МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Кафедра систем информатики

Ипполитова Ольга Александровна

Разработка распределенной версии системы проверки навыков программирования

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ
по направлению высшего профессионального образования

230100.68 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Тема диссертации утверждена распоряжением по НГУ №9 от «11» января 2012г.
Тема диссертации скорректирована распоряжением по НГУ №539 от «14» декабря 2012г.
Тема диссертации скорректирована распоряжением по НГУ № 183 от «14» мая 2013г.

Руководители

Иртегов Дмитрий Валентинович
б/с, доцент

Чурина Татьяна Геннадьевна
к.ф.-м.н., доцент

Новосибирск, 2013г.

МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Кафедра систем информатики

УТВЕРЖДАЮ

Зав. кафедрой М. М. Лаврентьев

…….
(подпись, дата)

ЗАДАНИЕ
на магистерскую диссертацию

студент Ипполитова Ольга Александровна

факультета ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Направление подготовки 230100.68 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Магистерская программа Технология разработки программных систем

Тема «Разработка распределенной версии системы проверки навыков программирования»

Цели работы: разработка распределенной версии системы NSUts посредством репликации баз данных нескольких копий. Включает: анализ существующих технологий, выбор применимой для решения данной задачи; адаптация структуры базы данных; разработка системы метаданных, их хранения и обновления; реализация алгоритма репликации.

Руководитель

Иртегов Дмитрий Валентинович
б/с, доцент

……..
(подпись, дата)

Чурина Татьяна Геннадьевна
к.ф.-м.н., доцент

……..
(подпись, дата)

Содержание

Введение....4
1 Проект NSUts....5
1.1 Общая информация....5
1.2 Структура системы....5
1.3 Использование....6
2 Обоснование задачи....7
3 Формальная постановка задачи....8
4 Поиск решения....9
4.1 Сравнение механизмов репликации....9
4.1.1 Репликация ведущий-ведомый....9
4.1.2 Репликация проталкиванием....9
4.1.3 Мультимастерная репликация Oracle....10
4.1.4 Репликация распространением слухов....10
4.2 Применение репликации распространением слухов к NSUts....11
5 Реализация....12
5.1 Атомарная единица данных....12
5.2 Метаданные....12
5.2.1 Время сохранения в локальную БД....12
5.2.2 Время фактического изменения....13
5.2.3 Флаг удаления....13
5.2.4 Уникальный идентификатор (UID)....13
5.3 Изменение структуры БД....13
5.3.1 Использование уникальных идентификаторов....13
5.3.2 Изменение способа генерации первичных ключей....14
5.3.3 Хранение дополнительной информации....15
5.4 Обновление метаданных....15
5.4.1 Обновление метаданных во время работы NSUts....15
5.4.2 Обновление метаданных во время репликации....16
5.5 Репликатор....17
5.5.1 Алгоритм репликации....17
6 Заключение....18
Литература....20
Приложение А....21
Приложение В....22

ВВЕДЕНИЕ

Система тестирования NSUts используется и поддерживается в НГУ более десяти лет. Она применяется как для проведения соревнований по программированию, так и в учебном процессе. С этой целью разработаны и включены в нее тематические тренировки по олимпиадному программированию и учебно-методический комплекс «Практические занятия по курсу Программирование».

Всем пользователям предоставляется доступ через веб-интерфейс. NSUts имеет централизованную архитектуру. Для проведения соревнований на нескольких площадках необходима распределенная версия системы NSUts. Она реализуется организацией репликации баз данных нескольких экземпляров системы.

В данной работе приведено сравнение существующих технологий и их применимость для решения поставленной задачи. Выявлено, что нет реализаций некой универсальной технологии репликации, отвечающей требованиям к распределенной версии NSUts. Репликация «распространением слухов» отвечает требованиям, но не имеет реализаций для реляционных БД.

Технология репликации «распространением слухов» позволяет обеспечить быструю синхронизацию данных, работает с любой топологией сети, а также работоспособна в условиях низкоскоростных и ненадежных каналов связи.

В работе решаются следующие задачи:

1) выбор технологии репликации;
2) разработка уточненного механизма репликации;
3) адаптация структуры базы данных для нужд репликации и работы в распределенном режиме;
4) адаптация исходного кода системы для нужд репликации и работы с измененной БД;
5) реализация алгоритма репликации;
6) тестирование полученной системы и реализованного алгоритма.

Реализованная система была успешно протестирована.

1 Проект NSUts

1.1 Общая информация

Система тестирования NSUts разрабатывается и поддерживается в лаборатории Parallels-НГУ. Она предназначена для автоматической проверки навыков программирования, в частности, для проведения олимпиад по программированию. Участникам, жюри и администраторам предоставляется доступ через web-интерфейс.

Участники олимпиады отсылают решения в систему, выбрав один из разрешенных языков программирования для данного тура. Система тестирования компилирует решение и, если компиляция проходит успешно, запускает его на наборе тестов. После отсылки решения участник может следить за его статусом: решение либо ожидает своей очереди, либо тестируется, либо уже протестировано. В последнем случае участник может узнать результаты тестирования решения или обнаружить ошибку компиляции.

На основе информации об успехе прохождения тестов решениями участников выстраивается рейтинг. Рейтинг доступен во время и после тура, но, как правило, с целью интриги «замораживается» (перестает обновляться) незадолго до окончания тура и «размораживается», когда тур окончен. Во время и после тура жюри имеет возможность отправлять какие-то из решений участников на перетестирование, если в этом возникает необходимость.

1.2 Структура системы

Система состоит из серверной части и тестирующего клиента (рис.1). Серверная часть работает под управлением веб-сервера Apache и отвечает за веб-интерфейс. Она построена в виде CGI-скриптов и модулей на языке Perl. Тестирующий клиент – программа, осуществляющая непосредственно тестирование решений. Он запрашивает у сервера решение участника, тестирует и пересылает серверу результаты (ошибки компиляции или результаты тестирования). С одним сервером может работать одновременно несколько тестирующих клиентов, расположенных на разных машинах.

Используется СУБД MySQL.

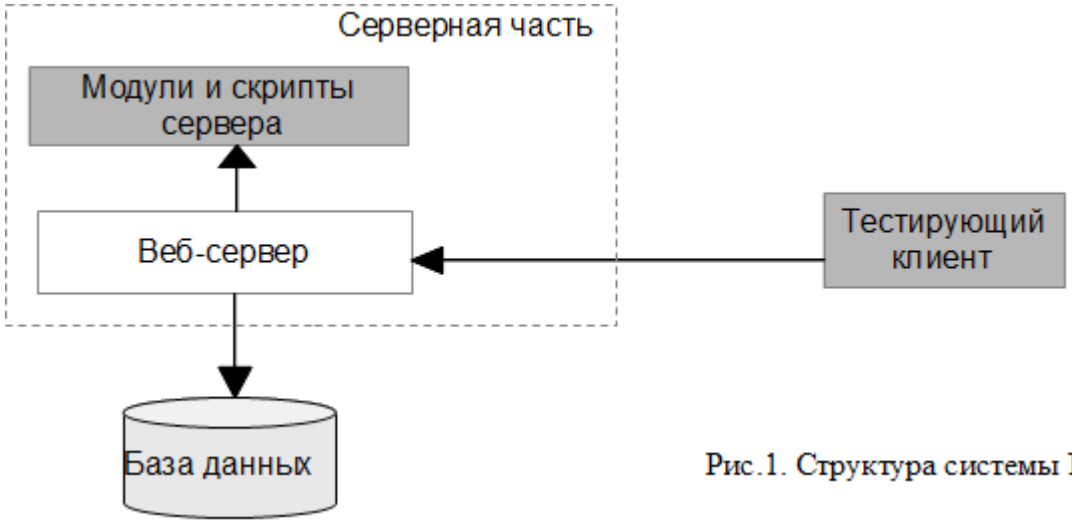

Рис.1. Структура системы NSUts

1.3 Использование

С помощью NSUts проводятся все основные соревнования по программированию в НГУ и в НСО, в том числе и такие крупные, как Открытая Всесибирская олимпиада по информатике и программированию им. И. В. Поттосина, Всесибирская открытая олимпиада школьников по информатике, школьные муниципальные и региональная олимпиады по информатике. На данной системе проводятся тренировки олимпиадных команд НГУ.

Также система используется для тестирования навыков программирования у студентов и школьников. Так, в ноябре 2011 с помощью NSUts было проведено тестирование по информатике и программированию в рамках мониторинга результативности обучения студентов (для Механико-математического, Физического факультетов и факультета Информационных технологий). На данной системе регулярно проводятся тестирования навыков программирования школьников по инициативе самих школ.

С сентября 2012 года система используется в учебном процессе, для приема и проверки задач по программированию у 1 и 2 курса ФИТ НГУ.

Система тестирования NSUts доступна круглосуточно по адресу https://olympic.nsu.ru/NSUts-test/.

2 Обоснование задачи

На данный момент для NSUts актуальна задача проведения соревнований в распределенном режиме на нескольких площадках. На каждой из таких площадок присутствует один или несколько представителей жюри, которые развертывают необходимое программное обеспечение, следят за его работой, подтверждают соблюдение регламента доступа участников к рабочим местам и т.д. То есть площадки равноценны с точки зрения участников. Это позволяет сэкономить на проезде участников к месту проведения соревнования (особенно в условиях Сибири и Дальнего Востока), решает проблему с получением виз иностранными участниками. Такое решение подходит тогда, когда нежелательно проводить соревнования по Интернет из-за возможности мошенничества, например, когда участие в соревновании дает льготы при поступлении в ВУЗ.

При организации нескольких площадок важно обеспечить равные условия для всех участников. С этой целью тестирование или соревнование проводят на одинаковых задачах. Во избежание утечки задач все участники должны проходить тестирование в одно и то же время. Следовательно, необходимо организовать обмен данными между площадками (как минимум, данными для построения общего рейтинга). То есть, если на каждой из площадок развернута своя копия системы тестирования, то необходимо пересылать по крайней мере часть данных этой системы. Развертывание же системы тестирования на единственной площадке ставит участников в неравные условия: из-за сетевых задержек участники с других площадок будут получать результаты тестирования своих решений позднее, чем участники с основной площадки. Также перебои в сетевом соединении могут привести к задержке отправки решения участником, при этом некоторые формулы расчета рейтинга учитывают точное время отправки. И наконец, при обрыве сетевого соединения между площадками на сколько-нибудь продолжительное время тур соревнования будет сорван.

Эта проблема решается с помощью распределенной системы тестирования [1].

3 Формальная постановка задачи

Необходимо реализовать распределенную версию системы тестирования NSUts на основе централизованной.

Под распределенной системой будем понимать набор независимых экземпляров системы тестирования, каждый из которых способен выполнять те же функции, что и централизованная система, и ведущих себя как единая система [2]. Такое поведение обеспечивается за счет синхронизации состояний всех экземпляров (репликации данных) [2, 3].

Перед распределенной версией NSUts ставятся требования:

1. Система должна быть работоспособна в условиях низкоскоростных и ненадежных каналов связи между площадками.
2. Репликация проводится как можно чаще, в идеале – за время, сопоставимое со временем тестирования одной задачи.
3. Противоречие между пунктами 1 и 2 может быть разрешено лишь за счет минимизации объема данных, передаваемых в каждом сеансе репликации. Желательно, чтобы при репликации передавались только данные, изменившиеся с момента предыдущей репликации.
4. Система должна обеспечивать прозрачное восстановление после разрыва связи. Помимо прочего, это означает, что система должна прозрачно восстанавливаться после разрыва связи во время сеанса репликации. Незавершенная репликация не должна приводить к несогласованным данным, а повторная попытка репликации после такого разрыва не должна приводить к дублированию данных или ложным сообщениям о конфликтах репликации.

Такой способ быстрой синхронизации баз данных может быть использован для решения других актуальных для NSUts задач: балансировки нагрузки и обеспечения отказоустойчивости.

4 Поиск решения

4.1 Сравнение механизмов репликации

Рассмотрим существующие технологии репликации, а также их применимость к решению поставленной задачи.

4.1.1 Репликация ведущий-ведомый

Репликация ведущий-ведомый (master-slave) основана на использовании неравноправных реплик [2, 3]. Все изменения вносятся в одну из реплик, которая и называется ведущей (master). Важным достоинством репликации master-slave является невозможность конфликтов репликации (невозможна ситуация, при которой одна и та же запись будет изменена в разных репликах). Другим достоинством является относительная простота метаданных, необходимых для того, чтобы определить, какое подмножество данных необходимо передавать. Уже из описания репликации ведущий-ведомый очевидно, что эта технология, во всяком случае в чистом виде, для нашей задачи неприемлема. Организация тестирования на нескольких площадках предполагает внесение изменения в базу данных на каждой из площадок, а репликация ведущий-ведомый этого не допускает.

4.1.2 Репликация проталкиванием

Репликация проталкиванием заключается в том, что при каждом внесении изменения в копию БД все остальные реплики оповещаются об этом изменении и обязаны внести это изменение в свою копию данных. Это гарантирует, что каждое изменение практически одновременно вносится во все реплики.

Этот подход широко применяется в кластерах СУБД, в том числе в кластерах MySQL [2-4]. Главным достоинством данного подхода является то, что при проталкивании изменений на другие сервера сохраняется порядок внесения изменений. В том числе, если какая-то группа изменений была объединена в транзакцию, то операции проталкивания также будут объединены в транзакцию. Это значительно облегчает обеспечение согласования данных во всех репликах.

Репликация проталкиванием допускает внесение изменений в несколько реплик (репликация с несколькими ведущими).

Важным недостатком этой схемы, который также делает ее неприемлемым для нашей задачи, является очень высокая чувствительность к сбоям связи и даже к задержкам. При работе кластерной СУБД, сервер может подтвердить завершение транзакции только после

того, как все остальные сервера кластера завершат эту транзакцию.

Все поставщики СУБД, поддерживающих кластеризацию, в том числе и разработчики MySQL, рекомендуют связывать серверы кластера высокоскоростной надежной линией связи с малыми задержками, лучше всего – прямым соединением через локальную сеть.

4.1.3 Мультимастерная репликация Oracle

Несколько большей гибкостью, чем ранее перечисленные технологии, отличается репликация СУБД Oracle. Кроме базовых механизмов репликации (снимков и materialized views), Oracle поддерживает мультимастерную репликацию на уровне таблиц и хранимых процедур [5]. При мультимастерной репликации на уровне таблиц, изменения, вносимые в таблицу, проталкиваются во все остальные реплики, в том числе, если это допускается настройками репликации, в асинхронном режиме. Асинхронный режим может использоваться в сочетании с относительно низкоскоростными каналами связи. Однако, поскольку репликация происходит на уровне таблиц, обеспечение целостности данных при этом возлагается на разработчика приложения, который должен предоставить соответствующие хранимые процедуры (триггеры) на специализированном диалекте языка SQL.

Использование репликации Oracle в распространяемой системе тестирования нецелесообразно, потому что Oracle представляет собой дорогостоящий коммерческий продукт. Его использование фактически исключило бы возможность свободного распространения системы и сделало бы развертывание системы запретительно дорогостоящим для конечного пользователя. Кроме того, поскольку целостность данных при репликации, в конечном итоге, обеспечивается разработчиками приложения за счет кода триггеров, использование мультимастерной репликации приведет к необходимости пересматривать код этих триггеров при каждом изменении модели данных, что, в свою очередь, приведет к значительному усложнению поддержки и развития системы.

4.1.4 Репликация распространением слухов

Многие успешные распределенные базы данных основаны на вариантах алгоритма, известного как «распространение слухов». Этот алгоритм допускает внесение изменений в несколько реплик и произвольную топологию связей между репликами. Кроме того, этот алгоритм обеспечивает легкое восстановление после неудачных попыток репликации и, таким образом, может применяться через ненадежные каналы связи.

Варианты распространения слухов применяются в таких системах, как NNTP (Network

News Transfer Protocol) [6], Novell eDirectory (ранее известна как NDS), Microsoft Active Directory, Lotus Notes/Domino [7], OpenLDAP syncrepl [8].

Идея алгоритма довольно проста и состоит в том, что при каждом акте репликации узлы предъявляют друг другу списки известных им изменений и обмениваются теми изменениями, которые им были неизвестны. Затем они прикладывают эти изменения к своей копии данных и добавляют их в свой список известных изменений.

Работа алгоритма опирается на то, что все узлы могут определить, какие изменения являются одинаковыми. Для этого, все реплики должны использовать универсальный механизм идентификации изменений.

При реализации репликации распространением слухов важной проблемой является обеспечение целостности данных. Фактически, это обеспечение возлагается на разработчика приложения. Так, в приведенном ранее списке примеров использования этого алгоритма репликации, три из пяти примеров — это специализированные базы данных, предназначенные для хранения данных служб каталогов.

4.2 Применение репликации распространением слухов к NSUts

Для реализации распределенной версии системы тестирования необходимо выполнить следующие задачи:

- определить атомарную с точки зрения репликации единицу данных;
- разработать механизм генерации глобального идентификатора единицы данных – уникальной для всех узлов метки;
- организовать хранение метаданных;
- реализовать механизм обновления метаданных;
- разработать решение для сохранения консистентности баз данных во время репликации;
- реализовать алгоритмы «клиента» и «сервера» репликации. Здесь «клиентом» будем называть узел, инициирующий репликацию, «сервером» – узел, принимающий соединение. В остальном алгоритмы их работы очень похожи.

5 Реализация

5.1 Атомарная единица данных

Для реализации реплицируемой системы необходимо определить атомарную единицу данных – тот объем, который при изменении какой-либо его части передается другим узлам целиком или не передается вообще.

В качестве такой единицы данных используется кортеж СУБД. Используемый в NSUts драйвер СУБД DBD::mysql работает с кортежами, что делает проще отслеживание изменений в базе данных и обновление метаинформации.

Использование более мелких атомов данных невозможно. Например, при создании кортежа на одном узле далеко не всегда можно передать его частями на другой узел. Использование же более крупных, чем кортежи, единиц данных (например, объектов) неоправданно, так как в NSUts нет слоя объектно-реляционного отображения. Таким образом, это лишь усложнит обновление метаданных и логику репликатора и приведет к росту объема передаваемых во время репликации данных.

5.2 Метаданные

Для корректной работы репликатора необходима дополнительная информация – метаданные. Они должны определять данные, которые будут переданы другим узлам во время репликации. В качестве решения используется механизм, аналогичный применяемому в Lotus Notes/Domino [7]. Метаданные приписываются каждой атомарной единице данных и состоят из следующих полей:

- уникальный идентификатор (UID);
- время сохранения в локальную БД;
- время последнего фактического изменения;
- флаг удаления.

Они хранятся в отдельной таблице реплицируемой базы данных. Во время репликации узлы сначала обмениваются списками метаданных, анализируют их и затем обмениваются измененными данными.

5.2.1 Время сохранения в локальную БД

Эта метка показывает, когда узел узнал о данном изменении. Она обновляется до текущего времени при любом изменении кортежа, как через систему NSUts, так и во время репликации. Время сохранения кортежа не передается другим узлам. Оно необходимо для

того, чтобы во время репликации узел пересылал метаданные не всех кортежей БД, а лишь тех, что были изменены после последней успешной репликации.

5.2.2 Время фактического изменения

Время фактического изменения – это, практически, версия кортежа. Оно обновляется до текущего времени только при изменении кортежа через систему NSUts. Если в сеансе репликации время фактического изменения кортежа совпадает на двух узлах, то этот кортеж не пересылается.

5.2.3 Флаг удаления

При модификации кортежа флаг удаления устанавливается в 0, при удалении – в 1. Он необходим для того, чтобы передавать информацию об удалении кортежа на другие узлы.

5.2.4 Уникальный идентификатор (UID)

Работа алгоритма репликации «распространением слухов» опирается на то, что все узлы могут определить, какие изменения являются одинаковыми. Для этого все реплики должны использовать универсальный механизм идентификации изменений. Изменение определяется уникальным идентификатором (UID) и временем модификации кортежа. Таким образом, UID должен быть уникален среди всех идентификаторов кортежей всех реплицируемых баз данных. Для этого UID составляется из двух чисел: номера реплики и значения счетчика, общего для всех таблиц одной базы данных.

5.3 Изменение структуры БД

Реализация распределенной версии NSUts потребовала изменения структуры данных. Причины: использование уникальных идентификаторов, изменение способа генерации первичных ключей и хранение дополительной информации, необходимой для репликации. Были реализованы скрипты миграции, изменяющие структуру данных централизованной системы так, чтобы их могла использовать распределенная. При этом централизованная система по-прежнему может работать с этими данными (кроме функции last_insert_id(), описанной в п.5.3.2).

5.3.1 Использование уникальных идентификаторов

Каждому кортежу БД необходимо поставить в соответствие уникальный идентификатор (UID). Предполагаемые решения:

1. Добавить атрибут, содержащий UID, к каждому отношению.

2. Использовать UID в качестве первичного ключа каждого отношения.

В теории второе решение предпочтительнее, т.к. оно значительно упрощает передачу кортежей, связанных внешним ключом. Предположим, что в качестве внешних ключей (как и в качестве первичных ключей) используются локальные идентификаторы (как в исходной, централизованной системе). Тогда передача кортежа данных для узла-отправителя должна сопровождаться поиском соответствия между локальными и уникальными идентификаторами и изменением кортежа. На узле-получателе перед сохранением должна проводиться такая же расшифровка и исправление на локальные ключи.

Анализ бизнес-логики и данных системы показал, что на практике второе решение также выигрывает. В большинстве таблиц БД централизованной версии в качестве первичных ключей используются значения auto increment MySQL [9]. Это означает, что первичные ключи в исходной версии системы чаще всего используются в качестве идентификаторов строк. Было выявлено только три отношения, где первичный ключ нес также и смысловую нагрузку. В двух из них первичный ключ генерируется не с помощью auto increment и удовлетворяет условию глобальной уникальности. Строки еще одной таблицы (решения участников) сортировались по auto increment-ключу, чтобы решения тестировались в порядке поступления. Сортировка по ключу была заменена на сортировку по времени посылки решения. Таким образом, использование UID как первичного ключа повлекло изменение исходного кода системы, работающего с одной таблицей.

5.3.2 Изменение способа генерации первичных ключей

Как говорилось ранее, в централизованной версии системы для генерации первичных ключей используется auto increment MySQL. Это решение не подходит для распределенной системы: ее база данных должна хранить кортежи, сгенерированные на разных узлах. Это значит, что первичные ключи кортежей одного отношения на разных узлах не должны дублироваться. UID удовлетворяют этому условию. Таким образом, использование UID в качестве первичных ключей автоматически решает проблему их дублирования.

Генерацию первичных ключей можно производить как в базе данных, так и в приложении. Используемый ранее auto increment требовал передачи неопределенного значения первичного ключа в INSERT-запросе [9]. В связи с этим для минимизации изменений в исходном коде решено реализовать генерацию уникальных идентификаторов в триггерах, срабатывающих перед INSERT-запросами. Код триггера одинаков для всех отношений, что

упрощает поддержку системы в дальнейшем.

Использование auto increment неотделимо от использования функции MySQL last_insert_id(). В системе NSUts все вызовы к драйверу СУБД происходят через модуль-обертку. Новая версия функции last_insert_id() была реализована в модуле-обертке как запрос значения переменной сессии [10]. Обновление этой переменной происходит в триггерах, после генерации нового идентификатора.

В Приложении А приведено тело триггера.

5.3.3 Хранение дополнительной информации

Как уже упоминалось, организация репликации между узлами требует хранения дополнительной информации. Для этого в основную базу данных были добавлены три таблицы. Они содержат:

- номер (имя) узла и значение счетчика, использующегогя для генерации UID;
- список известных узлов и даты последних успешных репликаций с ними;
- метаданные.

5.4 Обновление метаданных

5.4.1 Обновление метаданных во время работы NSUts

Система обновления метаданных была встроена в обертку метода драйвера СУБД, непосредственно выполняющего SQL-запрос.

Перед выполнением модифицирующего запроса к БД вызывается внутренняя функция 'prepareStatement'. Ее назначение – проанализировать запрос и подготовить вспомогательную информацию для обновления метаданных, например, SQL-запрос на обновление метаданных и аргументы для него.

Построение SQL-запроса на обновление метаданных различается для разных типов модифицирующих запросов. Рассмотрим три их типа: INSERT, UPDATE и DELETE.

- В запросах типа DELETE кортежи, подлежащие удалению, определяются блоком WHERE. Чтобы обновить метаданные для тех же самых кортежей, необходимо перед удалением запросить значения их уникальных идентификаторов. Запрос на обновление метаданных строится так:

```
UPDATE replication_meta
SET save_time=NOW(), mod_time=NOW(),  deleted=1
WHERE table_name=<table> AND entity_id IN (
```

```
SELECT <primary_key> FROM <table>
WHERE <where>
);
```

где значения <table> и <where> определяются путем анализа исходного запроса, <primary_key> определяется из структуры базы данных.

Запрос на обновление метаданных кортежей должен передаваться БД до их удаления.

- Для запросов типа UPDATE обновление метаданных происходит аналогично. После выполнения запроса набор кортежей, определяемых блоком WHERE может измениться. Поэтому запросы данного типа также должны выполняться после обновления метаданных.
- Запросам типа INSERT свойственна другая проблема: до их выполнения кортежам не присвоены уникальные идентификаторы. Поэтому добавление метаданных для новых кортежей происходит после их создания:

```
INSERT INTO replication_info
SET save_time=NOW(), mod_time=NOW(),  deleted=0,
table_name=<table>, entity_id=<id>;
```

где значение <table> определяется путем анализа исходного запроса, <id> – значение первичного ключа из исходного запроса либо, если первичный ключ генерируется с помощью триггера, результат вызова функции last_insert_id().

Результат вызова функции 'prepareStatement' – константа, указывающая, когда следует обновлять метаданные (до или после основного запроса). Ее возможные значения:

- BEFORE – для запросов типа UPDATE и DELETE;
- AFTER – для запросов типа INSERT;
- NEVER – для прочих запросов, например, SELECT.

Вся информация для обновления метаданных, построенная в процессе работы функции prepareStatement, кэшируется.

Далее выполняются основной запрос к базе данных и запрос на обновление метаданных в нужном порядке.

5.4.2 Обновление метаданных во время репликации

При обновлении кортежа во время репликации срабатывает механизм, описанный в предыдущем пункте. Однако в этом случае время модификации, как и сам кортеж, передается от другого узла. Оно используется в запросе на обновление метаданных вместо

текущего времени.

5.5 Репликатор

Протокол общения реплик основан на XML. Примеры сообщений приведены в Приложении В.

5.5.1 Алгоритм репликации

Ниже приведен алгоритм репликации. Логика работы клиента отличается от логики работы сервера тем, что клиент инициирует репликацию. Также было решено осуществлять сравнение списков метаданных на стороне клиента.

1) Узел-клиент инициирует начало репликации.
2) Оба узла строят списки изменений (наборы метаданных, время сохранения которых позднее времени последней успешной репликации), сервер передает свой список метаданных клиенту.
3) Узел-клиент строит списки обновления для себя и узла-сервера. При этом:
   a) если метаданные присутствуют в списке изменений только одного узла, они должны быть добавлены в список обновлений другого;
   b) если данные изменены на обоих узлах и время модификации совпадает, то это, скорее всего, одно и то же изменение, либо переданное обоим другими узлами, либо переданное одним из них другому во время прошлой незавершенной репликации. Его реплицировать не нужно;
   c) если данные изменены на обоих узлах и время модификации не совпадает, то скорее всего, клиент и сервер получали обновления от некого третьего узла и обновились до разных версий. В этом случае берется кортеж с более поздним временем изменения.

Также эта ситуация может означать конфликт репликации. Данный алгоритм не распознает конфликты, хотя, при объявленном в постановке задачи сценарии использования конфликтов быть не должно.

4) Клиент, используя свой список обновления, поочередно запрашивает данные у сервера и сохраняет их в свою базу данных, обновляя метаинформацию.
5) Клиент посылает изменения серверу, и сервер так же сохраняет их.
6) При успешном завершении узлы обновляют дату последней репликации. Это самая поздняя из дат сохранения в БД по всем переданным изменениям.

6 Заключение

В ходе данной работы был разработан механизм репликации «распространением слухов», в том числе были определены необходимые для репликации метаданные. Структура базы данных была адаптирована для работы в распределенном режиме и хранения метаданных. Также были реализованы механизмы автоматического обновления метаданных и репликации. При этом изменения в исходном коде системы были сведены к минимуму.

Система с измененной схемой идентификации успешно прошла автоматические функциональные тесты для NSUts.

Нагрузочное тестирование [11] выявило ухудшение производительности распределенной системы на 20-35% (в среднем на 25%) по сравнению с централизованной. Сравнение проводилось по среднему числу успешно сгенерированных страниц в секунду. Помимо исходной централизованной и полученной распределенной систем тестирование проводилось на системе, использующей исходный код распределенной и базу данных централизованной системы. Производительность такой системы ухудшилась на 6% по сравнению с централизованной. Таким образом, потеря 6% производительности связана с необходимостью сбора метаданных, а именно анализа всех SQL-запросов и исполнения дополнительных. Для части запросов можно улучшить производительность, выбирая значение идентификатора из запроса и не выполняя вложенный SELECT.

Потеря остальных 19% производительности связана с изменениями в базе данных, а именно с использованием длинных первичных ключей и добавлением триггеров. Возможно, к увеличению производительности приведет перенос логики триггера в модуль-обертку драйвера СУБД.

Реализованная система успешно решает поставленную задачу и дает возможность дальнейшего расширения функциональности. Технология быстрой синхронизации данных может также использоваться для решения других актуальных для NSUts задач: балансировки загрузки и обеспечения отказоустойчивости.

Один из возможных сценариев использования, помимо описанного, – решение проблемы возможного мошенничества на очном туре, когда участник имеет доступ к своим предыдущим решениям на других олимпиадах. Сейчас эта проблема решается полным закрытием доступа ко всем олимпиадам, кроме текущей. Причем доступ закрывается для всех пользователей, в том числе не участвующих в олимпиаде. Учитывая то, что система используется также в учебном процессе и для тренировки олимпиадных команд, этот путь не является хорошим.

В качестве решения предлагается проведение очного тура олимпиады на отдельном экземпляре системы. Для этого создается реплика с частичной копией данных (только тех, что необходимы для проведения тура). Участникам разрешается доступ только к выделенной, неполной системе. Репликация с основной системой не проводится до окончания тура. После окончания тура изменения передаются только в одну сторону – к основной системе. При таком подходе проведение, к примеру, школьной олимпиады не мешает студентам сдавать задания. Данное решение требует небольших дополнений реализованной системы: односторонней репликации и возможности создания частичной копии.

Один из возможных путей развития системы – добавление возможности распознавания и разрешения конфликтов. Это потребует расширения набора метаданных, логики репликатора и протокола общения реплик. Такая система предоставит более широкие возможности использования.

Литература

1. Иртегов, Д.В. и др. Исследование возможности реализации децентрализованной версии системы проверки знаний и навыков программирования.//Ершовская конференция по информатике. Труды НПО 2011. – С.117–123.
2. Танненбаум, Э. и др. Распределенные системы, принципы и парадигмы. СПб.: Питер, 2003. – 877с.
3. Иртегов, Д.В. Введение в сетевые технологии. СПб.: БХВ-Петербург, 2004. – 560с.
4. MySQL 5.0 Reference Manual, Chapter 17. MySQL Cluster // Электронный ресурс, режим доступа — свободный, http://dev.mysql.com/doc/refman/5.0/en/mysql-cluster.html
5. Garmany J., Freeman R. Oracle Replication: Snapshot, Multi-master and Materialized Views Scripts. Kittrell, North Carolina, USA: Rampant Tech Press, 2003. – 224p.
6. C. Feather, Request for Comments: 3977, Network News Transfer Protocol (NNTP), Network Working Group, 2006 // Электронный ресурс, режим доступа — свободный, http://tools.ietf.org/html/rfc3977
7. Lotus Domino Designer documentation // Электронный ресурс, режим доступа — свободный, https://www.ibm.com/developerworks/lotus/documentation/dominodesigner/
8. OpenLDAP 2.2. Administrators Guide: LDAP Syns Replication // Электронный ресурс, режим доступа — свободный, http://www.openldap.org/doc/admin22/syncrepl.html
9. MySQL 5.5 Reference Manual. 3.6.9. Using AUTO_INCREMENT // Электронный ресурс, режим доступа — свободный, http://dev.mysql.com/doc/refman/5.5/en/example-auto-increment.html
10. MySQL 5.6 Reference Manual. 5.1.5. Using System Variables // Электронный ресурс, режим доступа — свободный, http://dev.mysql.com/doc/refman/5.6/en/using-system-variables.html
11. Колбин Я.С. Нагрузочное тестирование автоматической проверки навыков программирования NSUts // Материалы 51-й международной студенческой конференции «Студент и научно-технический прогресс». Информационные технологии, 2013. – С.178.

Приложение А

(обязательное)

Листинг 1. Триггер, генерирующий уникальный идентификатор типа CHAR(19)

```
 BEGIN
declare n CHAR( 4 ) ;
declare l INT( 15 ) ;
SELECT node_id_hash, last_key INTO n, l FROM replication_info LIMIT 1 FOR UPDATE;
update replication_info set last_key=last_key+1;
SET @repl_last_id = concat( n, lpad(l, 18-length(n),'0') );
SET NEW.id = @repl_last_id;
END
```

Приложение В

(обязательное)

Листинг 1. Пример сообщения в протоколе общения реплик: запрос метаданных

```
<?xml version="1.0" encoding="utf-8"?>
<NSUts_replication content="meta-request" sender="testing_1.3" />
```

Листинг 2. Пример сообщения в протоколе общения реплик: пересылка списка метаданных

```
<?xml version="1.0" encoding="utf-8"?>
<NSUts_replication content="meta" sender="testing_1.3">
	<o t="team" id="112200000000013811" md="2013-04-18 12:13:53" d="0" />
	<o t="priv" id="112200000000013812" md="2013-04-18 12:13:54" d="0" />
</NSUts_replication>
```

Листинг 3. Пример сообщения в протоколе общения реплик: запрос данных

```
<?xml version="1.0" encoding="utf-8"?>
<NSUts_replication content="data-request" sender="testing_1.3">
	<o t="user_team" id="123400000000011450"/>
</NSUts_replication>
```

Листинг 4. Пример сообщения в протоколе общения реплик: пересылка данных

```
<?xml version="1.0" encoding="utf-8"?>
<NSUts_replication content="data" sender="testing_1.3">
	<t n="team">
		<a>id</a> <a>olympiad</a> <a>priv</a>
		<a>title</a> <a>creator</a> <a>old_userid</a>
	</t>
	<o t="team">
		<a>112200000000005891</a> <a>112200000000002799</a>
		<a>112200000000003091</a> <a>NSU 1</a> <a></a> <a>10021</a>
	</o>
</NSUts_replication>
```