目录

[一、 操作系统的限制 2](#_Toc393421805)

[二、 nginx自身的限制 2](#_Toc976899626)

[1) 编译优化 2](#_Toc851384455)

[2) Worker Processes 2](#_Toc1250503795)

[3) Worker Connections 3](#_Toc562899680)

[4) CPU Affinity 3](#_Toc2098042387)

[5) Keep Alive 3](#_Toc870240169)

[三、 硬件的限制 4](#_Toc1039074445)

[1) Access log 4](#_Toc845625171)

[2) Error log 4](#_Toc765305208)

[3) Open File Cache 4](#_Toc68347754)

[4) Buffers 4](#_Toc1025931854)

[5) 彻底避免磁盘IO 5](#_Toc972514339)

[6) 网络IO 5](#_Toc1462640418)

[四、 课堂优化内容 5](#_Toc1538186783)

[1) 如何自定义返回给客户端的404错误页面 5](#_Toc405003995)

[2) 如何查看服务器状态信息 6](#_Toc245644452)

[3) 优化并发量(如果客户端访问服务器提示“Too many open files”如何解决) 6](#_Toc677031966)

[a. 优化前使用ab高并发测试 6](#_Toc310305191)

[b. 修改Nginx配置文件，增加并发量 6](#_Toc1962446087)

[c. 优化Linux内核参数（最大文件数量） 6](#_Toc415392630)

[d. 优化后测试服务器并发量（因为客户端没调内核参数，所以在proxy测试） 7](#_Toc740611002)

[4) 优化Nginx数据包头部缓存(如何解决客户端访问头部信息过长) 7](#_Toc1589840589)

[5) 如何让客户端浏览器缓存数据(本地缓存静态数据) 7](#_Toc2146189718)

[6) 日志切割 8](#_Toc14023153)

[a. 把旧的日志文件重命名 8](#_Toc1974534985)

[b. Kill -USR1 PID(nginx的进程PID号) 8](#_Toc1195553064)

[7) 对页面进行压缩处理 8](#_Toc1417084408)

[8) 开启文件缓存功能 8](#_Toc1991020835)

[附:常见错误 8](#_Toc1968884750)

Nginx调优

引用http://blog.sina.com.cn/polygun2000

对于Nginx调优,事实上没有很好的办法很大程度的优化Nginx,因为Nginx本身已经优化的足够好,相比Apache,最大的优化在make install时已经产生.

那为什么要学习Nginx调优呢,一方面时因为nginx有很多的配置项,这些选项会影响Nginx的行为.但是,这些选项的默认设置并不是完全针对高负载的状态,所以需要针对性调整.另一方面,Nginx运行的OS的设置也会影响Nginx的运行,可以根据需求进行调整.

1. 操作系统的限制

Nginx是跨平台的,各个平台都有各自的高性能event polling(时间轮询)方法,所以选择自己熟悉的OS是最大的优化方案.在生产环境中,不要吧Nginx运行在Windows上,Nginx的作者明确说明不支持该方法.

在nginx中，可以在配置文件中指定所使用的event polling模型，指令如下:

use epoll;  Linux系统

`ulimit -a`命令所列出来的这些参数

这些参数是nginx无法超越的。例如，在许多系统中，默认允许打开的最大文件数为1024，如果nginx运行在这个环境下，那么当流量大时就会出现(24: Too many open files) 错误。而nginx的处理能力远不止1024，因此这个参数必须调整。

1. nginx自身的限制
2. 编译优化

    默认的nginx编译选项是用debug模式(-g)的（debug模式会插入很多跟踪和ASSERT之类），编译以后一个nginx可执行文件有好几MB。去掉nginx的debug模式编译，编译以后只有几百KB。

    在源码auto/cc/gcc中，在最后几行中找到如下内容:

        # debug

        CFLAGS=”$CFLAGS -g”

    把CFLAGS这一行删除或者注释掉，然后再编译即可。

    另外，如果只是把nginx作为web server，那么可以禁用一些用不到的modules，可以减少内存footprint(足迹)，提高服务器性能。

    ./configure

--prefix=/webserver/nginx

--without-mail\_pop3\_module

--without-mail\_imap\_module

--without-mail\_smtp\_module

--with-http\_ssl\_module

--with-http\_stub\_status\_module

--with-http\_gzip\_static\_module

1. Worker Processes

    worker process是整个nginx的关键，一旦主进程绑定到了指定的IP/Port上以后，主进程会使用指定的用户派生出worker process，这些worker process会处理所有的用户请求。worker process不是多线程的，不需要将每个connection在不同CPU内核间切换，因此运行多个worker process是必然的，通常每个CPU core(核)对应一个worker process。通常，如果超过4个worker process，那么CPU肯定不会成为瓶颈了，因为在CPU成为瓶颈之前，nginx的其他部分肯定已经不行了。

cat /proc/cpuinfo |grep 'core id'

1. Worker Connections

    worker connctions是一个有点"怪异"的概念，我不确定这个指令的确切目的。但是可以确定的是这个指令可以有效的限制每个worker process在同一时间可以维护多少个连接。如果非要我猜的话，我认为这个指令是一个保险机制，为了防止错误配置的keep-alive耗尽可用端口。

    在默认配制文件中，worker connections的值是1024。通常一个浏览器会针对一个站点打开2个连接，那么最大可以并发服务的用户数量为512个。这个数字看起来不小了，但是考虑到keep-alive默认的timeout值是65，那么意味着其实我们每秒只能处理8个connection。显然这个数字超过了大多数人的需求，尤其是当我们使用2-4个worker process的时候。但是对于大流量的站点，并且打开了keep-alvie，那么应该时刻考虑到这一点。

    当考虑worker connections的值的时候，其实很简单，流量增加就增加这个值。2048对于绝大多数人应该够用了，不过如果你的站点真的增长这么快的话，那么最大该设置多大，只有你自己尝试了。

1. CPU Affinity

    设置CPU affinity的意思是告诉nginx，每个worker process应该使用哪一个CPU core，指定以后该worker process只会使用你指定的那个CPU core。请小心的做这个配置，因为OS的CPU调度器在处理负载均衡方面远比人类做的更好。如果你确实认为你需要在CPU调度器层面做优化的话，你可以选择不同的CPU调度器，不过你需要清楚的知道你自己在干什么，如果不知道，不要碰这个配置。

1. Keep Alive

    keep alive是一个HTTP的特性，其允许用户agent(浏览器)和服务器之间保持连接，以便其他请求共用或者直到指定的timeout时间到达。这个特性事实上并不会改善我们的nginx服务器性能，因为nginx自己可以非常好的处理空闲连接。nginx作者指出，处理10000个空闲连接只耗费2.5MB的内存。

    在这里提到keep alive的目的很简单。keep alive对于终端用户感觉到的等待时间影响是巨大的。如果你的站点看起来载入很快，那么用户会很开心。Amazon做过调查，用户感觉的等待时间和最终业务成交量之间有直接的关系。

    keep alive可以避免在创建HTTP连接过程中许多无用操作，因此它的作用才如此明显。一般我们不需要65这么大的timeout值，但推荐你将这个值设置在10-20之间。

    keepalive\_timeout 15

1. tcp\_nodelay and tcp\_nopush

    这两个参数的用途很难理解，因为它们在很底层的网络部分影响nginx。一个简单肤浅的解释是这两个指令控制OS如何处理网络缓冲，何时将他们刷新给最终用户。因为这两个参数不会明显提高或改变任何事情，如果你不明白这两个参数的含义，那我建议你不用碰它，保持默认值就好了。

1. 硬件的限制

    上边已经讨论了OS，nginx本身的一些限制，现在让我们看看怎么把服务器的性能压榨到底。

    服务器上能成为瓶颈的就CPU,内存和IO。对于nginx来说，CPU和内存都不会是瓶颈。因此瓶颈只会产生在IO部分。硬盘相对于CPU和内存来说，是非常非常慢的设备。硬盘读取和写入是非常耗时的操作，因此我们需要尽量减少nginx对硬盘的读写操作。

1. Access log

    nginx默认会将每个请求写入到日志文件中，日志可以用来审计和统计。但是记录日志的操作会带来IO开销。如果你不需要记录日志，直接关闭这个选项就好。如果需要记录日志，最好将日志记录到内存中，然后定期将日志转储到磁盘上。这样可以避免频繁磁盘IO操作，极大提高性能。

    access\_log off;

1. Error log

    对于error log，其实不应该关闭的。但是为了降低磁盘IO操作，可以调整error log的级别，将这个参数设置成"warn"级别应该足够了，并且也不会产生很大的IO。

1. Open File Cache

    从文件系统读取数据包含文件打开和关闭操作，这部分也是磁盘块操作。为了减少这部分操作，可以缓存打开的文件描述符。这个操作使用[open file cache](http://wiki.nginx.org/HttpCoreModule" \l "open_file_cache) 实现，具体可以参考链接中的wiki。

1. Buffers

    调整buffer的大小对于nginx很重要，如果buffer设置得太小，nginx将不得不把upstream服务器返回来的响应存储在临时文件中。这会导致同时增加磁盘读写操作，流量越大影响越明显。

[client\_body\_buffer\_size](http://wiki.nginx.org/HttpCoreModule" \l "client_body_buffer_size) 指令指定了处理客户端请求body部分的buffer的大小。这个一般用于处理POST数据，表单提交，文件上传等操作。如果你要处理很多大的POST提交，那么这个值要设置足够大。

    client\_header\_buffer\_size 指令指定了处理客户端请求header部分的buffer的大小。设置成1K能满足绝大部分需求。

    client\_max\_body\_size 指令指定了可以接受的最大的用户请求body大小。通过HTTP头中的Content-Length确认，如果大小超过这个值，则客户端会得到“Request Entity Too Large” (413)错误。

    large\_client\_header\_buffers 指令分配用于处理从用户那里来的大文件头请求的buffer的最大数量和buffer大小。请求的header不能比其中的一个buffer大，否则nginx会返回“Request URI too large” (414)错误。最长的header行也必须小于一个buffer的大小，否则客户端会得到“Bad request” (400)错误。

[fastcgi\_buffers](http://wiki.nginx.org/HttpFcgiModule" \l "fastcgi_buffers)和[proxy\_buffers](http://wiki.nginx.org/HttpProxyModule" \l "proxy_buffers) 指令指定了处理upstream回应的buffer大小，也就是PHP,Apache或其他。上边也说到了，如果这个buffer太小，在响应用户之前，nginx将不得不把upstream服务器返回来的响应存储在临时文件中。注意，nginx的buffer是有上限的，这个上限由fastcgi\_max\_temp\_file\_size和proxy\_max\_temp\_file\_size控制。当然也可以通过将proxy\_buffering设置成off来关闭proxy connections的buffer(通常不是个好主意！)。

    示例如下:

    client\_body\_buffer\_size 8K;  
    client\_header\_buffer\_size 1k;  
    client\_max\_body\_size 2m;  
    large\_client\_header\_buffers 2 1k;

1. 彻底避免磁盘IO

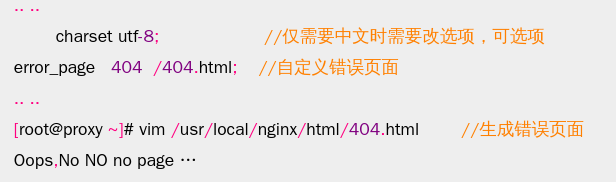
    最彻底的避免磁盘IO的方法是不使用磁盘，如果费用不是问题且数据量不大，那么可以使用ramdisk，将所有数据放到内存中。

1. 网络IO

    为了降低网络IO，我们使用gzip module来压缩传输的数据量。设置gzip\_comp\_level的值为4-5应该比较合适，如果设置更大没什么用处，白白浪费CPU。

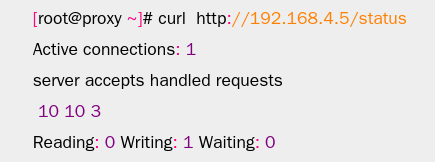
    gzip             on;  
    gzip\_comp\_level  5;  
    gzip\_min\_length  1000;  
    gzip\_proxied     expired no-cache no-store private auth;  
    gzip\_types       text/plain application/xml;  
    gzip\_disable     "MSIE [1-6]\.";

1. 课堂优化内容
2. 如何自定义返回给客户端的404错误页面



1. 如何查看服务器状态信息

编译安装时使用--with-http\_stub\_status\_module开启状态页面模块



Active connections：当前活动的连接数量。

Accepts：已经接受客户端的连接总数量。

Handled：已经处理客户端的连接总数量。

（一般与accepts一致，除非服务器限制了连接数量）。

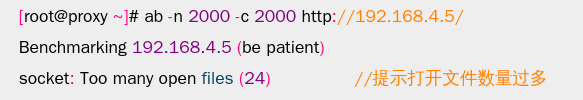
Requests：客户端发送的请求数量。

Reading：当前服务器正在读取客户端请求头的数量。

Writing：当前服务器正在写响应信息的数量。

Waiting：当前多少客户端在等待服务器的响应。

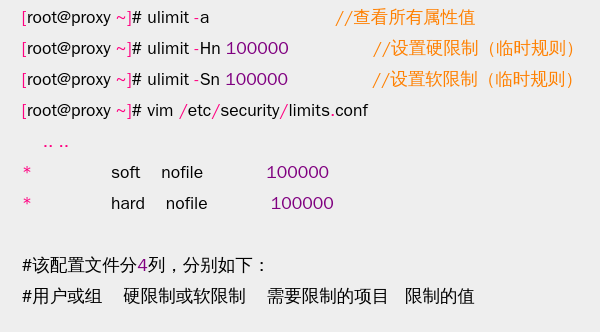
1. 优化并发量(如果客户端访问服务器提示“Too many open files”如何解决)
2. 优化前使用ab高并发测试



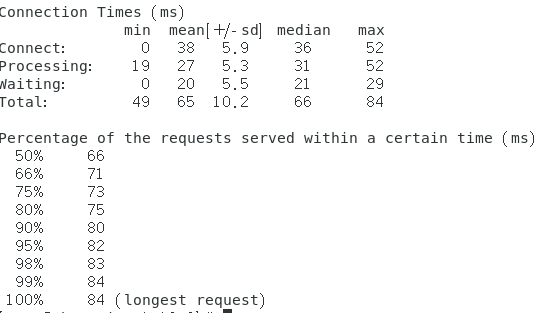
1. 修改Nginx配置文件，增加并发量



1. 优化Linux内核参数（最大文件数量）



1. 优化后测试服务器并发量（因为客户端没调内核参数，所以在proxy测试）



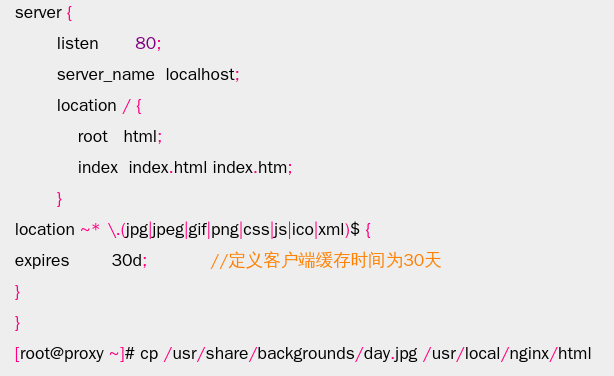
1. 优化Nginx数据包头部缓存(如何解决客户端访问头部信息过长)



1. 如何让客户端浏览器缓存数据(本地缓存静态数据)

地址栏内输入about:cache

清空firefox本地缓存数据



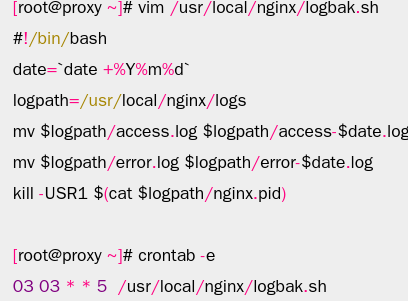
优化后，使用Firefox浏览器访问图片，再次查看缓存信息

1. 日志切割
2. 把旧的日志文件重命名
3. Kill -USR1 PID(nginx的进程PID号)

手动执行:# mv access.log access2.log

# kill -USR1 $(cat /usr/local/nginx/logs/nginx.pid)

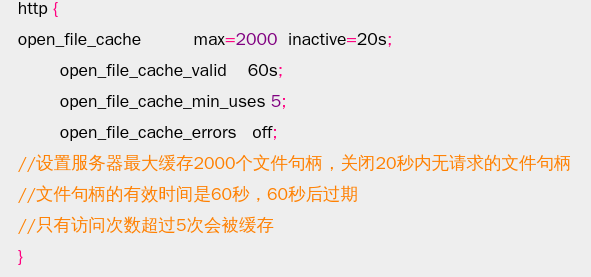
自动执行:



1. 对页面进行压缩处理



1. 开启文件缓存功能



附:常见错误

