



HighLoad

Техники масштабирования Веб-приложений

Петр Зайцев

pz@mysqlperformanceblog.com

Проблемы производительности

- Время отклика
 - Страница долго грузится, даже для одного активного пользователя
 - Типично обрабатывается много данных
- Пропускная способность (req/sec)
 - При большом числе активных пользователей время отклика растет и система не справляется
- Пути решения могут быть разные



В чем еще сложности?

- Число запросов часто растет вместе с объемом данных
 - $O(N)$ алгоритмы требуют $O(N^2)$ ресурсов
- Данные в памяти в 1000 раз быстрее чем на диске.
- Блокировки на уровне приложения, базы данных, операционной системы



Что масштабируем?

- Динамический контент
- Статический контент
- Базу данных



Динамический контент

- Пропускная способность
 - Решается увеличением числа Web серверов
 - Для одной и той же задачи может использоваться в 10 раз больше железа
 - Оптимизация скорости важна для эффективности, но не жизни проекта
- Время отклика
 - Изменение алгоритма
 - Оптимизация, например модули на C
 - Параллельные алгоритмы (на разных уровнях)

Статический контент

- Достаточность памяти и пропускной способности дисков
- nginx/lighttpd легко отдают 20.000+ маленьких объектов в секунду на сервер
- Типичная проблема – завязка на одну большую файловую систему
 - Разделить через проху-rewrite
 - Или не иметь проблему изначально
 - Photobucket
 - Отдавать напрямую лучше, чем через NFS

База данных

- Обычно наиболее сложный для масштабирования компонент
- Особенно сложно, если приложение создано без оглядки на масштабирование
- Очень важно кэширование (как впрочем и на других уровнях)



Пути масштабирования БД

- Купить более мощный сервер
 - Хороший подход, если результаты нужны быстро, а никакой возможности изменить приложение нет
 - Часто не решает проблемы времени отклика
 - Возможности роста ограничены
 - Финансово неэффективно
 - Особенно учитывая, что нужен второй такой же сервер (для надежности)

Репликация

- Относительно простое решение
 - Не так много изменений в приложении
- Достаточно большой предел для ряда приложений
 - YouTube еще год назад имела одну базу данных
- Важно ограничить запись
- Обеспечить данные в памяти
- Репликация не параллельна в MySQL



Потабличное разбиение

- Выделяются компоненты и их таблицы располагаются на разных серверах
 - И часто реплицируются
 - Например “сервер регистрационных данных”, “форумы” итд
- Относительная легкость изменения
- Не всегда легко выделить малозависимые компоненты
- Ограниченная масштабируемость



“Шардинг” – разбиение данных

- Данные распределены на много серверов
 - Которые обычно реплицируются
- На одном сервере может быть больше, чем один “шард”
 - Легкость балансировки, меньше блокировки
- Требует правильного разбиения данных
- Может требоваться более одного разбиения



Какой путь для вас?

- Оцените требования приложения к росту
- Объем базы данных
- Интенсивность записи
- Возможности кэширования
- Навыки персонала
- Временные рамки и бюджет разработки



Как разбивать данные

- Старые приложения
 - Так, чтобы минимизировать модификации
 - Часто схема полностью дублируется, включая имена баз данных, и добавляется логика определения нахождения объекта
- Новые приложения
 - Для обеспечения лучшей производительности и сопровождаемости
 - Несколько баз данных на сервере с разными именами



Критерий разбиения

- Хэш разбиение для объектов
 - Четные на сервер 1, нечетные на сервер 2
 - Просто, но очень негибко
- Таблица привязки для каждого элемента
 - Требуется обращение к словарю
 - Гибко, удобно балансировать данные
- Блочная структура
 - 1024 блока, которые приписаны к серверам
 - Гибко, но “блоки” можно переносить физически



Кластеризация разбиения

- Минимизация операций, требующих доступа ко многим (всем) группам
- Зависит от приложения – типично пользователь, сайт
 - Или группа – регион, школа итд
- Может требоваться несколько разбиений и дублирование данных
 - “Пользователь” и “Фильм”
- Некоторые приложения допускают произвольное разбиение



Проблемы с разбиением данных

- JOIN с системными/глобальными таблицами
 - “ручной join”
 - Нередко это бывает даже быстрее
 - “репликация” системных/глобальных таблиц на все сервера
 - Хорошо, если данных мало и меняются нечасто
 - Federated Storage Engine
 - Иногда работает, обычно мееедленно
 - Сложности с обеспечением доступности при падении мастера



Агрегация данных по всем разбиениям

- Действительно ли этого не избежать?
 - Дублирование данных
 - Пре-генерация данных итд
- Вручную пробежать все группы
 - Простое решение, работает для группировки
- UNION ALL VIEW на Federated Tables
 - Хорошо работает, когда время получения ответа не важно



Агрегация данных по всем разбиениям

- Самописные системы параллельной агрегации
 - Особенно хорошо, когда групп ограниченное число
 - Technorati Web Services Layer
 - HiveDB (система для решения этой проблемы)
- Sphinx
 - “Неожиданно” оказался полезным для глобальной агрегации далеко за пределами полнотекстового поиска



А как же MySQL Cluster?

- Зачем мучаться с разбиением данных вручную, давайте поставим MySQL Cluster и он все за нас сделает автоматически?
- Хорошо в теории, но плохо работает на практике для многих задач
 - Сложности с выполнением сложных запросов (по меньшей мере пока)
 - Не работает с большими наборами данных, даже в версиях 5.1.x
 - Достаточно хорошо работает, например, для хранения сессий в Web приложениях



Веб 2.0 мантра – кэшировать и еще раз кэшировать

- Набор технологий для избежания тяжелой работы по генерации данных
 - Классическое кэширование
 - Прегенерация статических данных
 - Таблицы содержащие суммарные данные итд
- Избежать работы лучше, чем ее оптимизировать!



Кэширование на всех уровнях

- “Железные” CPU Cache, RAID Cache
- Память как кэш
 - Файловый кэш и внутренние кэши БД
 - Требуют аккуратной настройки для лучшей производительности
 - Часто определяют размер данных, с которыми сервер будет эффективно справляться
- MySQL Query Cache
 - Кэш результатов запросов (вместо исходных данных)

Кэши уровня приложения

- Кэширование результатов ф-ий, объектов, малоизменяемых данных
 - APC/eAccelerator/xcache как кэш данных
 - высокая скорость, малый объем, локальность
 - Могут быть проблемы с блокировками и фрагментацией
 - MemCache
 - Распределенный “сетевой” кэш – единая копия данных и больше объем. Медленно
 - Диск (локальный или разделенный)
 - “Обернутый диск” – например BDB с memcache интерфейсом



НТТР и выше

- Регенерированные статические данные
 - Можно хитрить и генерить по 404 и стирать при инвалидации.
- SQUID и аналоги
 - Распределенный. TTL или инвалидация
- Кэш браузера
 - Правильно выставлять expires, etag
 - Если новой версии объекта давать новый URL, можно не беспокоиться об инвалидации



Политика кэширования

- Как избежать позавчерашних новостей?
 - TTL – объект имеет ограниченное время жизни
 - Просто, но менее эффективно чем могло бы быть
 - Инвалидация обновленных данных
 - Более сложно, но эффективно и оперативно
 - Часто используется с TTL “на всякий случай”
 - “Контроль версий объектов”
 - Знаем, что версия пользователя равна 1000 – значит, считаем недействительными все более старые зависимые объекты



Боремся с сетевыми задержками

- Получить больше данных за одно обращение
 - Не считывать таблицу строку за строкой
 - Использовать `multi_get` в memcache итд
- Делать запросы параллельно
 - Параллельная работа с внешними Web сервисами
- Избегать ненужных обращений
- “Отложенные операции”
 - Можно ли это сделать после того, как страница уже отдана?



Вот и все, или немного рекламы

- Persona Ltd
 - Мы занимаемся консалтингом в области MySQL, оптимизациями, дизайном архитектуры, обслуживанием MySQL
- Приходите к нам работать
 - Интересная работа для активных экспертов в Web технологиях и MySQL
- pz@mysqlperformanceblog.com
- <http://www.mysqlperformanceblog.com>

