Метод для согласованного выполнения семейства распределенных асинхронно взаимосвязанных транзакций

И.Г. Данилов

Научно-исследовательский центр супер-ЭВМ и нейрокомпьютеров

В работе предлагается метод обнаружения RW-конфликтов по разделяемым данным, возникающих во время конкурентного выполнения набора распределенных транзакций, который предназначен для предотвращения связанных с таким типом конфликтов аномалий выполнения.

1. Введение

В 90-х годах прошлого века M. Herlihy и J. E. B. Moss описали оригинальную аппаратную реализацию [1] синхронизации доступа нескольких процессоров к разделяемой памяти: конкурентные операции чтения/записи разделяемой памяти процессора могут объединяться в некоторый атомарный (неделимый) набор действий — *транзакцию* — и выполняться совершенно независимо от операций других процессоров. Предложенный механизм синхронизации, который в настоящее время является объектом активных научных исследований, был назван *транзакционной памятью, ТП* (англ. *transactional memory, TM*). Однако можно отметить, что сама идея не так уж и нова: впервые применять подобный механизм синхронизации для любых вычислительных процессов вообще, а не только для управления доступом к данным в *базах данных* (БД), предложил D. B. Lomet. В своей работе [2] Lomet описал концепцию *атомарных действий* (англ. *atomic actions*), позже реализованную Liskov и Scheifler в языке распределенного программирования Argus [3]. Появление идеи транзакционной памяти стало результатом обработки и осмысления многолетних исследований в области традиционной синхронизации вычислительных процессов и моделей согласованности памяти с одной стороны: схожие принципы и соответствующие конструкции можно встретить у Хоара — концепция *мониторов* [4], и с другой стороны — в области теории транзакционной обработки данных в базах данных.

В настоящее время актуальными являются исследования возможностей и преимуществ применения механизмов транзакционной памяти для масштабируемых вычислительных систем с распределенной памятью, в первую очередь для кластерных вычислительных систем. В силу своих особенностей ТП может оказаться более эффективным и масштабируемым по сравнению с традиционными решениями на основе алгоритмов распределенного взаимного исключения подходом к синхронизации в таких системах, что уже подтверждает ряд имеющихся исследований [5–7].

2. Обзор работ по тематике исследования

2.1. Свойства алгоритмов ТП

Свойством *живости* [8] для алгоритмов транзакционной обработки данных является критерий *согласованного выполнения транзакций* или просто *критерий согласованности*. Примером может служить классический критерий *сериализуемости* транзакций БД [9]. Однако, из-за существующих различий между транзакциями в БД и транзакциями памяти, прежде всего из-за большей автономности последних, традиционного критерия сериализуемости, который является основополагающим для транзакций БД, не хватает для определения корректности выполнения набора транзакций памяти. В нем рассматривают-

ся лишь *зафиксированные* транзакции и не затрагивается вопрос корректности выполнения *"живых"*, еще не зафиксированных транзакций. При использовании оптимистичных методов синхронизации такой критерий подходит только для полностью контролируемых сред или *песочниц*, какой является для транзакций СУБД, и не является удовлетворительным для транзакций памяти. Действительно, при конкурентном выполнении набора транзакций возможны ситуации гонок и, как следствие, несогласованные чтения: как из-за чередования конкурентных операций, так и, возможно, из-за несогласованного состояния памяти в случае использования клонирования (кэширования) данных, либо механизма откатов (см. классические проблемы конкурентного выполнения транзакций [10]), а валидация (обнаружение конфликтов по данным) происходит только перед самой фиксацией транзакции. Транзакции, выполняемые в еще не обнаруженном состоянии гонок, называются *транзакциями-зомби* [11] или *обреченными* (англ. *doomed*).

С целью избежания подобных ситуаций для алгоритмов ТП был предложен более строгий в плане корректности выполнения критерий, который является вариантом строгой сериализуемости, названный критерием *скрытности* (англ. *opacity*) [12]. Он предполагает, что: а) все операции, выполненные каждой зафиксированной транзакцией, представляются так, как если бы они были выполнены в одной неделимой точке жизненного цикла транзакции; б) ни одна операция любой прерванной транзакции не должна быть видна другим, в том числе выполняющимся транзакциям; в) каждая транзакция в любой момент времени наблюдает *согласованное* состояние системы (невозможны несогласованные чтения). Можно сказать, что критерий скрытности последовательно упорядочивает не только зафиксированные, но и прерванные транзакции, без наблюдения результатов их выполнения другими транзакциями набора. Таким образом, при возникновении конфликт должен быть сразу обнаружен (например, с помощью проверки на наличие изменений версий прочитанных ранее значений), и одна из конфликтующих транзакций прервана и перевыполнена заново.

2.2. Обнаружение конфликтов конкурентного выполнения распределенных транзакций

Одним из самых важных вопросов помимо выбора *стратегии обнаружения и разрешения* конфликтов конкурентного выполнения набора транзакций является реализация **метода обнаружения конфликтов**. При использовании *отложенной* стратегии обновления данных существует необходимость в обнаружении только RW-конфликтов, т.к. для записи используется промежуточный буфер (WR-конфликты невозможны), а сама память изменяется только в момент фиксации транзакции. Возникающие при этом "отложенные" WW-конфликты зачастую разрешаются на основе правила *"первая фиксация побеждает"* (англ. *first committer wins*, FCW): в момент фиксации транзакция проверяет перезаписываемые ею данные на наличие обновлений, сделанных конкурентными транзакциями, и, при обнаружении таких обновлений, откатывается. Для оптимистичных механизмов синхронизации [13] основной процедурой методов обнаружения конфликтов является валидация (проверка) множества считанных транзакцией значений *Rset*. Но то, как и когда данную процедуру использовать, зависит от реализуемого для отслеживания конкурентных обновлений разделяемых данных метода обнаружения конфликтов. Подходы, которые применяются в алгоритмах для мультипроцессорных систем, нельзя напрямую использовать для мультикомпьютерной или *распределенной транзакционной памяти, РТП* (англ. *distributed software transactional memory*, DSTM).

В работе [6] был предложен метод обнаружения RW-конфликтов с использованием версий для данных и валидации множества *Rset* путем отслеживания причинно-следственного порядка между операцией чтения транзакции и операциями записи остальных транзакций системы. Для этого авторы предложили специального вида логические часы [14], которые реализованы в виде целочисленного счетчика и подчиняются следующим правилам:

1. если a — успешное событие фиксации транзакции $T_i \in T$, выполняемой на узле системы N_k, то часы
$$C_k(a) \leftarrow C_k \leftarrow C_k + 1 \quad (1)$$

2. пусть a — событие приема на узле системы N_k сообщения q, с назначенной меткой времени, равной значению часов узла системы N_m в момент события b отправления сообщения q: $t_q = C_m(b)$; тогда после события a часы C_k устанавливаются в значение большее среди текущего значения часов C_k и метки t_q:
$$C_k \leftarrow \max(C_k, t_q) \quad (2)$$

Сам метод обнаружения конфликтов, названный методом *продвижения транзакции* (англ. *transaction forwarding*) заключается в следующем:

- при старте транзакция T_i, выполняемая на узле N_k считывает текущее значение часов C_k и сохраняет его в переменной wv;
- транзакция T_i *продвигает свое стартовое время* wv до значения $wv' > wv$, т.е. делает присваивание $wv = wv'$, только в случае успешной валидации $Rset$: для всех объектов в $Rset$ их текущая версия сравнивается с wv и, если версия какого-либо объекта превышает wv, то валидация заканчивается неуспешно, а транзакция откатывается;
- для чтения *удаленного* объекта транзакция T_i посылает сообщение-запрос на узел расположения объекта N_m; при получении ответного сообщения q к посланному ранее сообщению-запросу с текущей версией объекта транзакции необходимо продвинуть свое стартовое время wv до значения t_q в случае, если $wv < t_q$ (см. выражение 2 для логических часов); если же $wv \geq t_q$, то объект может быть считан безопасно;
- при чтении *локального* объекта, расположенного на узле выполнения транзакции, его версия проверяется и, если она больше, чем wv, то транзакция откатывается;
- в момент фиксации транзакция наращивает часы C_k (см. выражение 1) и назначает новую версию для всех перезаписываемых объектов равную новому значению часов C_k.

На базе предложенного метода авторами разработан алгоритм TFA, который по производительности превосходит все существующие конкурентные решения [6] и, как заявлено авторами, соответствуют критерию скрытности транзакций. Однако это не совсем так. Представим ситуацию, иллюстрация к которой изображена на рис. 1. В системе из четырех узлов: (N_1, N_2, N_3, N_4) выполняются две транзакции T_1 и T_2. При этом транзакция T_2 перезаписывает между $R_1(z)$ и $R_1(v)$ считанный ранее T_1 объект y, но в момент чтения $R_1(v)$ продвижения транзакции и, следовательно валидации $Rset$ не происходит, в силу того, что $wv_1 = C_4$. В результате чтение $R_1(v)$ получается несогласованным.

Кроме этого, недостатком предложенного подхода является его ориентированность на распределенные протоколы когерентности кэша (например Relay [15]), которые плохо масштабируются, и предоставление всего одной перезаписываемой копии объекта транзакциями системы. В работе [7] авторами был предложен подход и ряд принципов для эффективной реализации программно-организуемой транзакционной памяти на основе модели PGAS и односторонних коммуникаций. Предложенный подход был реализован в системе Cluster-STM, хорошо масштабируемой на кластерной системе с сотнями вычислительных узлов. Основной недостаток Cluster-STM — отсутствие поддержки динамического параллелизма на уровне потоков, ограничение “один процессор, одна транзакция” — был устранен в системе GTM [16], также построенной на основе модели PGAS и SPMD-подхода, но вместе с этим предоставляющей поддержку динамического параллелизма на уровне потоков и

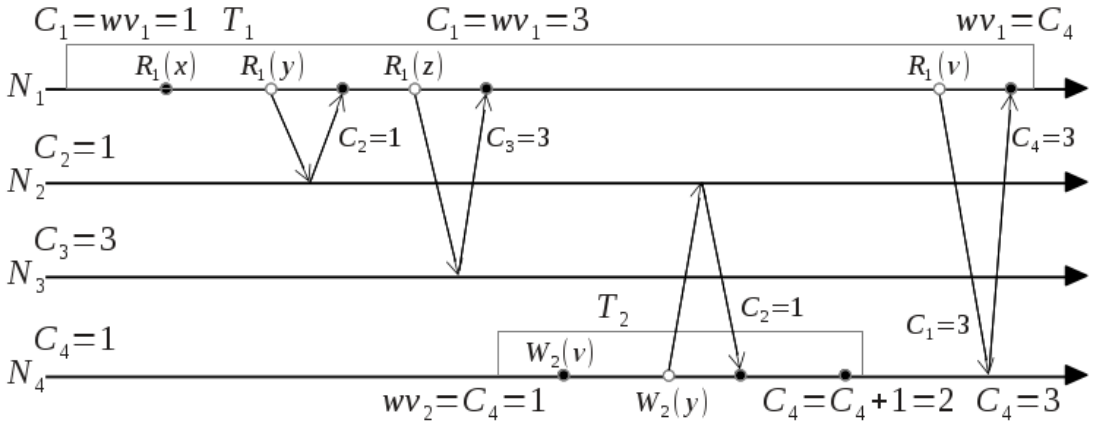

Рис. 1. Ситуация несогласованного чтения при использовании метода продвижения транзакции

удаленного вызова процедур. Недостатком обоих систем является предоставление только лишь гарантии сериализуемости; вопрос возможного возникновения гонок и, как следствие некорректное выполнение программы, не рассматривается.

Кроме вышерассмотренных существует ряд подходов, основанных на репликации и многоверсионности данных и предоставляющих более слабые по сравнению со скрытностью гарантии согласованности: GMU [17], SCORe [18] и др. Репликация позволяет только читающим транзакциям выполняться локально, без генерации сетевого трафика и валидации удаленных данных. В GMU применяется многоверсионная схема данных, *векторные логические часы* [19] для отслеживания причинно-следственной связи между событиями, а также *частичная истинная* схема репликации данных. GMU предоставляет слабый критерий согласованности EUS (англ. *extended update-serializability*), согласно которому различные только читающие транзакции могут видеть результат обновления неконфликтующих пищущих транзакций в различном порядке. Улучшенная версия протокола частичной истинной репликации данных, который назван SCORe, гарантирует более строгий критерий *выполнимой сериализуемости одной копии* (англ. *executing one-copy serializability*, 1CS) [18]. Протокол SCORe достаточно хорошо масштабируем и эффективен за счет использования локальной мультиверсионной схемы данных, позволяющей только читающим транзакциям всегда выполняться без отката, и за счет масштабируемой схемы синхронизации логических часов, в которой связанные транзакции обмениваются лишь одним скалярным числом — меткой часов.

Таким образом, можно сделать вывод, что в настоящий момент не разработано методов и алгоритмов РТП, гарантирующих выполнение критерия скрытности. Разработанные алгоритмы, дающие более слабые гарантии, реализуются на основе динамических, интерпретируемых языков, таких как Java, которые позволяют обнаружить и, возможно, устранить исключительные ситуации, возникающие из-за несогласованных чтений. Для более популярных в области выскопроизводительных вычислений не динамических языков, таких как C/C++ представлено всего несколько программных средств: Cluster-STM и DMV [20], гарантирующие сериализуемость, и HyflowCPP [21] — программный фреймворк, реализующий алгоритм TFA, который как было показано выше не предоставляет гарантию выполнения критерия скрытности.

3. Разработка метода обнаружения RW-конфликтов по данным в распределенной системе

3.1. Модель системы

Пусть $\mathbb{N} = \{n_1, n_2, \ldots, n_N\}$ — множество узлов *распределенной асинхронной вычислительной системы без сбоев* [8], а $\mathbb{T} = \{T_1, T_2, \ldots, T_M\}$ — множество распределенных в системе *транзакций*. Кроме транзакций система также включает в себя *память* с **последовательной** моделью согласованности, с помощью которой транзакции взаимодействуют. Будем подразделять память на два вида:

1. Plm_k — ***приватная***, доступная только транзакции $T_k \in \mathbb{T}$, $k = \overline{1, M}$ и относящаяся к или *выделенная* на узле выполнения T_k: $n_j \in \mathbb{N}$, $j = \overline{1, N}$ память. Множество L представляет собой совокупность всей локальной памяти в системе: $L = Plm_1 \cup Plm_2 \cup \ldots \cup Plm_M$;
2. G — ***разделяемая***, доступная всем транзакциям системы память такая, что $G = Pgm_1 \cup Pgm_2 \cup \ldots \cup Pgm_N$, где Pgm_j — часть разделяемой памяти, выделенная на узле $n_j \in \mathbb{N}$, $j = \overline{1, N}$.

Будем считать, что сами транзакции неподвижны и выполняются на том же самом узле, где были инициированы, а удаленные данные при этом могут перемещаться и кэшироваться на узле выполнения транзакции. Подобная модель, которая была предложена в работе [22], называется моделью с *потоком данных*. Также будем считать, что транзакции используют *отложенную* стратегию обновления данных.

Память представляет из себя множество специальных объектов — *ячеек*. Каждой ячейке памяти x соответствует некоторое определенное *значение*, задаваемое отображением $V(x)$. Обозначим символом $\mathcal{V}$ множество всех возможных значений, которые могут принимать ячейки памяти. Совокупность всех ячеек системы будем обозначать символом M : $M = L \cup G$. Будем говорить, что множество ячеек M_1 равно множеству M_2, когда:

$$|M_1| = |M_2| \wedge (\forall x \in M_1 \, \exists! \, y \in M_2 : x = y \wedge V(x) = V(y))$$

Для описания системы взаимодействующих распределенных транзакций воспользуемся по аналогии рядом базирующихся на понятии *машины состояний* определений, предложенных в [8], и адаптируем их для распределенной системы:

Определение 3.1. ***Локальным алгоритмом*** *транзакции будем называть пятерку* $(Z, I, \vdash^i, \vdash^r, \vdash^w)$*, в которой* Z *— множество состояний,* I *— это некоторое подмножества множества* Z*,* $\vdash^i$ *— отношение на множестве* $Z \times Z$*,* $\vdash^r$ *— отношение на множестве* $Z \times M \times Z$*,* $\vdash^w$ *— отношение на множестве* $Z \times M \times \mathcal{V} \times Z$*. Двуместное отношение* $\vdash$ *на множестве* Z *определяется соотношением:*

$$c \vdash d \Longleftrightarrow (c, d) \in \vdash^i \vee \exists x \in M((c, x, d) \in \vdash^r) \vee \exists x \in M, \exists v \in \mathcal{V}((c, x, v, d) \in \vdash^w)$$

Отношение $\vdash^i$ представляет собой переход между состояниями, связанный с *внутренними* событиями транзакции $T_k \in \mathbb{T}$, отношение $\vdash^r$ — событиями *чтения* значения из ячейки памяти, а $\vdash^w$ — событиями *записи* значения в ячейку памяти. Транзакция $T_k \in \mathbb{T}$ может выполняться на любом узле $n_j \in \mathbb{N}$ вычислительной системы (ВС). Выполнение транзакции T_k — выполнение системы переходов $(Z_k, I_k, \vdash^i_k, \vdash^r_k, \vdash^w_k)$. Состояние транзакции S_{T_k} описывается собственно состоянием транзакции c_{T_k} и состоянием или *слепком* доступной ей *локальной памяти* Plm_k: $S_{T_k} = (c_{T_k}, Plm_k)$. Внутренние события и события чтения, которые описываются с помощью отношений $\vdash^i_k, \vdash^r_k$ соответственно, могут изменять собственно состояние транзакции, а события записи в том числе и состояние доступной ей памяти: как локальной, так и глобальной.

Для описания распределенной системы в целом введем понятие *распределенного алгоритма*, составленного из соответствующих компонент транзакций, и системы переходов, которую данный алгоритм порождает:

Определение 3.2. ***Распределенным алгоритмом*** *семейства транзакций* $\mathbb{T} = \{T_1, T_2, \ldots, T_M\}$ *будем называть совокупность локальных алгоритмов, каждый из которых соответствует в точности одной транзакции из* $\mathbb{T}$.

Состояние семейства транзакций $\mathbb{T}$ будем называть *конфигурацией*. Каждая конфигурация системы определяется состояниями всех транзакций и состоянием разделяемой памяти G.

Поведение распределенного алгоритма семейства транзакций $\mathbb{T}$ описывается с помощью системы переходов, определяемой следующим образом:

Определение 3.3. *Пусть задано семейство транзакций* $\mathbb{T} = \{T_1, T_2, \ldots, T_M\}$, *а локальный алгоритм каждой транзакции* T_k *представлен пятеркой* $(Z_{T_k}, I_{T_k}, \vdash^i_{T_k}, \vdash^r_{T_k}, \vdash^w_{T_k})$. *Будем говорить, что система переходов* $\mathcal{S} = (\mathcal{C}, \to, \mathcal{I})$ ***порождена распределенным алгоритмом для семейства асинхронно взаимосвязанных транзакций*** $\mathbb{T}$, *если выполнены следующие соотношения:*

1. $\mathcal{C} = \{(S_{T_1}, \ldots, S_{T_M}, G) : (\forall T \in \mathbb{T} : S_T \in Z_T) \wedge (\forall x \in L, \forall y \in G : V(x), V(y) \in \mathcal{V})\}$, *где* $\mathcal{C}$ *— множество конфигураций семейства транзакций;*

2. $\to = (\bigcup_{T \in \mathbb{T}} \to_T)$, *где символом* $\to_T$ *обозначаются переходы, которые соответствуют изменениям состояния транзакции* T, *т.е.* $\to_{T_i}$ *— множество всех пар вида*

$$(S_{T_1}, \ldots, S_{T_k}, \ldots, S_{T_M}, G_1),\ (S_{T_1}, \ldots, S'_{T_k}, \ldots, S_{T_M}, G_2),$$

   таких, что выполняется одно из следующих условий:

   - $(S_{T_k}, S'_{T_k}) \in \vdash^i_{T_k}$ *и* $M_1 = M_2$;
   - *для некоторой ячейки* $x \in M_1$ *имеется событие чтения* $(S_{T_k}, x, S'_{T_k}) \in \vdash^r_{T_k}$ *и* $M_1 = M_2$;
   - *для некоторой ячейки* $x \in M_1$ *и значения* $v \in \mathcal{V}$ *имеется событие записи* $(S_{T_k}, x, v, S'_{T_k}) \in \vdash^w_{T_k}$ *так, что* $|M_1| = |M_2|$ *и* $\exists!\, y \in M_2 : x = y \wedge V(y) = v$;

3. $$\mathcal{I} = \{(S_{T_1}, \ldots, S_{T_M}, G) : (\forall T \in \mathbb{T} : S_T \in I_T) \wedge (\forall x \in L, \forall y \in G : V(x) = V_0(x), V(y) = V_0(y), V(x), V(y) \in \mathcal{V})\},$$

   где $\mathcal{I}$ *— множество начальных конфигураций семейства транзакций, а* $V_0(x) : \forall x \in L \cup G$ *— задает начальные значения ячеек памяти.*

Соответственно выполнением распределенного алгоритма называется *всякое* выполнение системы переходов, которая порождена этим алгоритмом. Конкретное выполнение E задается в видет последовательности конфигураций, через которые система проходит в данном выполнении: $E = (\gamma_0, \gamma_1, \ldots)$. Выполнение может быть либо конечным, если для него имеется *заключительная* конфигурация, т.е. такая конфигурация $\gamma \in \mathcal{C} : \nexists \delta \,|\, \gamma \to \delta$, либо бесконечным, если для него заключительной конфигурации не существует. С выполнением E связана последовательность событий $\bar{E} = (e_0, e_1, \ldots)$, где e_i — некоторое *допустимое* событие, обозначающее преобразование конфигурации γ_i в γ_{i+1}: $e_i(\gamma_i) = \gamma_{i+1}$.

Очевидно, что если два последовательных события в выполнении влияют на не пересекающиеся подмножества конфигураций, то эти события *независимы* и могут "выполняться" в другом порядке. В противном случае, если события нельзя поменять местами, то между ними существует *причинно-следственная зависимость*. Для любой последовательности событий $\bar{E}$, связанной с некоторым выполнением E выделим три вида такой зависимости.

Определение 3.4. *Будем говорить, что между двумя событиями $e_i, e_j \in \bar{E}$ существует:*

1. *истинная зависимость: $e_i \prec^{wr} e_j \iff$ событие e_i — событие записи в некоторую ячейку $x \in M$ значения $v \in \mathcal{V}$, а событие e_j — последующее после e_i событие чтения из ячейки x значения v;*

2. *выходная зависимость: $e_i \prec^{ww} e_j \iff$ событие e_i — событие записи в некоторую ячейку $x \in M$ значения $v_1 \in \mathcal{V}$, а событие e_j — последующее после e_i событие записи в эту же ячейку x значения $v_2 \in \mathcal{V}$;*

3. *антизависимость: $e_i \prec^{rw} e_j \iff$ событие e_i — событие чтения из некоторой ячейки $x \in M$ значения $v_1 \in \mathcal{V}$, а событие e_j — последующее после e_i событие записи в эту же ячейку значения $v_2 \in \mathcal{V}$.*

По аналогии с отношением причинно-следственного порядка, которое Лэмпорт ввел для систем с асинхронным обменом сообщениями [14], зададим такое же отношение для описанной выше асинхронной системы с распределенной общей памятью.

Определение 3.5. *Причинно-следственное отношение частичного порядка $\prec$ на множестве событий системы $\bar{E}$ выполнения E — это такое наименьшее отношение, удовлетворяющее следующим трем условиям:*

1. *если e и f — два события одной транзакции и e произошло раньше, чем f, тогда $e \prec f$;*

2. *пусть e и f два разных события, тогда если $e \prec^{wr} f \vee e \prec^{ww} f \vee e \prec^{rw} f$, то $e \prec f$;*

3. *отношение $\prec$ транзитивно, то есть если $e \prec g$ и $g \prec f$, то $e \prec f$.*

Определим два вида специальных событий, допустимых при выполнении распределенного алгоритма семейства транзакций. Первый вид событий — событие *отката* A_k: такое допустимое событие в последовательности событий транзакции T_k $\bar{E}_{T_k} = (e_0, \ldots, e_i, A_k, e'_0, \ldots) \,|\, \bar{E}_{T_k} \subseteq \bar{E}$, представленное парой $A_k = (S^i_{T_k}, S'_{T_k}) \in \vdash^i_{T_k}$, что $A_k(\gamma_{i+1}) = \gamma'_0$, где $\gamma'_0 = (S_{T_1}, \ldots, S'_{T_k}, \ldots, S_{T_M}, G') : S'_{T_k} \in I_{T_k} \wedge (\forall x \in Plm_k : V(x) = V_0(x) \in \mathcal{V})$. Второй вид специальных событий — событие *фиксации* C_k транзакции T_k: такое допустимое событие в последовательности событий $\bar{E}_{T_k} = (e_0, \ldots, e_i, C_k)$, представленное парой $C_k = (S^i_{T_k}, S'_{T_k}) \in \vdash^i_{T_k}$, что S'_{T_k} — *заключительное* состояние T_k в выполнении E.

3.2. Метод обнаружения RW-конфликтов по данным в распределенной системе

Будем считать, что с каждой ячейкой $x : x \in G$ связана некоторая *версия* или *метка* $ts^k_x : ts^k_x \in TSset$, присвоенная x транзакцией T_k. Определим на множестве $TSset$ функцию TS такую, что:

$$TS : TSset \to \{\langle j, tsn\rangle : j \in \mathbb{Z}_+, tsn \in \mathbb{Z}_{\geqslant 0}\},$$

где j — целое положительное число, номер узла ВС $n_j \in \mathbb{N}$, а tsn — целое неотрицательное число, которое является отметкой времени часов узла n_j.

Определим, аналогично тому, как это сделано в работе [6], логические часы CLK_m для узла ВС $n_m \in \mathbb{N}$:

1. если C_k — успешное событие фиксации транзакции $T_k \in \mathbb{T}$, выполняемой на узле системы n_m, то

$$CLK_m(C_k) \leftarrow CLK_m \leftarrow CLK_m + 1 \tag{3}$$

2. пусть e — событие чтения ячейки $x \in G$ транзакцией $T_k \in \mathbb{T}$, выполняемой на узле системы n_m; при этом если ячейке назначена версия $ts_x : TS(ts_x) = \langle j, tsn \rangle$, тогда после события e часы CLK_m устанавливаются в значение большее среди текущего значения часов CLK_m и отметки tsn версии ts_x:

$$CLK_m \leftarrow \max(CLK_m, tsn) \tag{4}$$

Установим следующие соотношения для всех возможных значений версий ячеек $ts_i \in TSset$. Пусть $\forall ts_1, ts_2 \in TSset : TS(ts_1) = \langle j_1, tsn_1 \rangle \wedge TS(ts_2) = \langle j_2, tsn_2 \rangle$, тогда выполняется:

$$\begin{gathered} ts_1 =, <, > ts_2 \Longleftrightarrow tsn_1 =, <, > tsn_2, \\ ts_1 \leq, \geq ts_2 \Longleftrightarrow tsn_1 \leq, \geq tsn_2, \\ ts_1 \equiv ts_2 \Longleftrightarrow tsn_1 = tsn_2 \wedge j_1 = j_2, \\ ts_1 \leqq, \geqq ts_2 \Longleftrightarrow ts_1 <, > ts_2 \vee ts_1 \equiv ts_2 \end{gathered} \tag{5}$$

Определим *множество прочитанных ячеек* транзакции T_k как: $Rset_k = \{\, x \,|\, x \in G \wedge \exists e : e \in \bar{E}_{T_k} \wedge e$ — событие чтения ячейки $x \,\}$. На множестве $Rset_k$ зададим функцию $RS_k : Rset_k \to \{ts \,|\, ts \in TSset\}$ — версия ячейки, которой соответствует некоторое значение $v \in \mathcal{V}$. Таким образом $RS_k(x) = ts_x$.

Определим для каждой транзакции T_k вектор $VC_k : |VC_k| = |\mathbb{N}| = N$ так, что $j-$й элемент вектора равен $VC_k[j] = tsn$ — некоторой отметке времени логических часов узла с номером равным j. Предлагаемый метод обнаружения RW-конфликтов в распределенной системе предполагает использование отложенной стратегия обновления данных и заключается в следующем:

- при старте транзакции T_k, выполняемой на узле n_m, вектор VC_k инициализируется нулевыми значениями: $VC_k[i] \leftarrow 0, \forall i = \overline{1, N}$;
- при чтении значения ячейки $x \in G$ с соответствующей версией $ts_x \in TSset : TS(ts_x) = \langle j, tsn \rangle$ проверяется, если $VC_k[j] < tsn$, то производится валидация объектов $Rset_k$ и присваивание $VC_k[j] \leftarrow tsn$, а также изменяются часы CLK_m (см. выражение 4 для логических часов); конфликты отсутствуют и объект может быть считан безопасно только в случае *успешной* валидации после чего в $Rset_k$ добавляется x: $RS_k(x) = ts_x$;
- при записи ячейки $x \in G$ производится определение ее текущей версии и при необходимости изменяются часы CLK_m согласно выражению 4, а новое значение v' сохраняется в буфере для последующей записи во время фиксации (отложенная стратегия обновления данных);
- валидация производится для всех объектов x чьи версии сохранены ранее в $Rset_k$ путем сравнения текущей версии объекта ts'_x относительно сохраненной в $Rset_k$ версии ts_x: $RS_k(x) = ts_x$ и если $ts'_x > ts_x$, то валидация заканчивается *неуспешно*;
- в момент события C_k фиксации транзакции T_k, выполняемой на узле n_m, наращиваются часы CLK_m (см. выражение 3), а все объекты *атомарно* перезаписываются новым значением v' с новой версией равной $ts^k_x = \langle m, CLK_m(C_k) \rangle$.

Сформулируем и докажем следующее утверждение:

Теорема 3.1. *Предложенный метод при условии атомарной перезаписи новыми значениями ячеек памяти во время фиксации транзакций позволяет гарантированно обнаруживать во время выполнения распределенного семейства асинхронно взаимосвязанных транзакций RW-конфликты по данным, которые могут привести к ситуации несогласованного чтения.*

Доказательство. Допустим, что это не так. Тогда для некоторой транзакции T_k узла ВС n_m в выполнении E распределенного алгоритма, реализующего предложенный метод, существует событие несогласованного чтения e'' ячейки $y \in G$ с версией $ts_y^s \in TSset$: $TS(ts_y^s) = \langle j, tsn^s \rangle$ и $VC_k[j] \geq tsn^s$, т.о. после e'' валидация $Rset_k$ не производится. При этом в выполнении E для T_k есть также предшествующее событие чтения e' ячейки $x \in G$ с версией, созданной транзакцией T_r, $ts_x^r \in TSset : TS(ts_x^r) = \langle i, tsn^r \rangle$ такое, что $e' \prec e''$.

Существование события несогласованного чтения e'' ячейки $y \in G$ означает то, что $\exists$ два события записи f' и f'' транзакции T_s узла n_j: f' — событие записи ячейки $x \in G$, а f'' — событие записи ячейки $y \in G$ с соответствующими версиями $ts_x^s, ts_y^s \in TSset : TS(ts_x^s) = TS(ts_y^s) = \langle j, tsn^s \rangle \wedge ts_x^s > ts_x^r$. Причем при условии атомарности перезаписи новыми значениями ячеек памяти во время фиксации транзакций выполняется: 1) $e' \prec^{rw} f'$, $e'' \prec^{rw} f''$, либо 2) $e' \prec^{rw} f'$, $f'' \prec^{wr} e''$. При этом $f' \prec f'' \vee f'' \prec f'$. В первом случае событие e'' не является событием несогласованного чтения, что противоречит первоначальному условию.

Во втором случае, так как после события чтения e'' выполняется соотношение $VC_k[j] \geq tsn^s$, то для транзакции T_k должно $\exists$ событие чтения g ячейки $z \in G$ с версией $ts_z^q \in TSset$: $TS(ts_z^q) = \langle j, tsn^q \rangle \wedge VC_k[j] = tsn^q \geq tsn^s$. Т.е. ts_z^q произведена некоторой транзакцией T_q. Тогда, если $tsn^q = tsn^s$, то T_q — это и есть транзакция T_s, поэтому при чтении ячейки $z \in G$ валидация $Rset_k$ должна отследить изменение версии ячейки $x \in G$ и дальнейшее событие несогласованного чтения e'' невозможно: т.о. приходим к противоречию. Если $tsn^q > tsn^s$, то T_q и T_s разные транзакции одного узла ВС n_j, но T_q завершилась позже T_s, следовательно, валидация $Rset_k$ после события чтения g также должна отследить изменение версии и ячейки $x \in G$, следовательно дальнейшее событие несогласованного чтения e'' невозможно. Опять приходим к противоречию.

Теорема доказана.

Стоит отметить, что условие атомарной перезаписи ячеек памяти при фиксации транзакции обычно выполняется в силу использования специальных протоколов фиксации, таких как двухфазный протокол фиксации (2PC).

4. Программная реализация предложенного метода

На основе разработанного метода предложены алгоритмы и создана программная система распределенной транзакционной памяти DSTM_P1 [23, 24]. Данная система реализована на языке C++ и предназначена для запуска и контроля над выполнением многопоточных приложений, написанных на языке Си с использованием метода синхронизации на основе распределенной кэшируемой программно-организуемой транзакционной памяти, на кластерных вычислительных системах под управлением операционной системы Linux.

DSTM_P1 можно условно разделить на две основные части: *среду выполнения приложений* и *прикладной программный интерфейс* (API), доступный для написания многопоточных приложений. Среда выполнения является своего рода программной *"песочницей"* для приложений пользователя и, по выбору пользователя. API системы DSTM_P1 подразделяется на:

- интерфейс для работы с распределенной разделяемой памятью *Idstm_malloc*;
- интерфейс для запуска и управления выполнением распределенных потоков *Idstm_pthread*;
- интерфейс для барьерной синхронизации распределенных потоков *Idstm_barrier*.

Общая схема работы DSTM_P1 может быть описана следующим образом: (1) пользователь системы загружает исходные коды многопоточного приложения, написанного на

языке C с использованием библиотеки Pthread и применением атомарных конструкций (__tm_atomic) в местах доступа к разделяемым данным; (2) приложение компилируется с помощью DTMC [8] в язык промежуточного представления; (3) далее полученное представление трансформируется в представление, содержащее соответствующие вызовы функций библиотеки транзакционной памяти для всех инструкций доступа к памяти атомарного блока; (4) представление, полученное на предыдущем шаге линкуется с необходимыми библиотеками (транзакционной памяти, «обертками» системных библиотек pthread и malloc); (5) на лету компилируется и исполняется main-функция приложения; (6) все вызовы функций библиотеки pthread и malloc переадресуются в соответствующие вызовы функций библиотек «оберток». Во время исполнения main-функции при вызове pthread_create определяется код функции потока, который распределяется в системе с помощью модуля балансировки нагрузки и исполняется в отдельном потоке на менее загруженном узле кластера.

С предварительными результатами экспериментальных исследований можно ознакомиться в [24].

5. Заключение

В работе представлено описание метода, который может быть использован для разработки алгоритмов синхронизации конкурентного выполнения набора распределенных асинхронно взаимосвязанных транзакций. Отличительной особенностью метода является его соответствие строгому критерию согласованности транзакций: критерию скрытности.

Литература

1. Herlihy M., Moss J. E. B. Transactional memory: architectural support for lock-free data structures // SIGARCH Comput. Archit. News. 1993. Vol. 21, No. 2. P. 289–300.

2. Lomet D. B. Process structuring, synchronization, and recovery using atomic actions // SIGOPS Oper. Syst. Rev. 1977. Vol. 11, No. 2. P. 128–137.

3. Liskov B., Scheifler R. Guardians and Actions: Linguistic Support for Robust, Distributed Programs // ACM Trans. Program. Lang. Syst. 1983. Vol. 5, No. 3. P. 381–404.

4. Hoare C. A. R. Monitors: an operating system structuring concept // Commun. ACM. 1974. Vol. 17, No. 10. P. 549–557.

5. Saad M. M., Ravindran B. HyFlow: a high performance distributed software transactional memory framework // Proceedings of the 20th international symposium on High performance distributed computing. HPDC '11. 2011. P. 265–266.

6. Saad M. M., Ravindran B. Transactional Forwarding Algorithm: Tech. rep.: ECE Dept., Virginia Tech, 2011.

7. Bocchino R. L., Adve V. S., Chamberlain B. L. Software transactional memory for large scale clusters // Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming. PPoPP '08. 2008. P. 247–258.

8. Тель Ж. Введение в распределенные алгоритмы. М.: Изд-во МЦНМО, 2009. 616 с.

9. Eswaran K. P., Gray J. N., Lorie R. A., Traiger I. L. The notions of consistency and predicate locks in a database system // Commun. ACM. 1976. Vol. 19, No. 11. P. 624–633.

10. Bernstein P. A., Goodman N. Concurrency Control in Distributed Database Systems // ACM Comput. Surv. 1981. Vol. 13, No. 2. P. 185–221.

11. Dice D., Shalev O., Shavit N. Transactional locking II // Proceedings of the 20th international conference on Distributed Computing. DISC'06. 2006. P. 194–208.

12. Guerraoui R., Kapalka M. On the correctness of transactional memory // Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming. PPoPP '08. 2008. P. 175–184.

13. Kung H. T., Robinson J. T. On optimistic methods for concurrency control // ACM Trans. Database Syst. 1981. Vol. 6, No. 2. P. 213–226.

14. Lamport L. Time, clocks, and the ordering of events in a distributed system // Commun. ACM. 1978. Vol. 21, No. 7. P. 558–565.

15. Zhang B., Ravindran B. Brief Announcement: Relay: A Cache-Coherence Protocol for Distributed Transactional Memory // Proceedings of the 13th International Conference on Principles of Distributed Systems. OPODIS '09. 2009. P. 48–53.

16. Sridharan S., Vetter J. S., Kogge P. M. Scalable Software Transactional Memory for Global Address Space Architectures: Tech. Rep. FTGTR-2009-04: Future Technologies Group, Oak Ridge National Lab, 2009.

17. Peluso S., Ruivo P., Romano P. et al. When Scalability Meets Consistency: Genuine Multiversion Update-Serializable Partial Data Replication // Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference. 2012. P. 455–465.

18. Peluso S., Romano P., Quaglia F. SCORe: a scalable one-copy serializable partial replication protocol // Proceedings of the 13th International Middleware Conference. Middleware '12. 2012. P. 456–475.

19. Mattern F. Virtual Time and Global States of Distributed Systems // Proc. Workshop on Parallel and Distributed Algorithms / Ed. by C. M. et al. North-Holland / Elsevier: 1989. P. 215–226.

20. Manassiev K., Mihailescu M., Amza C. Exploiting distributed version concurrency in a transactional memory cluster // Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming. PPoPP '06. 2006. P. 198–208.

21. Mishra S., Turcu A., Palmieri R., Ravindran B. HyflowCPP: A Distributed Transactional Memory Framework for C++ // 12th IEEE International Symposium on Network Computing and Applications. NCA 2013. Boston, USA: IEEE Computer Society, 2013.

22. Herlihy M., Sun Y. Distributed transactional memory for metric-space networks // Proceedings of the 19th international conference on Distributed Computing. DISC'05. 2005. P. 324–338.

23. Данилов И.Г. Прототип распределенной программной транзакционной памяти DSTM_P1 // Высокопроизводительные параллельные вычисления на кластерных системах. Материалы XI Всероссийской конференции (Н. Новгород, 2–3 ноября 2011 г.) / Под ред. проф. В.П. Гергеля. - Нижний Новгород: Изд-во Нижегородского госуниверситета. - 2011. - С. 102-107.

24. Данилов И. Г. Об одном подходе к реализации программной транзакционной памяти для распределённых вычислений // Известия ЮФУ. Технические науки. Тематический выпуск «Проблемы математического моделирования, супервычислений и информационных технологий». - Таганрог: Изд-во ТТИ ЮФУ, 2012. - № 6 (131), С. 91-95.