О проблеме организации кластеров мультиклиентских баз данных

Бойцов Евгений Александрович,
аспирант кафедры Теоретической Информатики
факультета ИВТ ЯрГУ им. П.Г. Демидова
boytsovea@yandex.ru

Аннотация— **Переход к парадигме SaaS несет как множество преимуществ конечным пользователям, так и множество проблем разработчикам программного обеспечения. Одной из таких проблем является организация хранилища данных, которое могло бы удовлетворить нужды клиентов компании-провайдера услуг, обеспечив при этом достаточно простой прикладной интерфейс для разработчиков, а также предоставить широкие возможности для администрирования и масштабирования. 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.**

Ключевые слова: базы данных; SaaS; мультиклиентская архитектура; масштабирование

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:

- 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.

Как известно, большинство приложений, разрабатываемых сегодня, основаны на работе с реляционными базами данных (РСУБД), которые, как и язык запросов SQL, стали де-факто стандартом в данной области. В случае облачного приложения все данные находятся в распоряжении и под ответственностью компании-провайдера услуги, которая обязуется обеспечить постоянный и быстрый доступ к ним для десятков или сотен тысяч своих клиентов одновременно. Невыполнение любого из этих требований повлечет наложение штрафных санкций на провайдера и, что наиболее важно, ущерб его репутации. Типовое соглашение о продаже услуг облачного сервиса гарантирует его доступность на уровне примерно 99%. Из данных обстоятельств следует, что эксплуатация облачного приложения требует больших затрат на организацию и обслуживание инфраструктуры хранения данных, поэтому естественным желанием разработчиков подобных сервисов является минимизация этих расходов и поиск архитектурных решений, которые позволили бы добиться такой минимизации без ущерба для производительности и функциональности приложения, насколько это возможно.

Одним из таких решений является мультиклиентская (multi-tenant) архитектура приложения (и, соответственно, базы данных). Основная идея данного подхода состоит в том, чтобы разделить один экземпляр приложения между несколькими клиентами (компаниями-покупателями услуг), тем самым, значительно сократив расходы на серверы приложений, веб-серверы и сопутствующие инфраструктурные элементы. Проектирование приложения 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.

Способы организации мультиклиентской архитектуры в облачных приложениях

В настоящий момент уже накоплен определенный опыт и известны два основных подхода для организации мультиклиентской базы данных [3,4,5].

Разделяемые таблицы

Данный подход является наиболее радикальным в вопросе о совместном использовании ресурсов сервера. При его применении к каждой таблице в БД добавляется дополнительный столбец, в котором хранится идентификатор клиента, которому принадлежит данная запись. К каждому SQL-запросу к БД, организованной подобным образом, требуется добавлять предикаты WHERE/HAVING, которые оставляют в результате только те данные, которые принадлежат именно данному клиенту. Также существуют проекты расширений языка SQL [1], которые добавляют такие предикаты автоматически, однако пока это только концепции, находящиеся на стадии разработки и исследования. Преимуществами такого подхода являются:

- лучшее использование дискового пространства;
- небольшой размер словаря данных;
- лучшее использование кэша планировщика запросов (то есть, меньшее время анализа запроса и генерации плана его исполнения).

Недостатки данного подхода таковы:

- раздувание размера таблиц и их индексов [2]. Из этого недостатка вытекает требование очень высокой квалификации разработчика, пишущего запросы к БД;
- необходимость каждый раз писать предикат отбора данных, принадлежащих конкретному клиенту, что ведет к появлению ошибок доступа, когда клиенты могут увидеть данные, которые им не принадлежат;
- сложность организации репликации и резервного копирования данных отдельного клиента.

В целом данный подход хорошо себя показывает в том случае, если схема данных приложения не обширная, а запросы относительно просты.

Выделенная схема для клиента

Данный подход является промежуточным между полной изоляцией данных клиентов друг от друга в отдельных БД и использованием общих таблиц для совместного хранения данных всех клиентов. Изоляция данных клиентов друг от друга достигается путем создания собственного набора объектов БД для каждого из них. Преимущества данного подхода:

- унификация кода запросов к БД и простота их написания [2];
- относительная простота выполнения резервного копирования и репликации данных отдельного клиента;
- уменьшение рисков, связанных с ошибками доступа к данным;
- упрощение администрирования.

Среди недостатков данного подхода можно выделить «раздувание» словаря БД, из-за которого уменьшается возможность использования кэша планировщика запросов, вынуждая его каждый раз строить план выполнения запроса заново [1], а также не такое оптимальное использование дискового пространства, как в случае подхода с разделяемыми таблицами [1]. В целом, данный подход хорош в том случае, если схема данных приложения сложна, а типовой запрос делает выборку из целого набора таблиц, выполняет вложенные подзапросы и другие сложные манипуляции с данными.

Ограничения существующих подходов и цели исследования

Оба подхода, несмотря на то, что большинство современных СУБД не поддерживают их напрямую и не рассчитаны на работу в указанных режимах, успешно находят свое применение в разработке. Однако, 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, нежели количество баз данных, является распределение 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. Целями данного исследования являются:

- разработка алгоритмов балансировки нагрузки в мультиклиентских кластерах баз данных облачных приложений;
- исследование разработанных алгоритмов на предмет эффективности, безопасности и корректности, включая имитационное моделирование и нагрузочное тестирование;

Архитектура системы управления мультиклиентским кластером

Среди основных характеристик мультиклиентской базы данных, с учетом которых следует разрабатывать систему управления кластером, можно выделить огромный и постоянно растущий суммарный размер базы, а также небольшой размер данных среднего клиента [3]. Исходя из указанных требований и особенностей, опишем предлагаемый проект системы управления облачным кластером.

Основная идея предлагаемого решения — это введение нового уровня абстракции между серверами БД и серверами приложений, главными задачами которого будут являться:

- маршрутизация запросов сервера приложений по идентификатору клиента на соответствующий сервер БД;
- управление размещением клиентских данных на серверах БД, динамическое перераспределение данных в зависимости от загрузки отдельных серверов и характера активности клиентов во времени;
- управление репликацией данных между серверами БД;
- управление резервным копирование данным;
- обеспечение отказоустойчивости в случае сбоя одного или нескольких серверов;
- оценка эффективности использования ресурсов и диагностика состояния системы.

Систему предполагается реализовать в виде набора взаимосвязанных служб, использующих собственную БД для поддержания карты кластера и сбора статистики о характере нагрузки. Общая архитектура системы представлена на рисунке 1.

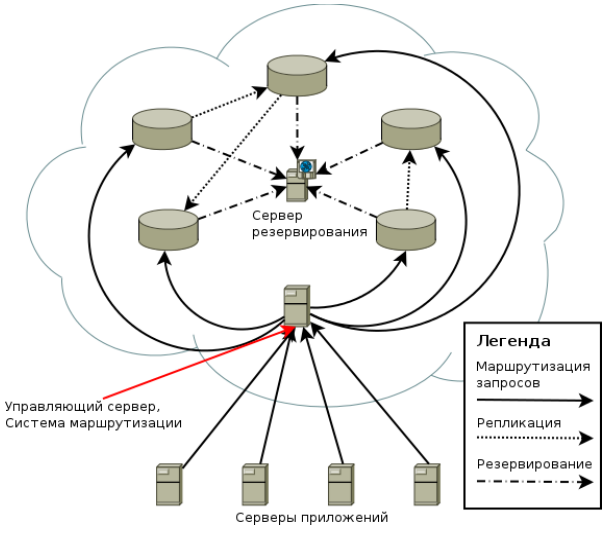

Рисунок 1 Архитектура мультиклиентского кластера баз данных

Перейдем к более подробному рассмотрению указанных выше задач системы.

Новым звеном в организации взаимодействия серверов приложений и серверов БД становится выделенный сервер системы управления кластером, маршрутизирующий запросы на основании идентификатора клиента, к данным которого адресован запрос, характера самого запроса, а также сведений о текущей

загрузке системы. Именно с ним должны будут иметь дело разработчики прикладных решений. Задачей данного компонента является максимально быстрое обслуживание поступающих запросов и передача их на исполнение.

Вторым и наиболее важным и сложным компонентом системы должен стать сервер управления размещением данных и балансировкой нагрузки. Его задачами являются:

- первичное размещение данных при развертывании системы, создании новых клиентских схем и добавлении серверов к кластеру;
- сбор статистики использования системы клиентами и их пользователями;
- анализ загрузки серверов, построение административных отчетов;
- управление размещением клиентских данных;
- диагностика состояния системы на предмет необходимости добавления вычислительных ресурсов и устройств хранения данных.

Именно данный компонент представляет наибольшую ценность, так как от успешности его работы будет зависеть производительность всего приложения. Ключевыми показателями, с помощью которых можно оценивать его эффективность, являются среднее время отклика сервиса, доступность сервиса, равномерность загрузки серверов.

Ядром данной службы должен стать алгоритм анализа загрузки кластера и необходимости перераспределения данных. При принятии решения о распределении данных клиентов он должен учитывать такие факторы, как производительность серверов и соотношение их возможностей, имеющиеся в распоряжении свободные ресурсы, а также историю активности отдельных клиентов.

Для такого алгоритма можно предложить различные стратегии и на данный момент не ясно, какой из них следует отдать предпочтение. Весьма вероятна ситуация, что эффективной в общем случае версии алгоритма не будет найдено, и конечная реализация будет содержать несколько вариантов, показавших себя наилучшим образом при различных внешних условиях. В этом случае выбор наиболее подходящего алгоритма из набора предложенных станет задачей администратора системы управления кластером.

[1] Oliver Schiller, Benjamin Schiller, Andreas Brodt, Bernhard Mitschang, “Native Support of Multi-tenancy in RDBMS for Software as a Service“, труды конференции EDBT 2011

[2] D. Jacobs, S. Aulbach, “Ruminations on Multi-Tenant Databases“, труды конференции BTW, страницы 514–521, 2007.

[3] F. Chong, G. Carraro, “Architecture Strategies for Catching the Long Tail“, веб-сайт корпорации Microsoft, 2006.

[4] F. Chong, G. Carraro, R Wolter, “Multi-Tenant Data Architecture“, веб-сайт корпорации Microsoft, 2006.

[5] K.S. Candan, W. Li, T. Phan, M. Zhou, "Frontiers in Information and Software as Services", труды конференции ICDE, 2009, страницы 1761-1768.

[6] E. Boytsov, V. Sokolov, “The Problem of Creating Multi-Tenant Database Clusters“, труды конференции SYRCoSE, 2012, страницы 172-177

[7] E. Boytsov, V. Sokolov, “Multi-tenant Database Clusters for SaaS“, труды конференции BMSD, 2012, страницы 144-149