СИНХРОНИЗАЦИЯ В СИСТЕМАХ, ОРИЕНТИРОВАННЫХ НА ОБРАБОТКУ ДАННЫХ МОНИТОРИНГА ОКРУЖАЮЩЕЙ СРЕДЫ

Ю.Г. Шаталова

Севастопольский национальный технический университет,
г. Севастополь, ул. Университетская, 33
E-mail: volnajulia@mail.ru

В статье рассматривается проблема синхронизации в распределенных системах с репликацией данных.

Введение. Для эффективного мониторинга состояния окружающей среды требуется проведение анализа изменений в показаниях, снятых за определенный промежуток времени. Эти показания необходимо хранить, обрабатывать и систематизировать. А, следовательно, необходим инструмент, технология, средство, позволяющие осуществить перечисленные операции. Наиболее подходящий инструмент для этого – базы данных. Причем использование распределенных баз данных наиболее удобно. Особенность применения распределенных баз данных в том, что данные хранятся не на одном компьютере, а на нескольких. Это, во-первых, позволяет рассматривать систему компьютеров как интегрированную вычислительную систему, с интегрированными же характеристиками, во-вторых, позволяет разместить данные в непосредственной близости от использующих их пользователей. Например, если результаты мониторинга обрабатываются несколькими исследователями, то каждому из них удобнее располагать свою часть данных на своем компьютере. Причем, главный принцип распределенных систем заключается в том, что для конечного пользователя распределенная система должна выглядеть так же, как нераспределенная.

В силу сказанного к распределенным базам данных предъявляются следующие требования [1]:

1. Локальная независимость.
2. Отсутствие опоры на центральный узел.
3. Непрерывное функционирование.
4. Независимость от расположения.
5. Независимость от секционирования.
6. Независимость от репликации.
7. Обработка распределенных запросов.
8. Управление распределенными транзакциями.
9. Аппаратная независимость.
10. Независимость от операционной системы.
11. Независимость от сети.
12. Независимость от типа СУБД.

Особенности управления распределенными транзакциями. Особое внимание уделим именно управлению распределенными транзакциями. Для обеспечения эффективной работы с распределенными данными, используется прием репликации данных, т.е. хранение копий таблиц на тех компьютерах, с которыми работают пользователи, обрабатывающие эти данные. Суть репликации заключается в том, что хранящиеся на различных компьютерах копии должны обновляться как только на одном из узлов это обновление будет внесено. И в связи с этим, возникает ряд проблем, которые необходимо решить для обеспечения качественной работы распределенной системы.

Одной из таких задач является синхронизация времени. Синхронизация необходима процессам для организации совместного использования ресурсов, таких как файлы или устройства, а также для обмена данными. В однопроцессорных системах решение задач взаимного исключения, критических областей и других проблем синхронизации осуществлялось с использованием общих методов, таких как семафоры и мониторы. Однако эти методы не совсем подходят для распределенных систем, так как все они базируются на использовании разделяемой оперативной памяти. Например, два процесса, которые взаимодействуют, используя семафор, должны иметь доступ к нему. Если оба процесса выполняются на одной и той же машине, они могут иметь совместный доступ к семафору, хранящемуся, например, в ядре, делая системные вызовы. Однако, если процессы выполняются на разных машинах, этот метод не применим, для

распределенных систем нужны новые подходы.

Синхронизация логических часов.

В централизованной однопроцессорной системе, как правило, важно только относительное время и не важна точность часов. В распределенной системе, где каждый процессор имеет собственные часы со своей точностью хода, ситуация резко меняется: программы, использующие время становятся зависимыми от того, часами какого компьютера они пользуются. В распределенных системах синхронизация физических часов (показывающих реальное время) является сложной проблемой, но очень часто в этом нет никакой необходимости: то есть процессам не нужно, чтобы во всех машинах было правильное время, для них важно, чтобы оно было одинаковое, более того, для некоторых процессов важен только правильный порядок событий. В этом случае мы имеем дело с логическими часами. Для распределенной системы обработки данных мониторинга требуется однозначность именно в порядке событий. То есть, транзакция, выполненная на одном из компьютеров должна рассматриваться как первоочередное событие, и ее прибытие на любой из компьютеров системы должно немедленно привести к изменению данных в соответствующих репликах.

Введем для двух произвольных событий отношение "случилось до". Выражение $a\ R\ b$ означает, что все процессы в системе считают, что сначала произошло событие a, а потом - событие b. Отношение "случилось до" обладает свойством транзитивности: если выражения $a\ R\ b$ и $b\ R\ c$ истинны, то справедливо и выражение $a\ R\ c$. Для двух событий одного и того же процесса всегда можно установить отношение "случилось до", аналогично может быть установлено это отношение и для событий передачи сообщения одним процессом и приемом его другим, так как прием не может произойти раньше отправки. Однако, если два произвольных события случились в разных процессах на разных машинах, и эти процессы не имеют между собой никакой связи, то нельзя сказать с полной определенностью, какое из событий произошло раньше, а какое позже.

Ставится задача создания такого механизма ведения времени, который бы для каждого события a мог указать значение времени $T(a)$, с которым бы были согласны все процессы в системе. При этом должно выполняться условие: если $a\ R\ b$, то $T(a) < T(b)$. Кроме того, время может только увеличиваться и, следовательно, любые корректировки времени могут выполняться только путем добавления положительных значений, и никогда - путем вычитания.

Одним из подходящих алгоритмов для решения этой задачи является алгоритм Lamport [2]. Суть его в том, что каждое сообщение должно нести с собой время своего отправления по часам машины-отправителя. Если в машине, получившей сообщение, часы показывают время, которое меньше времени отправления, то эти часы переводятся вперед, так, чтобы они показали время, большее времени отправления сообщения. Таким образом, задача синхронизации времени в постановке задачи определения порядка событий может быть решена.

Организация взаимного исключения при выполнении транзакций. Еще одна задача распределенных систем – обеспечение работы различных процессов с разделяемыми ресурсами, в данном случае с репликами отношений.

Распределенные системы, работающие с несколькими пользователями, можно организовать, используя так называемые критические секции [3]. Когда процессу нужно читать или модифицировать некоторые разделяемые структуры данных, он, прежде всего, входит в критическую секцию для того, чтобы обеспечить себе исключительное право использования этих данных, при этом он уверен, что никакой процесс не будет иметь доступа к этому ресурсу одновременно с ним. Это называется взаимным исключением. В однопроцессорных системах критические секции защищаются семафорами, мониторами и другими аналогичными конструкциями. В распределенных системах такие средства не подходят, но существуют свои алгоритмы [4, 5]. Один из них -- централизованный алгоритм.

Централизованный алгоритм. Наиболее очевидный и простой путь реализации взаимного исключения в распределенных системах - это применение тех же методов, которые используются в однопроцессорных системах. Один из процессов выбирается в качестве координатора (например, процесс, выполняющийся на машине, имеющей наибольшее значение сетевого адреса). Когда какой-либо процесс хочет войти в критическую секцию, он посылает сообщение с запросом к координатору, оповещая его о том, в какую критическую секцию он хочет войти, и ждет от координатора разрешение. Если в этот момент ни один из процессов не находится в критической секции, то координатор посылает ответ с разрешением. Если же некоторый процесс уже выполняет критическую секцию, связанную с данным ресурсом, то никакой ответ не посылается; запрашивавший процесс ставится в очередь, и после освобождения критической секции ему отправляется ответ-разрешение. Этот алгоритм гарантирует взаимное исключение, но вследствие своей централизованной природы обладает низкой отказоустойчивостью.

Распределенный алгоритм. Когда процесс хочет войти в критическую секцию, он формирует сообщение, содержащее имя нужной ему критической секции, номер процесса и текущее значение времени. Затем он посылает это сообщение всем другим процессам. Предполагается, что передача сообщения надежна, то есть получение каждого сообщения сопровождается подтверждением. Когда процесс получает такое сообщение, его действия зависят от того, в каком состоянии по отношению к указанной в сообщении критической секции он находится. Имеют место три ситуации:

1. Если получатель не находится и не собирается входить в критическую секцию в данный момент, то он отсылает назад процессу-отправителю сообщение с разрешением.

2. Если получатель уже находится в критической секции, то он не отправляет никакого ответа, а ставит запрос в очередь.

3. Если получатель хочет войти в критическую секцию, но еще не сделал этого, то он сравнивает временную отметку поступившего сообщения со значением времени, которое содержится в его собственном сообщении, разосланном всем другим процессам. Если время в поступившем к нему сообщении меньше, то есть его собственный запрос возник позже, то он посылает сообщение-разрешение, в обратном случае он не посылает ничего и ставит поступившее сообщение-запрос в очередь.

Процесс может войти в критическую секцию только в том случае, если он получил ответные сообщения-разрешения от всех остальных процессов. Когда процесс покидает критическую секцию, он посылает разрешение всем процессам из своей очереди и исключает их из очереди.

Алгоритм Token Ring. Совершенно другой подход к достижению взаимного исключения в распределенных системах предлагает этот алгоритм. Все процессы системы образуют логическое кольцо, т.е. каждый процесс знает номер своей позиции в кольце, а также номер ближайшего к нему следующего процесса. Когда кольцо инициализируется, процессу 0 передается, так называемый, токен. Токен циркулирует по кольцу. Он переходит от процесса n к процессу n+1 путем передачи сообщения по типу "точка-точка". Когда процесс получает токен от своего соседа, он анализирует, не требуется ли ему самому войти в критическую секцию. Если да, то процесс входит в критическую секцию. После того, как процесс выйдет из критической секции, он передает токен дальше по кольцу. Если же процесс, принявший токен от своего соседа, не заинтересован во вхождении в критическую секцию, то он сразу отправляет токен в кольцо. Следовательно, если ни один из процессов не желает входить в критическую секцию, то в этом случае токен просто циркулирует по кольцу с высокой скоростью.

Сравним эти три алгоритма взаимного исключения. Централизованный алгоритм является наиболее простым и наиболее эффективным. При его использовании требуется только три сообщения

для того, чтобы процесс вошел и покинул критическую секцию: запрос и сообщение-разрешение для входа, и сообщение об освобождении ресурса при выходе. При использовании распределенного алгоритма для одного использования критической секции требуется послать (n-1) сообщений-запросов (где n - число процессов) - по одному на каждый процесс и получить (n-1) сообщений-разрешений, то есть всего необходимо 2(n-1) сообщений. В алгоритме Token Ring число сообщений переменно: от 1 в случае, если каждый процесс входил в критическую секцию, до бесконечно большого числа, при циркуляции токена по кольцу, в котором ни один процесс не входил в критическую секцию.

К сожалению все эти три алгоритма плохо защищены от отказов. В первом случае к краху приводит отказ координатора, во втором - отказ любого процесса (парадоксально, но распределенный алгоритм оказывается менее отказоустойчивым, чем централизованный), а в третьем - потеря токена или отказ любого процесса.

Анализ достоинств и недостатков рассмотренных алгоритмов позволяет сделать следующие выводы:

Централизованный алгоритм:

1. Наиболее прост в реализации.
2. Наиболее устойчив к отказам.
3. При его использовании требуется наименьшее число сообщений.

Следовательно, централизованный алгоритм является наиболее эффективным, но требует модификации.

Предлагаются следующие пути повышения надежности и отказоустойчивости централизованного алгоритма:

1. Выбирать в качестве координатора процесс, непосредственно генерирующий транзакцию.

2. Принимать время отправки сообщения о транзакции за начальное.

3. Обеспечить выполнение транзакции на каждой реплике атомарно.

4. Перед выбором координатора применить алгоритм планирования, основанный на анализе состояний процессов, т.е. выбирать именно тот процесс, который в данный момент находится в состоянии готовности.

5. Использовать дублирующий координатор. В случае отказа первичного координатора, его может заменить дублер. При этом принцип выбора дублера – дополнительная возможность сделать алгоритм более гибким.

Заключение. В статье проанализированы некоторые проблемы организации распределенной базы данных, применяемой для хранения результатов мониторинга окружающей среды. Выбран алгоритм синхронизации времени в системе. Рассмотрены возможные алгоритмы организации взаимного исключения, обоснован выбор централизованного алгоритма. На основе анализа его недостатков предложены пути повышения надежности и отказоустойчивости выбранного алгоритма.

СПИСОК ЛИТЕРАТУРЫ

1. *Дейт К. Дж.* Введение в системы баз данных (седьмое издание). – М.: Вильямс, 2001. – 1072 с.
2. *Олифер Н. А., Олифер В. Г.* Сетевые операционные системы.– СПб.: Питер, 2007. – 544 с.
3. *Кузнецов С.Д.* Основы баз данных. – 2-е изд. – М.: Интернет-Ун-т информационных технологий, 2007. – 342 с.
4. *Коннолли Т., Бегг К.* Базы данных. Проектирование, реализация и сопровождение. Теория и практика - 3-е изд. – М.: Вильямс, 2003. – 1436 с.
5. *Семенов Ю.А.* Распределенные вычислительные системы в природе или истинный путь в развитии вычислительных комплексов. // Теория и технология программирования и защиты информации: тезисы докл. XIV междунар. научно-практ. конф. – Санкт-Петербург, 2009. – С. 72 – 75.