УДК 004.62

МАСШТАБИРОВАНИЕ БАЗЫ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ ШАРДИНГА

И.В. Слепов
Инженер-программист ЧУП «БелТехноСОфт»

А.Д. Тюменцев
Магистрант БГУИР

В.М. Бондарик
Декан факультета доуниверситетской подготовки и профессиональной ориентации БГУИР, кандидат технических наук, доцент

Белорусский государственный университет информатики и радиоэлектроники, Республика Беларусь
ЧУП «БелТехноСОфт», Республика Беларусь
E-mail: nikeslepov@gmail.com, lexatum@gmail.com, bondarik@bsuir.by

Аннотация. Описан один из возможных вариантов масштабирования базы данных с использованием шардинга для увеличения скорости обработки структурированных данных и производительности всей системы. Определены большинство возможностей и ограничений шардинга. Описано применение его на практике в приложении-прототипе.

Ключевые слова: масштабирование, шардинг, база данных, производительность.

Современные приложения при своей работе используют большое количество запросов к внешним базам данных. Существует несколько способов повышения производительности запросов к базам данных.

Можно повышать производительность экстенсивным путем: обновить оборудование для сервера, добавить количество оперативной памяти и т.д. Этот принцип называется вертикальным масштабированием. Однако этот способ может быть достаточно дорогим, длительным по времени, да и имеет свой предел роста. Можно купить лучшее оборудование, однако оно может не справиться со всеми требованиями приложения [1].

Второй способ, называемый горизонтальным масштабированием, предполагает расширение вычислительных ресурсов, доступных приложению, за счет увеличения количества серверов, на которых размещена база данных. Если раньше база данных была расположена на одном сервере, и в какой-то момент она перестала справляться с нагрузкой, то пользователи могли заметить значительное увеличение времени отклика приложения. Чтобы решить эту проблему, можно использовать стратегию шардинга.

Шардинг – это прием, который позволяет распределять данные между разными физическими серверами. Это существенно облегчает обработку данных [2].

Существует два типа шардинга: вертикальный и горизонтальный. В нашем случае применен горизонтальный шардинг – разделение базы данных на разные сервера. Это необходимо использовать для огромных таблиц, которые не умещаются на одном сервере и замедляют работу приложения. Разделение базы данных делается по следующему алгоритму:

на нескольких новых серверах создается одна и та же база данных (только структура, без данных);

определяются таблицы, данные которых должны присутствовать во всех базах данных, и те, у которых данные будут уникальными во всей системе;

перед каждым обращением к базе данных происходит выбор нужного соединения.

При реализации второго этапа алгоритма необходимо произвести классификацию таблиц, что позволит в дальнейшем в программном коде добавить процесс синхронизации и переноса данных.

Рассмотрим процесс применения шардинга в приложении более подробно. Само приложение написано с использованием языка программирования Ruby и фреймворка на его основе Ruby on Rails, роль базы данных представляет Percona Server. Для реализации шардинга был использован гем (библиотека) ar-octopus. Это лучший способ сделать шардинг базы данных в ActiveRecord [3]. ActiveRecord – шаблон проектирования приложений, популярный способ доступа к данным реляционных баз данных в объектно-ориентированном программировании. Схема Active Record – это подход к доступу к данным в базе данных. Таблица базы данных или представление обёрнуты в классы. Таким образом, объектный экземпляр привязан к единственной строке в таблице. После создания объекта новая строка будет добавляться к таблице на сохранение. Любой загруженный объект получает свою информацию от базы данных. Когда объект обновлён, соответствующая строка в таблице также будет обновлена. Класс обёртки реализует методы средства доступа или свойства для каждого столбца в таблице или представлении [4].

Для инициализации шардинга была создана таблица *shards*, в которой находится вся информация о наших шардах: уникальное имя шарда и данные подключения к серверу базы данных. Затем мы создали соответствующую модель в приложении (класс *Shard*), чтобы через ActiveRecord связать её с базой данных. После этого определили родительскую таблицу, в нашем случае *organizations*. Это было сделано для того, чтобы можно было идентифицировать те дочерние таблицы, которые каким-либо образом содержали данные, необходимые для участия в процессе переноса информации между шардами. Следующим шагом было создание таблицы *organization_shards*, чтобы связать между собой записи таблиц *shards* и *organizations*. Кроме этого, данные из этой таблицы помогли решить проблему с постоянным выбором соединения из третьего шага алгоритма разделения баз данных – мы заранее знали, с каким шардом работаем и у нас не было необходимости менять соединение между главным шардом по умолчанию и дополнительным шардом.

Далее рассмотрим процесс переноса данных между шардами. Сам процесс в реальном времени может быть длительным из-за большого количества данных, что уменьшит скорость работы как сервера, с которого будут переноситься данные, так и сервера, на который они будут мигрировать.

Перед самим процессом необходимо заблокировать все операции на таблицах, данные которых будут переноситься на другой шард. Затем, чтобы сократить время передачи, создадим специальный background процесс, который сгенерируют в файловой системе SQL файлы, в которых будут находиться все данные для импорта, а также создаст новый background процесс, который займется импортом этих данных. Все эти действия выполняются на специальном сервере, чтобы не замедлять работу самого приложения.

Однако у горизонтального шардинга есть и свои ограничения. Одним из таких ограничений является поиск и фильтрация данных. Данная проблема решилась путем написания дополнительного кода, в котором производится выборка данных со всех шардов, затем данные группируются в один массив, и в итоге в полученном массиве данных производится внутренняя фильтрация данных. Еще одним ограничением при горизонтальном масштабировании может стать уникальность данных, однако эта проблема решается довольно просто. Необходимо еще на этапе разработки архитектуры приложения заранее спланировать возможную нагрузку и на основе полученных результатов мы сможем понять, какое количество серверов может понадобиться для комфортной работы приложения. Например, нам понадобится 3 дополнительных сервера баз данных для шардинга. Чтобы не произошла ситуация, когда две или более записи с разных шардов имеют одинаковый ID, необходимо

произвести следующие операции с серверами баз данных:

на каждом сервере баз данных нужно установить начальное значение ID, т.е. на первом сервере это будет 1, на втором – 2, и т.д.;

определить значение auto_increment величине, равной количеству планируемых серверов.

Из последнего условия вытекает преимущество горизонтального масштабирования –мы можем бесконечно масштабировать нашу базу данных, однако лучше заранее определиться с возможным количеством серверов.

При разработке программного комплекса поддержки медицинского страхования было применено горизонтальное масштабирование с использованием шардинга, т.к. на стадии планирования приложения было решено изначально предотвратить возможное уменьшение производительности приложения. Разработанное ПО успешно работает в сетевой версии со множеством клиентов.

Процесс горизонтального масштабирования базы данных является очень эффективным способом, чтобы добиться как общей производительности системы и обработки больших объемов данных, так и снизить общую нагрузку на один единственный сервер. Данный процесс может быть применен как при планировании архитектуры приложения, так и на готовом проекте. Однако второй вариант содержит в себе много подводных камней и должен применяться только в крайнем случае, когда производительность приложения снизилась до критического уровня.

Список литературы

[1]. Горизонтальное масштабирование базы данных реального проекта с помощью SQL Azure Federations [Электронный ресурс] – Режим доступа: https://habrahabr.ru/company/epam_systems/blog/192984/

[2]. Масштабирование баз данных – партиционирование, репликация и шардинг [Электронный ресурс] – Режим доступа: https://web-creator.ru/articles/partitioning_replication_sharding

[3]. Octopus – Easy Database Sharding for ActiveRecord [Электронный ресурс] – Режим доступа: https://github.com/thiagopradi/octopus

[4]. ActiveRecord [Электронный ресурс] – Режим доступа: https://ru.wikipedia.org/wiki/ActiveRecord

SCALING A DATABASE WITH USING SHARDING

I.V. SLEPOV
Software Engineer «BelTechnoSoft »

A.D. TUMENTSEV
Master student of the Belarusian State University of Informatics and Radiolectronics

V.M. BONDARIK, PhD
Dean of the faculty of pre-university training and vocational guidance BSUIR, Assistant professor

Belarusian State University of Informatics and Radioelectronics, Republic of Belarus
E-mail: nikeslepov@gmail.com, lexatum@gmail.com, bondarik@bsuir.by

Abstract. The main goal of this research is to describe one of the possible ways of scaling a database using sharding to increase the processing speed of structured data and, in general, the performance of the entire system. During the research, I was able to determine most of the possibilities and limitations of sharding, and also applied it in practice in a prototype application.

Key words: scaling, sharding, database, performance.