DOI 10.12851/EESJ201512C06ART01

Anton A. Kostiukov,
Post-raduate,
Penza State University

Features of Settings Runtime for Heavy Web-based Systems

Keywords*: deploy environment, highload.*

Annotation*: In the process of deploying any highly loaded web-systems takes place of the head angle control system availability online resource (ie web server). Therefore, it is necessary to pay very close attention to setting up and testing its operation.*

Введение. Методы и стратегии настройки программного обеспечения распределённой вычислительной системы являются важным аспектом подготовки любой деятельности, связанной с выполнением задач на удалённом сервере: будь то научные исследования, эксперименты, обслуживание пользователей. На рисунке 1 приведена обобщённая схема взаимодействия в клиент-серверной архитектуре.

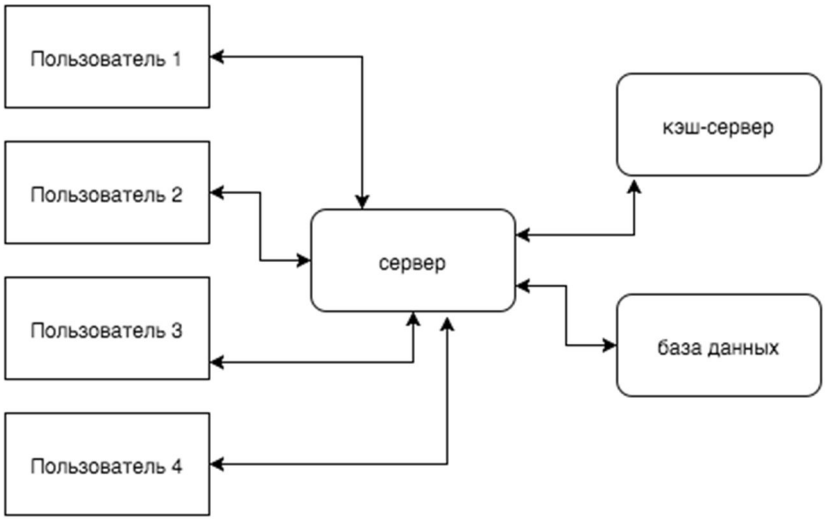

Рисунок 1 – Клиент-серверная архитектура

Обычно, для достижения наибольшего эффекта скорости работы используется стандартная связка Nginx + Apache. Nginx играет роль frontend сервера для выдачи статической информации ввиду присущей ему быстроты и лёгкости в данном аспекте. Например, необходимо обеспечить обслуживание 10000 тысяч запросов одновременно (так называемая “проблема 10к”). Ситуация осложняется тем, что у большинства клиентских машин медленные каналы связи. Если бы имело место использование сервера Apache для обслуживание данного пула клиентов, мы получили бы 1000 процессов httpd! Так как Apache имеет свойство на каждого клиента выделять отдельный поток ОС. И также на каждый из них ОС необходимо было бы выделить оперативную память, и эта память оставалась бы занятой до того момента, пока клиент не получит запрошенные данные, либо не получит сообщение-http-status. Если

применить описанную выше концепцию Nginx+Apache, то после того, как пришёл запрос от клиента, Nginx проксирует (пробрасывает) запрос до веб-сервера Apache и получает ответ намного быстрее, нежели в противоположном случае. А в случае со статическим контентом Nginx самостоятельно формирует ответ, не проксируя до Apache. В случае необходимости выполнения серверной логики (например, cgi-скрипт на языке Python или Clojure) Apache исполняет необходимый сценарий и передает результат (статический контент) в Nginx и освобождает память данного процесса для обработки других запросов. Nginx в силу своей заложенной лёгкости и быстроты отдаёт сгенерированный статический контент. При обеспечении грамотного кэширования, мы обнаруживаем титаническую экономию вычислительных ресурсов сервера. Такой сервер в первом приближении подходит для развёртки высоконагруженных приложений.

Настройка производительности веб-сервера. Для описываемой на рисунке 1 схемы, производительность сервера Apache не слишком критична, но если необходимо добиться максимальной производительности при экономии системных ресурсов – необходимо уделить пристальное внимание данному вопросу. Простейший способ увеличить производительность – это вертикальное масштабирование т.е. наращивание вычислительной мощи сервера через наращивание “железных” мощностей. Покупка нового процессора, оперативной памяти, сетевой карта. Но для начала лучше прибегнуть к оптимизации конфигурации сервера. Существует два вида конфигураций для сервера Apache: Требующие перекомпиляции сервера; Не требующие перекомпиляции сервера.

Основная часть функционала в Apache реализуется с помощью модулей, выполненных в виде интегрированных или динамических библиотек. На сегодняшний день абсолютное большинство модулей и библиотек для большинства дистрибутивов Apache работают как динамические модули. Эти модули можно легко отключить не прибегая к перекомпиляции. При уменьшении количества модулей, соответственно уменьшается объем потребляемой памяти. Поэтому если есть возможность перекомпилировать Apache, то к подбору модулей нужно подойти рационально используя apxs для Apache1 или apxs2 для Apache2. Для отключения ненужных скомпилированных модулей необходимо закомментировать лишние строки директивы LoadModule в конфигурационном файле httpd.conf. Если скомпилировать модули статически, то сервер будет потреблять намного меньше оперативной памяти, но минус состоит в том, что если нужно будет отключить один из них – сервер необходимо перекомпилировать. Для обработки каждого запроса в Apache назначается отдельный процесс или поток. Каждый процесс работает в точном соответствии с мультипроцессорной моделью (MPM). Тип MPM зависит от нескольких факторов, среди которых поддержка потоков в ОС, объем свободной памяти, требование к стабильности, требования к безопасности.

Например, если нужно обеспечить максимальную безопасность, существует мультипроцессорная модель peruser или Apache-itk. Если же нужно выжать из системы все в плане производительности, можно обратить внимание на prefork или worker. Так же нужно учесть, что для смены MPM необходимо скачать source-based дистрибутив Apache.

Директива DNS lookup (HostnameLookups). Директива HostnameLookups

активизирует обратные DNS запросы. Файл логирования при этом содержат DNS-хосты клиентов вместо их IP-адресов. Это сильно замедляет работу сервера, потому что необходимо сначала получить ответ от DNS-сервера. Необходимо выставить директиву HostnameLookups в Off. Если необходимы DNS-адреса, можно пропустить файл логов через утилиту logresolve. Кроме того необходимо следить за использованием IP-адресов для разметки директив DenyFrom и AllowFrom, а не целых доменных имён, чтобы избежать подмены хоста. Что также немаловажно в обратном случае случае сервер Apache ввиду стандартных настроек будет сначала раскрывать адрес сервера для того, чтобы узнать IP клиента и убедиться в его существовании.

Директива MaxClients. MaxClients управляет максимальным количеством параллельных запросов, которые будет поддерживать сервер. Значение MaxClients не должно быть очень маленьким, иначе некоторые запросы просто будут отклоняться. Напротив, слишком большое значение означает нехватку ресурсов и попросту “падение” сервера. Примерный расчет MaxClients:

MaxClients = доступная ОП / размер порождаемого процесса.

(Обычно на отдачу статики Apache резервирует под процесс 2-3 мегабайта. А для исполнения скриптов 16-32 мегабайта). Если сервер исчерпал количество доступных клиентов, остальные запросы попадают в ListenBacklog.

Группа директив MinSpareServers, MaxSpareServers и StartServers. Запрос на создание потока или процесса в систему это ресурсоемкая операция, поэтому Apache создаёт их про запас. Директивы MaxSpareServers и MinSpareServers устанавливают минимальное и максимальное число процессов/потоков, которые должны быть готовы принять запрос. Если значение MinSpareServers слишком мало и пришло много запросов, Apache расценит это как руководство к действию и создаст на каждый запрос отдельных поток, что очень негативно скажется на производительности сервера, так как создаст лишнюю нагрузку в пиковые моменты. Если MaxSpareServers слишком велико, Apache будет излишне нагружать систему, даже если число запросов минимально. Опытным путём нужно подобрать такие значения, чтобы Apache не создавал более 4 процессов/потоков в секунду. Если он создаст более 4, в ErrorLog будет сделана соответствующая запись. Проанализировав запись можно рассчитать, сколько минимально потоков необходимо серверу и записать эту цифру в директиву MinSpareServers.

Эффективность HTTP. Продвинутые клиенты и сервера поддерживают HTTP-сжатие. Использование сжатия позволяет понизить трафик между клиентом и сервером до 4-х раз, повышая при этом нагрузку на процессор сервера. Но, если сервер посещает много клиентов с медленными каналами, сжатие способно снизить нагрузку посредством уменьшения времени передачи сжатого ответа. При этом ресурсы, занятые дочерним процессом освобождаются быстрее, и уменьшается число одновременных запросов. Это особенно заметно в условиях ограничения памяти. При этом, значение степени сжатия gzip не рекомендуется устанавливать больше 4. Ввиду злоупотребления сжатием, усложняются вычисления для архивации, что даёт большую нагрузку на серверную систему, а выгода от сжатия такой ценой – невелика. Также, не следует сжимать файлы, формат которых уже подразумевает сжатие – это практически все мультимедийные файлы и архивы.

Кэширование на стороне клиента. Кэширование на клиентской машине –

очень важная часть работы по повышению отклика системы. Здесь помогут заголовки Expires для статических файлов. Для этого определён целый модуль Apache - mod_expires. Необходимо исследовать динамику изменения файлов на сервере, и если существуют файлы, которые никогда не обновляют свою версию, то всегда следует дать указание клиенту закешировать его. Если следовать всем инструкциям, можно сильно разгрузить сервер, что даст возможность обрабатывать больше количество запросов. Так же работа по разгрузке сервера со стороны базы данных (кэширующей базы данных), может производиться при помощи исследований основанных на алгебре кортежей (4), применяя которую можно найти лучшие пути оптимизации обхода участков базы данных. Алгебра кортежей является мощным инструментом для анализа и решения задач связанных с поиском лучших совпадений в кэше.

Сервер Nginx. Простой и лёгкий веб-сервер, специально предназначенный для обработки статических запросов. Причина его производительности в том, что процессы обработки запросов работают параллельно для всего множества соединений мультиплексируя их вызовами ОС (для различных версий UNIX – различными) Сервер имеет эффективную систему управления памятью с применением пулов. Ответы на запросы формируются в буферах оперативной памяти, поэтому достигается довольно высокая скорость обработки запросов. Также создаются цепочки буферов, которые в строгой последовательности передаются адресатам. Таким образом, при использовании в связке с Apache, Nginx настраивается на обработку статики и используется для балансировки нагрузки. Подавляющее время сервер выдаёт статический контент, функционирует на достаточно высокой скорости и с минимальными накладными расходами.

References:

1. *Ayvaliotis DA. Server Administration NGINX: Moscow, Moskniga, 2011; 300.*
2. *Analysis of the database using the algebra of tuples: VN. Belov, PP. Makarychev: Proceedings of the higher educational institutions. Volga region. Technical science, 2011, № 3; 25-36.*
3. *Arnold M, Almeida D, Miller K. Administration Apache. St. Petersburg, Peter MQM, 2013; 250.*
4. *Krishnamurthy B, Rexford G. Web-protocols. Theory and practice: Oreilly, 2010; 400.*
5. *Faith C. TCP; IP. Architecture. Protocols. Implementation. Moscow, Moskniga, 2011; 240.*
6. *Yurevich YG. WSGI, the introduction of WSGI - standard for exchanging data between the Web server (backend) and Web application (frontend). St. Petersburg, DMK, Peter, 2010; 200.*