ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ № 4(8), 2011, с. 139–152

УДК 004.652.8:004.056.2:515.1

А. А. Демидов

Топологические методы в проектировании системы синхронизации конкурирующих транзакций распределённой базы данных

Аннотация. Предлагается особый метод выделения объектов на основе топологического анализа внутренней структуры хранимых данных. Понятие функциональной зависимости обобщается до отношения обусловленности, которое используется далее для определения необходимости синхронизации конкурирующих процессов при доступе к данным в распределённой среде. В формализованной таким образом модели управление транзакциями существенно упрощается.

Ключевые слова и фразы: пассивный кэш, непротиворечивость, распределённая база данных, конкурирующие транзакции, синхронизационные издержки.

Введение

В настоящее время в мире происходит бурное развитие распределённых хранилищ данных сверхбольших объёмов, построенных на базе принципов нереляционных баз данных и условно объединяемых в рамках единой парадигмы NoSQL. Наиболее известны такие системы как Google BigTable, Amazon Dynamo, Apache Hadoop, Oracle Coherence и др. Но традиционные реляционные СУБД при этом не теряют свои позиции, наоборот — на тот же класс задач претендуют графовые СУБД: Neo4j, HyperGraphDB, InfiniteGraph и др. Почему же системы со столь впечатляющими характеристиками по объёмам хранимых данных, количеству обрабатываемых транзакций в секунду, надёжности и доступности сервиса не могут вытеснить классические реляционные и графовые базы данных? Почему в сравнении

Работа проводилась при финансовой поддержке Министерства образования и науки Российской Федерации по программе фундаментальных исследований Президиума РАН № 3, проект «Высоко-производительные масштабируемые средства работы с фактографическими базами большого объёма».

с бесплатным Apache Hadoop предпочтение отдаётся дорогим и менее производительным в абсолютных цифрах серверам на базе СУБД Oracle?

Дело в том, что речь идёт о различных типах транзакций, вернее, различны объекты, над которыми проводятся эти транзакции. Все хранилища обладают такими высокими характеристиками только на несвязанных или чрезвычайно слабо связанных данных, а с ростом сложности структур эффективность их применения существенно падает. Они не поддерживают транзакции над несколькими документами или парами ключ-значение, или же реализуют их недостаточно эффективно, поэтому имеют строго ограниченную область применения и не подходят для обработки сложно структурированной информации.

Графовые СУБД находятся на противоположной стороне спектра решений. Их основное предназначение — обработка сложно и разнообразно организованных, возможно — динамических структур данных. Они способны решать задачи, в которых структура представления данных плохо систематизируема или априори неизвестна. Ряд систем поддерживают распределённые транзакции. Так, в HyperGraphDB для управления транзакциями используется метод STM (Software Transactional Memory), при котором транзакция по завершении проверяет неизменность всех данных, использованных ею в своей работе. Успешно используемый в языке программирования Хаскель, в применении к базам данных метод STM имеет существенный изъян: здесь объём области памяти не фиксирован заранее, а количество переменных увеличивается динамически при добавлении новых записей. Поэтому транзакция не может знать, повлияли бы на её работу записи, добавленные конкурирующими транзакциями за время её выполнения, или нет.

Для точной проверки необходимо повторное выполнение всех запросов на выборку данных, которые производились в данной транзакции: если вновь добавленные записи не попадают в эти выборки, они не повлияли бы на выполнение транзакции, в противном случае такое влияние могло бы иметь место и транзакция должна откатиться. Ситуация ещё более усугубляется тем, что такая проверка должна производиться на неизменном состоянии базы данных и требует приостановки всех конкурирующих транзакций в системе, причём применение многоверсионности здесь не способно спасти ситуацию.

Кроме метода STM используются и другие способы организации транзакций, например, строго последовательное выполнение коротких транзакций с самого начала [1], что устраняет всякую конкуренцию в принципе. Для эффективной работы всё же необходимо отыскать более приемлемые методы организации конкурирующих процессов в распределённой среде. Один из возможных методов исследуется в данной работе.

1. Определения

Обозначим M пространство логически адресуемой серверной памяти распределённой системы, доступной для хранения информации базы данных. Размеры его атомарных элементов (машинных слов, страниц памяти) могут быть фиксированы. Семейство всех подмножеств M обозначим 2^M.

Будем предполагать, что логика системы поддерживается набором правил, описывающих множество процедур согласования. Каждая такая процедура исполняется при любом изменении входных данных и модифицирует выходные данные. Хорошо известная в теории реляционных баз данных функциональная зависимость элементов является частным случаем такой процедуры.

Будем говорить, что имеется направленная обусловленность $a \to b$ элемента b элементом a, если существует процедура согласования, читающая a и пишущая в b. Мы задали отношение направленной обусловленности $\mathcal{R} = \{(x, y) \in M \times M : x \to y\}$. Удобно считать, что она всегда содержит диагональ, т.е $a \to a \quad \forall a$. Множество $S \subset M$ назовём замкнутым если $x \to y \implies y \in S \quad \forall (x \in S, y \in M)$.

Замечание 1.1. *Пересечение $S = \prod\limits_{\alpha \in A} S_\alpha$ любого набора замкнутых множеств S_α замкнуто.*

Доказательство от противного: пусть S не замкнуто. Тогда

$$\exists (x \in S, y \in M) : (x \to y) \wedge \neg (y \in S)$$

и, значит, y не лежит хотя бы в одном из S_α, что противоречит замкнутости последнего.

Замыканием $[S]$ множества S назовём пересечение всех замкнутых подмножеств, его содержащих. Из следствия 1.1 вытекает, что это множество замкнуто и, значит, является наименьшим замкнутым множеством, содержащим S.

Введённый оператор замыкания имеет логическую природу, но удовлетворяет свойствам топологического замыкания [2, 3]

(C1): $[\varnothing] = \varnothing$,
(C2): $\forall A \in 2^M$, $A \subset [A]$,
(C3): $[A \cup B] = [A] \cup [B]$,
(C4): $[[A]] = [A]$.

. Замкнутое множество называется изолированной частью, если дополнение до неё замкнуто. Замкнутое множество называется связным, если любая его непустая изолированная часть совпадает с самим множеством.

Поскольку определение замыкания неконструктивно, то вычисление замыкания представляет нетривиальную задачу, для решения которой вводится оператор предзамыкания [4]

$$\mathcal{F}(S) = \{y \in M : \exists (x \in S) x \to y\},$$

в нашем случае удовлетворяющий (C1-C3).

Замечание 1.2. *Оператор замыкания получается многократным применением оператора предзамыкания*

Доказательство. Если при применении оператора предзамыкания ничего не добавилось, то уже получили замыкание. Поскольку добавляться может не больше раз, чем элементов в M, то процесс повторного применения завершится не позднее, чем через $|M|$ шагов, где M — это мощность множества M.

Значения оператора предзамыкания будем называть объектами. Объекты будем называть независимыми, если их замыкания не пересекаются. Объекты, совпадающие со своим замыканием, назовём замкнутыми.

Пример 1.1. Рассмотрим задачу хранения в базе данных персональной информации по определённому кругу лиц. Пусть каждая персона имеет такие атрибуты как возраст, занимаемая должность, гражданство и т.п.

Пусть И. И. Иванов, 64 лет, являясь гражданином РФ, состоит на госслужбе в мэрии города Переславля. Здесь можно выделить следующие атрибуты и связи:

имя $\to$ возраст,	имя $\to$ гражданство,
имя $\to$ должность,	страна $\to$ гражданство,
возраст $\to$ должность,	гражданство $\to$ должность.

Последние две связи возникают на основе закона о госслужбе РФ, в соответствии с которым государственную должность не имеют права занимать иностранные граждане или лица старше 65 лет. После очередного дня рождения или смены гражданства И. И. Иванов уже не сможет занимать свою должность, поэтому значение «госслужащий» атрибута «должность» потеряет смысл в рамках принятой модели данных и должно быть изменено.

В то же время при переходе И. И. Иванова на другое место работы ни его возраст, ни гражданство изменяться не обязаны — в обратном направлении никаких обусловленностей нет.

От атрибутов далее можно перейти к более крупным объектам — на уровень кортежей (то есть вершин с наборами атрибутов). Замкнутыми объектами здесь будут наборы атрибутов:

«должность», «гражданство–должность»,
«имя–возраст–гражданство–должность»,
«страна–гражданство–должность»,
объект с полным набором атрибутов.

2. Концепция непротиворечивости

В основе предлагаемого метода лежит концепция замкнутых объектов: если обеспечить атомарность операций с замкнутым объектом, то никаких дополнительных средств обеспечения непротиворечивости уже не потребуется. Действительно, по определению замкнутого объекта от него не зависит никакая другая область данных, поэтому его можно рассматривать изолированно от остальной базы данных. На этом базируются высокопроизводительные хранилища данных — документы или пары ключ-значение обычно являются замкнутыми объектами. Отсюда же проистекают и проблемы СУБД данного класса: при значительной степени связности данных применение замкнутых объектов становится невозможным по причине того, что каждый такой объект стремится занять весь объём базы целиком, а управление транзакциями вырождается в строго последовательное их выполнение: параллелизм в системе попросту исчезает.

Теорема 2.1. *Если несколько транзакций одновременно независимо обрабатывают данные, то либо существуют непересекающиеся замкнутые объекты, содержащие записываемые данные, либо существует атомарный элемент, который может быть записан более чем одной транзакцией.*

Доказательство. Если элемента, который может быть записан более чем одной транзакцией, нет, то замыкания множеств элементов, которые могут быть записаны отдельными транзакциями, образуют искомые замкнутые объекты. □

Проблему непротиворечивости можно свести к задаче обеспечения целостности объектов: никакой связный объект не должен быть изменён в транзакции лишь частично. При наличии пересечений это требование обеспечит непротиворечивое изменение всей последовательности объектов, использованных транзакцией. Действительно, если объекты $\{y_i\}$ обусловлены множеством объектов $\{x_i\}$, некоторые из которых были изменены транзакцией t_1, то транзакция t_1 обязана согласовать объекты $\{y_i\}$ этими изменениями по определению отношения обусловленности. В противном случае такая транзакция меняла бы базу данных противоречивым образом. Следовательно, если новые значения $\{x_i\}$ не допускают старых значений $\{y_i\}$, то транзакция t_1 обязана изменить $\{y_i\}$, чтобы привести базу данных в согласованное состояние. Согласование данных должно продолжаться до тех пор, пока объекты не придут в согласованное состояние.

Замечание 2.1. *Для обеспечения непротиворечивости базы данных достаточно установить блокировку на непосредственно участвующие в транзакции объекты, прочитанные или изменённые.*

Доказательство. Пусть объект j был изменён t_1 и на нём была установлена блокировка. Тогда, если значения обусловленных объектов $\{x_i\}$ не согласуются с текущими изменениями объекта j (в частности, добавление или удаление объекта j вызывает такое рассогласование), то транзакция t_1 должна была рекуррентно поменять значения и далее по всем последовательностям, включающим элементы из $\{x_i\}$. Если при этом встретится блокировка конкурирующей транзакции t_2, то операция не сможет завершиться и транзакция t_1 откатится или будет вынуждена ждать. Если же значения $\{x_i\}$ согласуются с текущими изменениями объекта j, то уже обусловленные ими значения $\{y_i\}$ далее по последовательностям транзакция ни проверять ни менять не обязана, и блокировки ниже по последовательностям обнаружены не будут. Однако объекты $\{y_i\}$ не обусловлены непосредственно объектом j, поэтому при его изменении не могут потерять актуальность, если актуальными являются промежуточные объекты $\{x_i\}$.

□

Хорошо известная проблема взаимной блокировки (deadlock) требует введения дополнительных блокировок потенциально подверженных изменениям объектов. Они ресурсозатратны и отчасти избыточны в ситуации, когда фактического изменения данных не происходит.

3. Алгоритмы управления транзакциями

Кроме метода пессимистических блокировок, известны оптимистические методы, позволяющие существенно снизить нагрузку на сервер за счёт отказа от блокировок при чтении данных. Блокировки на чтение позволяют предотвратить конфликты чтение-запись, возникающие при чтении объекта во время его изменения конкурирующей транзакцией. Распространённые методы управления распределёнными транзакциями с учётом CAP-тезиса Брюэра [5] были рассмотрены в работе [6], там же 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) [7], причём предложенная архитектура позволяла реализовать как слабую (BASE), так и строгую (ACID) непротиворечивость — в последнем случае синхронизация по доскам объявлений необходима также и при чтении объектов. Здесь хотелось бы дополнительно остановиться на деталях механизмов обеспечения непротиворечивости и синхронизации кэшей в рамках данной архитектуры — вопросе, который не был тогда изложен достаточно подробно.

Проблема синхронизации кэшей 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. Существует ещё один способ работы с кэшем, допускающий хранение в нём данных различной степени свежести. Будем исходить из следующей модели функционирования кэша:

(1) Клиентский кэш представляет собой прослойку между приложением и сервером базы данных; каждый клиент имеет свой собственный кэш.
(2) Между объектами отсутствуют неизвестные системе неявные обусловленности: если обусловленность существует, ей должна соответствовать связь в самом графе или в метаданных.
(3) Объекты подгружаются из базы данных в кэш по требованию приложения клиента в коротких транзакциях, по завершении загрузки порции данных транзакция завершается.
(4) Если при загрузке объекта выясняется, что версии главных или обусловленных объектов устарели, то система выполняет рекуррентное обновление данных с генерацией событий для приложения-клиента.
(5) Изменённый в кэше объект не может быть перезагружен в автоматической операции обновления данных до момента его сохранения в базе данных.
(6) Любой объект хранится в кэше только в одном экземпляре; многоверсионность или блокировки не обеспечиваются, 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 приложению-клиенту, чтобы оно 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 3.1. *В кэше графовой базы данных, хранящем модифицируемые, устаревающие данные с явными связями, при корректной работе последовательных транзакций с уровнем изоляции snapshot невозможно возникновение противоречий данных или deadlock-подобной блокировки изменёнными объектами процесса автоматического обновления данных.*

Доказательство. Последовательное выполнение транзакций гарантирует, что одновременно в кэше не могут существовать изменения более чем одной транзакции, поскольку изменения предыдущих транзакций либо подтверждены, либо откачены. Но само по себе это не гарантирует отсутствие тупиков обновления и непротиворечивость данных, поскольку кэш содержит данные различной степени свежести.

Допустим, текущая транзакция t_1 изменила объект a_k последовательности зависимых объектов $a_1 ... a_n$. При этом, в зависимости от структуры данных, транзакции могло понадобиться просмотреть часть предыдущих объектов последовательности. Если некоторые из этих объектов уже находилась в кэше на момент старта t_1, то при чтении недостающих элементов версии объектов могут не совпасть. Это 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. Поскольку по условию теоремы между объектами отсутствуют неявные обусловленности, то для того, чтобы добраться от объекта a_i к a_k, транзакция должна прочитать все элементы какой-либо из возможных последовательностей, ведущих из a_i в a_k. Может существовать множество различных путей из a_i в a_k, поэтому при загрузке какой-то одной последовательности

$a_i ... a_k$ некоторые объекты другой последовательности, содержащей объекты a_i и a_k, могут оказаться не загруженными в кэш.

Допустим теперь, что при первом изменении объекта a_k использовались устаревшие данные кэша, а теперь та же самая транзакция t_1 собирается изменить другой объект b_k, доступаясь к нему по пути $b_i ... b_k$, который ещё не был загружен в кэш полностью. Если первая и вторая последовательности не имеют общих элементов, то соответствующие объекты независимы, поэтому их совместное использование не может привести к противоречиям. В противном случае в процессе чтения последовательности $b_i ... b_k$ может потребоваться обновление объектов, одновременно входящих в первую последовательность устаревших элементов $a_i ... a_k$ – в случае изменения их в базе данных конкурирующими транзакциями после чтения первой последовательности.

Если при этом потребуется обновление изменённого объекта a_k, то это будет означать, что транзакция t_1 изменила (и пока не подтвердила) значение a_k в кэше без учёта его последних изменений в базе данных. Поскольку об изменениях кэша самой базе данных ничего не известно, то невозможно предотвратить изменение объекта a_k конкурирующими транзакциями. Следовательно, изменённое значение a_k следует считать противоречивым и необходимо выполнить откат t_1 и рекуррентно удалить или обновить в кэше элементы, вызвавшие конфликт, включая сам объект a_k. Связность зависимых областей чтения транзакции при этом гарантирует, что необходимость обновления элементов кэша всегда будет обнаружена.

Допустим теперь обратную ситуацию: пусть первая последовательность $a_i ... a_k$ является более свежей, чем вторая последовательность $b_i ... b_k$. Такая ситуация возможна только в том случае, если эти последовательности не пересекаются, иначе их общий элемент должен содержать свежие данные, а отсюда — и вся вторая последовательность. Остаётся только упомянуть, что свежие данные кэша не обязаны содержать самые последние актуальные 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. □

Замечание 3.1. *Как видно из доказательства, 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 алгоритм для 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.

Пассивный принцип функционирования кэша в отсутствие событийных механизмов синхронизации его состояния с изменениями базы данных хорошо согласуется с принятым в [6] подходом к организации BASE транзакций на основе моментального снимка многоверсионной базы данных на некоторый прошлый момент времени (конечно, при реализации событий между базой данных и кэшем ACID непротиворечивость тоже возможна: вместо моментального снимка там используется проверка изменений каждого считываемого объекта по доске объявлений).

В распределённой среде сама организация моментального снимка 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. Однако 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, алгоритмы функционирования которого были описаны в Теореме 3.1 и последующем замечании: 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. Репликация становится полезным, но не обязательным инструментом обновления устаревших данных. Например, если каждый узел изначально содержит полную копию всех данных системы, то без принудительной репликации придётся очень долго ждать, пока данные синхронизируются в результате рассогласования версий в транзакциях.

Итак, архитектура системы на основе распределённых досок объявлений позволяет реализовать следующие модели работы с данными:

(1) ACID-непротиворечивость. Процесс репликации сводится к удалению устаревших локальных копий данных для предотвращения разбухания базы данных. Для каждого локально считываемого объекта производится его проверка по доске объявлений на предмет изменений. При наличии изменений или при отсутствии локальной копии данные берутся с удалённого узла приписки, содержащего актуальную версию объекта. При записи объекта производится проверка отсутствия конфликта с конкурирующими транзакциями, данные передаются на узел приписки объекта, а также сохраняются локально.
(2) BASE-непротиворечивость. В системе организуется процесс репликации данных. Транзакция не может стартовать с меткой времени позже последней репликации. Поэтому всегда имеется некоторая задержка проявления изменений данных в системе. Данные считываются локально, а при отсутствии локальной копии — с удалённого узла приписки объекта. Запись объекта осуществляется по тем же правилам, что и в случае ACID.
(3) BASE-непротиворечивость с пассивным кэшем. Процесс репликации в системе полезен, но может быть сведён к удалению устаревших локальных копий данных. 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 случае наличия конфликта кэш клиента запрашивает более

новые версии данных, которые берутся с узлов приписки объектов, сохраняются локально и передаются кэшу. Запись объекта осуществляется по тем же правилам, что и в случае ACID.

4. Заключение

Таким образом, формализация понятия объекта на основе отношения обусловленности вместе с использованием методов топологии для анализа структуры данных позволили выявить интересные закономерности обработки данных в распределённой среде, что помогло понять — какие из ограничений существенны для непротиворечивого функционирования системы, а какие — неважны или могут быть ослаблены. По результатам исследования предложена концепция пассивного кэша, существенно снижающего требования к синхронизации конкурирующих транзакций в распределённой системе. Эта концепция хорошо уживается в разработанной в работе [6] архитектуре системы на основе досок объявлений. Полученные результаты используются в проекте глобального ресурса знаний для системы извлечения информации из текстов ИСИДА-Т.

Список литературы

[1] Abadi D., Thomson A. *The Case for Determinism in Database Systems* // 36th International Conference on Very Large Data Bases, Singapore, 2010, September 13–17 : Proceedings of the VLDB Endowment, 2010 Vol. **3**, no. 1, p. 70–80 ↑[]

[2] Келли Д. Общая топология. Пер. с англ. М. : Наука, 1968. — 385 с. ↑1

[3] Энгелькинг Р. Общая топология. Пер. с англ. М. : Мир, 1986. — 752 с. ↑1

[4] Largeron C., Bonnevay S. *A pretopological approach for structural analysis* // Information Sciences, 2002. **144**, p. 169–185, July ↑1

[5] Brewer E. A. *Towards robust distributed systems (abstract)* // Proc. of the 19th Annual ACM Symposium on Principles of Distributed Computing, Port-Land, Oregon, United States, 2000, July 16, p. 7 ↑3

[6] Демидов А. А. *Проектирование распределённых систем обработки объектных структур данных* // Труды XII Всероссийской научной конференции RCDL'2010. — Казань : Казанский университет, 2010, с. 441–447 ↑3, 3, 4

[7] Чардин П. *Многоверсионность данных и управление параллельными транзакциями* // Открытые системы, 2005, № 1, с. 64–69 ↑3

Demidov Aleksey. *Topological methods in the synchronization system design for concurrent transactions in distributed databases.*

Abstract. A special method of selecting objects is proposed based on topological analysis of the stored data internal structure. The concept of functional dependence is generalized to relation of conditionality, which is then used to determine the need for data access synchronization of concurrent processes in a distributed environment. This way formalized model essentially simplifies the transaction management.

Key Words and Phrases: passive cache, consistency, distributed database, concurrent transaction, synchronization costs.

Образец ссылки на статью:

А. А. Демидов. *Топологические методы в проектировании системы синхронизации конкурирующих транзакций распределённой базы данных* // Программные системы: теория и приложения : электрон. научн. журн. 2011. № 4(8), с. 139–152. URL: http://psta.psiras.ru/read/psta2011_4_139-152.pdf