УДК 004.051

ОПТИМИЗАЦИЯ ВЫСОКОНАГРУЖЕННЫХ ВЕБ-СЕРВЕРОВ

Попов Ю.В., Ушаков С. В.
Донецкий национальный технический университет
Кафедра прикладной математики и информатики
Email:justsat@gmail.com

Аннотация

Попов Ю.В., Ушаков С.В. Оптимизация высоконагруженных веб-серверов*. Рассмотрены основные способы и методы оптимизации высоконагруженных веб-серверов (в связке Apache+PHP+MySQL). Описаны самые ресурсоемкие места веб-проектов и примеры текущих решений оптимизации.*

Общая постановка проблемы

Высоконагруженные веб-сервера обслуживают от миллиона клиентов в сутки, а производительность системы не заключается лишь в её технических характеристиках. В следствии этого приходится оптимизировать программную часть системы. Цель данной оптимизации - заставить быстроработающую систему работать ещё быстрее.

Оптимизация сервера Apache

Существуют оптимизации, которые можно применить лишь при пересборке Apache, другие же можно применить без пересборки сервера. Apache - модульная программа, большая часть функций которой реализуется в модулях. При этом эти модули могут быть как вкомпилированы, так и собраны в виде DSO - динамических библиотеках. Большинство современных дистрибутивов поставляет apache с набором DSO, так что не нужные модули можно легко отключить без перекомпиляции. Рекомендуется запускать apache только с необходимыми модулями, чтобы уменьшить потребление памяти.

Очень важен выбор подходящего MPM. В apache каждый запрос обрабатывается в своем процессе или потоке. При компиляции apache позволяет выбирать один из нескольким MPM (Multi-processing module), которые отвечают за прослушивание портов, прием запросов и раздачу этих запросов дочерним процессам или потокам, в которых эти запросы будут обработаны. Выбор MPM зависит от нескольких факторов, таких как наличие поддержки потоков в ОС, количества свободной памяти, а также требований стабильности и безопасности.

Всегда следует настраивать директивы. Рассмотрим некоторые из них. За настройку DNS Lookup отвечает директива HostnameLookups, которая включает reverse DNS запросы, так что в логи будут попадать dns-хосты клиентов вместо ip-адресов. Разумеется, что это существенно замедляет обработку запроса, т.к. запрос не обрабатывается пока не будет получит ответ от DNS-сервера. Поэтому требуется чтобы эта директива всегда была выключена. В случае если директива AllowOverride не установлена в ‘None’, apache будет пытаться открыть .htaccess файлы в каждой директории которую он посещает и во всех директориях выше нее, поэтому если использование .htaccess не требуется - лучше отключить его импользование. Если для директории включена опция FollowSymLinks, сервер будет следовать по символическим ссылкам в этой директории. Если для директории включена опция SymLinksIfOwnerMatch, apache будет следовать по символическим ссылкам только если владелец файла или директории, на которую указывает эта ссылка совпадает с владельцем указанной директории. Так что при включенной опции SymLinksIfOwnerMatch apache делает больше системных запросов. Другая дорогостоящая операция это создание потока, и особенно процесса, apache создает их заранее. Директивы MaxSpareServers и

MinSpareServers устанавливают как много процессов/потоков должны ожидать в готовности принять запрос (максимум и минимум). Если значение MinSpareServers слишком маленькое и неожиданно приходит много запросов, apache вынужден будет создавать много новых процессов/потоков, что создаст дополнительную нагрузку в этой стрессовой ситуации. С другой стороны, если MaxSpareServers слишком велико, apache будет сильно нагружать систему этими процессами, даже если количество клиентов минимально. KeepAlive позволяет делать несколько запросов в одном TCP-подключении. Это особенно полезно для html-страниц с большим количеством изображений. Если KeepAlive установлен в Off, то для самой страницы и для каждого изображения будет создано отдельное подключение (которое нужно будет обработать master-процессу), что плохо и для сервера и для клиента.

Сжатие. HTTP-сжатие было определено в стандарте HTTP/1.1, и сейчас все современные клиентские программы и практически все сервера его поддерживают. Сервер может отдавать ответ в gzip или deflate, а клиентская программа незаметно для пользователя разжимает данные. Это уменьшает количество передаваемого трафика (до 75%), но конечно же повышает использование процессора. Однако если сервер посещает много клиентов с медленным подключение, сжатие может снизить нагрузку с вашего сервера из-за того что сервер сможет быстрее передать сжатый ответ и освободить ресурсы, занятые дочерним процессом. Особенно сильно этот эффект может быть заметен если у вас быстрый процессор, но мало памяти. Кеширование конфигурируется директивами модуля mod_deflate. Стоить иметь в виду, что не следует устанавливать степень сжатия gzip более 4-5 - это потребует существенно большего времени CPU, а эффект будет достаточно невелик. Разумеется не нужно пытаться сжать изображения в jpg, gif и png, музыку, видео файлы и все другие бинарные файлы, которые уже и так хорошо сжаты.

Не стоит забывать про возможность кеширование на стороне клиента. Для этого следует устанавливать Expires заголовки на статические файлы (модуль mod_expires). Если файл не изменяется, то его всегда следует попробовать закешировать на клиенте. Тогда у клиента будут быстрее загружаться страницы, а сервер освободится от лишних запросов.

Оптимизация MySQL

Оптимизация запросов важнейший компонент быстродействия сервера базы данных. Команда EXPLAIN - самая мощная команда в MySQL. EXPLAIN может в точности рассказать, что происходит, когда выполняется запрос. Эта информация позволит обнаружить медленные запросы и сократить время, затрачиваемое на обработку запроса, что впоследствии может значительно ускорить работу приложения.

MySQL индексы - объект базы данных, создаваемый с целью повышения производительности поиска данных. Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путем последовательного просмотра таблицы строка за строкой может занимать много времени. Индекс формируется из значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы и, таким образом, позволяет искать строки, удовлетворяющие критерию поиска. Ускорение работы с использованием индексов достигается в первую очередь за счёт того, что индекс имеет структуру, оптимизированную под поиск — например, сбалансированного дерева.

Кэширование запросов MySQL - может значительно ускорить производительность системы. Есть несколько способов кэшировать запросы. Начну со встроенного в MySQL механизма кэширование, который, однако не включен по умолчанию. После включения кэш работает автоматически:

- При каждом запросе типа SELECT вычисляет хэш-сумму строки запроса и ищет ее в кэше. Если находит - возвращает рузельтат из кэша, если нет - выполняет запрос, а результат заносит в кэш (если результат не больше значения query_cache_limit).

- При каждом запросе типа UPDATE, REPLACE, INSERT, DELETE, TRUNCATE или ALTER, удаляет из кэша все запросы, использующие таблицу, подвергшуюся обновлению.

Несмотря на то, что встроенный кэш запросов представляет мощный и ясный механизм, в ряде случаев он неприменим. Опишем наиболее распространенные:

Запрос использует пользовательские функции, но тем не менее является детерминистическим.

Оптимизация работы подзапросов потребовала разбить запрос на два с использованием временных таблиц.

Таблица часто обновляется, аннулируя кэш, но не обязательно делать актуальную выборку.

В таблице часто обновляются поля, не использованные в запросе.

Другой вариант это использование специальных программ для кэширования. Одна из самых распространенных - Memcached. Компьютерная программа, реализующая сервис кэширования данных в оперативной памяти на основе парадигмы хеш-таблицы. С помощью клиентской библиотеки (для C/C++, Ruby, Perl, PHP, Python, Java, CSharp/.Net и др.) позволяет кэшировать данные в оперативной памяти из множества доступных серверов. Для PHP также есть уже готовые библиотеки PECL для работы с memcached, которые дают дополнительную функциональность. Принцип работы очень прост: после установления соединения между клиентом (произвольное приложение, воспользовавшееся услугами одной из клиентских библиотек) и сервером (распределенной системой, состоящей из daemon'ов), клиенту предоставляется возможность выполнять четыре примитивных действия для организации кэширования:

- set — установить соответствие между ключом и указанным объектом;
- add — аналогично set, но только при условии, что объекта с таким ключом в кэше нет;
- replace — абсолютная противоположность add, выполняется только если такой объект в кэше есть;
- get — получить объект из кэша по указанному ключу.

В основном СУБД предоставляют встроенные средства кэширования, но на практике они умеют кэшировать только результаты запросов, что не всегда является именно тем, что необходимо веб-приложению. СУБД обычно полностью очищают кэш таблицы при каждом изменении данных, что приводит к полной его бесполезности при активном обновлении таблиц.

Современные алгоритмы заранее формируют для поиска так называемый полнотекстовый индекс — словарь, в котором перечислены все слова и указано, в каких местах они встречаются. При наличии такого индекса достаточно осуществить поиск нужных слов в нём и тогда сразу же будет получен список документов, в которых они встречаются. Sphinx - система полнотекстового поиска, отличительной особенностью является высокая скорость индексации и поиска. Сфинкс позволяет искать информацию хранящуюся в базе данных SQL или просто в файлах. Так же есть возможность индексирования и поиска данных на лету. При этом работа со Сфинксом в значительной степени ни чем не отличается от работы с сервером базы данных. Альтернотивой является MyISAM. MyISAM — одна из основных систем хранения данных в СУБД MySQL. Она основывается на коде ISAM и обладает в сравнении с ним рядом полезных дополнений. Таблицы MyISAM прекрасно подходят для использования в WWW и других средах, где преобладают запросы на чтение. Таблицы типа MyISAM показывают хорошие результаты при выборках SELECT. Во многом это связано с отсутствием поддержки транзакций и внешних ключей. Однако при модификации и добавлении записей вся таблица кратковременно блокируется, это может привести к серьёзным задержкам при большой загрузке. Таблицы MyISAM являются платформенно-независимыми. Табличные файлы можно перемещать между компьютерами разных

архитектур и разными операционными системами без всякого преобразования. Для этого MySQL хранит все числа с плавающей запятой в формате IEEE, а все целые числа — в формате с прямым порядком следования байтов. По умолчанию в каждой таблице может быть не более тридцати двух индексов, но это значение можно повысить до шестидесяти четырёх. Индексы создаются в виде двоичных деревьев. Разрешается индексировать столбцы типа BLOB и TEXT, а также столбцы, допускающие значения NULL. В таблицах MyISAM могут быть фиксированные, динамические либо сжатые записи. Выбор между фиксированным и динамическим форматом диктуется определениями столбцов.

Очередной способ улучшить быстродействие операций - это размещение таблиц в ОЗУ при помощи tmpfs. Tmpfs — временное файловое хранилище во многих Unix-like ОС. Предназначена для монтирования файловой системы, но размещается в ОЗУ вместо ПЗУ. Подобная конструкция является RAM диском, поэтому данные прийдется бекапить на жесткий диск с неким промежутком.

Шардинг - разделение данных на уровне ресурсов. Концепция шардинга заключается в логическом разделении данных по различным ресурсам исходя из требований к нагрузке. Например есть одна таблица и обращение к ней составляет 90% всех обращений к СУБД. В таком случае эту таблицу можно вынести на отдельный север. Что делать, если таблица стала настолько большой, что даже выделенный сервер под нее одну уже не спасает. Необходимо делать горизонтальный шардинг - т.е. разделение одной таблицы по разным ресурсам. Обычно, в качестве параметра шардинга выбирают ID - это позволяет делить данные по серверам равномерно и просто. Задачу определения конкретного сервера можно решать двумя путями: Хранить в одном месте хеш-таблицу с соответствиями “id=сервер”. Тогда, при определении сервера, нужно будет выбрать сервер из этой таблицы. В этом случае узкое место - это большая таблица соответсвия, которую нужно хранить в одном месте. Для таких целей очень хорошо подходят базы данных “ключ=значение”. Определять имя сервера с помощью числового преобразования. Например, можно вычислять номер сервера, как остаток от деления на определенное число. В этом случае узкое место - это проблема добавления новых серверов - придется делать перераспределение данных между новым количеством серверов. Горизонтальный шардинг имеет одно явное преимущество - он бесконечно масштабируем.

Партиционирование (partitioning) - это разбиение больших таблиц на логические части по выбранным критериям. Скорее всего есть несколько огромных таблиц. Причем чтение в большинстве случаев приходится только на самую последнюю их часть (т.е. активно читаются те данные, которые недавно появились). Партиционирование таблицы позволяет базе данных делать интеллектуальную выборку - сначала СУБД уточнит, какой партиции соответствует Ваш запрос (если это реально) и только потом сделает этот запрос, применительно к нужной партиции (или нескольким партициям). Партиционирование в MySQL - отлично реализовано на уровне СУБД (v. >= 5.1)

Репликация - это синхронное/асинхронное копирование данных с ведущих серверов на ведомые (или возможно тоже ведущие) сервера. Ведущие сервера называют мастерами (master), ведомые - слейвами (slave). Вариантов репликации бывает несколько, но для решения проблемы чтения, понадобится master-slave репликация. Это решение потребует некоторых изменений в приложении (обычно не большых), т.к. нужно будет читать данные с разных серверов БД, а не одного. Также, необходимо учесть репликационный лаг (задержка копирования данных на слейв с мастера) в работе приложения. Можно выбрать один из двух вариантов - синхронную или асинхронную репликацию. В случае первой не придется заботится о лаге, но это отразится на скорости отработки запросов на изменение/вставку данных. Репликация - это наращиваемое решение. Если одного слейва не хватает - ставится второй, третий и т.д. Принцип выбора слейвов на уровне приложения значения не имеет. Главное - это балансирование нагрузки на все слейвы. Естественно существует предельный

максимум слейв-серверов (и обычно он связан с тем, что на каком-то этапе уже мастер становится слабым звеном и начинаются проблеммы с записью) Репликация также помагает изолировать нагрузку. Если есть проблемные тяжелые запросы, которые выполняются не так часто, но несут за собой промедления работы всей СУБД, их можно вынести на слейвы.

Оптимизация PHP

Акселератор PHP — программа, ускоряющая исполнение сценариев PHP интерпретатором путём кэширования их байткода. Как выглядит обработка сценария на PHP обычным интерпретатором:

- Чтение файла
- Генерация байткода
- Выполнение кода
- Выдача результата

При этом процесс генерации байткода выполняется каждый раз и отнимает большую часть времени обработки сценария. Для обхода этого узкого места были разработаны акселераторы PHP — модули, кэширующие скомпилированный байт-код в памяти и/или на диске и в разы увеличивающие производительность PHP.

Оптимизация алгоритмов - наверное один из самых важных компонентов оптимизации. Чтобы выбрать оптимальный алгоритм, нужно знать какие алгоритмы бывают, анализировать быстродейсвтвие кода, изучать как решают ту или иную задачу другие.

Выводы

Повышение производительности сервера даже на 1% имеет смысл. Ведь если наш сервер обрабативал один миллион запросов, то после оптимизации он сможет обрабатывать на десять тысяч больше. К тому же оптимизация программной части за частую куда менее затратна, чем повышение технических характеристик серверов. Однако существует огромное количество спобов оптимизации и все они нуждаются в тестировании. Невозможно четко сказать какой метод будет самым оптимальным.

Список литературы

1. Highload Web. Электронный ресурс. Режим доступа [http://highload.com.ua]

2. Блог Highload. Электронный ресурс. Режим доступа [http://habrahabr.ru/hub/hi/posts/]

3. Techinfo - Оптимизация веб-сайта и веб-сервера. Электронный ресурс. Режим доступа [http://techinfo.net.ru/web/optimization/]