УДК 004.051

ПОВЫШЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ ВЕБ-СЕРВЕРОВ С БОЛЬШОЙ НАГРУЗКОЙ

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

Описаны методы оптимизации сервера LAMP (Linux, Apache, MySQL, PHP) с большим количеством посещений. Рассмотрены критические места сервера: какая именно часть требует большее количество процессорного времени, где есть возможность распараллелить процесс на потоки или даже на разные кластера.

Общая постановка проблемы

Нагрузка на веб-сервера обусловливается количеством посещений ресурса в сутки. Высоким принято считать обслуживание от миллиона клиентов. Логично предположить, что чем больше нагрузка тем лучше должна быть производительность системы. Однако она не заключается лишь в технических характеристиках, программная часть имеет не меньшую важность. Целью оптимизации является система работающая быстрее, даже на один процент.

Оптимизация сервера Apache

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. Существуют оптимизации, которые можно применить лишь при пересборке Apache, другие же можно применить без пересборки сервера. 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.

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.

Оптимизация MySQL

MySQL — свободная система управления базами данных. Важнейшим компонентом быстродействия является оптимизация запросов. 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. 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:

4

– 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.

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. 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.

Партиционирование (partitioning) - это разбиение таблиц на части по выбранным критериям. Чтение больших таблиц 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 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)

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

Так как существует очень большое количество способов и методов оптимизации серверов с высокой нагрузкой, то нельзя с точной уверенностью сазать какой из них оптимальнее и даст лучший результат. К тому же имеется вероятность наоборот - потерять производительность используя, к примеру, два несовместимых способа. Все они нуждаются в тестировании, однако можно сказать с точностью, что оптимизация программной части менее затратна нежели улучшение техничиских характеристик сервера.

Перечень источников

[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/]