nginx 優化突破十萬並行連線數

資料來源: http://www.yanghengfei.com/archives/326/

nginx 的安裝與基本配置文檔網上已經有很多了，但具體講優化的文章還比較少，偶爾發現有這麼一篇《nginx 優化突破十萬並發》仔細拜讀後，轉至此做以收藏，感謝原作的辛苦編寫。

一般來說 nginx 配置文件中對優化比較有作用的為以下幾項：

```
worker_processes 8;
```

nginx 進程數，建議按照 cpu 數目來指定，一般為它的倍數。

```
worker_cpu_affinity 00000001 00000010 00000100 00001000 00010000 00100000 01000000 10000000;
```

為每個進程分配 cpu，上例中將 8 個進程分配到 8 個 cpu，當然可以寫多個，或者將一個進程分配到多個 cpu。

```
worker_rlimit_nofile 102400;
```

這個指令是指當一個 nginx 進程打開的最多文件描述符數目，理論值應該是最多打開文件數（ulimit -n）與 nginx 進程數相除，但是 nginx 分配請求並不是那麼均勻，所以最好與 ulimit -n 的值保持一致。

```
use epoll;
```

使用 epoll 的 I/O 模型，這個不用說了吧。

```
worker_connections 102400;
```

每個進程允許的最多連接數，理論上每台 nginx 服務器的最大連接數為 worker_processes*worker_connections。

```
keepalive_timeout 60;
```

keepalive 超時時間。

```
client_header_buffer_size 4k;
```

客戶端請求頭部的緩衝區大小，這個可以根據你的系統分頁大小來設置，一般一個請求的頭部大小不會超過 1k，不過由於一般系統分頁都要大於 1k，所以這裡設置為分頁大小。分頁大小可以用命令 getconf PAGESIZE 取得。

```
open_file_cache max=102400 inactive=20s;
```

這個將為打開文件指定緩存，默認是沒有啟用的，max 指定緩存數量，建議和打開文件數一致，inactive 是指經過多長時間文件沒被請求後刪除緩存。

```
open_file_cache_valid 30s;
```

這個是指多長時間檢查一次緩存的有效信息。

```
open_file_cache_min_uses 1;
```

open_file_cache 指令中的 inactive 參數時間內文件的最少使用次數，如果超過這個數字，文件描述符一直是在緩存中打開的，如上例，如果有一個文件在 inactive 時間內一次沒被使用，它將被移除。

關於內核參數的優化：

```
net.ipv4.tcp_max_tw_buckets = 6000
```

timewait 的數量，默認是 180000。

```
net.ipv4.ip_local_port_range = 1024 65000
```

允許系統打開的端口範圍。

```
net.ipv4.tcp_tw_recycle = 1
```

啟用 timewait 快速回收。

```
net.ipv4.tcp_tw_reuse = 1
```

開啟重用。允許將 TIME-WAIT sockets 重新用於新的 TCP 連接。

```
net.ipv4.tcp_syncookies = 1
```

開啟 SYN Cookies，當出現 SYN 等待隊列溢出時，啟用 cookies 來處理。

```
net.core.somaxconn = 262144
```

web 應用中 listen 函數的 backlog 默認會給我們內核參數的 net.core.somaxconn 限製到 128，而 nginx 定義的 NGX_LISTEN_BACKLOG 默認為 511，所以有必要調整這個值。

```
net.core.netdev_max_backlog = 262144
```

每個網絡接口接收數據包的速率比內核處理這些包的速率快時，允許送到隊列的數據包的最大數目。

```
net.ipv4.tcp_max_orphans = 262144
```

系統中最多有多少個 TCP 套接字不被關聯到任何一個用戶文件句柄上。如果超過這個數字，孤兒連接將即刻被復位並打印出警告信息。這個限制僅僅是為了防止簡單的 DoS 攻擊，不能過分依靠它或者人為地減小這個值，更應該增加這個值(如果增加了內存之後)。

```
net.ipv4.tcp_max_syn_backlog = 262144
```

記錄的那些尚未收到客戶端確認信息的連接請求的最大值。對於有 128M 內存的系統而言，缺省值是 1024，小內存的系統則是 128。

```
net.ipv4.tcp_timestamps = 0
```

時間戳可以避免序列號的捲繞。一個 1Gbps 的鏈路肯定會遇到以前用過的序列號。時間戳能夠讓內核接受這種“異常”的數據包。這裡需要將其關掉。

```
net.ipv4.tcp_synack_retries = 1
```

為了打開對端的連接，內核需要發送一個 SYN 並附帶一個回應前面一個 SYN 的 ACK。也就是所謂三次握手中的第二次握手。這個設置決定了內核放棄連接之前發送 SYN+ACK 包的數量。

```
net.ipv4.tcp_syn_retries = 1
```

在內核放棄建立連接之前發送 SYN 包的數量。

```
net.ipv4.tcp_fin_timeout = 1
```

如果套接字由本端要求關閉，這個參數決定了它保持在 FIN-WAIT-2 狀態的時間。對端可以出錯並永遠不關閉連接，甚至意外當機。缺省值是 60 秒。2.2 內核的通常值是 180 秒，你可以按這個設置，但要記住的是，即使你的機器是一個輕載的 WEB 服務器，也有因為大量的死套接字而內存溢出的風險，FIN- WAIT-2 的危險性比 FIN-WAIT-1 要小，因為它最多只能吃掉 1.5K 內存，但是它們的生存期長些。

```
net.ipv4.tcp_keepalive_time = 30
```

當 keepalive 起用的時候，TCP 發送 keepalive 消息的頻度。缺省是 2 小時。

下面貼一個完整的內核優化設置：

```
net.ipv4.ip_forward = 0
net.ipv4.conf.default.rp_filter = 1
net.ipv4.conf.default.accept_source_route = 0
kernel.sysrq = 0
kernel.core_uses_pid = 1
net.ipv4.tcp_syncookies = 1
kernel.msgmnb = 65536
kernel.msgmax = 65536
kernel.shmmax = 68719476736
kernel.shmall = 4294967296
net.ipv4.tcp_max_tw_buckets = 6000
net.ipv4.tcp_sack = 1
net.ipv4.tcp_window_scaling = 1
net.ipv4.tcp_rmem = 4096 87380 4194304
```

```
net.ipv4.tcp_wmem = 4096 16384 4194304
net.core.wmem_default = 8388608
net.core.rmem_default = 8388608
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
net.core.netdev_max_backlog = 262144
net.core.somaxconn = 262144
net.ipv4.tcp_max_orphans = 3276800
net.ipv4.tcp_max_syn_backlog = 262144
net.ipv4.tcp_timestamps = 0
net.ipv4.tcp_synack_retries = 1
net.ipv4.tcp_syn_retries = 1
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_mem = 94500000 915000000 927000000
net.ipv4.tcp_fin_timeout = 1
net.ipv4.tcp_keepalive_time = 30
net.ipv4.ip_local_port_range = 1024 65000
```

下面是一個簡單的 nginx 配置文件：

```
user www www;
worker_processes 8;
worker_cpu_affinity 00000001 00000010 00000100 00001000 00010000 00100000 01000000;
error_log /www/log/nginx_error.log crit;
pid /usr/local/nginx/nginx.pid;
worker_rlimit_nofile 204800;

events
{
  use epoll;
  worker_connections 204800;
}
http
{
  include mime.types;
  default_type application/octet-stream;
  charset utf-8;
  server_names_hash_bucket_size 128;
  client_header_buffer_size 2k;
  large_client_header_buffers 4 4k;
  client_max_body_size 8m;
  sendfile on;
  tcp_nopush on;
```

```
keepalive_timeout 60;
fastcgi_cache_path /usr/local/nginx/fastcgi_cache levels=1:2
            keys_zone=TEST:10m
            inactive=5m;
fastcgi_connect_timeout 300;
fastcgi_send_timeout 300;
fastcgi_read_timeout 300;
fastcgi_buffer_size 16k;
fastcgi_buffers 16 16k;
fastcgi_busy_buffers_size 16k;
fastcgi_temp_file_write_size 16k;
fastcgi_cache TEST;
fastcgi_cache_valid 200 302 1h;
fastcgi_cache_valid 301 1d;
fastcgi_cache_valid any 1m;
fastcgi_cache_min_uses 1;
fastcgi_cache_use_stale error timeout invalid_header http_500;
open_file_cache max=204800 inactive=20s;
open_file_cache_min_uses 1;
open_file_cache_valid 30s;
tcp_nodelay on;
gzip on;
gzip_min_length 1k;
gzip_buffers 4 16k;
gzip_http_version 1.0;
gzip_comp_level 2;
gzip_types text/plain application/x-javascript text/css application/xml;
gzip_vary on;

server
{
  listen 8080;
  server_name backup.aiju.com;
  index index.php index.htm;
  root /www/html/;
  location /status
  {
     stub_status on;
  }
  location ~ .*\.(php|php5)?$
  {
     fastcgi_pass 127.0.0.1:9000;
```

```
        fastcgi_index index.php;
        include fcgi.conf;
    }
    location ~ .*\.(gif|jpg|jpeg|png|bmp|swf|js|css)$
    {
      expires 30d;
    }
    log_format access '$remote_addr - $remote_user [$time_local] "$request" '
             '$status $body_bytes_sent "$http_referer" '
             '"$http_user_agent" $http_x_forwarded_for';
    access_log /www/log/access.log access;
      }
}
```

關於 FastCGI 的幾個指令：

```
fastcgi_cache_path /usr/local/nginx/fastcgi_cache levels=1:2 keys_zone=TEST:10m
inactive=5m;
```

這個指令為 FastCGI 緩存指定一個路徑，目錄結構等級，關鍵字區域存儲時間和非活動刪除時間。

```
fastcgi_connect_timeout 300;
```

指定連接到後端 FastCGI 的超時時間。

```
fastcgi_send_timeout 300;
```

向 FastCGI 傳送請求的超時時間，這個值是指已經完成兩次握手後向 FastCGI 傳送請求的超時時間。

```
fastcgi_read_timeout 300;
```

接收 FastCGI 應答的超時時間，這個值是指已經完成兩次握手後接收 FastCGI 應答的超時時間。

```
fastcgi_buffer_size 16k;
```

指定讀取 FastCGI 應答第一部分需要用多大的緩衝區，這裡可以設置為 fastcgi_buffers 指令指定的緩衝區大小，上面的指令指定它將使用 1 個 16k 的緩衝區去讀取應答的第一部分，即應答頭，其實這個應答頭一般情況下都很小（不會超過 1k），但是你如果在 fastcgi_buffers 指令中指定了緩衝區的大小，那麼它也會分配一個 fastcgi_buffers 指定的緩衝區大小去緩存。

```
fastcgi_buffers 16 16k;
```

指定本地需要用多少和多大的緩衝區來緩衝 FastCGI 的應答，如上所示，如果一個 php 腳本所產生的頁面大小為 256k，則會為其分配 16 個 16k 的緩衝區來緩存，如果大於 256k ，增大於 256k 的部分會緩存到 fastcgi_temp 指定的路徑中，當然這對服務器負載來說是不明智的方案，因為內存中處理數據速度要快於硬盤，通常這個值的設置應該選擇一個你的站點中的 php 腳本所產生的頁面大小的中間值，比如你的站點大部分腳本所產生的頁面大小為 256k 就可以把這個值設置為 16 16k，或者 4 64k 或者 64 4k，但很顯然，後兩種並不是好的設置方法，因為如果產生的頁面只有 32k，如果用 4 64k 它會分配 1 個 64k 的緩衝區去緩存，而如果使用 64 4k 它會分配 8 個 4k 的緩衝區去緩存，而如果使用 16 16k 則它會分配 2 個 16k 去緩存頁面，這樣看起來似乎更加合理。

```
fastcgi_busy_buffers_size 32k;
```

這個指令我也不知道是做什麼用，只知道默認值是 fastcgi_buffers 的兩倍。

```
fastcgi_temp_file_write_size 32k;
```

在寫入 fastcgi_temp_path 時將用多大的數據塊，默認值是 fastcgi_buffers 的兩倍。

```
fastcgi_cache TEST
```

開啟 FastCGI 緩存並且為其製定一個名稱。個人感覺開啟緩存非常有用，可以有效降低 CPU 負載，並且防止 502 錯誤。但是這個緩存會引起很多問題，因為它緩存的是動態頁面。具體使用還需根據自己的需求。

```
fastcgi_cache_valid 200 302 1h;
fastcgi_cache_valid 301 1d;
fastcgi_cache_valid any 1m;
```

為指定的應答代碼指定緩存時間，如上例中將 200，302 應答緩存一小時，301 應答緩存 1 天，其他為 1 分鐘。

```
fastcgi_cache_min_uses 1;
```

緩存在 fastcgi_cache_path 指令 inactive 參數值時間內的最少使用次數，如上例，如果在 5 分鐘內某文件 1 次也沒有被使用，那麼這個文件將被移除。

```
fastcgi_cache_use_stale error timeout invalid_header http_500;
```

不知道這個參數的作用，猜想應該是讓 nginx 知道哪些類型的緩存是沒用的。

以上為 nginx 中 FastCGI 相關參數，另外，FastCGI 自身也有一些配置需要進行優化，如果你使用 php-fpm 來管理 FastCGI，可以修改配置文件中的以下值：

```
<value name="max_children">60</value>
```

同時處理的並發請求數，即它將開啟最多 60 個子線程來處理並發連接。

```
<value name="rlimit_files">102400</value>
```

最多打開文件數。

```
<value name="max_requests">204800</value>
```

每個進程在重置之前能夠執行的最多請求數。

下面貼幾張測試結果圖。

靜態頁面為我在 squid 配置 4W 並發那篇文章中提到的測試文件，下圖為同時在 6 台機器運行 webbench -c 30000 -t 600 http://www.yanghengfei.com/ 命令後的測試結果：

```
Active connections: 153037
server accepts handled requests
 9228400 9228400 9079725
Reading: 10240 Writing: 142797 Waiting: 0
```

使用 netstat 過濾後的連接數：

```
[root@backup ~]# netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'
TIME_WAIT 1051
FIN_WAIT1 16374
FIN_WAIT2 6
ESTABLISHED 112506
SYN_RECV 744
```

php 頁面在 status 中的結果（php 頁面為調用 phpinfo）：

```
Active connections: 80236
server accepts handled requests
 1497492 1497492 1465770
Reading: 18048 Writing: 62187 Waiting: 1
```

php 頁面在 netstat 過濾後的連接數：

```
[root@backup www]# netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'
TIME_WAIT 4692
FIN_WAIT1 45703
FIN_WAIT2 22
ESTABLISHED 65030
SYN_RECV 207
```

未使用 FastCGI 緩存之前的服務器負載：

```
top - 16:36:03 up  2:13,  1 user,  load average: 4.55, 2.08, 1.29
Tasks: 180 total,   9 running, 171 sleeping,   0 stopped,   0 zombie
Cpu0  :  9.3%us,  9.0%sy,  0.0%ni, 15.9%id,  0.0%wa,  0.0%hi, 65.8%si,  0.0%st
Cpu1  : 27.3%us,  9.3%sy,  0.0%ni, 59.7%id,  1.0%wa,  0.0%hi,  2.7%si,  0.0%st
Cpu2  : 27.8%us, 10.0%sy,  0.0%ni, 58.5%id,  0.0%wa,  0.0%hi,  3.7%si,  0.0%st
Cpu3  : 24.6%us, 10.3%sy,  0.0%ni, 61.5%id,  0.0%wa,  0.0%hi,  3.7%si,  0.0%st
Cpu4  : 29.3%us, 16.3%sy,  0.0%ni, 49.3%id,  0.0%wa,  0.0%hi,  5.0%si,  0.0%st
Cpu5  : 26.2%us,  9.6%sy,  0.0%ni, 60.5%id,  0.0%wa,  0.0%hi,  3.7%si,  0.0%st
Cpu6  : 26.6%us, 10.0%sy,  0.0%ni, 59.8%id,  0.0%wa,  0.0%hi,  3.7%si,  0.0%st
Cpu7  : 27.0%us,  9.0%sy,  0.0%ni, 61.0%id,  0.0%wa,  0.0%hi,  3.0%si,  0.0%st
Mem:   8174164k total,  6230168k used,  1943996k free,    35468k buffers
Swap:  4192956k total,        0k used,  4192956k free,  1980408k cached
```

此時打開 php 頁面已經有些困難，需要進行多次刷新才能打開。上圖中 cpu0 負載偏低是因為測試時將網卡中斷請求全部分配到 cpu0 上，並且在 nginx 中開啟 7 個進程分別制定到 cpu1-7。

使用 FastCGI 緩存之後：

```
top - 20:26:47 up  6:04,  1 user,  load average: 0.62, 0.16, 0.05
Tasks: 180 total,   5 running, 175 sleeping,   0 stopped,   0 zombie
Cpu0  :  1.7%us,  2.4%sy,  0.0%ni, 15.5%id,  0.0%wa,  0.0%hi, 80.5%si,  0.0%st
Cpu1  :  0.0%us,  0.3%sy,  0.0%ni, 98.3%id,  1.4%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2  :  0.3%us,  0.3%sy,  0.0%ni, 99.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  :  0.0%us,  0.3%sy,  0.0%ni, 99.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu4  :  2.4%us,  4.4%sy,  0.0%ni, 92.5%id,  0.0%wa,  0.0%hi,  0.7%si,  0.0%st
Cpu5  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu6  :  0.0%us,  0.3%sy,  0.0%ni, 99.3%id,  0.0%wa,  0.0%hi,  0.3%si,  0.0%st
Cpu7  :  0.0%us,  0.3%sy,  0.0%ni, 99.3%id,  0.0%wa,  0.0%hi,  0.3%si,  0.0%st
Mem:   8174164k total,  5590068k used,  2584096k free,    43756k buffers
Swap:  4192956k total,        0k used,  4192956k free,  2034756k cached
```

此時可以很輕鬆的打開 php 頁面。

這個測試並沒有連接到任何數據庫，所以並沒有什麼參考價值，不過不知道上述測試是否已經到達極限，根據內存和 cpu 的使用情況來看似乎沒有，但是已經沒有多餘的機子來讓我運行 webbench 了。囧，但至少可以說明加載 FastCGI 緩存之後，還有是些效果的。以圖為證嘛。

參考資料：

http://blog.chinaunix.net/u3/105004/showart_2087155.html

http://nginx.179401.cn/

http://blog.s135.com/nginx_php_v5/