

关于 nginx 的一些优化(突破十万并发)

一般来说 **nginx** 配置文件中对优化比较有作用的为以下几项:

```
worker_processes 8;
```

nginx 进程数, 建议按照 **cpu** 数目来指定, 一般为它的倍数。

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

为每个进程分配 **cpu**, 上例中将 8 个进程分配到 8 个 **cpu**, 当然可以写多个, 或者将一个进程分配到多个 **cpu**。

```
worker_rlimit_nofile 102400;
```

这个指令是指当一个 **nginx** 进程打开的最多文件描述符数目, 理论值应该是最多打开文件数(**ulimit -n**)与 **nginx** 进程数相除, 但是 **nginx** 分配请求并不是那么均匀, 所以最好与 **ulimit -n** 的值保持一致。

```
use epoll;
```

使用 **epoll** 的 **I/O** 模型, 这个不用说了吧。

```
worker_connections 102400;
```

每个进程允许的最多连接数, 理论上每台 **nginx** 服务器的最大连接数为 **worker_processes*worker_connections**。

```
keepalive_timeout 60;
```

keepalive 超时时间。

```
client_header_buffer_size 4k;
```

客户端请求头部的缓冲区大小, 这个可以根据你的系统分页大小来设置, 一般一个请求头的大小不会超过 **1k**, 不过由于一般系统分页都要大于 **1k**, 所以这里设置为分页大小。分页大小可以用命令 **getconf PAGESIZE** 取得。

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

这个将为打开文件指定缓存, 默认是没有启用的, **max** 指定缓存数量, 建议和打开文件数一致, **inactive** 是指经过多长时间文件没被请求后删除缓存。

```
open_file_cache_valid 30s;
```

这个是指多长时间检查一次缓存的有效信息。

```
open_file_cache_min_uses 1;
```

open_file_cache 指令中的 **inactive** 参数时间内文件的最少使用次数, 如果超过这个数字, 文件描述符一直是在缓存中打开的, 如上例, 如果有一个文件在 **inactive** 时间内一次没被使用, 它将被移除。

关于内核参数的优化：

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

timewait 的数量，默认是 180000。

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

允许系统打开的端口范围。

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

启用 timewait 快速回收。

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

开启重用。允许将 TIME-WAIT sockets 重新用于新的 TCP 连接。

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

开启 SYN Cookies，当出现 SYN 等待队列溢出时，启用 cookies 来处理。

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

web 应用中 listen 函数的 backlog 默认会给我们内核参数的 net.core.somaxconn 限制到 128，而 nginx 定义的 NGX_LISTEN_BACKLOG 默认为 511，所以有必要调整这个值。

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

每个网络接口接收数据包的速率比内核处理这些包的速率快时，允许送到队列的数据包的最大数目。

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

系统中最多有多少个 TCP 套接字不被关联到任何一个用户文件句柄上。如果超过这个数字，孤儿连接将即刻被复位并打印出警告信息。这个限制仅仅是为了防止简单的 DoS 攻击，不能过分依靠它或者人为地减小这个值，更应该增加这个值(如果增加了内存之后)。

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

记录的那些尚未收到客户端确认信息的连接请求的最大值。对于有 128M 内存的系统而言，缺省值是 1024，小内存的系统则是 128。

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

时间戳可以避免序列号的卷绕。一个 1Gbps 的链路肯定会遇到以前用过的序列号。时间戳能够让内核接受这种“异常”的数据包。这里需要将其关掉。

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

为了打开对端的连接，内核需要发送一个 SYN 并附带一个回应前面一个 SYN 的 ACK。也就是所谓三次握手中的第二次握手。这个设置决定了内核放弃连接之前发送 SYN+ACK 包的数量。

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

在内核放弃建立连接之前发送 SYN 包的数量。

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

如果套接字由本端要求关闭，这个参数决定了它保持在 FIN-WAIT-2 状态的时间。对端可以出错并永远不关闭连接，甚至意外当机。缺省值是 60 秒。2.2 内核的通常值是 180 秒，

你可以按这个设置，但要记住的是，即使你的机器是一个轻载的 WEB 服务器，也有因为大量的死套接字而内存溢出的风险，FIN-WAIT-2 的危险性比 FIN-WAIT-1 要小，因为它最多只能吃掉 1.5K 内存，但是它们的生存期长些。

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

当 keepalive 起用的时候，TCP 发送 keepalive 消息的频度。缺省是 2 小时。

下面贴一个完整的内核优化设置：

```
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
```

下面是一个简单的 **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 4k;
    fastcgi_buffers 8 4k;
    fastcgi_busy_buffers_size 8k;
    fastcgi_temp_file_write_size 8k;
```

```

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;
}
}

```

关于 FastCGI 的几个指令：

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

这个指令为 FastCGI 缓存指定一个路径，目录结构等级，关键字区域存储时间和非活动删除时间。

```
fastcgi_connect_timeout 300;
```

指定连接到后端 FastCGI 的超时时间。

```
fastcgi_send_timeout 300;
```

向 FastCGI 传送请求的超时时间，这个值是指已经完成两次握手后向 FastCGI 传送请求的超时时间。

```
fastcgi_read_timeout 300;
```

接收 FastCGI 应答的超时时间，这个值是指已经完成两次握手后接收 FastCGI 应答的超时时间。

```
fastcgi_buffer_size 4k;
```

指定读取 FastCGI 应答第一部分需要用多大的缓冲区，一般第一部分应答不会超过 1k，由于页面大小为 4k，所以这里设置为 4k。

```
fastcgi_buffers 8 4k;
```

指定本地需要用多少和多大的缓冲区来缓冲 FastCGI 的应答。

```
fastcgi_busy_buffers_size 8k;
```

这个指令我也不知道是做什么用，只知道默认值是 fastcgi_buffers 的两倍。

```
fastcgi_temp_file_write_size 8k;
```

在写入 fastcgi_temp_path 时将用多大的数据块，默认值是 fastcgi_buffers 的两倍。

```
fastcgi_cache TEST
```

开启 FastCGI 缓存并且为其制定一个名称。个人感觉开启缓存非常有用，可以有效降低 CPU 负载，并且防止 502 错误。

```
fastcgi_cache_valid 200 302 1h;
```

```
fastcgi_cache_valid 301 1d;  
fastcgi_cache_valid any 1m;
```

为指定的应答代码指定缓存时间，如上例中将 200, 302 应答缓存一小时，301 应答缓存 1 天，其他为 1 分钟。

```
fastcgi_cache_min_uses 1;
```

缓存在 fastcgi_cache_path 指令 inactive 参数值时间内的最少使用次数，如上例，如果在 5 分钟内某文件 1 次也没有被使用，那么这个文件将被移除。

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

不知道这个参数的作用，猜想应该是让 nginx 知道哪些类型的缓存是没用的。

以上为 nginx 中 FastCGI 相关参数，另外，FastCGI 自身也有一些配置需要进行优化，如果你使用 php-fpm 来管理 FastCGI，可以修改配置文件中的以下值：

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

同时处理的并发请求数，即它将开启最多 60 个子线程来处理并发连接。

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

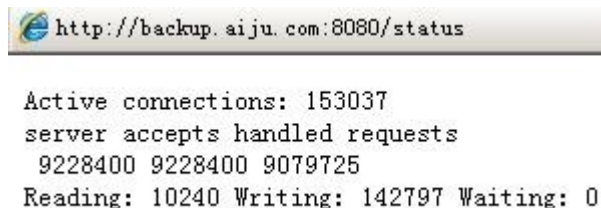
最多打开文件数。

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

每个进程在重置之前能够执行的最多请求数。

下面贴几张测试结果图。

静态页面为我在 squid 配置 4W 并发那篇文章中提到的测试文件，下图为同时在 6 台机器运行 webbench -c 30000 -t 600 http://backup.aiju.com:8080/index.html 命令后的测试结果：



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

使用 netstat 过滤后的连接数：

```
[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 页面在 status 中的结果（php 页面为调用 phpinfo）：

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

php 页面在 netstat 过滤后的连接数:

```
[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 缓存之前的服务器负载:

```
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
```

此时打开 php 页面已经有些困难, 需要进行多次刷新才能打开。上图中 cpu0 负载偏低是因为测试时将网卡中断请求全部分配到 cpu0 上, 并且在 nginx 中开启 7 个进程分别制定到 cpu1-7。

使用 FastCGI 缓存之后:

```
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
```


此时可以很轻松的打开 **php** 页面。

这个测试并没有连接到任何数据库，所以并没有什么参考价值，不过不知道上述测试是否已经到达极限，根据内存和 **cpu** 的使用情况来看似乎没有，但是已经没有多余的机器来让我运行 **webbench** 了。囧

参考资料:

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

<http://nginx.179401.cn/>

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