Константин Лепихов

Возможна ли жизнь без Apache?

Популярные решения не всегда являются оптимальными. Это особенно применимо к веб-серверам. В данной статье мы постарались раскрыть слабые места Apache и попытались предложить достойную альтернативу.

Как это ни странно, но я не люблю HTTP-сервер Apache. Да, это самый распространенный на сегодняшний день HTTP-сервер (по данным Netcraft), но большей частью эта популярность объясняется не его удобством, а инерцией администраторов, которые его устанавливают, потом передают своим приемникам и т. д. (вспомним «сладкую парочку» LAMP — Linux/Apache/MySQL/PHP, которую многие начинающие системные администраторы считают панацеей от всех бед и единственно верным решением).

Чем не угодил Apache

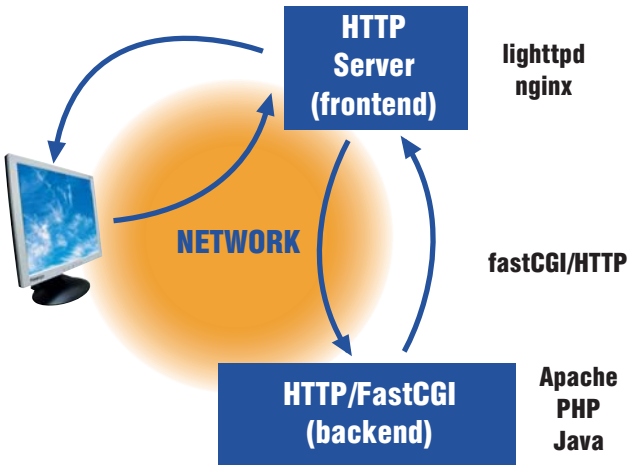
Раньше эта инерция кое-как была оправданна — в стародавние времена медленного Интернета и протокола HTTP/1.0 не было альтернатив. А сегодня, когда контент стал более динамичным, скорости перешагнули гигабитный рубеж и количество новых протоколов уже не поддается учету, позиции Apache не так безоговорочны. Рассмотрим основные причины, почему Apache нельзя назвать совершенным HTTP-сервером и почему надо серьезно подумать перед его установкой и, возможно, выбрать более гибкое и удобное решение, чем стандартный LAMP.

- Неоправданно большое потребление памяти. Согласно информации, приведенной в статье Дениса Смирнова (www.freesource.info/wiki/Anti_Apache), в Apache версии 1.3.x каждое параллельное соединение обрабатывается отдельной копией (дочерним процессом) Apache (в 2.x вместо копии используется нить исполнения, что несколько экономнее). Любая копия Apache требует для исполнения приблизительно 2 Мбайт памяти, то есть если нужно обработать 100 соединений, то для этого потребуется 200 Мбайт. Ну а если соединений несколько тысяч? Думаю, не у каждого есть в запасе несколько гигабайт свободной оперативной памяти (прибавим еще и SQL-сервер — в результате получаются совсем заоблачные цифры). А ведь несколько тысяч одновременных соединений совсем не редкость, многие форумы и популярные новостные сайты как раз имеют такую посещаемость, да еще и как среднее значение!
- Неправильная работа при большом количестве низкоскоростных подключений. Реализовано просто ужасно — на одно соединение выделяется копия объемом несколько десятков мегабайт, при этом средний размер передаваемых данных колеблется в пределах десятка килобайт. Согласитесь, память можно было бы расходовать и более экономно.
- Нерациональная работа со статическими данными. Отчасти это объясняется причинами, изложенными выше, и тем, что Apache уже староват для современного мира — например, передачу/чтение большого количества данных сегодня гораздо эффективнее делать не с помощью классических read/write, а более продвину-

тыми механизмами, например sendfile(), или использовать epoll/kqueue вместо стандартного poll().

▸ Небезопасная модель работы. Вспомним еще и такие неприятности как модульная архитектура сервера без надлежащего разделения привилегий между модулями, и ситуация окажется совсем безрадостной. Либо нам нужен суперкомпьютер и постоянный аудит всех сторонних модулей к серверу (что не всегда возможно), либо нужно выбрать что-то другое.

Кое-как все эти недостатки можно причесать и поправить. Например, для кеширования ответов (и, как следствие, экономии памяти) использовать модули поддержки так называемого режима reverse proxy (небезызвестный mod_accel Игоря Сысоева) или вообще специализированные решения — squid, oops. Но все остальные проблемы останутся с нами, особенно самая неприятная — невозможность стандартными средствами разделить в Apache понятия frontend (reverse proxy, или статический сервер) и backend (динамический сервер) из-за «неисправимых преимуществ» смешивания этих качеств в одной программе.



В отличие от Apache ядро и статический сервер у nginx и lighttpd — два разных компонента

Реальный пример из жизни

Чтобы усилить эффект внушения, приведу один пример — после замены связки Apache+mod_php на nginx+php-fcgi средний loading упал с 7 до 0,3! И это при том, что nginx не «подкручивался» по настройкам, а Apache был оптимизирован по всем пунктам — от конфигурации до флагов при сборке. Сайт, на котором производилась замена, имел постоянную посещаемость в районе нескольких тысяч разнородных запросов в час. Подозреваю, что если туда поставить squid, то нагрузку можно снизить еще больше. И при этом железо на сервере и его конфигурация далеки от идеала — это обычная машина с медленными IDE-дисками и небольшим объемом памяти, занятой другими, более «прожорливыми» программами.

Список участников

Итак, вы решились. Но что выбрать? Во-первых, это должен быть HTTP-сервер, удовлетворяющий целому ряду требований:

▸ поддержка CGI/FastCGI;

▸ поддержка современных возможностей передачи данных (как минимум sendfile/sendfile64,epoll/kqueue);

▸ малый расход памяти при большом количестве соединений;

▸ поддержка режима reverse proxy;

▸ поддержка url rewriting (с функционалом не хуже mod_rewrite);

▸ поддержка userdir/vhost (для массового хостинга);

▸ поддержка базовых возможностей HTTP-сервера: SSI, gzip-сжатие данных, SSL, TLS, авторизация пользователей;

▸ безопасность работы (поддержка chroot(), setuid/setgid).

Я расскажу о тех серверах, которые либо использовал сам, либо слышал отзывы об их работе. Конечно, этот список не велик, и многие другие интересные проекты остались «за бортом», но по крайней мере мой рассказ поможет лучше ориентироваться среди альтернативных HTTP-серверов и выбрать то, что будет оптимальным решением. Все функциональные возможности рассматриваемых серверов объединены в единую таблицу, для каждого сервера имеется краткое описание. В конце статьи на примерах показаны принципы настройки каждого из них.

Cherokee

Сайт проекта — www.0x50.org
Автор — Alvaro Lopez Ortega

Один из самых старых серверов: его разработка началась в 2001 году. Отличается гибкостью настройки и сборки (возможна даже сборка для встроенных устройств под названием u-Cherokee). Сервер имеет сильно модулированную архитектуру, где каждое расширение (handler) выполняет определенные функции (передача файлов, поддержка CGI/FastCGI, построение списка директорий и т. д.). Каждый модуль изолирован от «ядра» сервера, при этом есть возможность «сбросить» права при запуске или изолировать исполнение в отдельном каталоге с помощью вызова chroot(). К сожалению, работа сервера не отличается стабильностью — у меня он довольно часто аварийно завершал работу даже при включении стандартных расширений и возможностей, например поддержки epoll. Не совсем корректно реализована поддержка сжатия данных с помощью gzip — например, даже не проверяется, сможет ли клиент их обработать. Присутствуют ошибки в реализации SSL/TLS, причем довольно грубые. Но в целом, если вам нужен быстрый и нетребовательный к ресурсам HTTP-сервер для небольшого сайта с PHP/CGI/FastCGI, или если вы заинтересованы в разработке встроенного HTTP-сервера или клиент-серверных приложений на базе библиотек Cherokee, то этот проект может вас заинтересовать. Готовые сборки Cherokee есть в Linux-дистрибутивах Debian, Gentoo и в портах xBSD-систем — NetBSD, OpenBSD, FreeBSD.

lighttpd

Сайт проекта — www.lighttpd.org
Автор — Jan Kneschke

Самый продвинутый из «легкого» семейства. Обладает всеми возможностями «взрослого» HTTP-сервера, включая поддержку массового хостинга, FastCGI, SSL/TLS и средства управления трафиком. Очень прост в конфигурировании, при этом не в ущерб функциональности. Для lighttpd в Сети существует очень много разнообразных HOWTO по настройке

и положительных отзывов об использовании, его собственная документация отличается подробным описанием всех опций и параметров. Поддерживается режим работы в изолированном каталоге с помощью вызова chroot().

nginx

Сайт проекта — www.sysoev.ru/nginx
Автор — Игорь Сысоев

Сайт дополнительной документации и полезные статьи по использованию этого сервера — www.nginx.info.

Очень динамично развивающийся проект. Хотя номер версии еще далек от заветного 1.x, сервер вполне функционален и стабилен в работе. По словам автора, различные версии nginx работают на многих загруженных сайтах и нескольких серверах Rambler. Как и lighttpd, он обладает всеми «взрослыми» возможностями, включая такие полезные как встроенный geo-модуль, позволяющий в зависимости от IP-клиента задавать различные действия, и модуль интеллектуального сжатия данных с помощью алгоритмов gzip/deflate. Сервер поддерживает все новшества ядер Linux/FreeBSD, такие как sendfile/sendfile64/kqueue/sendfilev, epoll для Linux, /dev/poll для Solaris и accept-фильтры для FreeBSD. По заверениям разработчика, на 10 000 неактивных keep-alive-соединений расходуется всего 2,5 Мбайт памяти! Если вы захотите собрать nginx самостоятельно, то будете приятно удивлены интересной системой сборки сервера — она отличается от стандартных autoconf/automake, но позволяет более гибко регулировать все шаги процесса, включая даже разновидности компилятора (кроме обычного gcc поддерживается icc, ccc, acc и bcc). Еще одна интересная и полезная особенность nginx — это способ управления. В отличие от стандартных SIGTERM/SIGHUP для перезапуска и обновления конфигурации nginx можно управлять аж шестью сигналами, при этом дочерние процессы поддерживают дополнительно еще три сигнала. Все это позволяет выполнять плавную перезагрузку сервера в случае изменения конфигурации или завершения работы процессов. Таким образом, даже в случае большого количества открытых соединений клиенты даже не заметят «подмены» в работе сервера. К сожалению, пока на сайте проекта очень мало документации, так как Игорь больше занят разработкой, а не составлением описания работы, но этот недостаток легко восполняется русскоязычным списком рассылки, где можно задать вопрос разработчику и другим пользователям nginx. И, наверное, самый хороший источник информации по дополнительным параметрам работы — это непосредственно исходный код nginx: да-да, его можно просто читать и получать удовольствие от этого.

Приступаем к конфигурации

Напоследок я решил продемонстрировать, как можно настроить сайт на базе вышеперечисленных серверов. В качестве контента возьмем популярную программу imp (www.horde.org/imp), которая вполне годится на роль feature-rich-приложения. Она представляет собой основанную на протоколе IMAP почтовую систему. Написан IMP на PHP. В простейшем виде, наша конфигурация должна выглядеть так:

Оригинальный httpd.conf

```
#
# File: horde.conf
#
# This is the Horde Apache configuration file; it is included from the
# Apache httpd.conf file.
#
Alias /horde /var/www/html/addon-modules/horde/
Alias /horde/ /var/www/html/addon-modules/horde/

<Directory /var/www/html/addon-modules/horde>
  Options Indexes FollowSymLinks
  AllowOverride None
  order allow,deny
  allow from 127.0.0.1
</Directory>
<Directory "/var/www/html/addon-modules/horde/config">
  order deny,allow
  deny from all
</Directory>
<Directory "/var/www/html/addon-modules/horde/lib">
  order deny,allow
  deny from all
</Directory>
<Directory "/var/www/html/addon-modules/horde/locale">
  order deny,allow
  deny from all
</Directory>
<Directory "/var/www/html/addon-modules/horde/po">
  order deny,allow
  deny from all
</Directory>
<Directory "/var/www/html/addon-modules/horde/scripts">
  order deny,allow
  deny from all
</Directory>
<Directory "/var/www/html/addon-modules/horde/templates">
  order deny,allow
  deny from all
</Directory>
#
# File: imp.conf
#
# This is the IMP Apache configuration file; it is included from the
# Apache httpd.conf file.
#
<Directory "/var/www/html/addon-modules/horde/imp/config">
  order deny,allow
  deny from all
</Directory>
<Directory "/var/www/html/addon-modules/horde/imp/lib">
  order deny,allow
  deny from all
</Directory>
<Directory "/var/www/html/addon-modules/horde/imp/locale">
```

```
    order deny,allow
    deny from all
  </Directory>
  <Directory "/var/www/html/addon-modules/horde/imp/po">
    order deny,allow
    deny from all
  </Directory>
  <Directory "/var/www/html/addon-modules/horde/imp/scripts">
    order deny,allow
    deny from all
  </Directory>
  <Directory "/var/www/html/addon-modules/horde/imp/tem-
plates">
    order deny,allow
    deny from all
  </Directory>
```

Cherokee

К сожалению, Cherokee не поддерживает FastCGI, и пришлось использовать обычный CGI-модуль для PHP. Ниже приведен пример конфигурационного файла cherokee.conf:

```
  DocumentRoot /var/www/html//var/www/html
  Directory /horde {
        Handler common
              DocumentRoot
  /var/www/html//var/www/html/addon-modules/horde
              Allow from 127.0.0.1
  }
  # Cherokee не поддерживает параметр deny from all, поэтому
  ничего про это не пишем
  # Rest of the virtual host configuration..
  # Add your parameters here..
  #
  Extension php, php4 {
        Handler phpcgi {
              Interpreter /usr/bin/php-cgi
        }
  }
```

Как видите, конфигурационный файл совсем небольшой, но все вроде работает.

lighttpd

Пример конфигурационного файла lighttpd.conf

```
  # сначала подключаем нужные модули:
  # access (для управления доступом)
  # compress (для gzip-сжатия трафика)
```

Название сервера	Cherokee	lighttpd	nginx
Версия	0.4.25	01.03.14	0.1.41
Лицензия	GPL	BSD	BSD
Поддерживаемые ОС	Windows Linux OpenBSD FreeBSD NetBSD Sun Solaris	Linux *BSD[1] SGI IRIX Windows (Cygwin)	Linux *BSD[1] Tru64 UNIX HP/UX Windows
Поддержка sendfile()	•	•	•
Поддержка sendfile64	•	•	•
Поддержка sendfilev()	•	•	•
Поддержка epoll()	•	•	•
Поддержка kqueue()	–	•	•
Поддержка /dev/poll	–		•
Поддержка accept_filter	–	•	•
Поддержка TCP_DEFER_ACCEPT	•	–	•
Поддержка режима reverse proxy	–	•	•
Поддержка CGI	•	•	•
Поддержка FastCGI	–	•	•[2]
Поддержка chroot() и сброса привилегий	•	•	–
Поддержка распределения нагрузки (load balancing)	–	•	•
Поддержка сжатия gzip/deflate	•	•	•[3]
Поддержка vhost	•	•	•
Поддержка userdir	•	•	–
Поддержка SSL/TLS	GNUTLS/OpenSSL	OpenSSL	OpenSSL
Поддержка PCRE-синтаксиса	собственная реализация	через библиотеку pcre	через библиотеку pcre
Поддержка URL rewriting	•	•	•
Поддержка HTTP-авторизации	•	•	•
Ограничение скорости отдачи ответов	–	•	•
Стандартные возможности HTTP-сервера (SSI, пользовательские коды ошибок и т. д.)	частично (SSI не поддерживается)	•	•

[1] Подразумевается все семейство ОС BSD, включая Sun Solaris.
[2] nginx не умеет самостоятельно управлять FastCGI-сервером.
[3] nginx наиболее правильно поддерживает этот режим, так как учитывает параметры запроса клиента.

```
# fastcgi (для поддержки FastCGI версии PHP)
# alias (для создания alias)
server.modules = (
                    "mod_access",
                    "mod_compress",
                    "mod_fastcgi",
                    "mod_alias",
                    "mod_accesslog")
# задаем DocRoot
server.document-root       = "/var/www/html"
# подгрузка fastcgi
fastcgi.server= ( ".php" =>
              "localhost" =>
              (
              "socket" => "/var/spool/lighttpd/php-fastcgi.socket",
               "bin-path" => "/usr/bin/php-cgi"
               )
             )
          )
# задаем alias
alias.url = ( "/horde/" => "/var/www/html/addon-
modules/horde/" )
# задаем ограничения на доступ
$HTTP["url"] =~ "^/(config|lib|locales|templates|po|scripts)/" {
        url.access-deny = ( "" )
        server.dir-listing = "disable"
}
$HTTP["host"] !~ "127.0.0.1"
        url.access-deny = ( "" )
}
```

Как видно, получилось даже лучше, чем с Apache.

nginx

Пример конфигурационного файла nginx.conf:

```
# передаем все запросы на php backend
# к сожалению, nginx не умеет самостоятельно запускать
fcgi-сервер, поэтому
# запустим его через утилиту spawn-fcgi из lighttpd.
location ~* ^.+\.php$ {
      fastcgi_pass   <fcgi host>:<fcgi port>;
      fastcgi_index  index.php;
      fastcgi_param  SCRIPT_FILENAME
/var/www/html/forum$fastcgi_script_name;
      fastcgi_param  QUERY_STRING     $query_string;
      fastcgi_param  REQUEST_METHOD   $request_method;
      fastcgi_param  CONTENT_TYPE     $content_type;
      fastcgi_param  CONTENT_LENGTH   $content_length;
      # занимает ресурсы, без нужды не использовать
      fastcgi_param  REMOTE_ADDR      $remote_addr;
      fastcgi_param  SERVER_PORT      $server_port;
      fastcgi_param  REDIRECT_STATUS  200;
      # специально для php-cgi без этого переменная
PHP_SELF
      # будет пустая
      fastcgi_param  SCRIPT_NAME      $fastcgi_script_name;
}
# теперь можно заходить по адресу http:/..../imp
server {
         listen <ip>:80;

         sendfile        on;
         tcp_nodelay     on;

         gzip on;
         root /var/www/html/addon-modules/horde;
      index index.php;
      location = / {
         root /var/www/html/addon-modules/horde/imp;
         index index.php;
            allow from 127.0.0.1;
      }
      location / {
            allow from 127.0.0.1;
         rewrite ^/horde(.*)$ $1 break;
         rewrite ^/favicon.ico$ /graphics/favicon.ico break;
      }
      location ~* \.php {
         rewrite ^/horde(.*) $1 break;
          }
# настроим доступ
         location ~= ^/(config|lib|locales|templates|po|scripts)/ {
            deny all;
         }
}
```

Дополнительные источники информации

Полезные ссылки

Денис Смирнов, «В этом мире есть не только Apache»:
www.freesource.info/wiki/Anti_Apache

Константин Лепихов, «PHP FastCGI»:
www.freesource.info/wiki/Stat'i/PhpFastCGI?v=179h

Ralf S Engelschall, «Practical approaches for distributing HTTP traffic»:
www.engelschall.com/pw/wt/loadbalance/article.html

Dan Kegel, «The C10K problem»:
www.kegel.com/c10k.html

Так есть ли жизнь без Apache?

Из приведенных примеров видно, что конфигурационные файлы альтернативных HTTP-серверов проще, чем у Apache. И это без потери их функциональности. Так стоит ли идти на поводу у привычки и закостенелых взглядов и использовать привычный многим пользователям сервер Apache? Попробуйте одну из рассмотренных нами легковесных программ, и, возможно, сервер вашего популярного веб-ресурса еще долго не придется подвергать апгрейду. |