УДК 004.72

ПРОТОКОЛЫ ОБЕСПЕЧЕНИЯ КОГЕРЕНТНОСТИ КЭШ-ПАМЯТИ В РАСПРЕДЕЛЕННЫХ СИСТЕМАХ. ИХ ПРИМЕНЕНИЕ В БИЛЛИНГОВЫХ СИСТЕМАХ

Дерикьянц Павел Павлович
аспирант
Кубанский государственный аграрный университет, Краснодар, Россия

Освещена тема применения протоколов обеспечения когерентности данных в оперативной памяти модулей биллинговых систем. Рассмотрены проблемы и преимущества использования данных протоколов

Ключевые слова: ПРОТОКОЛЫ, КОГЕРЕНТНОСТЬ, КЭШ-ПАМЯТЬ, РАСПРЕДЕЛЕННЫЕ БИЛЛИНГОВЫЕ СИСТЕМЫ

UDC 004.72

REPORTS OF COHERENT MAINTENANCE OF CACHE-MEMORY IN THE DISTRIBUTED SYSTEMS. THEIR APPLICATION IN BILLING SYSTEMS

Derikjanz Pavel Pavlovich
post-graduate student
Kuban State Agrarian University, Krasnodar, Russia

The theme of reports of coherent maintenance of data in operative memory of billing systems modules application is taken up. Problems and advantages of use of the given reports are considered

Keywords: REPORTS, COHERENT, CACHE-MEMORY, DISTRIBUTED BILLING SYSTEMS

Осознав то, что объемы данных, которые необходимо обрабатывать, достигнут огромных размеров и обрабатывать их приемлемой скоростью, используя внешнюю память, вскоре будет невозможно, разработчики АСР начали искать возможные пути выхода. Идею одного из таких решений почерпнули из области проектирования микропроцессорных систем и систем с распределенной памятью. Она вылилась в реализацию отдельных модулей БС, использующих собственный или разделяемый с другими модулями кэш-данных. Это позволило резко увеличить время отклика и сократить нагрузку на СУБД.

Далее будут описаны модели, позволяющие обеспечить актуальность данных, находящихся в различных экземплярах оперативной памяти каждого модуля. И так как эти модели родились в мультипроцессорных системах с разделяемой памятью и заимствованы именно оттуда, то рассматривать мы их будем применительно именно к таким системам, делая необходимые поправки и уточнения, где это будет необходимо.

Для модификации удаленных копий данных используются коммуникационный протокол и протокол согласованности (когерентности)

состояния памяти. Первый применяется для доставки изменений, а второй призван предотвращать использование копий данных, подвергшихся модификации в другом процессоре или сервере в распределенной сети.

Конкретную реализацию разделяемой памяти характеризуют:

уровень реализации (аппаратный, программный, аппаратно-программный);

структура разделяемых данных (слово, кэш-строка, страница, сегмент, фрагмент, объект и т.д.);

модель состоятельности памяти, определяющая допустимые последовательности доступа в память (SRSW — «один читатель/один писатель», MRSW — «много читателей/один писатель», MRMW — «много читателей/много писателей»);

политика обеспечения когерентности данных (модификация, объявление несостоятельными);

механизм управления распределением памяти и размещением данных (централизованный/распределенный, статический/динамический).

В данный момент используется несколько моделей состоятельности памяти.

Строгая (strict): каждая операция чтения возвращает последнее записанное значение.

Последовательная (sequential): все процессоры в системе наблюдают один и тот же порядок выполнения операций записи и чтения (процессор/сервер, выполняющий запись приостанавливается до получения подтверждений об объявлении несостоятельными всех копий модифицируемых данных или о модификации этих копий).

Процессорная (processor): наблюдаемый в двух процессорах порядок выполнения операций чтения/записи может не совпадать, но порядок выполнения записей, производимых каждым процессором, должен быть одним и тем же.

Слабая (weak): вводящая разграничение между обычным и синхронизованным доступами в память при выполнении требования состоятельности памяти только при доступах в точках синхронизации и разрешением несогласованности данных вне этих точек.

Свободная (release): уточненная модель слабой состоятельности, требующая состоятельности памяти только между парой операций синхронизации acquire-release (acquire — открывает критический интервал и обеспечивает исключительный доступ процессора к разделяемым данным, release — закрывает критический интервал, разрешая всем процессорам доступ к разделяемым данным). При этом операции синхронизации, выполняемые в разных процессорах, должны удовлетворять модели последовательной или процессорной состоятельности памяти.

Неторопливо-свободная (lazy release): модель свободной состоятельности с отложенным до момента непосредственного обращения распространением модификаций разделяемой памяти так, что при операции acquire передаются все предшествующие результаты команд записи, которые выполнялись до наступившего момента синхронизации.

Нетерпеливо-свободная (eager release): модель свободной состоятельности с передачей изменений разделяемой памяти при реализации операции release, безотносительно к тому, когда эти изменения будут востребованы.

Интервально свободная (scope): модель свободной состоятельности с разбиением исполнения на глобальные и локальные интервалы и гарантированием состоятельности разделяемых данных для всех процессоров только в конце каждого глобального интервала, а также гарантированием состоятельности разделяемых данных для последовательных локальных интервалов внутри каждого процессора.

Последовательно-свободная (entry): вариант модели свободной состоятельности, который требует доступа к разделяемым данным, защищенного синхронизирующими переменными, значения которых должны модифицироваться согласно модели последовательной согласованности (все процессоры должны иметь одну и ту же последовательность обращений к каждой из этих переменных).

Так как АСР и ИБС это программные продукты и не имеют аппаратной платформы, то рассмотрение моделей обеспечения когерентности кэш-данных на аппаратной основе мы опустим и перейдем непосредственно к рассмотрению программных реализаций.

Системы на базе передачи сообщений.

Сегодня системы на базе коммерчески доступных материнских плат и сетевых интерфейсов широко используются как дешевая аппаратная альтернатива системам с аппаратной реализацией распределенной разделяемой памяти. Протоколы когерентности памяти при программной реализации служат надстройкой над аппаратными средствами передачи сообщений. Это заведомо создает проблемы с недостаточной пропускной способностью при создании удаленных копий данных, передаче изменений и миграции данных, однако ничем не ограничивает разнообразие протоколов согласованности состояния памяти, позволяя учитывать особенности прикладных программ.

Строгая когерентность требует, чтобы каждый вычислительный модуль видел все доступы в память в одном и том же порядке. Однако обычно в параллельных программах используются операции синхронизации для управления доступом к разделяемым переменным или для установления порядка выполнения операций. В первом случае используются операции синхронизации для установки критических интервалов, во втором — барьерная синхронизация. Программа может не учитывать изменения разделяемых переменных вне критического

интервала, что позволяет использовать модели свободной состоятельности памяти, поэтому ослабленная модель состоятельности памяти различает нормальный и синхронизированный доступ в память.

Модели ослабленной состоятельности памяти позволяют повысить производительность и расширяют спектр аппаратных платформ с различными коммуникационными протоколами, допускающими программную организацию разделяемой памяти, например, при определенных ограничениях на последовательности команд записи/чтения в прикладных программах. Реализация этих ограничений возлагается на примитивы синхронизации, посредством которых программист формирует допустимые последовательности команд.

В большинстве программных реализаций наименьшими элементами разделяемой памяти служат страницы, что обусловлено использованием существующих аппаратных средств организации виртуальной памяти для обнаружения записи в разделяемые страницы (например, запись в страницу, доступную только для чтения, или несостоятельную страницу). Однако сравнительно большой размер разделяемых страниц создает проблему ложного разделения, при котором конфликтными признаются модификации разными процессорами различных ячеек памяти одной страницы, не конфликтующие между собой. Решением служит либо введение логических разделяемых единиц (объектов, типов данных), либо отображение многих виртуальных страниц на одну физическую.

Программный уровень реализации разделяемой памяти может базироваться либо на компиляторе, который выявляет доступы к разделяемым переменным и вставляет в исполняемый код вызовы примитивов синхронизации и процедур обеспечения когерентности, либо на библиотеке процедур, статически или динамически связываемых с прикладной программой. В любом случае, обращение к разделяемым данным, расположенным в другом вычислительном модуле, вызывает

пересылку через коммуникационную среду с обеспечением когерентности всех копий одних и тех же данных в разных модулях. Обнаружение модифицированных данных требует либо внесения в код программы дополнительных команд для установки флагов модификации с целью указания измененных данных, либо создания перед первой записью в страницу копии разделяемых данных, которая используется по завершении интервала вычислений процесса для сравнения текущего состояния данных страницы с их сохраненной копией. Для выявления моментов модификации разделяемых данных вводятся отметки логического времени, привязывающие изменения к интервалам синхронизированного доступа к разделяемым данным.

Эффективность программных реализаций распределенной разделяемой памяти ограничивается большим размером разделяемых страниц, временем выполнения операций синхронизации, включая определение модифицированных данных, а также задержкой и пропускной способностью коммуникационной среды. В программных реализациях задержка доступа в удаленную память на порядок и более превосходит задержку в аппаратных реализациях, поэтому сравнимая производительность может быть достигнута лишь на некоторых классах задач.

При программной реализации разделяемая память доступна только ограниченному спектру прикладных программ, написанных с использованием четко определенных языковых конструкций, ориентированных на поддержку выявления компилятором записей в разделяемые переменные или использование библиотек для доступа к ним. Не все ячейки памяти одинаково доступны, поэтому разделяемая память не может использоваться для реализации синхронизации. Для этого используется ограниченное число примитивов синхронизации, например, критические интервалы (lock) и барьеры (barrier), реализованные

посредством передачи сообщений. Стремление к повышению эффективности ведет к применению моделей состоятельности памяти, значительно отличающихся от используемых в системах с аппаратной реализацией разделения памяти. Программы для систем с аппаратной разделяемой памятью трудно переносить на системы с программной разделяемой памятью.

Модель свободной состоятельности памяти

Для реализации модели свободной состоятельности могут быть использованы различные протоколы, в том числе неторопливой (lazy release consistency — LRC) и нетерпеливой (eager release consistency — ERC) свободной состоятельности. В их рамках задержки, вызванные передачей модифицируемых данных, скрываются за счет буферизации записей или их конвейеризации в пределах критических интервалов. При этом в критическом интервале возможно объединение групп записей в одну, что повышает эффективность их передачи.

Вычислительные модули выполняют обычные операции чтения/записи и операции синхронизации acquire и release. Согласно требованиям модели необходимо обеспечить, чтобы все модули видели один и тот же порядок выполнения операций синхронизации. Одним из механизмов реализации частичной упорядоченности этих операций служит введение в каждом модуле локальной отметки времени и вектора отметок времени. При свободной состоятельности время протекания процесса в каждом модуле делится на интервалы. После выполнения каждой операции acquire или release начинается новый интервал, а локальная отметка времени модуля увеличивается на 1. Вектор отметок времени служит для поддержки временных зависимостей между всеми модулями: его компоненты однозначно сопоставляются модулям. Каждая модификация памяти описывается отметкой записи write-notice. Когда вычислительный модуль выполняет операцию записи в разделяемую

страницу, он создает отметку записи для этой страницы, помещая в эту отметку значение локальной отметки времени интервала, адрес и значение изменяемого слова. Модули формируют список отметок записи для каждой страницы.

Реализация протокола LRC

При реализации модели LRC операция синхронизации acquire, открывающая критический интервал, обеспечивает состоятельность всех разделяемых страниц для продолжения выполнения критического интервала. Критический интервал завершается операцией release, после чего всем вычислительным модулям становятся доступны переменные, модифицированные в этом критическом интервале. Однако отметки записи для модифицированных разделяемых данных завершаемого критического интервала, необходимые для поддержания состоятельности страниц, могут быть получены другим модулем только при выполнении им операции acquire.

Когда вычислительный модуль выполняет операцию acquire, он посылает модулю, выполнившему перед ним операцию release, копию своего вектора отметок времени. Тот в ответ высылает отметки записей, отсутствующие в модуле, выполняющем операцию acquire. В пересылаемом списке отметок записи имеются сведения не только о записях, произведенных самим вычислительным модулем, но и о записях, о которых ему стало известно в результате выполнения им операций синхронизации с другими модулями.

После получения отметок записи вычислительный модуль запоминает присланные отметки записей и модифицирует вектор отметок времени, присваивая каждому компоненту максимум из собственного значения и из значений полученного вектора. Он также делает несостоятельными копии всех страниц, для которых получены отметки записей, так как эти отметки содержат сведения о модификации страниц после последнего

предыдущего согласования копий. При первом доступе к несостоятельной копии страницы вырабатывается сигнал «промах», обозначающий необходимость модификации страницы. Возможны, по крайней мере, два варианта доставки состоятельной страницы. Первый вариант основан на распределенном хранении внесенных в страницу изменений, а второй — на поддержке резидентной копии каждой страницы и ее пересылке в затребовавший модуль.

При распределенном хранении изменений при первом доступе к странице по записи создается копия этой страницы. Затем в течение интервала страница может многократно модифицироваться. При выполнении операции release текущее состояние страницы сопоставляется с копией и вычисляется «разность» diff, совокупность адресов и новых значений измененных ячеек. Модуль связывает diff с отметкой записи. Когда модуль нуждается в состоятельной копии страницы, он просматривает имеющийся у него список отметок записей и определяет, какие diff ему необходимы. Далее он запрашивает из других модулей требуемые ему diff и после их получения создает состоятельную копию страницы, применяя diff в требуемом порядке.

При реализации протокола LRC разность diff формируется и передается по явному запросу, а в случае ERC отметки записи передаются сразу после своего формирования. Недостаток LRC состоит в необходимости «сборки мусора» для удаления невостребованных копий страниц и diff. Приходится решать, до каких пор хранить эти данные.

При поддержке резидентной копии каждой страницы в конце интервала при выполнении release вычисленные diff для модифицированных страниц высылаются в резидентные для них вычислительные модули. Резидентные модули, используя получаемые diff, модернизируют соответствующие страницы. Недостатком этого протокола служит то, что при обращении многих модулей с запросом копии

страницы возникает «узкое горло» при последовательной отсылке страницы каждому модулю.

Реализация протокола ERC

При реализации протокола ERC записи буферизируются до тех пор, пока не потребуется сделать их доступными при очередном выполнении операции release; для этого используется трансляционная рассылка измененных данных. Основное отличие протокола ERC от LRC состоит в том, что состоятельность страниц поддерживается операцией release, а не acquire. При этом ERC может выполнять лишние передачи данных для поддержания состоятельности копий страниц, даже если в некоторых модулях в них не будет обращений. Поэтому недостаток протокола состоит в том, что после обращения к несостоятельной странице требуется ожидание, пока необходимая разность diff будет вычислена и передана этому модулю.

Предотвращение использования несостоятельных данных основывается на отметках логического времени. Получение состоятельных копий страниц вместо применения diff основано на механизме записей в удаленную память и чтения из удаленной памяти. В каждом вычислительном модуле каждой странице сопоставляется вектор отметок времени слива, где отмечаются все изменения страниц, наблюдаемых модулем. В каждом модуле разделяемым страницам памяти сопоставляется вектор отметок синхронизации, устанавливающий, какие преобразования конкретной страницы необходимы, прежде чем модуль может получить к ней доступ. Собственно, этот вектор имеет для каждой страницы смысл списка отметок записи протокола LRC. Для ускорения операции acquire каждый вычислительный модуль содержит отметку глобального времени синхронизации, имеющую максимальное значение логического времени из всех постраничных отметок этого модуля. Если элемент в векторе отметок времени слива страницы меньше, чем

соответствующий элемент вектора отметок синхронизации, то страница эта уже не состоятельна. При обратной ситуации модуль имеет более новую копию страницы по сравнению с той, которая ему необходима в соответствии с моделью состоятельности памяти.

Протокол Copyset-2 поддерживает одновременно состоятельными разделяемые страницы не более чем в двух модулях. Аппаратные/программные средства удаленной записи в память копируют записи в память этих модулей; при обращении к разделяемой странице со стороны третьего модуля копия страницы в одном из разделявших ее ранее модулей, объявляется несостоятельной, а второй модуль создает копию страницы для третьего и устанавливает с ним взаимное удаленное копирование разделяемой страницы. При выполнении операции release модуль увеличивает значение своего логического времени и использует его значение для удаленного изменения соответствующих элементов векторов отметок времени слива для страниц, которые подверглись модификации в завершаемом интервале.

При инициации выполнения операции acquire вычислительный модуль посылает другому модулю, последним исполнившим операцию release, свой текущий глобальный вектор отметок времени синхронизации, как это делается в протоколе LRC. Последний, получив этот вектор, устанавливает, какие страницы несостоятельны в запрашивающем модуле, и высылает отметки записи для всех этих страниц. По получении этих отметок первый модуль, продолжая выполнять acquire, модифицирует свой вектор отметок синхронизации, присваивая элементам, соответствующим страницам, для которых поступили отметки записи, максимальное значение логического времени из отметок записи и текущего значения этих элементов. Если каждый элемент вновь сформированного вектора отметок синхронизации не больше соответствующего элемента вектора отметок времени слива, то модуль узнает, что у него все страницы

состоятельны. В противном случае, если не все страницы состоятельны, они помечаются как отсутствующие. Их модификация еще не завершена, но уже инициирована. Если модуль при доступе к странице обнаруживает ее несостоятельность, то вновь сравнивает элементы вектора отметок синхронизации с элементами вектора отметок времени слива. Если обнаруживается состоятельность соответствующей страницы, то она помечается как состоятельная, а модуль продолжает выполнение интервала. В противном случае модуль ожидает, пока страница станет состоятельной, сравнивая элементы названных векторов.

Протокол Copyset-N использует резидентное размещение разделяемых страниц в вычислительных модулях. Другие модули, имеющие копию страницы, настраивают отображение копии этой страницы в резидентную страницу для автоматического изменения последней при модификации копии. При выполнении операции release модуль увеличивает значение своего логического времени и использует это значение для удаленного изменения соответствующих элементов векторов отметок времени слива в каждой из резидентных страниц, которые подверглись модификации в завершаемом интервале. При выполнении операции acquire модуль действует так же, как в протоколе Copyset-2, однако не ждет, пока страницы станут состоятельными, а запрашивает их из резидентных модулей. При этом запрашивающий модуль получает также вектор отметок времени слива, что позволяет ему оценить состоятельность поступившей страницы. Протокол AURC сочетает оба протокола, работая в случае двух копий как Copyset-2, а при большем числе копий — как Copyset-N.

Надо отметить, что применение полноценного механизма поддержания когерентности кэш-данных для биллинговых систем, работающих с использованием индивидуальных кэшей, вовсе не является обязательным. Вполне приемлем механизм, когда один модуль записывает

измененные данные в БД. А все остальные модули узнают об этих изменениях, периодически зачитывая специальную таблицу, в которой отражается какой атрибут, какого абонента, был изменен. Это довольно простой в реализации способ, но когерентности данных в любой момент времени он не обеспечивает, так как нет никакой гарантии, что за время между очередным зачитыванием изменений не произойдет обращение к атрибуту, измененному другим модулем. А если принимать во внимание, что такое решение может применяться на довольно большой абонентской базе, то в периоды пиковой нагрузки, СУБД может не отвечать своевременно на запрос и в результате период зачитывания изменений может неопределенно возрастать. Последствия такой ситуации можно описать на примере. Допустим, prepaid абонент имел баланс 54 руб. 13 коп. и в период, когда возникла задержка с зачитыванием изменений из такой синхронизационной таблицы, ему была начислена абонентская платя за использование определенной услуги, которая опустила его баланс до 4 руб. 13 коп. Вполне вероятно, что этот абонент может начать пользоваться услугами сотовой связи. При этом модуль тарификации, еще не скорректировав баланс после начисления абонентской платы, позволит абоненту пользоваться услугами на сумму 54 руб. 13 коп. После чего, скорректировав баланс на сумму абонентской платы, он может получиться отрицательным, а это уже дебиторская задолженность, которую в этом случае оператор берет на себя. В реальных условиях эта сумма может возрастать до миллионов.

Специфика работы кэш-памяти для программных модулей БС ставит перед ней те же требования, что и перед кэш-памятью мультипроцессорных систем. Сходство заключается в том, что при использовании архитектуры с индивидуальной кэш-памятью можно провести аналогию с локальным кэшем процессора или локальной памятью отдельного сервера, входящего в состав распределенной

вычислительной системы. Это делает алгоритмы и протоколы, применяемые в мультипроцессорных системах на основе передачи сообщений, пригодными для использования их в сфере биллинговых решений. Основная проблема, стоящая перед разработчиками, это выбор той или иной модели в качестве средства для поддержания актуальности данных во всех модулях. Хотя надо отметить, что проблема быстродействия и времени получения данных из удаленной копии, для БС играет меньшую роль, чем для микропроцессорных систем, так как ситуация возникновения «промаха» при работе модулей довольно редкая. Объясняется это тем, что при работе, модули БС обращаются к атрибутам разных абонентов. И вероятность обращения двух или более модулей, за время меньшее периоду синхронизации данных, к атрибутам одного и того же абонента очень мала. Проблемы могут возникнуть в период массового изменения данных по каждому абоненту. Допустим в период начисления абонентской платы или проведении биллинга. В этом случае при использовании протокола ERC резко возрастает нагрузка на каналы передачи данных, что в некоторых случаях крайне не желательно. Протокол ERC, при большой абонентской базе, предъявляет более жесткие требования к коммутационной среде. Дело в том, что при наличии большого количества активных абонентов, которые совершают вызовы, пополняют счет и т.д., генерация сообщений рассылаемых всем остальным модулям для внесения в них изменений довольно велико и это может повлечь серьезные задержки при недостаточной пропускной способности сети. Рассмотрим конкретный пример. Допустим, имеется абонентская база данных, численность которой составляет 4 млн. Топология сети имеет вид, представленный на рис. 1.

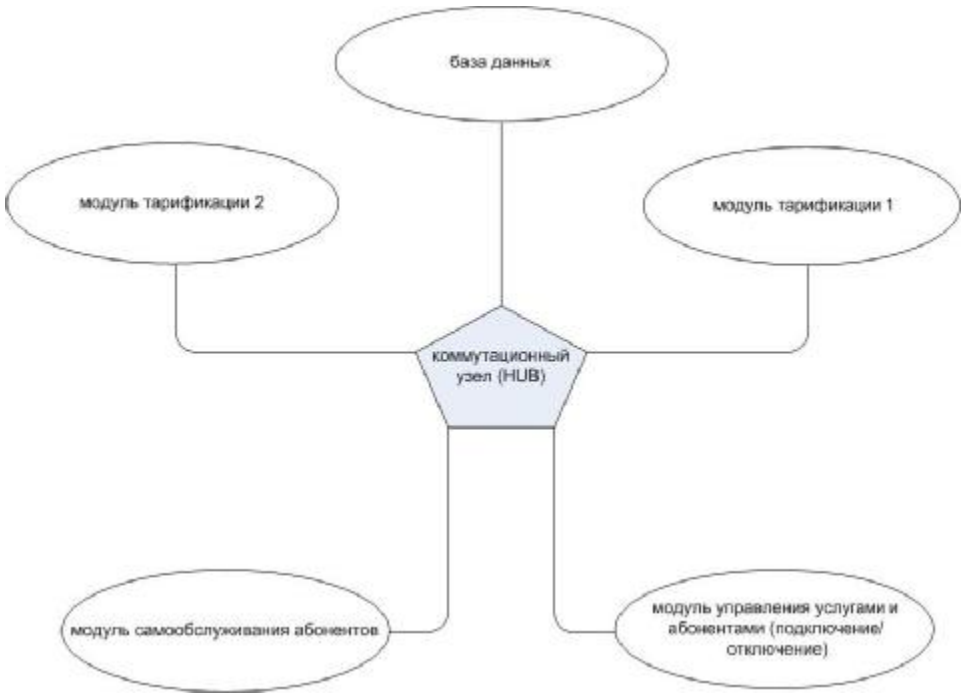

Рис.1 Топология сети биллинговой системы с 4-мя модулями.

В среднем для того, чтобы передать актуальные данные о конкретном атрибуте одного абонента (id абонента, значение атрибута, а так же служебная информация метка времени и т.п.) необходимо около 512 байт. Такое количество абонентов в часы пиковой нагрузки совершает способно совершать до 2000 «событий» в секунду, которые потребуют обновления данных об абоненте (изменение баланса, подключение/отключение услуги и т.п.). Путем нехитрых вычислений можно подсчитать, какой объем трафика понадобиться передать за секунду, чтобы обеспечить когерентность данных во всех копиях работающих модулей. 512*2000*4=4 096 000 байт. Понятно, что даже для сети с пропускной способностью 100Мб/с это не проблема. Однако, на практике число модулей, которые работают у операторов намного больше. Связано это с тем, что, например, два модуля тарификации не могут обеспечить нужную скорость тарификации. А при использовании конвергентного биллинга в сети работают еще и модули, обеспечивающие принятие решения о предоставлении абоненту возможности пользования услугой. Другими словами в реальных условиях у сотовых операторов в сети может работать

до 12 модулей различного назначения, использующих копию кэш-данных в своей оперативной памяти. При таких условиях объем трафика может возрастать до 12,5 Мбайт/с и потребуют наращивания пропускной способности сети. Ситуация усугубляется еще и тем, что эти же каналы передачи данных используются для передачи обработанной информации от модуля к модулю и в БД биллинга.

Проблема с генерацией большого количества сообщений заключается еще и в том, что операции их формирования, приема и применения изменений довольно дорогие и требуют значительного процессорного времени. Связано это в первую очередь с тем, что объем требуемый для хранения информации по всем абонентам может достигать нескольких гигабайт и поиск нужного на таких объемах может достигать пары секунд. При этом один из процессоров занят процессом поиска нужных данных. В этот момент производительность всего сервера в целом будет «проседать». Исходя из этого, можно сделать вывод, что использование алгоритма ERC на больших абонентских базах практически невозможно, так как количество сообщений при его использовании довольно велико, а операции поиска по оперативной памяти очень дорогие. Однако он безболезненно может применяться на небольших абонентских базах, численностью до 1 млн. Нагрузка на коммутационные каналы и объем кэшируемых данных в оперативной памяти в этом случае не велики. Поиск нужного абонента и внесение изменений в его атрибуты на таком объеме не занимает много процессорного времени. Для абонентских баз с численностью более 1 млн. предпочтительнее применять протокол LRC. Его использование позволяет снизить объем передаваемых сообщений в связи с этим он более подходит для поддержания копий данных в оперативной памяти каждого модуля в когерентном состоянии.

Но нужно заметить, что протоколы когерентности в данном случае применяются не в мультипроцессорных системах, а в системе массового

обслуживания и некоторые процессы и события, происходящие внутри нее, поддаются описанию и прогнозированию. Так, например, с легкостью можно спрогнозировать рост числа звонков в пиковые часы, праздники, оценить рост нагрузки при росте числа абонентов и т.п. Таким образом, проблему с пропускной способностью сети и актуальностью данных в копии каждого модуля можно снизить путем ввода в нее механизмов управления алгоритмами когерентности. Допустим, использовать в часы пиковой нагрузки алгоритм LRC для минимизации рассылки сообщений. И наоборот переключить все модули на алгоритм ERC в ночные часы, когда нагрузка на сервера минимальна.

Для обеспечения отказоустойчивости и уменьшения нагрузки на каналы передачи данных схема построения сети может выглядеть, так как представлено на рис. 2.

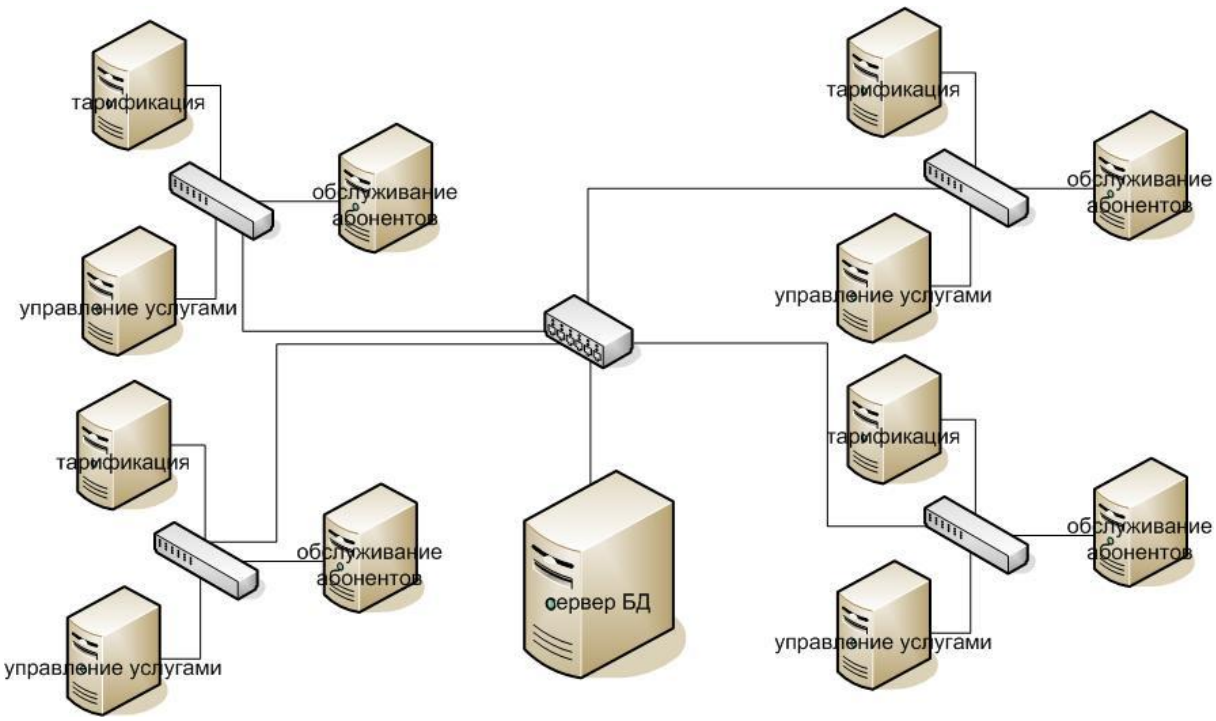

Рис. 2 Схема организации сети с использованием сегментов

Такая организация позволяет добиться снижения потока данных передаваемых через центральный узел коммутации (switch) за счет «отсечения» сообщений, которые будут передаваться внутри сегмента.

Процентное снижение, для случая представленного на рис.2, составляет приблизительно 16,7%. Кроме снижения трафика, такая схема позволяет повысить отказоустойчивость при сбое. Допустим при выходе из строя одного из сегментов, работа биллинговой системы в целом не останавливается, так как оставшиеся сегменты содержат внутри себя экземпляры модулей способных решать все задачи связанные с тарификацией, контролем над подключением/отключением услуг, самообслуживанием абонентов и т.п. Это дает возможность без потерь восстановить работоспособность сбойного сегмента. Становится возможным проведение профилактики на отдельных серверах.

Внедрение решения на основе использования кэширования данных и применения протоколов когерентности позволяет сотовому оператору минимизировать потери, связанные с дебиторской задолженностью и ошибками тарификации с использованием устаревших данных. Но в то же время оно сопряжено с определенными рисками, так как использование БД в качестве средства сигнализации об изменении атрибутов абонента хоть и не гарантирует их актуальность в каждый момент времени, но реализует практически 100%-ю доставку изменений даже при сбое. Чего нет при использовании обработки данных в оперативной памяти сервера. В том случае если по каким-либо причинам возник сбой, то это влечет за собой потерю всех данных и остановку сервиса до тех пор пока модуль не получит из базы данных все необходимые для работы данные. Если принять во внимание, что все операторы сотовой связи работают по принципу - «если неопределенность, то дать абоненту пользоваться услугой», то при отказе модуля, который отвечает за обслуживание prepaid абонентов, сразу возникнет дебиторская задолженность и компания понесет убытки. Поэтому основная проблема для компаний, использующих такое решение это отказоустойчивость. Для решения этой проблемы в логику работы всех модулей необходимо заложить

возможность работы с несколькими копиями. При штатной работе этот принцип можно использовать для балансировки нагрузки, а в случае выхода из строя динамически переключаться на работу с оставшимися копиями. Такой подход позволяет восстановить сбойный экземпляр модуля без остановки работы биллинговой системы.

Следующая проблема, возникающая при эксплуатации – это вероятность сбоя сети или нехватка пропускной способности. При этом для минимизации потерь, связанных с невозможностью поддерживать кэш данных в актуальном состоянии между всеми модулями, они должны иметь возможность работать автономно с последующим «накатом» изменений при появлении связи. Во избежание возникновения подобных сетевых проблем приоритетной схемой построения сети является схема когда, вся сеть в которой работают сервера, поделена на сегменты и каждый сегмент соединен с максимальным количеством смежных. В случае недоступности какого-либо сегмента все остальные остаются работоспособными.

Переход на работу с использованием протоколов когерентности, так же потребует от оператора изменения некоторых бизнес процессов связанных с внесением новых данных об абоненте. Отныне все операции такого рода в штатном режиме должны проходить через соответствующие модули дабы не была нарушена когерентность во всех модулях.

Заключение

Основные преимущества, которые дает применение протоколов обеспечения когерентности данных на основе передачи сообщений с технической стороны это:

1. значительное снижение нагрузки на СУБД. Она уже не принимает участие в обработке вставленных в таблицы данных, а лишь служит как приемник конечных данных

2. за счет передачи подавляющей части сообщений непосредственно от модуля к модулю по сети, позволяет снять с СУБД работу по формированию синхронизационных сообщений
3. становится возможным освободившиеся мощности выделить под другие задачи.

С экономической стороны такой принцип работы обеспечивает практически 100% защиту от возникновения дебиторской задолженности, связанных с ошибками тарификации на основе неверных данных о балансе абонента. Кроме того, он позволяет избежать задержек с реакцией системы на заказы услуг абонентов, что значительно повышает уровень их лояльности. Данный показатель является одним из важнейших для сотовых компаний.

В статье предложена схема сегментной организации сети, при которой в каждом сегменте содержится полный набор модулей, способных выполнять все задачи по обслуживанию абонентов. Это позволяет достичь необходимой отказоустойчивости и снизить нагрузку на каналы передачи данных.

Литература

1. Е. Коваленко. Система Sequent NUMA-Q. // http://www.osp.ru/os/1997/02/6.htm
2. В. Корнеев. Архитектуры с распределенной разделяемой памятью. // http://www.osp.ru/os/2001/03/179969/
3. В. Корнеев, А. Киселев. Современные микропроцессоры. 2-е издание. М.: Нолидж. 2000
4. В.В. Рудометов, В.С. Семенов. Анализ когерентности кэш-памятей для повышения эффективности тестирования подсистемы памяти.