基于Nginx的Web服务器研究与优化

杨繁荣
江西软件职业技术大学　江西南昌　330000

摘　要：当前互联网服务器每天都要面对数量庞大的用户访问需求，这些大量的需求对Web服务器的性能要求极高，在大型服务器过于昂贵，传统Apache服务器又过于占用内存且处理高并发能力不够的情况下，人们开始着手于Nginx这种轻量级的Web服务器的研究之上。因为Nginx的高并发处理能力强、占用内存资源少、稳定性高、配置简单且代码开源等优点，研究Nginx的运行机制和源代码具有很大的意义。本文就从集群技术、负载均衡技术为引初步对Nginx的数据结构、进程模型、负载均衡算法以及源代码优化等方面简要的对Nginx进行分析。

关键词：服务器；Nginx；研究；轻量级

Research and Optimization of Web server based on Nginx

Fanrong Yang
Jiangxi Software Vocational and Technical University, Jiangxi Nanchang 330000

Abstract: the current Internet server every day to face a large number of user access demand, these large demand for Web server performance requirements, in the large server is too expensive, traditional Apache server too memory and high concurrency capacity, people began to Nginx this lightweight Web server research.Because of Nginx's high concurrent processing ability, less memory resources, high stability, simple configuration and open source code, it is of great significance to study the operation mechanism and source code of Nginx.In this paper, Nginx briefly analyzes the data structure, process model, load balancing algorithm and source code optimization of cluster technology and load balancing technology.

Keywords: Server, Nginx, Research, Lightweight

引言：

现阶段，互联网的应用已经与我们的生活密不可分，衣食住行都在互联网发展的带动下变得越来越简单，越来越便捷。截止到2022年，我国使用互联网的人数已经达到10.32亿，互联网的普及率高达百分之七十三。如此庞大的用户基数，自然带来了爆发式的并发Web访问请求。如此大量的访问请求导致服务器处理高并发式请求的压力增大，如果这个问题得不到良好的处理，对于我国服务器运营企业的发展将会产生很大的影响，从而限制企业的发展。大型企业因为其资金雄厚，体量巨大，可以通过革新服务器的硬件性能，或者更换更加大型的服务器设备来解决这个问题；但是对于更多的中小型企业来说，大型服务器设备的价格过于昂贵，根本承受不起。于是从软件角度出发改善服务器性能的技术也就应运而生了，这种技术就叫做集群技术。

一、集群技术含义

集群技术也就是了通过利用多台服务器组成一个集群系统，通过系统给用户提供服务。集群技术可以将一群松散的，独立的服务器联结在一起，形成单一系统的管理模式，服务器的数量可以根据访问数量的多少进行相应配置；而当集群中一台服务器出现技术故障时，还可以将访问请求转发给其他的服务器。相比于更换大型服务器，集群技术的应用不仅成本更加低廉，系统的稳定性和持续性也得到了巨大的提升。

二、负载均衡技术含义

集群技术因为需要对众多的服务器进行合理的调度和并发式请求的分配，于是就引出了一个概念，叫做负载均衡技术。

负载均衡技术是集群技术的核心，一般应用于并发式请求量过高，服务器平均响应时间过程的环境当中。通过相应的算法得出的结果制定合适的分配策略，将客户端的请求合理的分配给后端的服务器。这样可以降低服务器对于客户端请求的响应时间，增加吞吐量，合理利用空闲的服务器资源。

目前在实现负载均衡的方式上也有硬件方式和软件方式两种；硬件方式是利用F5 BIG-IP或者深信服AD系列等负载均衡器来实现负载均衡，但是负载均衡器的价格也很昂贵，会对中小型服务器企业带来很大的经济负担。因此大部分企业会选择价格便宜，配置简单，使用灵活的软件优化方式来实现负载均衡。

在集群系统当中，各个服务器会因为自身硬件配置等原因在处理能力和负载性能产生差异，因此如果负载均衡器对于客户端的请求分配不合理，就容易导致服务器超载，使集群系统的整体性能降低，负载均衡进行分配的原则是要达到以下几种目标：

（1）将客户端发来的请求按能者多劳的形式分配给后端服务器。

（2）争取以最小的消耗来获得最快的响应时间。

（3）尽可能做到最大的资源利用率。

三、对Nginx的研究

通过软件优化方式实现负载均衡技术自然避免不了对于HTTP服务器软件的研究，常用的HTTP服务器软件有重量级Web服务器（例如Apache和IIS）和轻量级Web服务器（例如Nginx和Lighttpd）两种，但重量级的Web服务器，虽然支持的功能多，但是其处理速度很慢，会占用大量的内存资源；相比之下Nginx这种轻量级的Web服务器由于其内存消耗低，高并发，代码开元等特点，受到了国内外企业和开发者的喜爱。在我国，大家熟知的百度、京东、腾讯、淘宝、网易等都是使用的Nginx。

作为轻量级Web服务器，Nginx在很多方面有着其他轻量级甚至是重量级Web服务器无法比拟的优势：

Nginx采用的是内核poll模型和分阶段资源分配技术，可以在支持更多并发连接的同时，占用很低的内存资源，性能极高。

Nginx效率远高于其他服务器。

Nginx可以实现无缓存的反向代理，从而提高网站运行速度。

Nginx可以直接在内部支持HTTP代理服务器的对外服务，还支持利用算法实现负载均衡。

Nginx支持热部署，不仅启动速度极为迅速，而且可以在不断开服务的情况下进行版本和配置的升级，可以全天无休的运作[1]。

1.Nginx的模块化结构

Nginx的架构高度模块化的架构，就像是拼装机器人一样；除了核心代码，其他所有功能的实现都依赖于模块化的设计，其模块大致可以分为以下几类：

（1）事件模块

事件模块也就是ngx_events_module，其主要作用是负责收集、管理、分发事件的处理机制，根据服务器的操作系统和配置参数的不同，来选择不同的事件驱动方式。

（2）阶段处理模块

阶段处理模块也就是phase handler，其主要作用是接收用户请求并给予回应。

（3）Upstream模块

Upstream模块的作用是选择一台服务器来实现Web请求的反向代理。

（4）过滤模块

过滤模块filter可以完成已接收的来自处理模块的数据请求的一些附加功能。

（5）负载均衡模块

负载均衡模块load-balance的作用是选择不同的后台服务器并交给upstream模块进行反向代理。

2.Nginx进程模型和事件驱动架构

在Nginx服务器中，网卡、磁盘作为事件发生源负责产生事件；事件模块作为事件收集器负责事件的收集和分发；所有模块都是事件的处理器，事件模块会根据其他模块对于客户端请求的诉求将请求访问分配到适合的处理模块当中进行处理，这样的一整个流程就是Nginx的事件驱动架构[2]。

Nginx的进程模型是多进程的运行方式；在Nginx当中会直接使用worker进程来处理不同的请求，同时因为Nginx的事件驱动结构会使Nginx中Worker的数量保持一个恒定的水平。

当Nginx启动后，它的第一步动作是创建一个master进程，这个进程起到一个监督，分配的作用。它会将实际的处理工作分配给下面的worker进程，让每一个worker进程以平等竞争的方式来争取到处理连接的机会，因为各个worker进程之间是相互独立的，所以不会出现同一请求同时被两个进程处理的情况。

3.Nginx三大核心数据结构

（1）ngx_str_t数据结构

ngx_str_t是对字符串类的重新定义，其原型是：

```
typedef struct {
size_t      len;
u_char     *data;
} ngx_str_t;
```

（2）Ngx_array_t数据结构，该数据结构代码形式如下[3]：

```
typedef struct ngx_array_s  ngx_array_t;
struct ngx_array_s {
```

```
    void       *elts;  //具体的数据区域的起始地址
    ngx_uint_t   nelts; //已经存储了的元素数量
    size_t       size;  //单个元素的字节大小
    ngx_uint_t   nalloc; //数组容量，即数组预先分配的内存大小
     ngx_pool_t   *pool;  //内存池,用其保存分配此数组的内存池地址。
};
```

（3）Ngx_pool_t数据结构，也称内存池结构，该数据结构定义如下：

```
00048 : typedef struct {
00049 :    u_char      * last ;  //当前内存池分配到此处，即下一次分配从此处开始
00050 :    u_char      * end ;   //内存池结束位置
00051 :    ngx_pool_t  * next ;  //内存池里面有很多块内存，这些内存块就是通过该指针连成链表的
00052 :    ngx_uint_t   failed ; //内存池分配失败次数
00053 : } ngx_pool_data_t ;       //内存池的数据块位置信息
00054 :
00055 :
00056 : struct ngx_pool_s {      //内存池头部结构
00057 :     ngx_pool_data _t  d ;    //内存池的数据块
00058 :     size_t       max ;   //内存池数据块的最大值
00059 :     ngx_pool_t       * current ; //指向当前内存池
00060 :     ngx_chain_t        * chain ;  //该指针挂接一个ngx_chain_t结构
00061 :     ngx_pool_large_t    * large ;  //大块内存链表，即分配空间超过max的内存
00062 :     ngx_pool_cleanup_t * cleanup ; //释放内存池的callback
00063 :     ngx_log_t          * log ;   //日志信息
00064 : } ;
```

四、Nginx搭载的负载均衡算法

在Nginx中搭载的负载均衡算法有：加权轮询、IP哈希、最小连接数等，同时它也支持搭载第三方的负载均衡算法，例如fair和url哈希算法等

Nginx中负载均衡的基础文件是ngx_http_upstream_module，其对应的模块就是upstream模块，它通过对各个服务器的权值大小进行赋值，再通过proxy_pass等指令将请求合理分配到不同服务器上，例如：

```
upstream backend {
    server backend1.example.com weight=5;
    server 127.0.0.1:8080    max_fails=3 fail_timeout=30s;
    server unix:/tmp/backend3;
    server backup1.example.com  backup;}
server {
    location / {
        proxy_pass http://backend; }}
```

像这样的一个定义就可以实现组调度功能了。

1.加权轮询算法

如果Nginx的配置文件Upstream当中没有配置任何负载均衡算法时，就会默认负载均衡算法为加权轮询算法，其源代码文件名为ngx_http_upstream_round_robin.c。它的部分代码如下：

```
http {
    upstream cluster {
        server 192.168.1.2 weight=5;
        server 192.168.1.3 weight=3;
        server 192.168.1.4 weight=1;
    }
    ...
location / {
        proxy_set_header X-Real-IP $remote_addr;   //返回真实IP
        proxy_pass http://cluster;      //代理指向cluster
    }
```

在Nginx加权轮询算法中，每个节点都有三个权重变量，分别是：weight、currentWeight、effectiveWeight，它们代表的含义分别是配置的权重、节点当前权重和有效权重。

它的整个算法流程是首先访问所有节点，计算出当前状态下所有节点的有效权重之和

再把所有节点中当前权重中最大的一个节点作为选中节点；选中节点的权重算法是currentWeight = currentWeight – totalWeight。

加权轮询算法相比于轮询算法来说，它给每一个服务器增加了权重的概念，对于服务器本身的真实性能反应不一定准确，而且由于客户端每次的请求处理时间都有所不同，会造成不均衡的状况发生；也就是说当权重大的服务器正在处理一个用户请求时，如果又来了一个请求，依然会被分配到这台服务器当中，导致这台服务器超载，而其他服务器空闲的情况出现。

2.IP Hash算法

IP哈希算法的处理原则是根据IP地址进行hash，来找到相应后端服务器并进行分配。其核心算法如下：

（1）首先是根据客户端的IP地址计算得到一个数值：

```
for (i = 0; i < 3; i++) {
```

```
    hash = (hash * 113 + iphp->addr[i]) % 6271;
  //iphp->addr[i]为ip的点分十进制法的第i段
}
```

（2）接着根据计算出来的数值，寻找对应的后端：

```
w = hash3 % total_weight;
while (w >= peer->weight) {undefined
    w -= peer->weight;
    peer = peer->next;
    p++;
}
```

IP哈希算法和加权轮询算法的问题相同，都是没有考虑到后端服务器的真实负载情况，当请求都来自同一个IP地址的时候就很容易超载[4]。

3.加权最小连接数算法

加权最小数方法相比于前两种，它是通过连接数和权重相乘的结果来对服务器负载情况进行衡量，因此加权最小数不仅会考虑到后端服务器的连接数也会考虑到每台服务器的性能问题。

但是它对于服务器权重的配置也是根据管理员经验配置得出的，多以加权最小数算法利用连接数与权重的乘积对各服务器的真实负载情况计算也不准确。

五、Nginx服务器配置文件的优化

1.在全局块中，worker_processes会决定创建的worker数量，前面已经说过，worker数量对于Nginx的性能影响很大，因此需要将worker数量设定在合理的范围内，低了会使服务器资源空闲，高了会产生过多的进程切换，因此可以将worker_processes设置为CPU的内核数量。同时可以为每一个worker进程绑定一个固定额CPU核心以减小多个worker进程争抢一个内核的现象发生。

2.在events块的设置当中，可以设置每个worker进程的最大连接数，理论上应为1024，和全局块中一样；另外需要根据操作系统的不同设置不同的事件驱动模型，在BSD操作系统下可以设置kqueue事件驱动模型，在Linux操作系统下可以设置epoll事件管理机制，对于性能的提升有很大帮助[5]。

3.对于HTTP块的设置中，设置一个连接超时时间上限，来将连接超时的请求进行关闭，释放出系统资源来应对其他请求。面对高并发数量请求时很有用。设置server_tokens off隐藏Nginx的版本号，以防止被恶意攻击。启动Linux的系统函数sendfile（）来发送文件，提高发送效率；设置open_file_cache：文件缓存和时间间隔，定时淘汰未访问元素；设置open_file_cache_valid：定期检查元素的有效性；设置open_file_cache_min_uses：来确定不被淘汰的最小访问次数。设置gzip on来将数据在发送前进行压缩，降低数据量和传输时间；设置gzip_static来查询资源是否被压缩过；设置gzip_min_length来确定压缩资源的门限，减少不必要的压缩；设置gzip_comp_level和gzip_type来规定压缩等级和类型。

六、关于优化Linux的内核参数

Nginx的内核参数一般是针对的通常情况，而在高并发情况下，应对能力会有所不足，因此需要修改部分内核参数：

1.修改fs.file--max=999999；这个参数的设置直接限制最大并发连接数，在高并发情况下应设置到最大。

2.设置net.ipv4.tcp_tw_reuse=1；这个设置可以将处在等待状态的连接请求重新建立TCP连接，在高并发下，当这个状态的请求很多的，这样设置可以提高响应速度。

3.设置net.ipv4.tcp_fin_timeout=30；这个参数表示在服务器主动关闭连接情况下，socket保持在FIN-WAIT-2状态下的最大时间为30秒。

4.设置net_ipv4.tcp_keepalive_time=600；设置此参数可以在建立长连接的时候更快的清理无效的连接。

5.设置net.ipv4.tcp_max_tw_buckets=5000；这个参数的设置可以清理过多的TIME-WAIT套接字，防止Web服务器因为TIME-WAIT套接字的数量过多而变慢。

6.设置net.ipv4.tcp_max_syn.backlog=262144；设置这个参数可以防止出现因Nginx繁忙而来不及建立accept新连接的时候，Linux丢失客户端发来的请求问题的出现[6]。

七、结束语

本文阐述了集群技术、负载均衡技术，对Nginx的内在结构和内置负载均衡算法进行了剖析，分析了其性能高的运行原理，对其配置文件和内核参数进行了优化，使其在高并发请求情况下可以提高运行性能，提升响应效率。

参考文献：

[1]姚兆凡.轻量级Web服务器Nginx的研究与优化[D].南京邮电大学，2017.

[2]戴伟，马明栋，王得玉.基于Nginx的负载均衡技术研究与优化[J].计算机技术与发展，2019，29（3）：4.

[3]李思莉，杨井荣，苟强.轻量级Web服务器的高并发技术研究与实现[J].计算机技术与发展，2020，30（10）：5.

[4]李若兰.基于Nginx的Web服务器优化的应用研究[J].科技风，2021（9）：2.

[5]戴伟.基于Nginx高性能Web服务器的理论研究与性能改进[D].南京邮电大学，2020.

[6]高原.基于Nginx的web服务器负载均衡策略研究[D].海南大学，2019.