]. Во-первых, путём логического разделения данных на небольшие независимые блоки, каждый из которых в отдельности обрабатывается атомарно, но транзакции с несколькими блоками не поддерживаются — таковы хранилища данных «ключ-значение». Во-вторых, на основе принципов BASE транзакций, где непротиворечивость достигается лишь со временем, а ответственность за разрешение конфликтов возлагается на приложение.

Между тем, CAP-тезис Брюера вынуждает жертвовать согласованностью информации, но не её непротиворечивостью. Поэтому должны существовать системы, не подпадающие под действие CAP-тезиса, которые допускали бы рассогласованность копий данных, но при этом гарантировали непротиворечивость глобального состояния. Анализ источников не выявил исследований в данном направлении. Наиболее близкие идеи содержатся в концепциях транзакционной памяти (Transactional Memory, TM [9]) и хранилищ данных «ключ-значение».

Транзакционная память контролирует всю прочитанную транзакцией область данных на предмет отсутствия изменений конкурирующими транзакциями. Имеются модификации данного метода с отложенной проверкой, когда допускается временное существование противоречий в базе данных [10]. Отличие ACI-транзакций состоит в стратегии работы с прочитанными элементами: здесь транзакция всегда завершается, а вместо контроля изменений все прочитанные ею элементы перезаписываются вместе с изменёнными элементами с новой меткой времени, образуя одну непротиворечивую окрестность.

Хранилища данных «ключ-значение» разделяют базу данных на независимые области данных — значения, прямым наследником которых являются предлагаемые открытые объекты. Поскольку разные значения совершенно не связаны, то меняющие их транзакции не конфликтуют между собой. ACI-транзакции накладывают меньше ограничений на структуру хранимых данных: открытые объекты могут пересекаться, что даёт возможность обрабатывать взаимозависимые области данных со связями. Своеобразной платой за это является потеря транзакциями свойства D — долговечности произведённых изменений.

3. Существо идеи

Непротиворечивость глобального состояния (как одного из возможных сечений пространства событий) не может быть сохранена, если имеет место хотя бы одно из следующих условий:

Каждый процесс имеет возможность свободно обращаться к любому другому процессу системы — тогда возможен конфликт версий данных разных процессов.

Существует реальная система, отражением которой является глобальное состояние — тогда устаревшие версии могут конфликтовать с актуальным состоянием дел.

Оба случая объединяет одно допущение, что реальная система и вычислительные процессы существуют в мире с мгновенными коммуникациями, а согласование версий происходит в отдельном мире с конечной скоростью распространения сигналов. Фактически, здесь происходит смешение двух несовместимых физических концепций: дальнодействия в идеальном пространстве Аристотеля $R \times T$ и близкодействия в реальном пространстве Минковского $R \times iT$. Такая вольность не всегда допустима: реальная система функционирует в рамках физических законов, накладывающих абсолютные ограничения на скорость распространения сигналов.

Тогда, может быть, следует устранить это недоразумение и отказаться от физически абсурдного дальнодействия? Это — здравая идея, однако оказывается, что выделенное непротиворечивое глобальное состояние самим своим существованием обязано допущению мгновенных коммуникаций. Такое состояние по определению должно быть одинаковым для всех процессов, и любые его изменения одним процессом мгновенно должны становиться доступны остальным процессам системы. Это известная физическая проблема — допущение мгновенных коммуникаций влечёт появление абсолютной системы отсчёта. Понятие текущего состояния оказывается сверхидеализацией, не реализуемой никаким реальным вычислительным процессом.

Другими словами, никакой реальный нелокальный динамический процесс, строго говоря, не может быть описан с помощью абстракции текущего состояния. Вместо этого необходимо рассматривать весь конгломерат событий системы в пространстве-времени, взаимоотношение которых и определяет её эволюцию на самом деле. Любое непротиворечивое сечение пространства событий может выступать в качестве «состояния системы», но дело в том, что таких сечений много, и сечения, доступные разным процессам, могут не совпадать. Только по завершении эволюции системы все последующие сечения начнут совпадать друг с другом.

В рассматриваемом классе задач имитационного моделирования не существует выделенного глобального состояния как такового, вместо него имеется множество равноправных взаимопересекающихся эволюционирующих сечений. Бессмысленно говорить о мгновенном изменении несуществующего состояния, поэтому копии данных в различных кэшах системы не устаревают сразу после изменения процессом одной из них и не требуют синхронного обновления. Каждый процесс может использовать собственное непротиворечивое глобальное состояние; глобальные состояния, доступные разным процессам, могут отличаться в течение какого-то времени, поэтому мгновенных коммуникаций здесь не требуется.

4. Понятие ACI-транзакций

ACI-транзакциями назовём способ работы с данными, обеспечивающий известные свойства: атомарности (Atomicity), непротиворечивости (Consistency), изолированности (Isolation), но не гарантирующий сохранности подтверждённых изменений (Durability). Отсутствие последнего свойства позволяет конкурирующим ACI-транзакциям работать без откатов: если система работоспособна, то все транзакции всегда завершаются, однако использование подтверждённых изменений в последующих транзакциях не гарантируется, то есть эти изменения могут быть утрачены.

Открытые объекты позволяют определить области данных, которые могут изменяться независимо друг от друга. Чем они мельче, тем более высокого уровня параллелизма на этих данных можно достичь. В вырожденном случае, когда каждый элемент данных может меняться независимо, получается в точности модель NoSQL хранилища данных «ключ-значение».

Эти основные идеи и составляют концепцию ACI-транзакций:

- **Atomicity** — соблюдается на уровне открытых объектов.
- **Consistency** — при операциях над открытыми объектами.
- **Isolation** — конкурирующие транзакции не видят друг друга.

Следует особо отметить, что все эти свойства ACI-транзакций, включая атомарность, могут быть реализованы без синхронизации. Транзакции не будут конфликтовать при записи, если каждая из них создаст собственную версию записываемых данных. Читающие транзакции не будут конфликтовать с пишущими транзакциями, если будут видеть фиксированное состояние базы данных (режим чтения snapshot). Мы рассмотрим даже более эффективный способ реализации атомарности ACI-транзакций, использующий понятие непрерывности операций.

5. Открытые объекты

Выделим в системе предметный слой программного обеспечения, где будем определять объекты данных; совокупность низлежащих программно-аппаратных уровней назовём ядром. Для предметного слоя внутреннее устройство ядра не имеет значения, важен только предоставляемый им программный интерфейс, который обычно обеспечивается движком базы данных или операционной системой.

Определение 1. Операция с данными называется атомарной на текущем уровне, если для остальной части системы она выглядит мгновенной, то есть если во время выполнения этой операции никакая другая операция с участием тех же данных невозможна.

Обозначим G множество элементов данных — таких единиц хранения информации, операции с которыми атомарны на уровне ядра. С точки зрения предметного слоя эти операции будут элементарными, то есть атомарными и неделимыми на субоперации. Чтобы обеспечить атомарность последовательностей элементарных операций, на уровне предмет-

ного слоя вводятся средства группировки операций, такие как критические секции или транзакции.

Определение 2. Транзакция действует рационально, если при отсутствии конкуренции меняет глобальное состояние непротиворечивым образом, не считывает и не записывает данные без причины.

Будем полагать, что каждая транзакция в отдельности действует рационально. Противоречивое глобальное состояние обычно содержит несовместные значения элементов данных, совокупность которых не отвечает никакой осмысленной модели предметной области — допустимые сочетания значений элементов данных определяются ограничениями выбранной модели.

Определение 3. Будем говорить, что элемент данных $a \in G$ ограничивает значения элемента данных $b \in G$ и записывать $a \to b$, если множество допустимых значений элемента b непосредственно зависит от значения, которое имеет элемент a.

Множество пар вида $a \to b$ определяет отношение на множестве G, которое назовём отношением (непосредственной) зависимости элементов данных. Это отношение не обязательно транзитивно:

$$(a \to b) \wedge (b \to c) \not\Rightarrow (a \to c).$$

Элемент данных c может также *опосредованно* зависеть от элемента данных a, если имеет место ситуация $a \to b \to c$. Множество допустимых значений элемента c ограничено значением элемента b, но поскольку это значение само принадлежит множеству, ограниченному значением элемента a, то элемент a ограничивает и значения элемента c посредством элемента b. Такое отношение является транзитивным замыканием отношения непосредственной зависимости, его учитывать не нужно.

Каждая транзакция реализует некоторый алгоритм, поэтому она в курсе того, какие элементы данных потенциально могут ограничивать значение изменяемого ею элемента, в частности, предыдущее значение элемента может ограничивать следующее его значение: $a \to a$. Чтобы узнать, какие из этих ограничений актуальны, транзакция должна поинтересоваться текущими значениями ограничивающих элементов. Как правило, это означает, что она должна прочитать значения этих элементов, хотя некоторые из них могут оказаться вычислимыми на основе других. Поэтому зависимости элемента данных могут быть найдены на основе того, какие элементы необходимо прочитать или вычислить перед его изменением.

Пример 2. Пусть внешний ключ реляционной таблицы $\mathbf{B}$ ссылается на первичный ключ таблицы $\mathbf{A}$ и существуют записи $a \in \mathbf{A}$ и $b \in \mathbf{B}$. Элементами данных здесь являются записи, поскольку операции с ними атомарны в базах данных. Поля первичного ключа записи a менять нельзя, иначе нарушится ссылочная целостность. Поля внешнего ключа записи b можно изменять, сослав на любую другую запись таблицы $\mathbf{A}$. Поэтому $b \to a$. С другой стороны, при изменении ключевых полей записи b необходимо проверить, существует или нет запись $a' \in \mathbf{A}$, на которую устанавливается ссылка. Поэтому $a' \to b$.

Данный пример показывает, что отношение зависимости определено на отдельных элементах (записях или объектах), а не на структурах концептуальной модели (таблицах или классах). На структурах данных индуцируется более грубое отношение $\mathbf{A} \leftrightarrow \mathbf{B}$ взаимной зависимости, более того, единая концептуальная модель существует не всегда: изменение элементов данных может повлечь изменение концептуальной модели, но эти изменения будут не мгновенными и не одновременными для разных процессов системы.

В каждой точке $g \in G$ определим базис $\pi(g)$ фильтра $\tau(g)$ [11, § 1.3] из единственного множества, составленного из элементов данных, ограничивающих значения элемента g (при этом полагаем $g \to g$):

$$\pi(g) = \{\{x \in G : x \to g\}\},$$
$$\tau(g) = \operatorname{fil}\pi(g).$$

Определение 4. Произвольное подмножество $S \in \tau(g)$ называется предокрестностью точки $g \in G$.

Определение 5. Множество называется открытым, если оно является предокрестностью каждой из своих точек. Множество называется замкнутым, если его дополнение открыто.

Определение 6. Предокрестность точки, содержащая открытое множество, называется её окрестностью. Окрестность точки, совпадающая с открытым (замкнутым) множеством, называется открытой (замкнутой) окрестностью соответственно.

Отображение $\tau : G \to 2^{2^G}$ определяет предтопологию, а совокупность открытых множеств — топологию на множестве G.

Пример 3. Пусть $G = \{a, b, c\}$ и $a \to b$, $a \to c$. Базисом фильтра в точке a будет система множеств $\mathcal{B}_a = \{\{a\}\}$,

в точке b — система множеств $\mathcal{B}_b = \{\{a,b\}\}$,

в точке c — система множеств $\mathcal{B}_c = \{\{a,c\}\}$.

Тогда фильтрами в точках a, b и c будут системы множеств

$$\mathcal{F}_a = \{\{a\},\{a,b\},\{a,c\},\{a,b,c\}\},$$
$$\mathcal{F}_b = \{\{a,b\},\{a,b,c\}\},$$
$$\mathcal{F}_c = \{\{a,c\},\{a,b,c\}\}.$$

Открытыми здесь являются множества

$$\mathcal{O} = \{\varnothing,\{a\},\{a,b\},\{a,c\},\{a,b,c\}\}.$$

Совокупность систем множеств $\mathcal{F}_a$, $\mathcal{F}_b$, $\mathcal{F}_c$ определяет предтопологию, а система множеств $\mathcal{O}$ — топологию на G.

Определение 7. Открытую (замкнутую) окрестность, атомарность операций чтения и записи которой обеспечена, будем называть открытым (замкнутым) объектом соответственно.

Открытые объекты примечательны тем, что совершенно не зависят от состояния остальной части данных. В свою очередь, от замкнутых объектов в остальной части данных тоже ничего не зависит.

Поскольку открытый объект не зависит от состояния остальной части данных, то достаточно обеспечить атомарность операций с таким объектом, чтобы обеспечить непротиворечивость любых операций с разделяемыми данными, не меняющих топологию. Действительно, если элемент b зависит от элемента a, транзакция не вправе менять значение элемента b, не поинтересовавшись значением элемента a. Если транзакция не считывает значение элемента a, то она либо вычисляет его — и тогда это значение не должно меняться конкурирующими транзакциями, либо рискует привести систему в противоречивое состояние. То, что другие данные зависят от открытого объекта, не представляет проблемы — рациональная транзакция сама производит все необходимые изменения зависимых объектов.

Замечание 1. Реализация произвольных операций, в том числе — меняющих топологию, возможна с использованием свойства непрерывности и будет рассмотрена далее.

6. Предобъекты и непротиворечивость

Пусть, по-прежнему, G есть множество элементов данных, операции с которыми атомарны на уровне ядра, а транзакции действуют рационально, не считывают и не записывают данные без причины.

Определение 8. Предокрестность точки $g \in G$, атомарность операций чтения и записи которой обеспечена, будем называть предобъектом с атрибутом g.

В силу определения 5 каждый предобъект содержится в охватывающем открытом объекте, поэтому предобъекты мельче открытых объектов, что снижает вероятность коллизий конкурирующих транзакций и положительно сказывается на динамике системы. Но в отличие от открытых объектов, которые определяют топологию, предобъекты определяют лишь предтопологию [12, 13, 15] и лишены многих полезных свойств открытых объектов. Предобъекты не являются независимыми от остальной части данных, однако их использование всё равно оказывается полезным.

Пример 4. Пусть транзакция меняет значение элемента данных c открытого объекта $a \to b \to c$. Тогда она должна поинтересоваться значением ограничивающего его элемента b, то есть прочитать элемент b. Если старое значение элемента b не конфликтует с новым значением элемента c, то считывать значение элемента a нет необходимости — оно ни на что не влияет. Здесь $b \to c$ и есть предобъект.

Пусть A — множество допустимых значений элементов данных, тогда A^G — множество возможных глобальных состояний системы. Оператор $\varphi: A^G \to A^G$ изменения глобального состояния может, вообще говоря, менять зависимости элементов данных. Поэтому формально можно было бы рассматривать отношение зависимости на множестве пар $A \times G$: при разных значениях элемента данных он может ограничивать значения разных элементов (пример 2), — однако это усложнило бы изложение. Равно можно считать, что предтопология на G не постоянна: являясь проекцией индуцированной предтопологии глобального состояния $w \in A^G$, $w \subset A \times G$, она меняется при изменении этого состояния оператором φ. Действие $\varphi(w)$ оператора φ на состояние w непрерывно, если не усиливает предтопологию на G. Разложим оператор φ в композицию $\varphi = \sigma^{-1}\delta\sigma$ операторов: чтения элементов данных в локальную память, изменения атрибута предобъекта и записи элементов обратно на сервер.

Утверждение 1. Каков бы ни был оператор δ изменения атрибута, считывающий значения ограничивающих элементов данных явно, сохранить непротиворечивость глобального состояния

невозможно, если действие оператора чтения σ^{-1} не является непрерывным.

Доказательство. Отображение f одного топологического пространства в другое непрерывно, если образ фильтра произвольной точки x тоньше, чем фильтр точки $f(x)$ [11, § 9.2].

Если оператор σ^{-1} не является непрерывным, то прочитанные окрестности оказываются мельче исходных, и это значит, что часть непосредственных зависимостей при чтении теряется (например, из-за малого размера локального буфера). Оператор δ не может гарантировать непротиворечивость при изменении атрибута, поскольку не имеет в своём распоряжении полного набора ограничений. ■

Определение 9. Чтение данных непрерывно, если сохраняет предокрестности, то есть каждая предокрестность при чтении имеет вид, какой она имела бы после записи в некоторой транзакции.

Утверждение 2. Непрерывность чтения предобъектов гарантирует непротиворечивость изменения глобального состояния.

Доказательство. Обозначим $U_S = \bigcup_{s \in S} \pi(s)$ минимальный предобъект, включающий множество $S \subset G$. Множества A и B элементов данных, изменяемые двумя разными транзакциями, могут:
совпадать: $A = B$;
пересекаться: $A \cap B \neq \varnothing$;
касаться: $(U_A \cap B \neq \varnothing) \vee (A \cap U_B \neq \varnothing)$;
быть отделены: $(U_A \cap B = \varnothing) \wedge (A \cap U_B = \varnothing)$.

Если множества отделены или, с учётом атомарности операций, совпадают, то предобъекты U_A и U_B независимы друг от друга. Это справедливо даже в случае изменения зависимостей, когда предтопологии, в которых построены U_A и U_B, не совпадают между собой.

Действительно, обозначим U'_A и U'_B минимальные предобъекты, которые включали бы A и B в конкурирующей предтопологии. Допустим, что U_A и U_B в разных предтопологиях отделены, а U'_A и U_B в одной предтопологии касаются друг друга: $U'_A \cap B \neq \varnothing$. Последнее означает, что в результате второй транзакции образовались зависимости $B \to A$. Первая транзакция при изменении A не проверяла содержимое элементов B, так как $U_A \cap B = \varnothing$; она могла лишь вычислить эти значения на основе прочитанных ею элементов U_A. Поскольку вторая транзакция не меняла элементы U_A, то их старые значения согласуются с новыми значениями B. Но раз первая транзакция не проверяла явно элементы B, то она не может знать точно, какое значение — старое или новое — они имеют. Поэтому её изменения должны быть согласованы с обоими вариантами. Случай старта транзакций из несогласованных состояний, когда прочитанные значения в U_A и U_B несовместны, нарушает условие непрерывности.

Если множества пересекаются или касаются, тогда предобъекты U_A и U_B содержат зависимые элементы данных, новые значения которых могут оказаться несовместны между собой. Чтобы избежать возникновения противоречия, необходимо искусственно дополнить первоначальными значениями элементов каждый из зависимых предобъектов до объединения $A \cup B$ — после чего ситуация сводится к случаю, когда множества A и B совпадают. ■

Если система допускает совместное хранение нескольких равноправных версий, например, с помощью локальных кэшей процессов или временных меток [15], то пишущие транзакции не конфликтуют между собой, так как множества оказываются отделены. При этом может возникать неоднозначность: транзакции будет видна та из версий, которую она начнёт считывать первой. Противоречия же можно избежать, если не допускать смешения элементов конкурирующих версий в одной транзакции.

Замечание 2. Идеи версионности реализуются в понятии фрагментов, которые позволяют осуществлять старт конкурирующих транзакций из несогласованных состояний и окончательно избавляют от необходимости синхронизации процессов.

7. Сборка предобъектов из фрагментов

Непрерывность чтения не требует, чтобы пересекающиеся предокрестности были изменены в одной транзакции — она предоставляет гораздо больше свободы, чем кажется на первый взгляд.

Определение 10. Предокрестность, все элементы которой сохранены в одной транзакции, будем называть фрагментом (подмножество фрагмента, являющееся предокрестностью, — фрагмент).

Утверждение 3. Реализация непрерывности чтения фрагментов без поддержки версий невозможна.

Доказательство. Если версии не поддерживаются, то любое изменение элемента данных конкурирующей транзакцией автоматически исключает его из одного фрагмента и включает в другой. ■

Чтобы обеспечить непрерывность чтения данных, наряду с изменёнными элементами данных необходимо сохранять в версии все прочитанные транзакцией элементы в предположении, что они ограничивают значения изменённых элементов. Тогда читающая транзакция сможет определить, стыкуются ли значения в пересекающихся фрагментах в одно непротиворечивое сечение или нет.

Замечание 3. Транзакция сама обозначает зависимости, считывая значения ограничивающих элементов при внесении изменений. Зависимости также могут быть обозначены явно с помощью связей — это позволяет сократить размер фрагментов за счёт устранения балласта из опосредованных зависимостей.

В ряде случаев транзакция вычисляет значение ограничивающего элемента данных на основе значений других элементов, так что даже при наличии зависимости $a \to b$ реального считывания значения элемента a не происходит. Часто для этого используется ограничение $b \to b$, когда текущее значение элемента ограничивает варианты своего изменения. Неявное получение значений требует сохранения начального значения всех прочитанных транзакцией элементов данных, в том числе и тех, которые затем были ею изменены.

Пример 5. Вычисляются многие значения. Так, в примере 2 зависимость $\mathbf{A} \to \mathbf{B}$ имеет мощность 1:1, в то время как обратная зависимость $\mathbf{B} \to \mathbf{A}$ — N:1, поскольку при изменении полей первичного ключа записи из $\mathbf{A}$ необходим поиск связанных записей в $\mathbf{B}$, для чего просматривается вся таблица $\mathbf{B}$. Но если для ускорения поиска использовать индекс, то можно прочитать только связанные записи таблицы $\mathbf{B}$, а все остальные считать не связанными.

Читающая транзакция всегда пытается выбрать самые последние доступные ей версии элементов данных, но она жёстко ограничена необходимостью непрерывного чтения предобъектов, поэтому в ряде случаев вынуждена в зависимости от выбранной стратегии работы с данными либо отбрасывать неподходящие фрагменты, либо инициировать откат.

В ситуации на рис. 1a транзакцией будет выбрана верхняя, самая последняя версия элемента b. Если эта транзакция затем изменит считанное значение, то при её подтверждении необходимо будет сохранить оба значения: начальное и изменённое, — чтобы обозначить зависимость $b \to b$. Если транзакция не считывала предварительно значение элемента b, а сразу подала запрос на его изменение, то сохранять начальное значение не нужно.

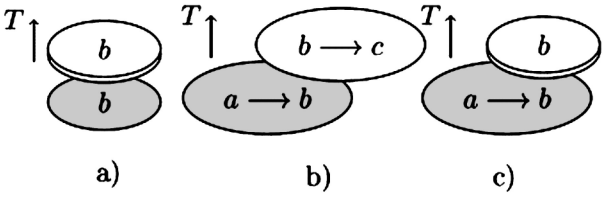

Рис. 1. Варианты пересечения фрагментов

Типичная процедура сборки предобъекта представлена на рис. 1b. Нижний фрагмент был создан транзакцией, прочитавшей значение элемента a и изменившей значение элемента b, а верхний — транзакцией, прочитавшей значение элемента b и изменившей значение элемента c. В итоге значение элемента b фрагмента $a \to b$ оказывается равным значению этого элемента фрагмента $b \to c$, что позволяет читающей транзакции объединить фрагменты в один предобъект $a \to b \to c$, не разрывая предокрестностей.

Необходимость сохранения всех прочитанных значений демонстрирует рис. 1c. Верхняя версия элемента b была сохранена без считывания значения элемента a, которое было вычислено на основе начального значения элемента b. Пусть теперь транзакция начала чтение фрагмента $a \to b$ с элемента a и собирается прочитать значение элемента b. Если начальное значение элемента b не было сохранено изменившей его транзакцией, то невозможно определить наличие зависимости $b \to b$; невозможно даже сказать, является ли верхняя версия элемента b более поздней, чем нижняя — сборка фрагментов оказывается невозможной. Поэтому начальное значение изменяемого элемента b необходимо сохранять.

Замечание 4. Версии необходимо хранить ограниченное время, пока существуют заинтересованные транзакции — после того, как определённая версия данных станет доступна всем процессам системы и все более старые транзакции будут завершены, более ранние версии этих данных можно будет безопасно удалить.

Фрагменты позволяют организовать совместное хранение нескольких вариантов зависимостей элементов данных с разной предтопологией, что даёт возможность преодолеть ограничение утверждения 2, не позволяющее использовать непрерывность для разрешения конфликта транзакций, стартовавших из несогласованных состояний. Пополнение пересекающихся или касающихся фрагментов при записи не требуется — конкурирующие версии могут храниться совместно.

Фрагменты позволяют окончательно отказаться от синхронизации конкурирующих процессов в пользу отложенного асинхронного согласования версий разделяемых данных.

Пишущие транзакции порождают фрагменты, частично упорядоченное множество которых образует пространство событий системы. Читающие транзакции собирают фрагменты в предобъекты по непрерывности, причём восстанавливаемое таким образом глобальное состояние может не совпадать с глобальным состоянием, в разрезе которого эти фрагменты были созданы. Фрагменты выступают в роли посредника, своеобразного буфера между конкурирующими транзакциями, осуществляющего развязку и позволяющего им работать независимо друг от друга — синхронизация процессов заменяется асинхронным согласованием версий данных.

Непрерывность чтения фрагментов гарантирует восстановление непротиворечивого глобального состояния. Поскольку каждый фрагмент был создан транзакцией в разрезе некоторого состояния, то все необходимые для сборки фрагменты присутствуют в системе, задача состоит в том, чтобы избежать ситуации, когда ни один из фрагментов не может быть добавлен к восстанавливаемому глобальному состоянию без потери непротиворечивости.

При восстановлении глобального состояния читающей транзакцией фрагменты, созданные разными процессами в разрезе разных глобальных состояний, могут смешиваться друг с другом, если это не нарушает непрерывность их чтения. Так реализуется принцип объединения совместных историй процессов. При этом важно, с одной стороны, не допускать объединения несовместных версий в одно состояние, что привело бы к возникновению противоречия, а с другой — избегать полного разделения, изоляции глобальных состояний, что привело бы к неконтролируемому росту количества вариаций и падению эффективности системы в целом.

8. Пассивный локальный кэш

ACI-транзакции позволяют построить систему без синхронизации, а в качестве локальной памяти процессов применить пассивный кэш.

Условие **C1** (раздел 3) не допускает множественных связей между процессами без использования механизма согласования версий. Поэтому любой процесс в системе, в том числе — клиентский, если он имеет возможность обращаться к нескольким конкурирующим процессам, должен осуществлять коммуникации только через посредство кэша, который является своего рода прослойкой между процессом и остальной частью системы и умеет разрешать коллизии при согласовании версий получаемых данных.

В отличие от активного кэша, обновляющего содержащиеся в нём копии разделяемых данных всякий раз по приходу события об изменении этих данных конкурирующими процессами, пассивный кэш не проявляет собственной инициативы вообще. Всякий раз, когда локальный процесс запрашивает отсутствующие в кэше данные, пассивный кэш перенаправляет запрос другим процессам и сохраняет копии полученных данных.

Если при этом нарушается непрерывность и данные в кэше не стыкуются в непротиворечивое сечение, значит, локальные копии данных устарели и должны быть удалены, а вызвавшая конфликт транзакция — отменена. С точки зрения приложения принцип работы с разделяемыми данными одинаков, независимо от того, на базе какой системы — с активным или пассивным кэшем — оно выполняется. В обоих случаях начало и конец группы операций с разделяемыми данными помечаются специальными инструкциями, образуя критическую секцию или транзакцию, позволяющую подсистеме кэша организовать безопасный доступ к этим данным.

Заключение

Представлена новая парадигма управления конкурирующими процессами в частном случае задач имитационного моделирования, не требующих наличия выделенного непротиворечивого глобального состояния. Изложена концепция ACI-транзакций и пассивного кэша, которые позволяют обеспечить масштабируемость системы без необходимости жертвовать свойством непротиворечивости.

Предтопологические методы находят своё применение в структурном анализе [16–20], однако в области параллельной обработки данных подобные подходы ранее не применялись.

В настоящее время на основе предложенной методики ведётся разработка высокопроизводительной базы знаний для системы извлечения информации из текстов ИСИДА-Т [21]. Структура организации знаний представляет собой направленный граф, вершинами которого выступают понятия предметной области, а дугами — выявленные отношения между ними. Навигация между вершинами осуществляется посредством прохождения по рёбрам графа. Преобъектом в данном случае является модифицируемая вершина вместе с теми своими непосредственными соседями, которые были прочитаны в той же транзакции перед произведённым изменением.

Литература

1. *Таха Х. А.* Введение в исследование операций. 6-е изд.: Пер. с англ. М.: Издательский дом «Вильямс», 2001.
2. *Котов В. Е.* Сети Петри. М.: Наука, 1984.
3. *Lamport L.* Time, clocks, and the ordering of events in a distributed system // Communications of the ACM, 1978, July, Vol. 21, No 7, 558–565.
4. *Дейт К. Дж.* Введение в системы баз данных. 7-е изд.: Пер. с англ. М.: Издательский дом «Вильямс», 2001.
5. *Babao O., Marzullo K.* Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms // Distributed Systems. Frontier Series // New York: ACM Press, 1993, Vol. 4, 55–96.
6. *Brewer E. A.* Towards robust distributed systems // PODC'00 Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing, 2000.
7. *Gilbert S., Lynch N. A.* Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services // SIGACT News, 2002. Vol. 33, No 2, 51–59.
8. *Abadi D., Thomson A.* The problems with ACID, and how to fix them without NoSQL // DBMS Musings, 2010.
9. *Guerraoui R., Kapalka M.* The Theory of Transactional Memory // Bulletin of the EATCS, 2009, February, No 97, 84–105.
10. *Знаменский С. В.* На пути к новой теории информационных систем // Электронные библиотеки: перспективные методы и технологии, электронные коллекции // Труды XIV Всероссийской научной конференции RCDL'2012. — Переславль-Залесский: Изд-во Университет города Переславля, 2012, с. 265–273.
11. *Кутателадзе С. С.* Основы функционального анализа. Современная математика — студентам и аспирантам. 4-е изд., испр. Новосибирск: Изд-во Ин-та математики, 2001.
12. *Kent D. C., Min W. K.* Neighborhood spaces // International Journal of Mathematics and Mathematical Sciences, 2002, Vol. 32, No 7, 387–399.
13. *Rath N.* Action of convergence groups // Topology Proceedings, 2003, Vol. 27, No 2, 601–612.
14. *Rath N.* Applications of convergence groups // International Journal of Pure and Applied Mathematics, 2008, Vol. 48, No 3, 415–433.
15. *Bernstein P. A., Goodman N.* Multiversion Concurrency Control — Theory and Algorithms // ACM Transactions on Database Systems, 1983, Vol. 8, No 4, 465–483.
16. *Largeron Ch., Bonnevay S. B.* A pretopological approach for structural analysis // Information Sciences, 2002, July, Vol. 144, No 1–4, 169–185.
17. *Stadler B. M. R., Stadler P. F., Shpak M., Wagner G. P.* Recombination Spaces, Metrics, and Pretopologies // Z. Phys. Chem., 2002, Vol. 216, 217–234.
18. *Allam A. A., Bakeir M. Y., Abo-Tabl E. A.* Some methods for generating topologies by relations // Bulletin of the Malaysian Mathematical Sciences Society. Second Series, 2008, Vol. 31, No 1, 35–45.
19. *Levorato V., Bui M.* Modeling the Complex Dynamics of Distributed Communities of the Web with Pretopology // IICS // Proceedings of the 10th International Conference on Innovative Internet Community Services (I2CS). LNI — Bangkok, Thailand: GI, 2010, June 3, Vol. 165, 306–320.
20. *Levorato V.* Modeling Groups in Social Networks // Proceedings of the 25th European Conference on Modelling and Simulation (ECMS'2011). — Krakow, Poland, 2011, June 7, 129–134.
21. *Кормалев Д. А., Куршев Е. П., Сулейманова Е. А., Трофимов И. В.* Технология извлечения информации из текстов, основанная на знаниях // Программные продукты и системы. — 2009. — № 2(86). — С. 62–66.

Демидов Алексей Александрович. М. н. с. ИПС им. А. К. Айламазяна РАН, Исследовательский центр искусственного интеллекта. Окончил в 1966 г. РГАЕА им. П. А. Соловьева. Количество печатных работ: 8. Область научных интересов: искусственный интеллект, параллельные вычисления, системы управления базами данных. E-mail: alex@dem.botik.ru