基于 Nginx 负载均衡的动态改进算法

张宇星[1],马明栋[2],王得玉[2]
(1. 南京邮电大学 通信与信息工程学院,江苏 南京 210003;
2. 南京邮电大学 地理与生物信息学院,江苏 南京 210003)

摘 要:随着过去几十年互联网服务的指数增长,各大网站的访问量急剧上升。海量的用户请求使得热门网站的网络请求率可能在几秒钟内大规模增加。一旦服务器承受不住这样的高并发请求,由此带来的网络拥塞和延迟会极大地影响用户体验。负载均衡是高可用网络基础架构的关键组件,通过在后端引入一个负载均衡器,将工作负载分布到多个服务器来缓解海量并发请求对服务器造成的巨大压力,提高后端服务器和数据库的性能以及可靠性。而 Nginx 作为一款高性能的 HTTP 和反向代理服务器,正越来越多地应用到实践中。文中将分析 Nginx 服务器负载均衡的体系架构,研究默认的加权轮询算法,并提出一种改进后的动态负载均衡算法,实时收集负载信息,重新计算并分配权值。通过实验测试,对比不同算法下的负载均衡性能,改进后的算法能有效提高服务器集群的性能。

关键词:Nginx;负载均衡;反向代理;高并发;动态算法
中图分类号:TP393.0 **文献标识码**:A **文章编号**:1673-629X(2020)03-0073-04
doi:10.3969/j.issn.1673-629X.2020.03.014

A Dynamic Improvement Algorithm Based on Nginx Load Balancing

ZHANG Yu-xing[1], MA Ming-dong[2], WANG De-yu[2]
(1. School of Telecommunications & Information Engineering, Nanjing University of Posts and Telecommunications, Nanjing 210003, China;
2. School of Geographical and Biological Information, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

Abstract: With the exponential growth of Internet services in the past few decades, visits to major websites have risen dramatically. Massive user requests make it possible for popular websites to see a massive increase in the rate of web requests in a few seconds. Once the server can't withstand such high concurrent requests, the resulting network congestion and delays can greatly affect the user experience. Load balancing is a key component of high-availability network infrastructure. By introducing a load balancer in the back end, the workload is distributed to multiple servers to alleviate the huge pressure on servers caused by massive concurrent requests, and improve the performance and reliability of the back-end servers and databases. As a high-performance HTTP and reverse proxy server, Nginx is more and more applied in practice. In this study, we will analyze the architecture of Nginx server load balancing, study the default weighted polling algorithm and propose an improved dynamic load balancing algorithm to collect load information in real time, recalculate and distribute weights. The experiment shows that the proposed algorithm can effectively improve the performance of server cluster by comparing the load balancing performance of different algorithms.

Key words: Nginx; load balancing; reverse proxy; high concurrence; dynamic algorithm

0 引 言

随着互联网的快速发展,PC 端、移动端的逐渐兴起,早期的单台服务器已经难以处理用户多样化的并发请求。通过服务器集群的负载均衡技术能有效地分散单台服务器的访问压力,还能有效避免当一台 Web 服务器发生故障或遭到攻击无法对外提供服务时,整个服务无法进行的状况[1]。当集群中的 Web 服务器宕机时,通过负载均衡技术可以将访问流量转移给集

收稿日期:2019-04-02 **修回日期**:2019-08-07 **网络出版时间**:2019-11-07
基金项目:江苏省自然科学基金-青年基金项目(BK20140868)
作者简介:张宇星(1996-),男,硕士研究生,研究方向为网络编程与服务端开发;马明栋,博士,教授,研究方向为地理信息系统平台软件设计与开发等;王得玉,博士,副教授,研究方向为水环境遥感、GIS 软件设计与开发。
网络出版地址:http://kns.cnki.net/kcms/detail/61.1450.TP.20191107.0918.074.html

群中其他的服务器。负载均衡技术通过均衡后端服务器的压力,高效利用所有的后端服务器,提高整个服务器集群的性能。负载均衡的实现主要包括硬件和软件两个方面[2]。与基于硬件的负载均衡设备相比,基于软件的负载均衡不需要独立的机器设备,通过在服务器上安装和配置软件来实现负载均衡,配置灵活,可扩展性强,成本也较低[3]。LVS 和 Nginx 是比较常见的基于软件的负载均衡,LVS 工作在网络的第四层,作为请求的转发,没有流量,而 Nginx 工作在网络的第七层,可以高效地处理 HTTP 应用,可使用的场合也多于 LVS,因此文中将主要研究 Nginx 的负载均衡。

Nginx 是由俄罗斯人开发的一款开源 Web 服务器软件,占用资源低,处理请求快,模块化设计,可扩展性好,配置灵活,采用异步、非阻塞、事件驱动的方式处理请求,并发量高,具有反向代理和缓存服务功能[4-5]。在实际中应用非常广泛,目前国内的很多企业也都采用 Nginx 作为负载均衡和反向代理的 Web 服务器[6]。

负载均衡算法对 Nginx 负载均衡技术起着关键作用,文中通过研究 Nginx 自带的负载均衡算法,提出一种动态的改进算法,以有效提高服务器集群的性能。

1 Nginx 体系架构

Nginx 服务器启动后,产生一个 Master 进程,由 Master 进程产生多个 Worker 进程。Master 进程负责管理多个 Worker 进程,而 Worker 进程则负责接收并处理客户端的请求[7]。Nginx 的主要配置文件 nginx.conf 在编译安装好的 nginx/conf 目录下可以找到,打开文件可以观察到其大致结构,如图 1 所示。在文件中配置 worker 进程数、连接数等参数可以达到 Nginx 性能调优的效果。其中全局块主要是配置 Nginx 的用户组,工作进程数,进程 pid 的存放路径等。events 块主要配置工作进程的最大连接数。http 全局块主要配置 sendfile 和超时信息以及文件的引入等。server 块和 location 块主要配置主机的服务信息等。

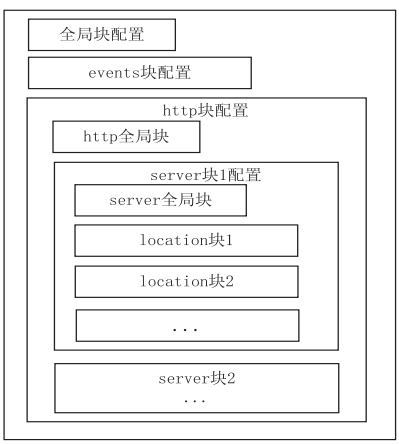

图 1 nginx.conf 配置文件结构

Nginx 负载均衡的实现就是源于 Nginx 高效的反向代理服务[8]。反向代理服务的对象是服务器端,客户端并不知道实际处理请求的机器是哪一台。反向代理服务器收到来自客户端的请求,将请求转发给后端服务器处理。外部客户不能直接访问真实的服务器,即使遭到恶意攻击,也能保证后端服务器的安全[9]。Nginx 的反向代理服务由 proxy_pass URL 来配置,反向代理配置如下:

```
upstream mynginx{
server 192.168.168.161;
server 192.168.168.163;
}
server {
listen      80;
server_name   localhost 192.168.168.162;
resolver8.8.8.8;
location / {
proxy_pass http://mynginx;
include proxy_params;
}
}
```

当客户端的请求发送过来时,Nginx 利用 proxy_pass 指令进行代理转发,然后在 upstream 模块中根据负载均衡算法将请求分配给后端服务器[10]。

Nginx 服务器接收客户端的请求时,并不会立即转发给上游的 Web 服务器,而是如图 2 所示,在 Nginx 反向代理服务器上设置缓存,以键值对的形式存储在内存或者硬盘中,获得完整的 HTTP 请求包后再将完整的请求发给上游服务器[11]。并且对于一些静态资源,Nginx 通过使用 proxy_cache 将用户的请求缓存到本地一个目录。当下一次有相同请求转到代理服务器时,会先查看本地缓存,如果找到了该请求则直接返回给客户端,如果本地缓存中没有找到,再将请求转发到后端 Web 服务器,减少了 Nginx 与后端 Web 服务器的连接次数,加快了响应速度,提高了 Nginx 处理性能[12]。

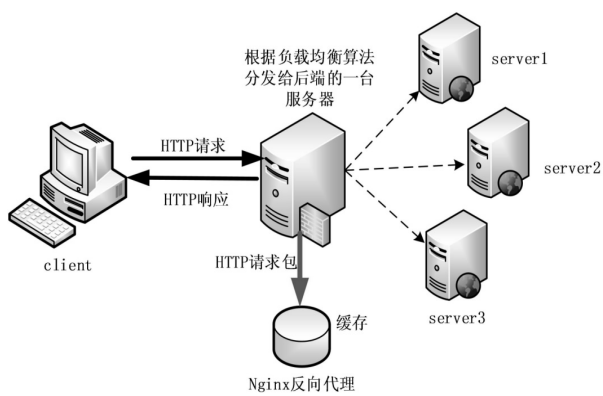

图 2 Nginx 反向代理模型

2　算法改进设计

常见的负载均衡算法主要有加权轮询、最小连接数、ip_hash、url_hash 和 fair 算法[13]。加权轮询算法按照预先分配的权重轮流分发给各个服务器节点,没有考虑服务器的连接数和实时负载。ip_hash 和 url_hash 针对 Http 应用本身来做分流策略,将客户端请求按 ip 或 url 的哈希结果分配[14]。可以避免用户每次访问的服务器不同,使得 session 落在不同的服务器上,造成用户端 cookie 验证失败等问题,但是容易导致某台服务器节点的压力变得非常大而其他节点却空闲的不均衡现象。最小连接数和 fair 算法考虑的指标也过于单一,因此算法的均衡效果也不是很明显[15]。

文中在 Nginx 的官网下载 nginx-1.14.2 版本的源码,在源码的 src/http 目录下可以找到默认的加权轮询算法 ngx_http_upstream_round_robin.c 文件。文件中的 ngx_http_upstream_get_peer 函数计算每一个后端服务器的权值,在函数中定义指针 * peer 和 * best,通过 for 循环遍历后端服务器。再通过 best 指针选取权值最大的服务器,将 best 指针作为 ngx_http_upstream_get_peer 函数的返回值。函数中权值的部分计算代码如下:

```
peer->current_weight+=peer->effective_weight;
total+=peer->effective_weight;
if(peer->effective_weight<peer->weight){
peer->effective_weight++;
}

if(best==NULL||peer->current_weight>best->current_weight){
best=peer;
p=i;
}
```

文中提出的动态改进算法是在加权轮询算法的基础上通过动态收集实时负载和硬件利用率,建立数学模型,得出动态反馈的负载均衡算法。

首先用 S_i 表示服务器集群中的每个节点,C_i 表示每个服务器节点的性能,服务器节点的默认权值为:

$$W_{\mathrm{d}}(i)=\frac{C_i}{\sum_{i=1}^{n}C_i} \tag{1}$$

用 $U_{\mathrm{c}}(i)$,$U_{\mathrm{m}}(i)$,$U_{\mathrm{d}}(i)$ 和 $U_{\mathrm{b}}(i)$ 分别表示周期 T 内单个节点的 CPU、内存、磁盘 IO 和网络带宽的占用率,不同指标对服务器性能影响不同,K_{c},K_{m},K_{d},K_{b} 分别表示 CPU、内存、磁盘 IO 和网络带宽对服务器的影响因子[16],并且 $K_{\mathrm{c}}+K_{\mathrm{m}}+K_{\mathrm{d}}+K_{\mathrm{b}}=1$。每个节点性能的实时负载比重 $W_{\mathrm{N}}(i)$ 为:

$$W_{\mathrm{N}}(i)=\frac{K_{\mathrm{c}}*U_{\mathrm{c}}(i)+K_{\mathrm{m}}*U_{\mathrm{m}}(i)+K_{\mathrm{d}}*U_{\mathrm{d}}(i)+K_{\mathrm{b}}*U_{\mathrm{b}}(i)}{K_{\mathrm{c}}+K_{\mathrm{m}}+K_{\mathrm{d}}+K_{\mathrm{b}}} \tag{2}$$

在最小连接数算法中,当有请求分发过来时,连接数加 1,当请求处理完成时,连接数减 1。文中提出的动态改进算法也将考虑每个节点的实时连接数 $N_i=\{N_1,N_2,\cdots,N_n\}$,每个节点的连接数占比为:

$$W_{\mathrm{c}}(i)=\frac{N_i}{\sum_{i=1}^{n}N_i} \tag{3}$$

当服务器节点负载较重时,请求的响应时间就比较长,所以响应时延也是衡量服务器节点实时负载的一个重要因素。然后考虑请求的响应时间,在 Nginx 的日志文件中提取每个节点服务器在周期 T 内的最近一次请求时间,$T_i=\{T_1,T_2,\cdots,T_n\}$,后端服务器集群最近一次的响应平均时间为:

$$\mathrm{Avg}(T)=\frac{1}{N}\sum_{i=1}^{n}T_i \tag{4}$$

服务器节点的响应时间比重为:

$$W_{\mathrm{t}}(i)=T_i/\mathrm{Avg}(T) \tag{5}$$

综合节点的实时性能,连接数和响应时间建立数学模型:

$$W_i=(1-W_{\mathrm{N}}(i))*W_{\mathrm{d}}(i)+(1-W_{\mathrm{c}}(i))*W_{\mathrm{d}}(i)+(1-W_{\mathrm{t}}(i))*W_{\mathrm{d}}(i) \tag{6}$$

将三个因素的权值动态相加得到实时的权值,即每个服务器节点的权值计算如下:

$$W_i=(3-W_{\mathrm{N}}(i)-W_{\mathrm{c}}(i)-W_{\mathrm{t}}(i))*W_{\mathrm{d}}(i) \tag{7}$$

将计算所得结果与 ngx_http_upstream_get_peer 函数中的 effective_weight 结合,通过 best 指针返回权值最大的服务器节点。

3　实验环境搭建与测试

实验在虚拟机中搭建了五台服务器,一台作为客户端测试连接,一台作为反向代理服务器,其余三台作为后端服务器。在后端服务器中搭建 Nginx+php 环境来实现动态的网站运行环境。Nginx 通过 fastcgi 接口来实现与 php 文件的交互。在 server 模块的 location 配置中引入一个 index.php 动态资源文件,放置在 location 中的 root 目录下,location 的配置如下:

```
location ~ \.php$ {
root    html;
fastcgi_index    index.php;
fastcgi_pass    127.0.0.1:9000;
fastcgi_param    SCRIPT_FILENAME    $document_root$fastcgi_script_name;
include    fastcgi_params;
}
```

实验环境搭建好后使用 Httperf 和 Autobench 来进行性能测试,Httperf 可以根据每一次不同的测试要求产生不同的工作负载,而 Autobench 是一款基于 Httperf 开发的 perf 脚本工具,可以根据参数设置自动进行并发测试,并将结果保存在. tsv 文件中[17-18]。在客户端中键入如下命令进行测试:

autobench --single_host --host1 192. 168. 168. 162 --port1 80 --uri1 /index. php --quiet --low_rate 20 --high_rate 200 --rate_step 20 --num_call 10 --num_conn 500 --timeout 5 --file results. tsv

实验中分别对默认轮询算法、最小连接数算法以及文中提出的动态反馈算法进行测试,将得到的三个 results. tsv 文件进行整理,得到的统计数据如表 1 和表 2 所示。

表 1 三种算法的响应时间

并发连接数	默认轮询算法/ms	最小连接数算法/ms	动态反馈算法/ms
200	29.3	27.9	29.6
400	30.3	29.5	31.2
600	31.1	31.6	32.4
800	151.4	104.5	64.8
1 000	247.5	157.9	121.1
1 200	317.4	216.5	174.2
1 400	366.3	270.8	212.2
1 600	398.1	303.3	253.7
1 800	441.5	333.9	308.9
2 000	560.7	356.3	329.1

表 2 三种算法的实际并发连接数

并发连接数	默认轮询算法	最小连接数算法	动态反馈算法
200	198	199	199
400	394	394	394
600	579	579	580
800	632	670	722
1 000	649	747	802
1 200	648	759	857
1 400	645	758	852
1 600	648	766	861
1 800	617	763	841
2 000	476	760	834

从表中数据可以看出,当并发量较低时,三种算法的响应时间相差不大,但当并发量达到 600 以上时,三种算法的响应时间都迅速增大,并且最小连接数算法的响应时间要小于默认轮询算法,而文中提出的动态反馈算法响应时间则要优于其他两种。Autobench 测试时设置了 timeout 为 5 秒,超时 5 秒的连接都会当作请求失败的连接。从表 2 中也可以看出,当并发数大于 1 200 时,实际处理的连接数都出现了不同程度的下降,但从整体来看,文中提出的动态算法成功处理的连接数要高于其他两种算法。

4 结束语

服务器集群技术和负载均衡技术能有效减轻 Web 并发请求所带来的后端服务器的压力。而负载均衡算法又对负载均衡技术有着重要的影响。文中所提出的负载均衡优化算法不用每次都去实时地更新权值,而是在周期 T 之后重新收集负载信息,计算权值,分配权值,减少了每次都去重新计算、分配权值带来的不必要的消耗。通过实验数据对比,验证了优化后的算法对负载均衡的性能有一定的提升。同时,在研究过程中,该算法也存在不足之处,对于时间周期 T 的取值,周期太长不能有效反映当前服务器节点的性能,周期太短频繁取值造成性能下降,对于这两方面的平衡还有待进一步的研究。

参考文献:

[1] 杜晋芳. 基于 NGINX 的网络安全管理平台后台框架技术的研究与实现[D]. 北京:北京邮电大学,2016.

[2] 兰 翔. 基于 Nginx 的负载均衡技术的研究与改进[D]. 广州:华南理工大学,2012.

[3] 汪文君. 基于 Nginx 服务器集群负载均衡方案的研究和改进[J]. 电子世界,2017(2):179-181.

[4] 陈大才. 基于 Nginx 的高并发访问服务器的研究与应用[D]. 沈阳:中国科学院大学(中国科学院沈阳计算技术研究所),2018.

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

[6] 汪佳文. 基于移动应用 App 的高并发性能研究与应用[D]. 杭州:浙江理工大学,2018.

[7] 杜 星. 轻量级 Web 服务器 Nginx 的理论与技术研究[D]. 南京:南京邮电大学,2016.

[8] 张 云,许江淳,李玉惠,等. 基于 Nginx 服务器负载均衡技术的研究与改进[J]. 软件,2017,38(8):6-12.

[9] 苏翔宇,朱爱群. CentOS7 下基于 Nginx 的反向代理及负载均衡研究与实现[J]. 现代计算机,2018(10):61-64.

[10] BRCIC P. Improving the performance of physical servers using a proxy servers accelerators[C]//2013 21st telecommunications forum Telfor (TELFOR). Belgrade: IEEE, 2013: 865-868.

[11] 王利萍. 基于 Nginx 服务器集群负载均衡技术的研究与改进[D]. 济南:山东大学,2015.

[12] PAL M B, JAIN D C. Enhancing the web pre-fetching at proxy server using clustering[J]. Engineering Universe for

(下转第 81 页)