Модел. и анализ информ. систем. Т. **20**, № **4** (2013) 136–149

УДК 004.652

Проектирование и разработка имитационной модели мультиклиентского кластера баз данных

Бойцов Е. А.

Ярославский государственный университет им. П. Г. Демидова
150000 Россия, г. Ярославль, ул. Советская, 14

e-mail: boytsovea@yandex.ru

получена 10 октября 2013

Ключевые слова: базы данных, SaaS, мультиклиентская архитектура, масштабирование, имитационная модель

Одной из главных тенденций последних лет в проектировании программного обеспечения стал переход к парадигме Software as a Service (SaaS), которая несет ряд неоспоримых преимуществ как для компаний-разработчиков ПО, так и для конечных пользователей. Однако вместе с этими преимуществами данный переход несет и новые архитектурные вызовы, одним из которых является организация хранилища данных, которое могло бы удовлетворить нужды компании-провайдера услуг, обеспечив достаточно простой прикладной интерфейс для разработчиков. Для разработки эффективного решения в данной области следует принимать во внимание особенности архитектуры облачных приложений, ключевыми из которых являются потребность в простом масштабировании и быстрой адаптации к меняющимся условиям. В данной работе проводится краткий анализ существующих проблем в области организации облачных систем хранения данных, основанных на реляционной модели, а также предлагается концепция кластера РСУБД, предназначенного для обслуживания приложений с мультиклиентской архитектурой. Кроме того, в статье дается описание имитационной модели подобного кластера, а также основных этапов ее разработки и принципов, заложенных в ее основу.

Введение

Одной из наиболее заметных тенденции в разработке программного обеспечения в наши дни является переход к парадигме Software as a Service (SaaS). Суть данного подхода можно определить следующими ключевыми положениями [1]:

- Приложение разрабатывается в виде системы распределенных веб-сервисов, взаимодействующих друг с другом;
- Вычислительные мощности и инфраструктуру, необходимые для работы приложения, предоставляет компания-провайдер услуг;

- Оплата за приложение взимается не единовременно, при покупке лицензии, а по факту использования.

Главным преимуществом данного подхода к разработке для конечного потребителя является то, что все расходы по организации необходимой инфраструктуры компания-провайдер берет на себя. Однако переход «в облака» несет не только преимущества, но и новые проблемы, в основном для разработчиков и администраторов систем подобного рода. Ключевой особенностью приложений, разрабатываемых согласно парадигме SaaS, является потребность в простом масштабировании: так как размер клиентской базы велик, а динамика ее изменения непредсказуема, то обеспечение масштабируемости становится жизненно важной задачей разработчиков.

Одним из подходов для обеспечения высокой степени масштабируемости является использование мультиклиентской (multi-tenant) архитектуры. Данный подход предполагает, что основные ресурсы приложения совместно используются многими компаниями-арендаторами, а за изоляцию их данных отвечает программная логика приложения. Проектирование решения в соответствии с подобными архитектурными принципами позволяет, при наличии достаточных вычислительных мощностей, поставить для обслуживания клиентов практически неограниченное количество экземпляров приложения. Однако указанные соображения не распространяются на серверы БД, так как данный компонент системы масштабируется плохо. В традиционных клиент-серверных системах проблема масштабирования стоит менее остро, так как типичная современная СУБД в единственном экземпляре вполне способна удовлетворить нужды среднего и даже достаточно крупного предприятия.

Многие современные реляционные СУБД предоставляют широкий набор возможностей, таких, как поддержка транзакций, процедурные расширения, различные механизмы репликации и тюнинга сервера под конкретный профиль нагрузки. Подобное обилие функционала, с одной стороны, делает такие системы очень удобными при разработке прикладного ПО и позволяет сэкономить много времени и денег компании-разработчику, а с другой стороны – затрудняет горизонтальное масштабирование, которое, как уже упоминалось выше, является одной из ключевых потребностей облачного приложения. Данный факт ставит компании-разработчики перед нелегким выбором:

- попытаться воспользоваться решениями по вертикальному масштабированию РСУБД, что требует больших вложений в инфраструктуру, лицензии на ПО и оплату труда высококвалифицированных сотрудников;

- перейти к использованию гораздо менее функциональных, но более масштабируемых NoSQL-решений, тем самым увеличив стоимость разработки за счет необходимости обучения персонала новым технологиям и реализации дополнительной функциональности силами собственных разработчиков.

Как правило, конкретные решения всегда оказываются где-то посередине. Какие-то части приложения, скорее всего не имеющие дела с критически важной информацией (кэши, хранилища временных данных и т.п.), переводятся на использование NoSQL-СУБД, которые гораздо лучше справляются с обслуживанием подобных задач, какие-то части по-прежнему используют РСУБД.

Что же касается поддержки мультиклиентской архитектуры на уровне РСУБД, то, несмотря на то, что в этой области имеются некоторые наработки и паттерны проектирования [2–4], в целом можно сказать, что данная техника не находит должной инструментальной поддержки со стороны разработчиков ПО подобного рода. Разработчикам предлагается самостоятельно заботиться об изоляции данных клиентов, а получающиеся базы данных достаточно тяжело администрировать ввиду их крайней сложности и большого размера. При этом облачному приложению, которое нацелено на широкую аудиторию, требуются десятки баз данных подобной структуры, так как, во-первых, все клиенты просто физически не поместятся в одну базу, а во-вторых, жизненно важно иметь хотя бы одну копию данных каждого клиента как из необходимости обеспечения устойчивости к сбоям и сохранности данных, так и из соображений производительности приложения и распределения нагрузки. Еще более существенным вопросом, чем количество баз данных, является балансирование нагрузки и оптимальное использование вычислительных мощностей компании-провайдера. Чтобы обеспечить должный уровень обслуживания, облачное приложение должно быть способно динамически адаптироваться под изменения профиля нагрузки путем автоматического перераспределения ресурсов. На данный момент программных средств, способных решить данную задачу комплексно, не существует, как не существует и четких требований к системам подобного рода и проверенных алгоритмов, которые можно было бы использовать для их построения. Соответственно возникает идея разработки теоретической модели системы подобного рода и исследования ее свойств. Указанную модель мы назовем мультиклиентским кластером баз данных. В последующих главах будут изложены основные характеристики предлагаемого подхода, сейчас же хотелось бы остановиться на существующих достижениях в исследуемой области.

1. Обзор существующих решений

Ряд исследователей уже обращались к проблеме разработки надежного хранилища данных, адаптированного под нужды облачных приложений. В целом работы ведутся в двух направлениях:

- разработка принципиально новых NoSQL-СУБД и способов обработки данных;
- адаптация реляционных СУБД под нужды облачных приложений с мультиклиентской архитектурой.

Развитие первого направления привело к появлению ряда новых решений, занявших свою нишу в области разработки прикладного ПО. Здесь стоит упомянуть такие широко известные NoSQL-СУБД, как Memcached, Redis, Cassandra и MongoDB, нашедшие свое применение в тысячах приложений по всему миру. Данный класс систем делает упор на производительности и масштабируемости, жертвуя функциональностью. Наиболее "простые" СУБД из приведенного списка (это Memcached и Redis) теоретически позволяют обслуживать вплоть до 100000 простых запросов в секунду [5,6]. При этом по мере наращивания функциональности (различные способы индексации данных, сложные выборки) они также подвержены деградации

производительности и сталкиваются с такими же проблемами, как и традиционные РСУБД, что неявно подтверждает истинность теоремы CAP [7], утверждающей, что в любой реализации распределённых вычислений возможно обеспечить не более двух из трёх следующих свойств:

- согласованность данных (англ. consistency) — во всех вычислительных узлах в один момент времени данные не противоречат друг другу;
- доступность (англ. availability) — любой запрос к распределённой системе завершается корректным откликом;
- устойчивость к разделению (англ. partition tolerance) — расщепление распределённой системы на несколько изолированных секций не приводит к некорректности отклика от каждой из секций.

Также в ряду технологий, облегчающих жизнь разработчиков облачных приложений, следует отметить модель распределённых вычислений MapReduce и фреймворк для распределенных вычислений Apache Hadoop.

В области традиционных РСУБД можно выделить четыре направления, которые так или иначе могут быть использованы при разработке облачных распределенных приложений:

- решения по кластеризации и вертикальному масштабированию СУБД;
- архитектурные концепции для организации мультиклиентской инфраструктуры в рамках одной СУБД;
- прототипы решений по встроенной поддержке мультиклиентской архитектуры на уровне СУБД;
- прототипы решений по управлению распределенными кластерами РСУБД с мультиклиентской архитектурой.

Решения из первой категории являются традиционными для РСУБД и имеют многолетнюю историю. Здесь можно встретить предложения различного характера: от дополнительных программных опций до специализированного аппаратного обеспечения. Следует отметить, что исторически решения подобного рода разрабатывались для других целей, а именно обеспечения функционирования пакетов ПО, обслуживающего крупные организации, которые владеют огромными массивами данных, когда для построения одного отчета требуется обработка десятков миллионов записей. Данный сценарий работы не соответствует потребностям типичного облачного приложения, основными пользователями которых выступают мелкие и средние компании, не генерирующие по отдельности большой вычислительной нагрузки. В случае облачного приложения нагрузка создается количеством запросов, а не их "тяжестью". Соответственно, хотя указанные решения и могут быть применены, компании-разработчику рано или поздно придется описывать дополнительный слой программной логики, ответственный за распределение клиентов и маршрутизацию запросов. Более релевантные исследования [2] велись в области поиска способов организации мультиклиентского окружения в существующих СУБД. В рамках данных исследований были предложены два подхода:

- Использование общих таблиц для хранения данных всех клиентов компании-провайдера. При применении такого подхода к каждой таблице в БД добавляется дополнительный столбец, в котором хранится идентификатор клиента, которому принадлежит данная запись.

- Выделенная схема для клиента. При использовании данного подхода изоляция данных клиентов друг от друга достигается путем создания собственного набора объектов БД для каждого из них.

На основе данных исследований некоторыми разработчиками были предложены концепции расширений для РСУБД, призванных обеспечить встроенную поддержку мультиклиентской архитектуры на уровне сервера [3]. Подобные системы позволяют проектировать структуру БД в соответствии с требованиями мультиклиентской архитектуры, выстраивая иерархию схем клиентов в объектно-ориентированном стиле.

Однако для работы облачного приложения одного сервера БД мало. Для того чтобы обслуживать запросы от десятков и сотен тысяч клиентов, необходим кластер. Проблема организации подобного кластера также поднималась рядом исследователей. В частности, рассматривались вопросы миграции данных в кластерах с мультиклиентской архитектурой и был предложен протокол для осуществления такой миграции [8]. Кроме того, рассматривались вопросы минимизации стоимости владения кластером [9, 10] в условиях использования in-memory баз данных и размещения инфраструктуры на основе услуг IaaS-провайдеров, а также способы размещения клиентов для минимизации издержек с соблюдением соглашений об уровне обслуживания (Service Level Agreements, SLA) [11].

2. Архитектура мультиклиентского кластера баз данных

Прежде чем приступить к описанию предлагаемой архитектуры мультиклиентского кластера баз данных, сделаем несколько замечаний относительно характера тех облачных приложений, для обслуживания которых он предназначается. Как известно, облачные решения предпочитают мелкие и средние компании, которые с их помощью сокращают свои издержки на поддержание IT-инфраструктуры. Чтобы привлечь внимание таких компаний, провайдеру требуется предоставлять свое решение по относительно низкой (в сравнении с традиционным "коробочным" вариантом) цене [12]. Следовательно, для того чтобы окупить разработку приложения, обслуживаемая клиентская база должна быть достаточно велика. Таким образом, рассматриваемое нами приложение имеет дело с потоком запросов, который можно разделить на N независимых и не пересекающихся подпотоков от каждого клиента компании. Так как клиенты – это мелкие и средние предприятия, то они не генерируют вычислительно трудоемких запросов, однако общее количество запросов велико.

Общая идея предлагаемого подхода к организации кластера состоит в том, чтобы реализовать дополнительный слой абстракции, предоставляющий программный ин-

терфейс, который был бы максимально близок к интерфейсу традиционных СУБД. Основные преимущества такого решения:

- упрощение разработки приложений, связанное с возможностью использования разработчиками уже имеющихся знаний, не заботясь об изоляции данных клиентов друг от друга;
- упрощение администрирования системы.

Предлагаемый подход может быть использован, так как БД приложения с рассматриваемой архитектурой представляет собой совокупность большого количества небольших, логически изолированных БД для каждого клиента. Основными задачами мультиклиентского кластера станут [13]:

- обеспечение изоляции данных клиентов друг от друга;
- управление размещением клиентских данных на серверах БД, динамическое перераспределение данных в зависимости от загрузки отдельных серверов и характера активности клиентов во времени;
- обеспечение отказоустойчивости в случае сбоя одного или нескольких серверов;
- маршрутизация запросов серверов приложений на соответствующий сервер БД;
- оценка эффективности использования ресурсов и диагностика состояния системы.

Схематично архитектура кластера представлена на рисунке 1. Рассмотрим подробнее некоторые из указанных функций.

2.1. Изоляция данных клиентов

В приложении рассматриваемой архитектуры данные различных клиентов должны быть изолированы друг от друга. Выше уже упоминались два способа обеспечения подобной изоляции, применяемые сегодня в РСУБД: это использование общих таблиц с идентификацией записей клиента по специальному дополнительному столбцу и изоляция данных клиентов на уровне схем. В контексте рассматриваемого решения может быть применен только подход с использованием изоляции на уровне отдельных схем, т.к. альтернативный подход с общими таблицами, во-первых, достаточно тяжело обобщить, а во-вторых, он значительно затрудняет адаптацию схемы БД под нужды конкретного клиента. Подход же с отдельной схемой для каждого клиента хотя и считается менее эффективным с точки зрения совместного использования ресурсов сервера, легко обобщается для нужд проектируемой системы. Сценарий работы кластера выглядит следующим образом:

1. при регистрации нового клиента на одном из серверов БД в составе кластера создается пустая схема для размещения его данных;

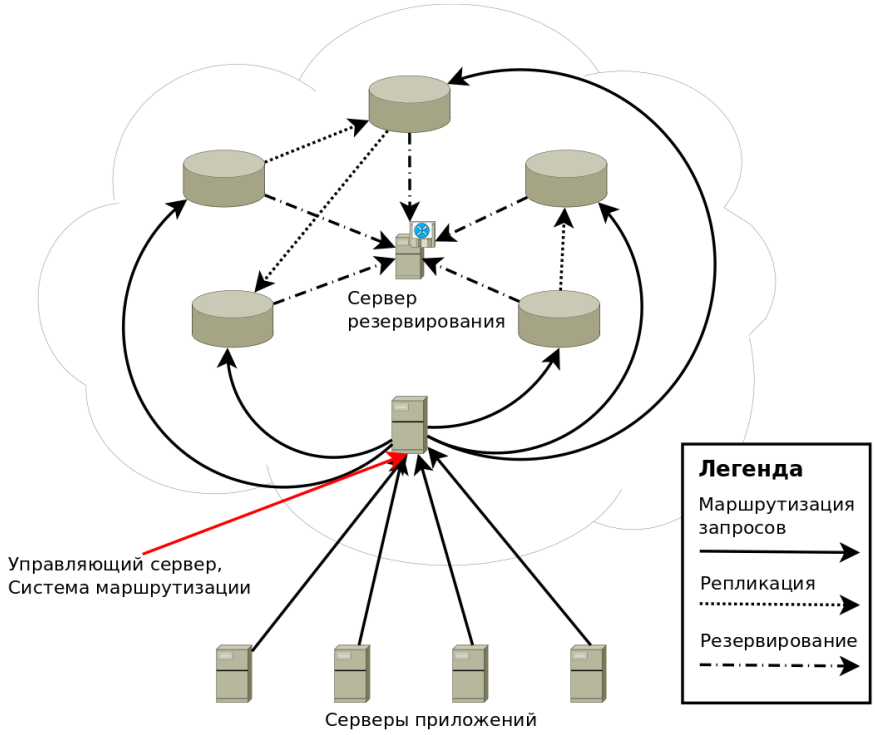

Рис. 1. Архитектура мультиклиентского кластера баз данных

2. при установлении соединения с кластером разработчик указывает идентификатор клиента, с данными которого он хочет работать; на основе этой информации кластер перенаправляет запросы на соответствующий сервер БД и адресует их в соответствующую схему;

3. разработчик инициализирует созданную схему, создавая таблицы, поля, индексы и другие объекты БД, после чего она становится готова к работе

За исключением нового звена – маршрутизатора запросов в кластере – описанная технология новой не является и упоминается, например, в [14].

2.2. Управление размещением данных на серверах БД

Одной из основных задач проектируемой системы является эффективное управление размещением данных клиентов. Для решения данной задачи система управления кластером должна располагать статистикой об интенсивности и характере потоков запросов отдельных клиентов, а также протоколом, позволяющим по необходимости максимально оперативно перемещать данные клиента с одного узла кластера на другой. Применяемые алгоритмы должны быть способны распределять клиентов по узлам кластера таким образом, чтобы обеспечить их максимально равномерную загрузку.

2.3. Обеспечение отказоустойчивости

Одной из наиболее важных потребностей облачного приложения является постоянная доступность решения, которая немыслима без обеспечения устойчивости к

сбоям отдельных узлов кластера. Основным способом обеспечения отказоустойчивости на уровне подсистемы хранения данных является репликация. В контексте рассматриваемой концепции организации кластера наиболее разумным вариантом кажется использование частичной асинхронной репликации master-slave. Репликацию необходимо настраивать таким образом, чтобы на другие узлы реплицировалась целиком вся схема клиента. При размещении реплик следует учитывать распределение клиентов по серверам кластера и загруженность каждого сервера. В идеальном случае реплики должны быть размещены таким образом, чтобы отказ любого сервера из состава кластера не приводил ни к отказу в обслуживании для любого из клиентов, ни к перегруженности тех серверов, на которые была распределена нагрузка выбывшего. Помимо обеспечения отказоустойчивости репликация позволит повысить производительность кластера путем переноса части транзакций, не модифицирующих данные клиента, на реплики.

2.4. Маршрутизация запросов

Предлагаемая концепция мультиклиентского кластера предполагает, что разрабатываемая система станет основной точкой взаимодействия приложения с серверами БД. При поступлении запроса от клиента, требующего выборки данных из БД, приложение устанавливает соединение с точкой входа кластера (возможно, установление соединения будет чисто логической операцией, физически же соединение будет поддерживаться постоянно) и указывает, с данными какого клиента и в каком режиме (чтение/чтение и модификация) ему необходимо работать. На основе этих данных подсистема маршрутизации запросов должна определить, какой из серверов БД, располагающих копией данных запрашиваемого клиента, будет обслуживать данный запрос. Запрос, предполагающий модификацию данных, может быть обслужен только мастер-сервером данного клиента, запрос же только на чтение может быть опционально перенаправлен на одну из реплик. Следует отметить, что компонент кластера, ответственный за маршрутизацию запросов, будет являться одним из наиболее нагруженных компонентов системы и, следовательно, не может использовать дорогостоящих алгоритмов балансировки нагрузки. Таким образом, эффективность распределения нагрузки по кластеру будет почти полностью определяться компонентом, ответственным за размещение данных клиентов и настройку репликации.

3. Разработка имитационной модели кластера

Первым этапом исследования предлагаемой концепции стала разработка имитационной модели мультиклиентского кластера баз данных [15]. Для того чтобы точнее отобразить особенности облачных приложений в модели, было осуществлено небольшое исследование одного из таких приложений.

3.1. Исследование существующих приложений

В качестве образца облачного приложения с мультиклиентской архитектурой была выбрана онлайн-система, предназначенная для сдачи налоговой отчетности в го-

сорганы и обмена электронными документами между предприятиями. На данный момент клиентская база приложения насчитывает примерно 43000 клиентов и продолжает расти. Приложение основано на классической трехуровневой архитектуре:

- клиент;
- сервер приложений;
- сервер БД.

В качестве сервера БД используется открытая СУБД Postgres SQL. Основные задачи по поддержке и обслуживанию кластера БД выполняются при помощи ряда специализированных приложений и сервисов. На данный момент в основном кластере компании насчитывается примерно 60 серверов баз данных, и его размер продолжает увеличиваться по мере роста клиентской базы. Для изоляции данных клиентов используются выделенные схемы, маршрутизация запросов осуществляется при помощи системы специализированных служб и метаданных, ассоциированных с сессией клиента.

Для анализа работы приложения разработан специализированный сервис, который собирает некоторую статистику об основных характеристиках: числе клиентов, их активности, размеру данных, количеству ошибок. Данный сервис активируется раз в 24 часа и проводит анализ журналов работы приложения. Ввиду некоторых особенностей реализации данного сервиса, собираемая статистика не очень точна (сбор полной статистики очень трудоемок и не оправдывает себя). Т.к. сервис способен анализировать лишь весь клиентский RPC-запрос в целом, который в общем случае может подразумевать исполнение нескольких SQL-запросов либо не подразумевать исполнения SQL-запросов вовсе, то мы не можем полагаться на точность его данных относительно таких численных характеристик запросов, как, например, среднее время выполнения. Однако для выявления основных закономерностей данная статистика вполне применима.

В рамках разработки имитационной модели мультиклиентского кластера наиболее интересным был вопрос моделирования клиентской базы приложения и порождаемого ею потока запросов. На первом этапе исследования в качестве основной характеристики клиента был взят размер его схемы на сервере БД. Соответственно на основе данных вышеописанного сервиса сбора статистики исследовались следующие вопросы:

- каково распределение размера данных клиентов;
- как взаимосвязаны (и взаимосвязаны ли) количество запросов от клиента и размер его данных;
- как взаимосвязаны (и взаимосвязаны ли) среднее время обработки запроса клиента и размер его данных.

Для нахождения ответа на первый вопрос была построена выборка вида (идентификатор клиента, размер его данных), позже преобразованная в выборку (размер данных, число клиентов со схемой данного размера). Результат представлен на рисунке 2. Анализ полученных данных показал, что размер средней схемы клиента

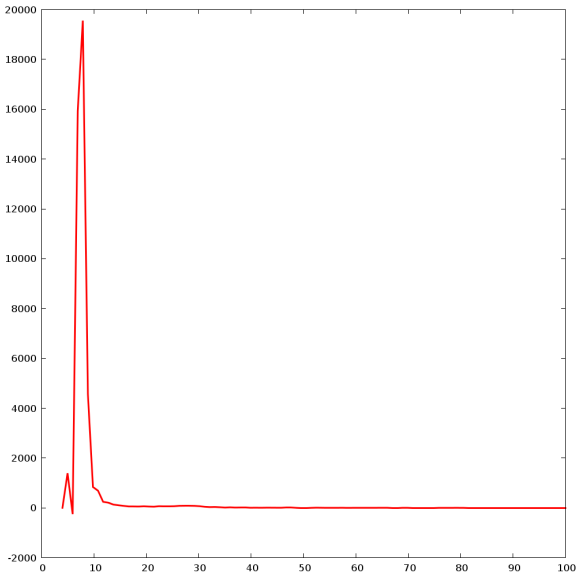

Рис. 2. Распределение размеров схем клиентов

составляет 8 мегабайт, а кривая распределения в целом соответствует кривой логнормального распределения.

Для исследования взаимосвязи между размером схемы клиента и количеством запросов, которые он генерирует, была построена выборка из примерно 40000 записей вида (размер схемы клиента, число запросов за промежуток времени). На основе данной выборки был вычислен коэффициент корреляции между указанными значениями, который оказался равен 0.7, что свидетельствует о достаточно сильной взаимосвязи между рассматриваемыми величинами.

Последним исследовался вопрос о взаимосвязи между размером схемы клиента и средней продолжительностью исполнения генерируемых им запросов. Для этого была построена выборка вида (размер данных клиента, средняя продолжительность запроса) на основе данных работы приложения за 24 часа, которая содержала приблизительно 7500 записей. На ее основе также был вычислен коэффициент корреляции между исследуемыми величинами, который оказался равен 0.03, что свидетельствует об отсутствии взаимосвязи.

Ввиду того, что исследование статистических данных показало, что размер схемы клиента и средняя продолжительность запроса никак не связаны между собой, было необходимо выяснить, как распределена средняя продолжительность запросов. Итог данного исследования представлен на рисунке 3. Из данного графика можно сделать вывод, что среднее время выполнения запросов также распределено логнормально.

3.2. Архитектура модели кластера

На основании собранных данных была разработана имитационная модель мультиклиентского кластера баз данных. Модель является графическим приложением под операционную систему Linux и разработана с использованием языка программиро-

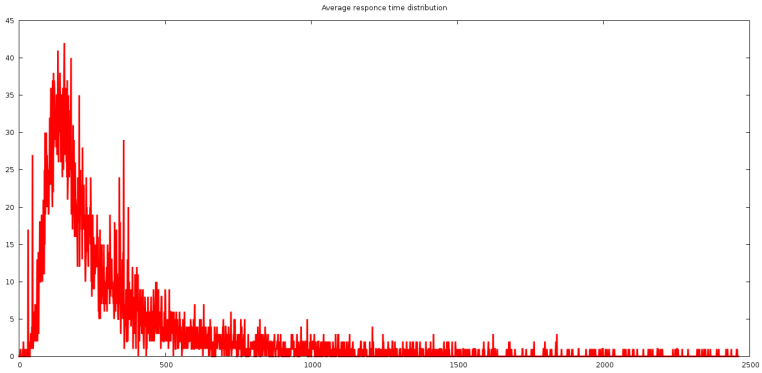

Рис. 3. Распределение средней продолжительности исполнения запросов

вания C++ и фреймворка Qt. Модель работает в условном дискретном времени, от одного события до другого.

Существует несколько основных сущностей, способных генерировать события:

- генератор запросов от клиентов и запросов на размещение в кластере новых клиентов;
- серверы БД в составе кластера – события окончания обработки запросов;
- генератор «сбоев» – события временного выхода из строя серверов кластера.

Архитектура модели схематично представлена на рисунке 4.

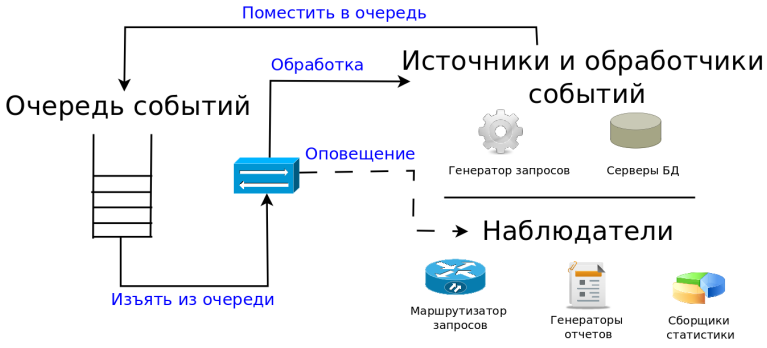

Рис. 4. Архитектура имитационной модели

Модель состоит из четырех основных типов объектов: очереди событий, диспетчера событий, обработчиков и источников событий, а также наблюдателей. Источники событий генерируют новые события и помещают их в очередь на обработку. После этого диспетчер извлекает первое (ближайшее) событие из очереди и направляет его соответствующему обработчику. Например, событие "поступление входящего запроса" сначала направляется на маршрутизатор запросов, а затем – тому серверу БД, который был выбран маршрутизатором для его обработки. Обо всех этапах обработки каждого события ядро модели оповещает всех наблюдателей, которые ответственны за сбор статистики о работе моделей, построение отчетов и выполнение ряда служебных задач. Одним из таких наблюдателей, в частности,

является алгоритм распределения данных, что позволяет ему собирать статистику и анализировать работоспособность кластера с целью дальнейшей оптимизации его работы. В ряде случаев окончание обработки одного события ведет к появлению нового, например, окончание обработки SQL-запроса, модифицирующего данные клиента, приводит к появлению запросов на репликацию данных изменений на сервера-реплики.

Генератор запросов моделирует пуассоновский поток событий. Интенсивность потока может задаваться как абсолютно (число запросов в единицу модельного времени, например, 100), так и относительно числа клиентов (например, 0.05 запроса на клиента в единицу модельного времени). При работе во втором режиме общая интенсивность потока равна произведению числа клиентов и указанного значения интенсивности, то есть динамически изменяется по мере размещения новых клиентов. К каждому генерируемому запросу прикрепляется специальный весовой коэффициент, влияющий на время выполнения запроса сервером. В соответствии с полученными статистическими данными, указанные коэффициенты распределяются согласно логнормальному закону, параметры которого можно указать при настройке генератора.

Разработанная модель способна генерировать большое количество графических отчетов о своей работе (среднее время ответа на последние 100 запросов, размеры очередей на отдельных серверах и кластере в целом, распределение потока запросов по серверам и т.д.).

4. Выводы и дальнейшие направления исследования

Первые эксперименты с использованием разработанной модели были посвящены анализу эффективности простых стратегий управления кластером. В частности, исследовались простейшие алгоритмы распределения данных (уравновешивание сервером по числу клиентов или суммарному объему данных) и алгоритмы маршрутизации запросов. В ходе моделирования было выявлено, что даже в тех случаях, когда интенсивность входного потока существенно меньше теоретической пропускной способности кластера, использование примитивных стратегий управления приводит к тому, что в определенные промежутки времени на определенных серверах возникают достаточно продолжительные периоды «всплеска» активности, ведущие к формированию очередей из запросов и, следовательно, к резкому падению производительности обслуживаемого облачного приложения с точки зрения тех клиентов, данные которых расположены на указанных серверах. Данный факт означает, что для реальной работы указанные стратегии неприменимы. В рамках исследования планируется дальнейшее совершенствование модели и выявление ключевых факторов, влияющих на эффективность работы предлагаемой системы управления кластером.

Список литературы

1. Candan K.S., Li W., Phan T., Zhou M. *Frontiers in Information and Software as Services* // ICDE, 2009.

2. Chong F., Carraro G., Wolter R. *Multi-Tenant Data Architecture* // Microsoft Corp. Website, 2006.

3. Schiller O., Schiller B. , Brodt A., Mitschang B. *Native Support of Multi-tenancy in RDBMS for Software as a Service* // Proceedings of the 14th International Conference on Extending Database Technology EDBT '11, 2011.

4. Jacobs D. , Aulbach S. *Ruminations on Multi-Tenant Databases* // Proceedings of BTW Conference, 2007.

5. Zawodny J. *Redis: Lightweight key/value Store That Goes the Extra Mile* Linux Magazine, QuarterPower Media, 2009.

6. *Benchmarking Top NoSQL Databases*. DATASTAX Corporation, 2013.

7. Eric A. Brewer, *Towards robust distributed systems* // Symposium on Principles of Distributed Computing - PODC, 2000.

8. Elmore A.J., Das S., Agrawal D., El Abbadi A. *Zephyr: Live Migration in Shared Nothing Databases for Elastic Cloud Platforms* // SIGMOD Conference, ACM, 2011.

9. Schaffner J., Januschowski T., Kercher M., Kraska T., Plattner H., Franklin M., Jacobs D. *RTP: Robust Tenant Placement for Elastic In-Memory Database Clusters* // SIGMOD Conference, ACM, 2013.

10. Fan Yang,Jayavel Shanmugasundaram, Ramana Yerneni. *A Scalable Data Platform for a Large Number of Small Applications* // Yahoo! Research Tech Report, 2008.

11. Lang W., Shankar S., Patel J.M., Kalhan A. *Towards Multi-tenant Performance SLOs* // ICDE, 2012.

12. Chong F., Carraro G. *Architecture Strategies for Catching the Long Tail* // Microsoft Corp. Website, 2006.

13. Boytsov E.A. , Sokolov V.A. *The Problem of Creating Multi-Tenant Database Clusters* // Proceedings of SYRCoSE 2012, 2012.

14. Riggs S., Krosing H. *PostgreSQL 9 Administration Cookbook*. Packt Publishing, Birmingham-Mumbai, 2010.

15. Boytsov E.A., Sokolov V.A. *The Development of an Imitation Model of a Multi-Tenant Database Cluster* // Proceedings of BMSD 2013, 2013.

Designing and Development of an Imitation Model of a Multi-Tenant Database Cluster

Boytsov E. A.

P.G. Demidov Yaroslavl State University, Sovetskaya str., 14, Yaroslavl, 150000, Russia

Keywords: databases, SaaS, multi-tenant architecture, scalability, imitation model

One of the main trends of recent years in software design is a shift to a Software as a Service (SaaS) paradigm which brings a number of advantages for both software developers and end users. However, along with these benefits this transition brings new architectural challenges. One of such challenges is the implementation of a data storage that would meet the needs of a service-provider, at the same time providing a fairly simple application programming interface for software developers. In order to develop effective solutions in this area, the architectural features of cloud-based applications should be taken into account. Among others, such key features are the need for scalability and quick adaptation to changing conditions. This paper provides a brief analysis of the problems in the field of cloud data storage systems based on the relational model and it proposes the concept of database cluster designed for applications with a multi-tenant architecture. Besides, the article describes a simulation model of such a cluster, as well as the main stages of its development and the main principles forming its foundation.

Сведения об авторе:
Бойцов Евгений Александрович,
Ярославский государственный университет им. П.Г. Демидова,
аспирант кафедры теоретической информатики