Что нового в nginx?

Максим Дунин

Профессиональная конференция разработчиков высоконагруженных систем

Про ложь, наглую ложь и статистику

W3Techs:

```
0.x.x -  2.5%
1.x.x - 97.5%
```

Жизнь наладилась!

https://w3techs.com/technologies/details/ws-nginx/all/all
“Nginx is used by 31.4% of all the websites whose web server we know.”

HL 2016 HighLoad++

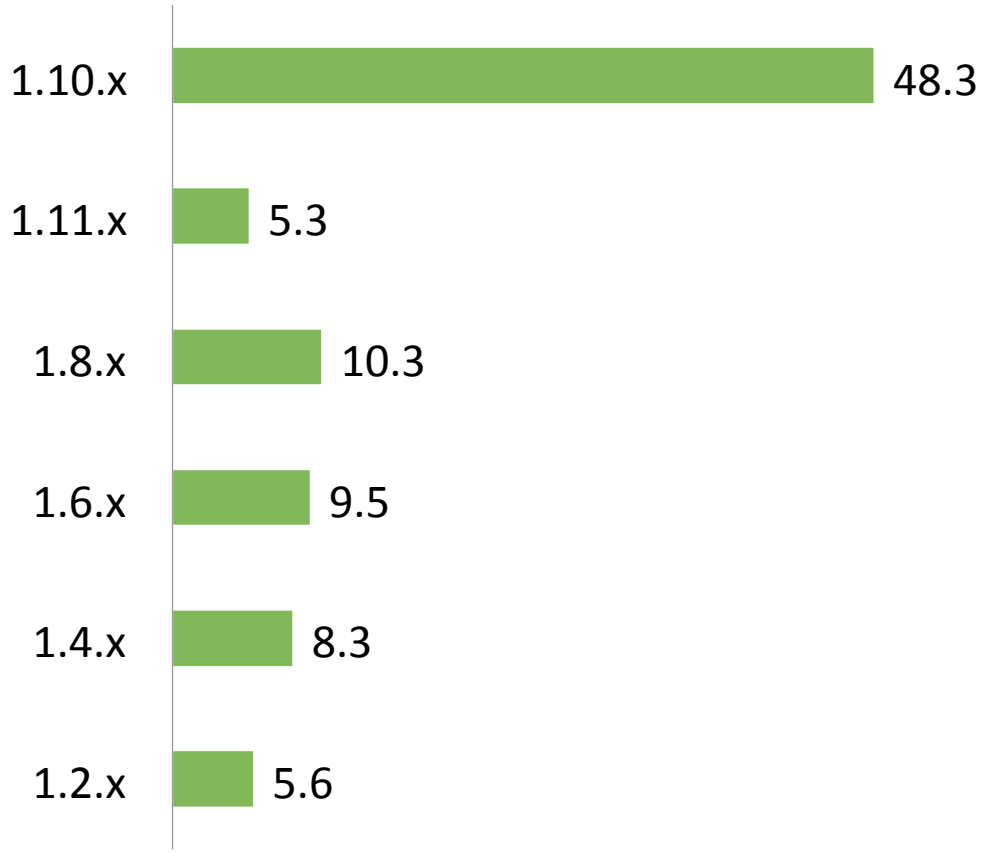
Подробнее про 1.x.x

Новое:

```
1.10.x  - 48.3%
1.11.x  -  5.3%
```

Старое:

```
1.8.x   - 10.3%
1.6.x   -  9.5%
1.4.x   -  8.3%
1.2.x   -  5.6%
```



HL 2016 HighLoad++

1.9.x

Научили nginx показывать порт клиента в различных местах, добавили знание о неидемпотентных запросах в proxy_next_upstream, в кеш теперь можно писать асинхронно, а cache manager умеет следить за собой. Появились динамические модули, worker_cpu_affinity обзавёлся ручкой auto для ленивых, модуль slice позволяет загружать и кешировать большие файлы по частям, resolver научился делать запросы по TCP, HTTP/2 пришёл на смену SPDY, sub_filter теперь умеет два раза, nginx -T показывает конфиг, а nginx -V – версию OpenSSL, “listen ... reuseport” для быстрой обработки новых соединений, и модуль stream для тех, кому и целого HTTP мало.

HL 2016 HighLoad++

Порт клиента

- `set_real_ip_from` теперь ставит не только адрес, но и порт

```
proxy_pass http://backend;
proxy_set_header X-Real-IP
                 $remote_addr:$remote_port;
```

- переменные `$proxy_protocol_port`, `$realip_remote_port`

HL 2016 HighLoad++

Идемпотентность

Идемпоте́нтность — свойство объекта или операции при повторном применении операции к объекту давать тот же результат, что и при одинарном.

GET, HEAD, PUT, DELETE, … – да
POST, LOCK, PATCH – нет

- Не повторяем неидемпотентные запросы
- Вернуть старое поведение:

```
proxy_next_upstream error timeout
                    non_idempotent;
```

HL 2016 HighLoad++

Запись в кеш

Запись – это быстро
Но может и заблокироваться

Решение:

```
aio threads;
aio_write on;
```

HL 2016 HighLoad++

Cache: боремся за память

Ошибки “could not allocate node”?

- Cache умеет удалять старые записи
- Но другой процесс может успеть раньше

Теперь cache manager следит за количеством записей.

HL 2016 HighLoad++

Динамические модули

- Упростить сборку пакетов
  - внешние зависимости у модулей!
- Упростить отладку
  - 3rd party модули должно быть просто выключить

HL 2016 HighLoad++

Динамические модули

Собрать:

```
./configure --with-http_image_filter_module=dynamic
./configure --add-dynamic-module=/path/to/module
```

nginx.conf:

```
load_module modules/ngx_http_image_filter_module.so;
```

HL 2016 HighLoad++

Динамическая загрузка своего модуля

- Меняем `config`-скрипт на `auto/module`
- Для сложных модулей:
  - `ngx_count_modules()`
  - `ngx_modules` меняем на `cycle->modules`

HL 2016 HighLoad++

Динамическая загрузка своего модуля

Было:

```
ngx_addon_name="ngx_http_delay_module"
HTTP_MODULES="$HTTP_MODULES ngx_http_delay_module"
NGX_ADDON_SRCS="$NGX_ADDON_SRCS $ngx_addon_dir/ngx_http_delay_module.c"
```

Стало:

```
ngx_addon_name="ngx_http_delay_module"

ngx_module_type=HTTP
ngx_module_name=ngx_http_delay_module
ngx_module_incs=
ngx_module_deps=
ngx_module_srcs=$ngx_addon_dir/ngx_http_delay_module.c
ngx_module_libs=

. auto/module
```

HL 2016 HighLoad++

CPU affinity

Пример из документации:

```
worker_processes    4;
worker_cpu_affinity 0001 0010 0100 1000;
```

Что делать, если у нас много процессоров?

```
worker_cpu_affinity auto;
worker_cpu_affinity auto 0101010101010101;
```

HL 2016 HighLoad++

Кешируем большие файлы

Проблемы:

- С бекенда скачивается файл целиком
- Range-запросы ждут начала файла
- Часто нужно только начало

Решение:

- Качать и кешировать кусками

HL 2016 HighLoad++

Модуль slice

```
location / {
    slice              1m;
    proxy_cache        cache;
    proxy_cache_key    $uri$is_args$args$slice_range;
    proxy_set_header   Range $slice_range;
    proxy_cache_valid  200 206 1h;
    proxy_pass         http://upstream;
}
```

- Подзапросы на отдельные части
- Кешируем отдельные части

HL 2016 HighLoad++

SPDY умер, HTTP/2 за него

```
server {
    listen 443 ssl http2;
    ...
}
```

- Мультиплексирование запросов в одном соединении
- Теперь работает `proxy_request_buffering`
- Недоделанность реализаций всё там же, осторожнее

HL 2016 HighLoad++

Несколько sub_filter за раз

Пример:

```
sub_filter http://one.example.com https://one.example.com;
sub_filter http://two.example.com https://two.example.com;
sub_filter_once off;
```

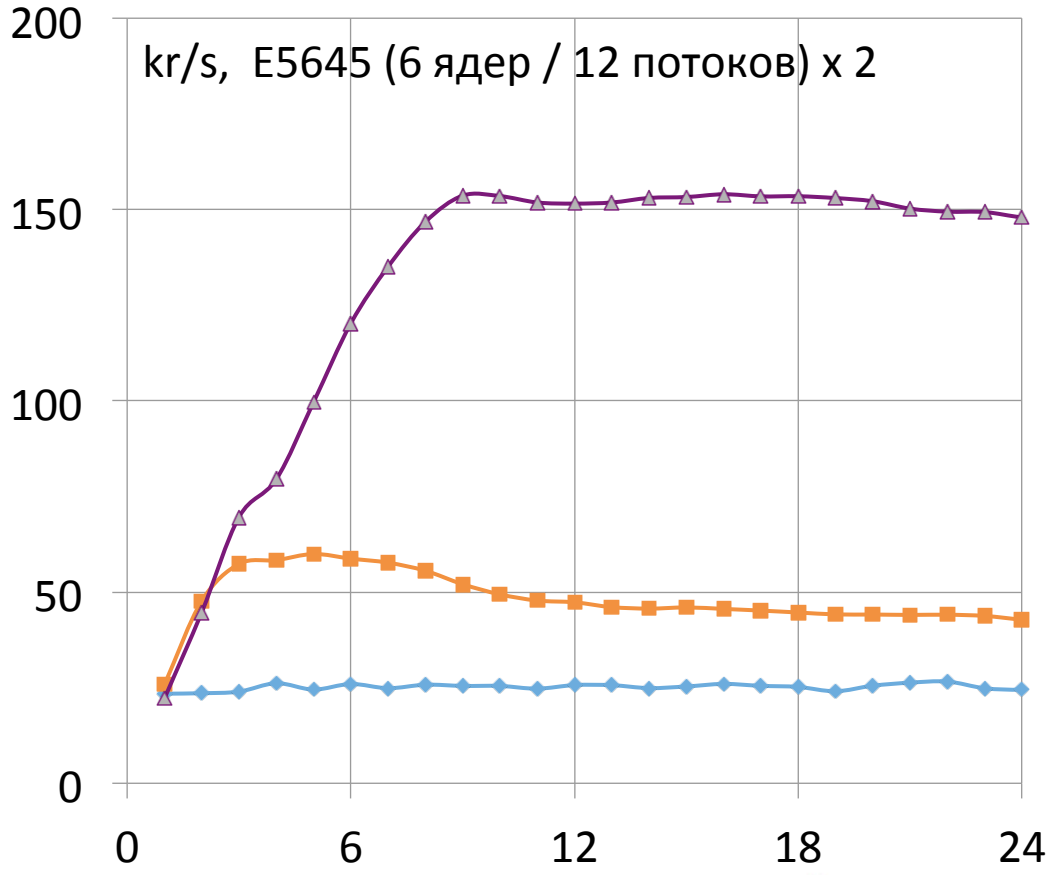
Заодно он научился делать это быстрее.

HL 2016 HighLoad++

Быстрая обработка новых соединений

- Тестируем
  - accept_mutex!
- Один listen-сокет – contention в ядре

```
listen 80 reuseport;
```



HL 2016 HighLoad++

Stream: для тех, кому HTTP мало

```
stream {
    upstream backend {
        server 10.0.0.2:12345;
        server 10.0.0.3:12345;
    }
    server {
        listen 12345;
        proxy_pass backend;
    }
}
```

- Балансировка произвольных соединений
- SSL от клиентов, SSL к бекендам
- Ограничения количества соединений и скорости
- PROXY protocol к бекендам
- … и даже UDP

HL 2016 HighLoad++

Разное

- Resolver научился TCP
- Блоки upstream{} в разделяемой памяти
- Разделяемая память на Windows с ASLR (Vista и новее)
- SSLv3 по умолчанию запрещён
- $upstream_connect_time
- Полный конфиг по “nginx -T”
- Версия OpenSSL в “nginx -V”

```
nginx version: nginx/1.11.2
built by gcc 4.9.2 (Debian 4.9.2-10)
built with OpenSSL 1.0.1k 8 Jan 2015 (running with OpenSSL 1.0.1t 3 May 2016)
...
```

HL 2016 HighLoad++

Всё это доступно в nginx 1.10.2

HL 2016 HighLoad++

1.11.x

Переменные в модуле stream, возможность "подсматривать" в SSL handshake, и множество всего ещё. Больше нет quick recovery, но есть max_conns, а cache manager научился быть незаметным, proxy_bind теперь умеет transparent, и умеет использовать IP_BIND_ADDRESS_NO_PORT. Мы выключили accept_mutex, научились использовать EPOLLEXCLUSIVE вместо него, и начали полагаться на EPOLLRDHUP, map научился сложным значениям, переменная $request_id для злопамятных. Теперь можно несколько SSL-сертификатов (наконец-то RSA и ECDSA одновременно!), умеем списки кривых для ECDH, выключили DH, упростили сборку динамических модулей.

HL 2016 HighLoad++

Stream: переменные

А также map, return, geo, geoip, split_clients, real ip, access log…

Теперь можно:

```
map $remote_addr $limit {
    127.0.0.1     "";
    default       $binary_remote_addr;
}

limit_conn_zone $limit zone=addr:10m;
limit_conn addr 1;
```

HL 2016 HighLoad++

Stream: $ssl_preread_server_name

Если нужно посмотреть в SSL-соединение,
но нельзя снимать SSL:

```
map $ssl_preread_server_name $backend {
    backend.example.com      192.0.2.1:12345;
    default                  192.0.2.2:12345;
}

server {
    listen       12345;
    proxy_pass   $backend;
    ssl_preread on;
}
```

HL 2016 HighLoad++

Количество соединений к бекендам

Пример:

```
upstream u {
    server 192.0.2.1:8080 max_conns=10;
    server 192.0.2.2:8080 max_conns=10;
    server 192.0.2.3:8080 max_conns=10;
}
```

Исходно для NGINX Plus.

HL 2016 HighLoad++

Transparent proxy

Передать адрес клиента, традиционные решения:

- X-Forwarded-For / X-Real-IP
- PROXY protocol

Если очень надо:

```
proxy_bind $remote_addr transparent;
```

SO_BINDANY (NetBSD), IP_TRANSPARENT (Linux), IP_BINDANY (FreeBSD), нужен root

HL 2016 HighLoad++

Linux и IP_BIND_ADDRESS_NO_PORT

Сколько соединений можно установить к одному бекенду?
К двум?

А теперь пишем в конфиг:

```
proxy_bind 192.0.2.1;
```

и всё плохо. Почему?

- Bind() выбирает локальный порт
- IP_BIND_ADDRESS_NO_PORT (Linux 4.2, glibc 2.22)

HL 2016 HighLoad++

Accept mutex, Linux и EPOLLEXCLUSIVE

Thundering herd problem.

Традиционный ответ – accept mutex.

Но:

- вообще-то у нас процессов мало
- часто мешает в тестах
- нельзя использовать на Windows
- не нужен с “listen … reuseport”
- на Linux появился EPOLLEXCLUSIVE (Linux 4.5, glibc 2.24)

HL 2016 HighLoad++

Accept mutex, Linux и EPOLLEXCLUSIVE

Новый default:

```
accept_mutex off;
```

Ну и EPOLLEXCLUSIVE поддерживаем.

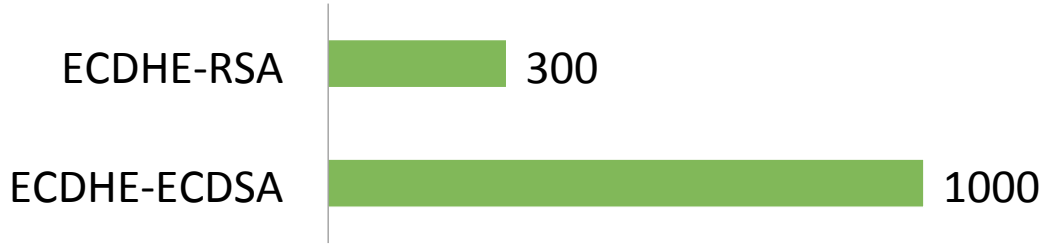
HL 2016 HighLoad++

SSL и сертификаты

RSA – совместимо, но дорого

ECDSA – быстро, но не будет работать в XP (Android до 4.0, OpenSSL до 1.0.0)

```
rsa2048:      400 signs/s
ecdsap256:   5400 signs/s
```



Теперь можно так:

```
ssl_certificate     example.com.rsa.crt;
ssl_certificate_key example.com.rsa.key;
ssl_certificate     example.com.ecdsa.crt;
ssl_certificate_key example.com.ecdsa.key;
```

HL 2016 HighLoad++

SSL и другое

- `ssl_ecdh_curve` умеет список кривых (OpenSSL 1.0.2+)
  - “auto” по умолчанию
  - OpenSSL 1.1.0: X25519:prime256v1:secp521r1:secp384r1
- `ssl_dhparam` по умолчанию отсутствует
  - DHE-шифры выключены
  - Медленно
  - Атаки на DHE, weakdh.org

HL 2016 HighLoad++

Динамические модули: совместимость

Проблема:

- Структуры зависят от опций сборки
- Собирать модули надо с теми же опциями

Теперь проще:

```
./configure --with-compat ...
./configure --with-compat --add-dynamic-module=/path/to/module
```

Бонус: совместимость с NGINX Plus.

HL 2016 HighLoad++

Разное

- На EPOLLRDHUP мы теперь полагаемся, если ядро новое
- HTTP/2 – исправления и улучшения
- Map научился сложным комбинациям переменных и строк

```
map $host $log {
    ~foo      /var/log/$host.log;
    default   /var/log/default.log;
}
```

- $request_id

Всё это доступно в nginx 1.11.5

Still counting

HL 2016 HighLoad++

Где взять дене^Wсвежий nginx?

- Исходные коды, как обычно:
  http://nginx.org/ru/download.html

- Для FreeBSD:
  используйте порты nginx и nginx-devel, там всё новое
  Спасибо Сергею Осокину!

- Для Linux разных версий:
  системные пакеты – обычно старые, свежие пакеты доступны на
  http://nginx.org/ru/linux_packages.html

  (отдельные пакеты для модулей geoip, image filter, njs, perl, xslt)

HL 2016 HighLoad++

Спасибо!

Вопросы?

Максим Дунин

mdounin@mdounin.ru