ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ № 4(22), 2014, с. 195–205
УДК 004.652.8:004.75

А. А. Демидов

Об особенностях организации СУБД в MPP-системе

Аннотация. В настоящее время ведутся работы по созданию экзафлопсных компьютеров, способных выполнять 10^{18} операций с плавающей точкой в секунду. Полезная производительность комплекса на реальных задачах обычно оказывается существенно ниже из-за накладных расходов на синхронизацию конкурирующих процессов. В данной работе излагается концепция пассивного кэша, который позволяет в ряде задач отказаться от синхронизации процессов, а также предлагается архитектура высокопроизводительной СУБД на его основе.

Ключевые слова и фразы: параллельные и распределённые вычисления, сверхбольшие базы данных.

Введение

Существуют две основные параллельные архитектуры: мультипроцессорная с общей памятью (SMP, *Symmetric Multiprocessing*) и мультипроцессная с распределенной памятью (MPP, *Massively Parallel Processing*) [1, 2]. В SMP-системе все разделяемые данные находятся в общей памяти, к которой процессы получают организованный доступ. В MPP-системе разделяемые данные распределяются между всеми процессами, имеющими возможность запрашивать эти данные друг у друга. Различие архитектур иллюстрирует рис. 1.

Вычислительные системы, построенные на базе SMP-архитектуры называют параллельными, а системы на базе MPP-архитектуры — распределёнными; иногда оба подхода пересекаются. В параллельной системе вычислительные потоки распределяются между процессорами в рамках одного физического или виртуального компьютера.

Проект проводится при финансовой поддержке государства в лице Минобрнауки России (идентификатор № RFMEFI61314X0030) и РФФИ 12-07-00533-а.

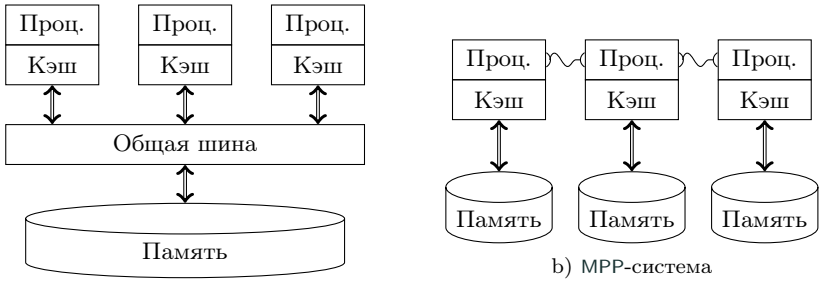

Рис. 1. Отличия архитектур SMP и MPP

В распределённой системе потоки запускаются на независимых узлах, обрабатываемые данные также разделяются на части и располагаются на различных узлах сети.

Система управления транзакциями любой современной СУБД позволяет развернуть её на SMP-системе: конкурирующий доступ потоков к данным будет регламентирован средствами самой СУБД. Для создания MPP-системы внутренних возможностей СУБД недостаточно: функции общей шины, обеспечивающей организованный доступ конкурирующих транзакций к данным в SMP-системе, здесь распределяются между узлами, что накладывает ограничения на протокол взаимодействия процессов между собой. Эти ограничения на самом деле определяет принятый в системе протокол когерентности кэшей, регламентирующий процедуру согласования версий конкурирующими процессами с целью сохранения непротиворечивости глобального состояния.

Протокол когерентности, в свою очередь, определяется выбранной моделью непротиворечивости, которая является своего рода контрактом между процессами и памятью, гласящим, что при соблюдении процессами определенных правил память будет работать предсказуемым образом, в противном случае корректность операций с нею не гарантируется [**3**, §6.2]. В SMP-системе протокол когерентности реализуется с помощью общей шины, в MPP-системе его необходимо прописывать явно.

В данной работе предлагается способ построения распределённой СУБД на базе MPP-архитектуры, предполагающий организацию единого виртуального адресного пространства путём добавления подсистемы динамической подгрузки данных в кэш и механизма трансляции

адресов.

1. Обзор состояния исследований

На уровне языка программирования существование протокола когерентности выражается в необходимости использования средств группировки операций, таких как критические секции или транзакции, и механизмов синхронизации. Находясь внутри критической секции, процесс может быть уверен, что используемые им разделяемые данные не изменяются конкурирующими процессами, которые вынуждены ждать, пока текущий процесс не выйдет из критической секции. Разделяемые данные, прочитанные процессом вне критической секции, могут потерять свою актуальность в любой момент, поэтому после входа в критическую секцию процесс должен заново перечитать все используемые им значения.

Такой групповой способ работы с данными, когда все необходимые элементы считываются в локальный буфер, изменяются, и затем записываются обратно в общую память, резко контрастирует с идеей кэша как таковой, подразумевающей многократное повторное использование считанных из общей памяти значений. Большинство решений данной проблемы (протоколы MSI, MESI, write-once, Firefly, Dragon и другие [4]) приводят к концепции активного кэша, который либо рассылает другим кэшам немедленные уведомления сразу после изменения содержащегося в нём значения общей памяти, либо всякий раз перед использованием такого значения запрашивает другие кэши на предмет наличия изменений. Любая разновидность активного кэша осуществляет поддержку текущего состояния системы и, следовательно, нуждается в синхронизации: пока все кэши не восстановят согласованность, следующая операция с данными не допускается. Поэтому ни один из перечисленных протоколов, вообще говоря, не является хорошо масштабируемым.

Для достижения высокой степени масштабируемости распределённых СУБД применяются различные методы ослабления гарантий непротиворечивости транзакций, условно объединяемые в рамках парадигмы NoSQL: это может быть логическое разделение базы данных на независимые друг от друга сущности («документы», пары «ключ-значение»), отказ от требования строгой ACID-непротиворечивости данных и переход на принципы BASE (*eventual consistency*) или аналогичные. Эти методы объединяет необязательность наличия определённого глобального состояния системы в каждый момент

времени (проблема *Global Predicate Evaluation* [5]): на определённом этапе версии данных на разных узлах могут различаться, однако впоследствии эти различия устраняются системой на основе принятого протокола когерентности.

2. Пассивный кэш

NoSQL решения позволяют избежать синхронизации за счёт отказа от постоянной актуализации глобального состояния, что даёт возможность предложить концепцию пассивного кэша, реализующего модель причинной непротиворечивости. Пассивный кэш, основной задачей которого является простое хранение данных, не проявляет собственной инициативы вообще — всякий раз, когда процесс запрашивает данные из общей памяти, пассивный кэш сохраняет полученные значения, и при следующем запросе тех же данных передаёт их процессу без обращения к общему ресурсу. Пассивный кэш допускает рассогласованность копий данных в системе, поэтому не обеспечивает существование глобального состояния. Бессмысленно говорить о мгновенном изменении несуществующего состояния, поэтому копии данных в различных кэшах не устаревают сразу после изменения одной из них и не требуют синхронного обновления.

В достаточно сложной программно-аппаратной системе, как правило, можно выделить несколько уровней или слоёв. Выделим 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 его программный интерфейс — API, который обычно обеспечивается сервером локальной базы данных или операционной системой.

Определение 2.1. Операция называется атомарной на данном уровне, если для остальной части системы она выглядит мгновенной, то есть если во время выполнения данной операции никакая другая операция с участием этих данных невозможна.

Алгоритмы функционирования пассивного кэша конкретизируются выбранной моделью непротиворечивости данных. Так, если выбрана модель хранения данных в виде «ключ-значение», где обеспечена атомарность записи отдельной пары, то этого достаточно для поддержки непротиворечивости на уровне пар — поскольку значения разных пар не зависят друг от друга и связей между ними нет, то

в результате встречи двух конфликтующих версий одной и той же пары, одна из них будет целиком потеряна в результате записи другой без каких-либо последствий для непротиворечивости. Если объекты данных, которыми оперирует приложение, более сложны и состоят из нескольких пар «ключ-значение», непротиворечивость операций с такими наборами пар необходимо обеспечивать дополнительно. Эти дополнительные средства обеспечения непротиворечивости как раз и будут составлять спецификацию протокола когерентности.

3. Архитектура MPP-СУБД

Каждый уровень распределённой системы имеет собственный протокол, регламентирующий интерфейс взаимодействия между объектами, расположенными на разных узлах. Совокупность этих протоколов образует стек — иерархически организованный набор протоколов, достаточный для организации взаимодействия узлов в сети [6]. Полное описание стека протоколов, необходимого для создания MPP-системы, в данной работе не предполагается — во-первых, это представляет слишком сложную задачу, а во-вторых, такое описание накладывало бы необоснованные ограничения на архитектуру системы.

Пассивный кэш, как архитектурное воплощение протокола когерентности, образует уровень между предметным слоем, содержащим объекты предметной области, и низлежащими уровнями ядра, обеспечивающими доступ к общей памяти. Эскиз архитектуры представлен на рис. 2.

- На уровне *модели предметной области* данные представляются в произвольной форме, процесс имеет к ним свободный доступ. Структура этих данных определяется самим процессом, чаще всего данные представляются в виде объектов.
- На уровне *трансляции адресов* организуется единое виртуальное адресное пространство неограниченного объёма, не все диапазоны которого разрешаются — прямой доступ по таким адресам запрещён и приводит к ошибке. Те же данные вместе с возможной служебной информацией представлены здесь в форме объектов данных, которые могут содержать методы динамической загрузки содержимого.
- На уровне *представления данных* физически хранится локальное содержимое объектов данных в виде более мелких структур данных, организуется наполнение этих структур данными из пакетов и поддержка непротиворечивости. Пассивный кэш имеет механизм

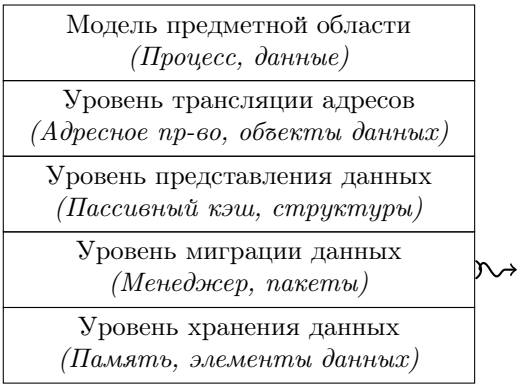

Рис. 2. Уровни MPP-системы с пассивным кэшем

вытеснения редко используемых данных, удаляющий локальное содержимое лишних объектов данных — если только они не были изменены текущей транзакцией.

- На уровне *миграции данных* менеджер выполняет размещение, репликацию и поиск данных по узлам системы в форме пакетов. При чтении проверяется целостность пакетов и отсеиваются те, элементы которых повреждены в результате сбоев или отказа узлов (и поэтому более сильная проверка атомарности здесь бесполезна).
- На уровне *хранения данных* физически хранятся разделяемые данные в форме элементов данных, снабжённых идентификатором версии. Операции с отдельным элементом данных атомарны.

Процесс имеет доступ к единому виртуальному адресному пространству системы: возможность запросить выделение блока виртуальной памяти под новые данные, освободить виртуальную память, а также обратиться по прямому адресу в виртуальной памяти. Механизм трансляции адресов преобразует виртуальный адрес в физический адрес в пассивном кэше (виртуальный адрес элемента данных фактически является его уникальным идентификатором). Если необходимые процессу данные уже содержатся в пассивном кэше, то они передаются процессу. В противном случае пассивный кэш транслирует запрос на получение данных менеджеру, который определяет их местоположение в системе и либо перенаправляет запрос другим узлам, либо считывает данные из памяти. После получения данные сохраняются в пассивном кэше, и далее всё происходит по описанному

выше сценарию.

Протокол когерентности должен ограничивать коммуникации между процессами с целью не допустить одновременного использования рассогласованных данных, способного привести к возникновению противоречия. Поэтому любой процесс в системе, в том числе — клиентский, если он имеет возможность обращаться к нескольким процессам, должен осуществлять свои коммуникации в соответствии с протоколом когерентности. На практике это проще реализовать, предусмотрев на клиенте уровень пассивного кэша.

С точки зрения приложения принцип работы с разделяемыми данными совершенно 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, позволяющими подсистеме кэша организовать безопасный доступ к этим данным. Находясь внутри такой помеченной группы операций, которую обычно называют критической секцией или транзакцией, процесс может быть уверен, что его работа с разделяемыми данными происходит на основе принятого в системе протокола когерентности. В случае возникновения неустранимой ошибки транзакция может быть аварийно прервана системой, а все внесённые ею изменения — отменены.

Имеет место следующая совместимость архитектур: корректно написанное приложение для системы с пассивным кэшем без переделки переносится на систему с активным кэшем. Обратная совместимость не гарантируется, кроме случая, когда возможно переформулировать саму задачу без привлечения понятия текущего состояния.

4. Протокол когерентности

Для каждого из конкурирующих процессов протокол когерентности должен обеспечить непротиворечивую эволюцию соответствующего состояния путём ограничения доступа к несогласованным данным достаточного, чтобы не допускать противоречия. Согласование версий данных производится в пассивном кэше при сборке структур данных из пакетов. Необходимым условием работоспособности метода является атомарность операций с объектами данных. По сравнению с общей памятью SMP-системы, имеющей все версии элементов данных в своём распоряжении, в распределённой памяти MPP-системы эти элементы могут оказаться разбросаны по разным узлам, в

том числе — вышедшим из строя, поэтому обеспечение атомарности операций становится нетривиальной задачей.

Протокол когерентности не регламентирует функции компонентов, лежащих вне уровня представления данных, однако он предъявляет определённые требования к поступающим от них данным, распространяя таким образом своё косвенное влияние и на другие уровни MPP-системы.

- На уровне *модели предметной области* (рис. 2) влияние протокола когерентности никак не ощущается. Единственное проявление — откат изменений внутри критической секции при возникновении исключения — в обычной системе он может быть обусловлен естественным действием конкурирующих процессов. Выделение памяти влечёт создание элементов данных, освобождение — их удаление, адрес виртуальной памяти является уникальным идентификатором элемента данных и может выделяться из пула доступных процессу адресов. Виртуальный адрес элемента данных не определяет места его физического хранения — этими вопросами ведает менеджер на уровне миграции данных. Виртуальное адресное пространство неограниченно, хотя доступность диапазонов адресов зависит от наличия работающих узлов.

Замечание 4.1. *Никаких внешних особенностей у такой системы нет, и обычные программы могут запускаться на ней без переделки.*

- На уровне *трансляции адресов* протокол когерентности вынуждает использовать объекты данных для хранения содержимого памяти. Объект данных — это чистая форма, оболочка данных. При обращении по виртуальному адресу объекта данных он инициирует загрузку своего содержимого и только потом становится доступен — это называется загрузкой по требованию, *lazy load* [7]. Содержимое объекта данных, если оно не было изменено, может свободно вытесняться из памяти. Для доступа к содержимому необходимо транслировать виртуальный адрес объекта данных в физические адреса составляющих его структур данных, для чего можно использовать хэш-функцию, либо завести служебное поле объекта данных, в котором хранить физические адреса соответствующих структур.
- На уровне *представления данных*, где функционирует пассивный кэш, протокол когерентности регламентирует процедуры размещения данных из пакетов по структурам и объединения элементов данных

в новый пакет для сохранения изменений. Спецификация протокола когерентности определяется моделью непротиворечивости, принятой в системе.

- На уровне *миграции данных* протокол когерентности предусматривает наличие менеджера, занимающегося размещением и сбором пакетов по узлам системы. В целях быстродействия на этот уровень делегируется задача фильтрации версий: в запросе указывается список элементов, прочитанных текущей транзакцией, и несовместные версии отсеиваются заранее. Некоторые пакеты могут оказаться повреждены в результате сбоев, что делает бесполезными попытки обеспечить атомарность объектов данных при записи. Поэтому при чтении контролируется целостность пакетов, повреждённые пакеты игнорируются — для пассивного кэша, допускающего потерю подтверждённых изменений, это эквивалентная замена. Задача размещения может решаться как с оптимизацией — хорошо согласуется с идеей объектов подход [8], так и без оной — каждый элемент данных на основе его виртуального адреса (уникального идентификатора) прикрепляется к своему узлу приписки, который служит центром притяжения изменений и обязательно содержит одну из версий элемента данных, хотя не обязательно последнюю.
- На уровне *хранения данных* протокол когерентности требует атомарности операций с элементами данных. Кроме того, если алгоритмы менеджера на уровне миграции данных слабы и не обеспечивают эффективного поиска необходимых элементов, то может оказаться полезным хранить в памяти узла несколько версий элемента данных — например, на узле приписки.

Пассивный кэш, руководствуясь потребностями текущей транзакции, формирует для менеджера параметры отбора элементов данных, включающие диапазоны значений этих элементов данных или их виртуальные адреса (или уникальные идентификаторы), а также примерный временной промежуток, за который будут отбираться версии элементов данных. Менеджер собирает запрошенные данные по узлам системы, при этом в итоговом множестве могут оказаться несколько версий каждого элемента данных (или не оказаться ни одной). Пассивный кэш должен выбрать из предлагаемых версий наиболее подходящую и разместить её в локальной памяти, или выполнить повторный запрос к менеджеру с другими параметрами, или инициировать откат текущей транзакции, если данные получить не удалось.

5. Заключение

В работе предложена концепция пассивного кэша, позволяющего отказаться от синхронизации процессов в классе задач, не требующих наличия в системе глобального состояния. На его основе предложена архитектура высокопроизводительной СУБД в MPP-системе. Разработаны базовые принципы представления данных в такой системе и высокоуровневые алгоритмы их обработки.

Список литературы

[1] R. J. Chevance, *Server Architectures: Multiprocessors, Clusters, Parallel Systems, Web Servers, and Storage Solutions*, Elsevier Digital Press, Amsterdam, 2005, 739 pp. ↑ 195.

[2] К. Хьюз, Т. Хьюз, *Параллельное и распределённое программирование с использованием C++*, Пер. с англ., Издательский дом «Вильямс», М., 2004, 672 с. ↑ 195.

[3] Э. Таненбаум, М. ван Стеен, *Распределённые системы. Принципы и парадигмы*, Пер. с англ., Серия «Классика computer science», Питер, СПб., 2003, 877 с. ↑ 196.

[4] J. Handy, *The Cache Memory Book*, 2nd ed., Academic Press Inc., San Diego, 1998, 229 pp. ↑ 197.

[5] O. Babao, K. Marzullo, “Consistent global states of distributed systems: Fundamental concepts and mechanisms”, *Distributed Systems*. v. 4, Frontier Series, Addison-Wesley, New York, 2003, pp. 55–96 ↑ 198.

[6] В. Г. Олифер, Н. А. Олифер, *Компьютерные сети. Принципы, технологии, протоколы*, 4-е изд., Питер, СПб., 2010, 944 с. ↑ 199.

[7] М. Фаулер, Д. Райс, М. Фоммел, Э. Хайет, Р. Ми, Р. Стаффорд, *Шаблоны корпоративных приложений*, Пер. с англ., Signature Series, Вильямс, М., 2010, 544 с. ↑ 202.

[8] C. Curino, E. Jones, Y. Zhang, S. Madden, “Schism: a Workload-Driven Approach to Database Replication and Partitioning”, 36th International Conference on Very Large Data Bases (Singapore, 2010, September 13–17), *Proceedings of the VLDB Endowment*, **3**:1, pp. 48–57 ↑ 203.

Рекомендовал к публикации *д.ф.-м.н. Н. Н. Непейвода*

Об авторе:

Алексей Александрович Демидов

Младший научный сотрудник Исследовательского центра искусственного интеллекта ИПС им. А. К. Айламазяна РАН.

e-mail: alex@dem.botik.ru

Образец ссылки на эту публикацию:

А. А. Демидов. *Об особенностях организации СУБД в MPP-системе* // Программные системы: теория и приложения: электрон. научн. журн. 2014. Т. 5, № 4(22), с. 195–205.

URL http://psta.psiras.ru/read/psta2014_4_195-205.pdf

Aleksey Demidov. *About the features of DBMS architecture in MPP-system.*

Abstract. Today a lot of effort is worldwide put into development exaplops computers, which will be able to perform 10^{18} floating-point operations per second. Effective performance of the cluster on real tasks is considerably less of that value due to synchronisation costs of concurrent processes. This paper introduces a concept of passive cache, which eliminates any need for process synchronization in some tasks, and proposes a system architecture of high performance distributed DBMS, based on the passive cache. (in Russian).

Key Words and Phrases: parallel and distributed computing, very large databases.