



PERCONA
Performance Consulting Experts

Масштабирование MySQL с помощью шардинга и репликации

Окт 8, 2008

HighLoad++ Мастер Класс

Москва, Россия

Зайцев Петр, Percona Inc

Проблемы Веб Приложений

- Уровень Генерации Страниц
 - Легко добавить больше серверов
 - Большинство приложений не имеют зависимостей между серверами
- Статические Веб Данные
 - Картинки Видео Итд
 - Нет зависимостей легко добавлять сервера диски
 - CDN может взять на себя нагрузку
- База данных
 - Обычно наиболее сложная для масштабируемости из за сложных взаимосвязей.

Классы Веб Приложений

- Новые возможности существующих приложений
 - Рекомендация продукта в Amazon.Com
 - Сразу большая нагрузка
- Типичные стартапы
 - Медленный но ускоряющийся рост (10% в месяц)
 - Часто есть время для решения проблемы
- Моментальные хиты
 - FaceBook приложения хороший пример
 - Очень быстрый рост нагрузки и размера базы данных
 - Миллионы пользователей в течении нескольких недель

Подходы к Дизайну Приложений

- “Думай о сегодня”
 - Решим сегодняшние проблемы а с завтрашними разберемся завтра.
 - Намеренный выбор или по неопытности
 - Типично для “студенческих” стартапов
- “Высокопрофессиональный подход”
 - Планирование производительности качества заранее
 - Все делается правильно сразу даже если это требует больше времени
 - Типично для опытных команд
- Большинство приложений где-то посередине

Какой разумный подход ?

- Установите временной горизонт для текущей архитектуры
 - “Я делаю прототип и возьму денег через 3 месяца и найму умных людей чтобы они все сделали”
- Определите требования производительности (нагрузка размер базы данных итд).
- Архитектура должна их обеспечивать
 - Не достаточно масштабируемая убьет приложение
 - Но и переусложнение чревато тем что продукт никогда не увидит свет

А может быть есть панацея ?

- MySQL Cluster ?
- Continuent/Sequoia ?
- KickFire ?
- MySQL Proxy ?
- BigTable ?
- SimpleDB ?
- У всех есть свои сложности
 - И вам лучше знать о них заранее

Рост MySQL проекта

- Обычно все начинается с одного сервера
 - Быстрые JOIN, данные легко доступны
- Бустро упирается в CPU или Ввод Вывод
 - И не стоит забывать о проблемах масштабирования MySQL на многопроцессорах
- “Scale-UP” ограничен и дорог
 - Особенно когда интересна скорость обработки в одном потоке
- Дальнейшие обычные выводы:
 - Функциональный Партишенинг
 - Репликация

Функциональный Партишенинг

- “Давайте положим базу данных форумов на другой сервер”
 - Выбор набора таблиц который максимально независим от других
 - Простые Join запросы можно сделать руками или использовать Federated таблицы
- Проблемы
 - Эти “части” могут вырастать слишком большими
 - И дальнейшее функциональное разбиение может быть очень сложным или невозможным

Репликация

- Многие приложения только читают данные
 - Правда многие из этих чтений идут из memcache или другого кэша
- Используем слейвы для выполнения чтений
- Репликация асинхронна
 - Нужно аккуратно чтобы не читать старые данные
 - Слейвы могут быть на разных позициях
- Не помогает масштабировать запись
 - Слейвы обычно могут выдерживать меньше записей чем мастер так как выполняют записи в один поток
- Кэши слейвов обычно очень дублированы

Что делать с асинхронностью

- Уровень запросов
 - Слейв для отчетов не критичных по времени
- Уровень сессий
 - Если пользователь не модифицировал данные можно давать старые данные (но консистентно)
 - Сохранять binlog позицию когда была модификация
- Версия/timestamp
 - Если данные этого пользователя не были модифицированы читаем для него все со слейва
- MySQL Proxy
 - Идет работа для автоматической маршрутизации запросов

Репликация и Запись

- Очень быстрая деградация (упрощение)
 - Занятость мастер 50% - 2 слейва 50% свободны для чтений
 - Эквивалент одного сервера делающего только чтения
 - Выросла загрузка на 50% стала 75% На слейвах осталось 25% свободно для чтения
 - Получаем эквивалент $\frac{1}{2}$ сервера то есть в 2 раза меньше
- Проблема одного потока
 - Использует один процессор
 - Один параллельный запрос к диску (в основном)
 - Больше всего проблема на medium/high end servers

Оптимизация Репликации

- Используйте Persona патчи для того чтобы найти какие запросы грузят репликацию
- “Построчная” репликация в MySQL 5.1
 - Не нужно тратить всемя на Slave на поиск
- Замена сложных запросов на обновление на выборку и обновление
 - INSERT ... SELECT <Очень сложный запрос>
 - Меняем на
 - SELECT
 - Сохраняем результат в память приложения
 - INSERT

Минимизация Задержки

- Один поток – длинные запросы все блокируют
- Разбиение запросов
 - DELETE LIMIT 100 в цикле
 - Часто хорошо с заменой на выборку и удаление
- ALTER TABLE делать локально
- Можно использовать “помошник” для сложных запросов
 - Master вставляет задания в таблицу очередь
 - Скрипт смотрит и выполняет на каждом слейве
 - Можно управлять какие слейвы должны его выполнить

Репликация и Кэширование

- Пример 20GB база данных и 16GB памяти
 - Она почти полностью влазит в память и записи почти нет
- Данные выросли до 100GB и мы добавили 5 слейвов – можем делать в 6 раз больше чтений
 - Но так как теперь на каждом слейве в память влазит 1/6 то все тормозит
- Можно чуть улучшить
- Дублируется вся база данных тоже
 - Быстрые RAID диски не дешевы
 - А если вы используете SSD то это большая проблема

Как улучшить Кэширование

- Роли слейвов
 - Слейв для сложных отчетов
 - Слейв для полнотекстового поиска
- Маршрутизация запросов
 - Все запросы для данной сессии/пользователя идут на один слейв (stickiness)
 - Фиксированное разбиение (четные нечетные user_id)
- Полностью избежать пересечение не удастся
- Содержимое кэша из за записей дублируется на всех серверах

Шардинг

- Когда ф-ое разбиение и репликация не помогают
- Разбиваем данные на маленькие кусочки и храним на многих серверах
- “Единственное” решение для крупного масштаба
- Нужно аккуратное планирование
- Может быть не просто реализовать
 - Особенно если это не предусмотрено дизайном
- Как разбивать данные очень критичный вопрос
 - Часто бывает надо несколько копий данных с разным разбиением

Шардинг и Репликация

- Шардинг используется вместе с репликацией
 - В основном для высокой доступности
- Один сервер падает раз в 3-5 лет
 - 200 серверов – падение раз в неделю
 - И данные недоступны для ряда клиентов
- Мы любим использовать Master-Master репликацию – просто использовать.
- Так же помогает с обслуживанием
 - Обновление – замена железа или операционки?
 - ALTER/OPTIMIZE MySQL Таблиц ?

Как разбивать данные ?

- Большинство запросов должно быть к одному шарду
- Шард не становится слишком большим
 - Хорошо: Блоги по `user_id`
 - Плохо: Шардинг по `country_id`
 - Большая часть траффика может быть из одной страны
- Множество разбиений одновременно
 - По Книге И по пользователю одновременно
- Полное дублирование данных или сохранение только указателя

Технологии Шардинга

- Фиксированное хеширование
 - Четные на сервер А, нечетные на сервер В
 - Не гибко. Может быть улучшено через консист. хеш.
- Словарь данных
 - Данные пользователя 25 на сервере D
 - Гибко но словарь может стать узким местом
- Смешенное хеширование
 - Объекты хэшируются не большое число объектов которые отображаются через словарь
- Прямое указание адреса - `<shardid><objectid>`

Таблицы и Шарды

- Каждый пользователь получает свою группу таблиц (или баз данных)
 - Слишком много таблиц
- Один набор таблиц на сервер
 - Таблицы могут быть слишком большие
 - Тяжелее перемещать данные между серверами
 - Проще для “старых” приложений
- Что-то среднее
 - Много пользователей на группу; много групп на сервер
 - Наиболее гибко но сложнее для реализации.

Планирование загрузки

- Удобно если можно динамически добавлять шарды
- Надо оставлять место на рост (часто можно прикинуть сколько объектов на шард подходит)
- Надо учитывать характер использования
 - Например многие из пользователей играют с системой пару месяцев и уходят
- Параметры роста данных и характер доступа к ним

Архивирование Данных

- Иногда кроме простого разбиения по пользователем используется разбиение по времени
- Старые данные могут храниться на архивных серверах
- Модификация словаря с указанием даты до которой данные архивированы.

Перемещение Данных

- Требуется для балансировки рано или поздно
- Перемещаем по одному пользователю
 - Делая пользователя только для чтения временно
 - Часто достаточно сложно
 - Особенно если идентификаторы не содержат user_id в себе
 - Долго
 - Минимально затронута ф-ость системы
- Перенос потаблично/целая база данных
 - Легко (стандартные инструименты) и быстро
 - Больше ограничение ф-ости системы

Что управляет шардингом

- Уровень доступа к данным
 - Хорошо если разработка делается для шардинга сразу
- Уровень доступа данных делает парсинг запросов
 - Выделяя `user_id=X` из запроса и направляя
- HiveDB <http://www.hivedb.org>
- HSCALE <http://www.hscale.org>
- Spock Proxy

Доступ к глобальным данным

- Часто нужно объединить данные с какими то глобальными таблицами.
 - Информация о пользователях страны итд
- Делать Join руками
 - Так же делает кэширование более эффективным
- Глобальное Дублирование таблиц
 - Может быть MySQL репликация или ручное копирование
- Доступ через Federated таблицу
 - Очень аккуратно но для простых запросов работает
 - Сложно реализовать High Availability с ней

Агрегация данных

- Глобальный поиск, аналитика, рейтинг, лента
- Несколько Шардов или все шарды
 - Думайте о таких доступах при дизайне шардинга
- Создание саммари таблиц
- Параллельное выполнение на многих шардах
 - Может быть сложно сделать для некоторых языков
- Экспорт данных для аналитики
 - Может у кого есть Netezza или Kickfire для анализа ?
- Использование других решений
 - Nutch, Sphinx, Hbase, итд

Кеширование

- Шардинг не избавляет от необходимости
- Обязательно для больших приложений
- Может уменьшить нагрузку к базе данных в 10x+ раз
- Только откладывает время когда система должна быть реплицирована или произведено разбиение.

Спасибо что пришли

- Вопросы
 - pz@percona.com
- Да мы занимаемся консалтингом по MySQL и веб масштабированию
 - <http://www.percona.com>
- Наша книга
 - Переписанное 1е издание

