浙江理工大学学报(自然科学版),第35卷,第4期,2016年7月
Journal of Zhejiang Sci-Tech University (Natural Sciences)
Vol. 35, No. 4, Jul. 2016

DOI:10.3969/j.issn.1673-3851.2016.07.021

基于Nginx的Web服务器性能优化研究

黄　静,李　炳
(浙江理工大学信息学院,杭州 310018)

摘　要:动态负载均衡算法能最小化并行集群中应用服务的响应时间或执行时间,并且对解决高度并行系统中不可预测负载估计问题至关重要。通过加权最小连接算法与upstream机制相结合,提出一种动态反馈负载均衡算法,解决Nginx负载均衡的动态反馈问题,提高了负载均衡分配的准确性。压力测试结果表明,该算法能避免因动态负载信息采集而引入影响整体性能的通信负载开销,提高算法可行性,减小Web服务器平均响应时间并提升正确响应率,为服务器性能优化提供新的解决方案。

关键词:加权最小连接;upstream;Nginx;性能优化;动态反馈;
中图分类号:TP393　　**文献标志码**:A　　**文章编号**:1673-3851(2016)04-0600-07　　**引用页码**:070601

0　引　言

随着计算机技术的发展和互联网的深度渗透,Web已经成为主要的内容服务形式,并极大促进了Internet用户剧烈增长和流量爆炸式增长。这也为Web服务器带来了巨大挑战。通过负载均衡技术,能有效缓解Web服务器在应对高并发、高访问量情景下的性能压力,提升Web服务器的服务质量。负载均衡技术能拓展现有的网络带宽、增加吞吐量、加强网络数据处理能力、提升网络的灵活性和可用性,有效解决数据访问和业务信息急剧增长给服务器带来的巨大处理压力。

负载均衡技术的实现主要取决于负载均衡调度策略或算法。常见的负载均衡算法包括轮询算法(round robin, RR)、加权轮询算法(weighted round robin, WRR)、最小连接算法(least connection, LC)、加权最小连接算法(weighted least connection, WLC)算法[1]。其中RR、WRR为静态算法,静态算法的实现简单,分析运用方便,但是算法并行性较差,只在特定的情况下才能达到较高的效率;LC、WLC为动态算法,动态算法相比静态算法,性能可以提升40%左右,但是以连接为粒度来衡量服务器负载状况不够全面,不同连接对服务器产生的负载量存在差异。不论静态算法与动态算法,都仅从负载均衡器单方面衡量服务器负载状况,缺少获取服务器实时负载信息为依据进行负载分配。

为了更准确的、合理的分配客户端请求负载,不少学者对动态反馈负载均衡算法做了大量研究。龚梅等[2]提出一种集群系统的透明动态反馈负载均衡算法,基于LVS下行同步信号周期性采集服务器多种负载信息,并引入负载容余进行反馈调节。该算法虽然解决了动态反馈问题,却因周期性采集,增加负载采集通信开销。王玥等[3]提出一种旨在最小化负载均衡开销的动态自适应算法,着重考虑CPU间消息发送、接受的数量,缺乏对服务器负载信息的全面描述。Ren等[4]通过添加单指数平滑预测机制,改进加权最小连接算法,以降低服务器负载倾斜,该算法通过历史数据进行预测,缺乏与服务器的实时动态反馈。因此本研究基于Nginx反向代理构建Web服务器,为优化服务器性能,对动态负载均衡算法进行研究,提出一种基于加权最小连接,结合Nginx的upstream机制的动态反馈负载均衡算法,在不增加通信负载开销的基础上,实现服务器负载信息的动态反馈,提升负载分配的准确性。

收稿日期:2015-11-26
基金项目:浙江省自然科学基金项目(LY12F03012)
作者简介:黄静(1965-),女,辽宁沈阳人,教授,主要从事生态、生物、环保、信息结合技术以及物联网在农业上的应用研究。

1　动态反馈负载均衡算法研究

1.1　加权最小连接算法

加权最小连接算法[5]是根据服务器节点当前服务的连接数与其配置权值的比值,进行请求的分配,连接数越小,权值越大,分配负载的概率越大。该算法是一种动态算法,在加权轮询算法的基础上,既考虑集群中各个服务器节点的性能差异,又考虑各节点实时的处理状态信息,有效提升系统整体性能。

加权最小连接算法主要分为 6 个步骤:a)检查节点是否轮询标志位;b)判断当前节点是否正常工作;c)判断当前节点的失败连接次数是否超过最大连接次数,然后检查连接是否超时;d)计算连接数与权值的比值;e)出现多个相同比值的最优节点时,进行加权轮询;f)将最优节点的当前权值毒化,连接数加一。

假设服务器集群 $S=\{S_1,S_2,\cdots,S_n\}(n>1)$,每个服务器的权值用 $W(S_i)(1\leqslant i\leqslant n)$ 表示,每个服务器当前所处理的连接数用 $C(S_i)(1\leqslant i\leqslant n)$ 表示,即当节点满足式(1)时,负载均衡器与该节点建立连接,并转发请求。

$$\frac{C(S_j)}{W(S_j)}=\min\{\frac{C(S_i)}{W(S_i)}\}(1\leqslant i\leqslant n),(1\leqslant j\leqslant n) \tag{1}$$

服务器权值 $W(S_i)$ 为正,由式(1)可简化为:

$$C(S_j)W(S_i)\leqslant C(S_i)W(S_j) \tag{2}$$

S_i 即为均衡选择的最优节点。具体流程如图 1 所示。

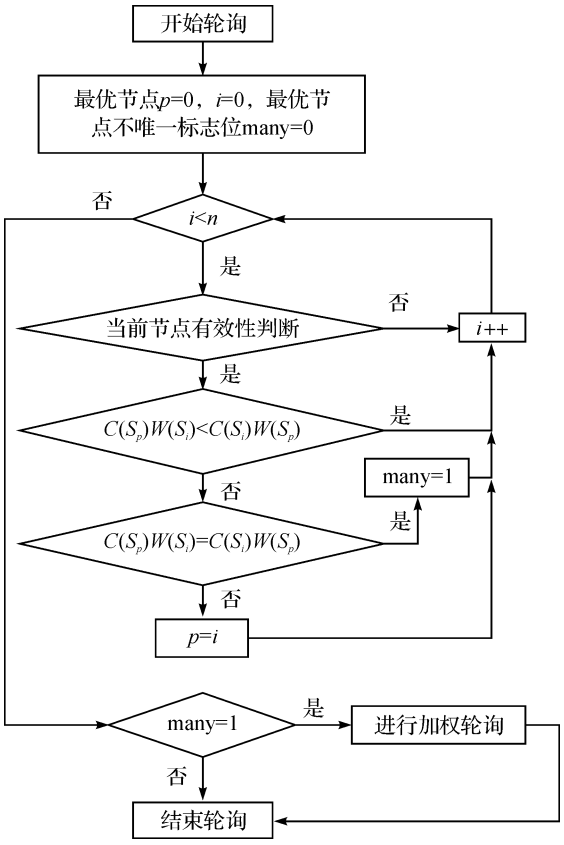


图 1　加权最小连接流程

1.2　upstream 机制

upstream 机制[6]是 Nginx 与上游服务器通信的主要机制,是 HTTP 模块中反向代理的主要模块。客户端的请求通过 Nginx 的 upstream 模块转发给上游服务器,上游服务器将处理好的响应通过 upstream 机制返回给 Nginx,再转发给客户端,或直接从上游服务器返回给客户端。upstream 机制的设计目的主要包括以下几个方面:a)通过反向代理将客户端与后端服务器进行隔离,客户端至 Nginx 称为下游,Nginx 至后端服务器称为上游,降低了 Nginx 的业务处理,将大量耗费资源的请求转发给上游服务器集群,从而轻松应对高并发请求场景;b)通过 upstream 机制可以将客户端发起的请求进行划分、优化、以及生成多个子请求向上游集群中不同的服务器节点进行转发,通过并行处理方式缩短单个请求平均响应时间;c)协调上下游的通信协议;d)通过反向代理进行负载均衡操作,使上游服务器各节点分配相同的请求个数,达到负载均衡效果,转发上游服务器的响应;e)在 Nginx 和上游服务器间构建高速通信网络,提升响应速度。具体的 upstream 机制如图 2 所示。

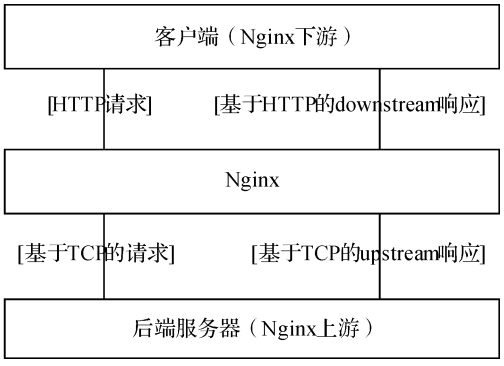


图 2　upstream 机制示意

客户端请求经 Nginx 访问上游服务器的流程分为 6 个阶段:启动 upstream,连接上游服务器,发送请求,接收上游服务器响应包头,处理响应包体,结束请求。其具体的实现流程如图 3 所示。

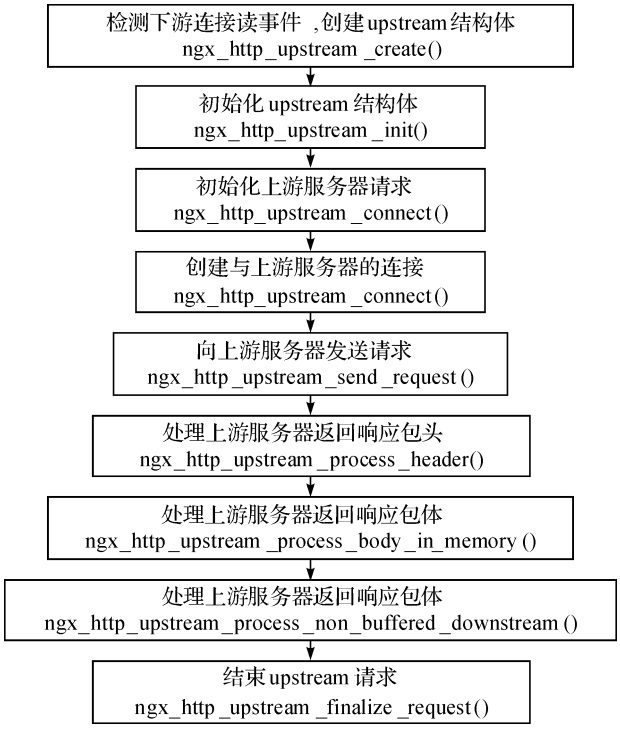


图 3 Nginx 与上游服务器通信流程

1.3 负载信息评价

负载实时、有效、全面的描述是动态反馈负载均衡算法中重要的一个环节，通过采集上游服务器各节点的实时负载信息，负载均衡器动态修改服务器节点的实际权值，为合理、科学、有效地进行请求调度提供依据，提升服务器请求分配的准确性。

本研究构建的负载信息评价模型综合文献[7]与文献[8]有关负载综合评价模型的研究，提出两部分内容。首先，负载信息评价模型通过静态性能指标进行上游服务器节点的静态性能进行描述，其次通过动态负载指标进行节点的实时负载信息描述。

1.3.1 静态性能指标模型 $P(S_i)$

静态性能指标参数包括：CPU运行速率 P_{cpu}、内存容量大小 P_{mem}、磁盘读写速率 P_{disk}、网络传输速率 P_{net}。通过上述 4 个参数构建线性加权函数如式(3)所示。

$$P(S_i)=w_{cpu}\times P_{cpu}+w_{mem}\times P_{mem}+w_{disk}\times P_{disk}+w_{net}\times P_{net} \quad (3)$$

其中，$S_i(1\leqslant i\leqslant n)$ 表示上游服务器节点，w 代表参数权值，各参数的权值比设置为 $\{0.4,0.2,0.2,0.2\}$[9]，Nginx 通过静态性能指标模型为上游服务器节点配置静态权值。

1.3.2 动态负载指标模型 $L(S_i)$

动态负载指标参数包括：CPU 使用率 L_{cpu}、内存使用率 L_{mem}、磁盘 IO 占用率 L_{disk}、网络带宽使用率 L_{net}。通过上述 4 个参数构建线性加权函数如式(4) 所示。

$$L(S_i)=w_{cpu}\times L_{cpu}+w_{mem}\times L_{mem}+w_{disk}\times L_{disk}+w_{net}\times L_{net} \quad (4)$$

其中，$S_i(1\leqslant i\leqslant n)$ 表示上游服务器节点，各参数的权值配置同静态性能指标模型，上游服务器通过动态负载指标模型对负载信息进行描述。

2 动态反馈负载均衡算法实现

动态负载均衡算法能最小化并行集群中应用服务的响应时间或执行时间，并且对解决高度并行系统中不可预测负载估计问题至关重要。集中式动态负载均衡算法在均衡负载过程中引入了影响整体性能的通信负载开销，降低了算法的可行性[10]。本研究基于 Nginx 的 upstream 机制，实现负载信息反馈避免了通信负载开销。

动态反馈负载均衡可以分解为 3 个部分：负载的调度分配，负载信息的动态反馈，负载信息的采集。通过结合 Nginx 内置加权最小连接算法、upstream 机制以及上游服务器负载信息采集器，提出动态反馈的负载均衡算法(dynamic feedback load balancing，DFLB)。

2.1 动态反馈负载均衡算法设计

概念引入：采集指令(GetLoadInfo)，用于 Nginx 向上游服务器发送采集负载信息指令；负载率(LoadRate)，由动态负载指标模型计算而得，用于描述上游服务器节点实时的负载情况，值在 0～1 之间，Nginx 根据负载率作为权值更新依据；连接阈值(ConnectionThreshold)，用于判定是否发送采集指令的依据，基于服务器节点的静态性能进行设置。

动态反馈负载均衡算法设计流程如图 4 所示。

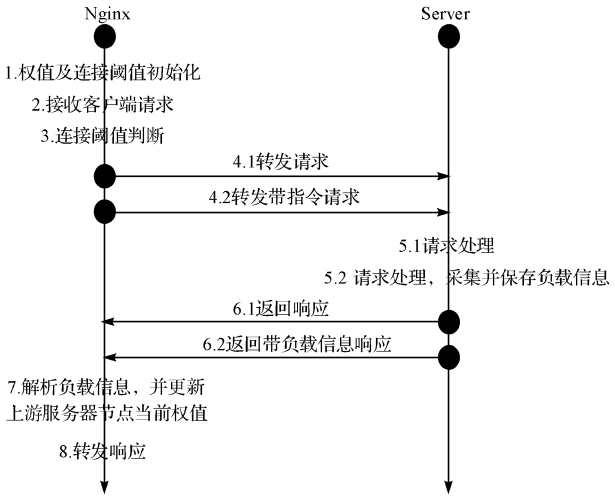


图 4 动态反馈负载均衡算法设计流程

2.2　动态反馈负载均衡算法步骤

动态反馈负载均衡算法的实现包括 Nginx 反向代理服务器与上游服务器两部分。Nginx 负责采集指令的发送,动态权值的修改;上游服务器负责负载信息的采集。主要步骤包括:a)连接数判断,Nginx 接收来自客户端的请求,利用加权最小连接算法,寻找上游负载最优节点 p,比较连接阈值(*ConnectionThreshold*)与最优节点的当前连接数,当前节点 p 连接数大于所设连接阈值时,执行步骤 b),否则执行步骤 c);b)采集指令发送,Nginx 通过 upstream 机制向选择的节点创建连接,并将采集指令(GetLoadInfo)填入连接的请求头;c)将客户端请求打包向上游服务器发送;d)请求指令解析,步骤 a)中选择的上游服务器节点 p,接收来自 Nginx 连接请求,并进行请求头解析,判断是否包含采集指令(GetLoadInfo),如果解析出指令,则进行步骤 e),否则,进行步骤 f);e)采集指令的反馈,上游服务器部署负载信息采集器,进行负载信息采集,依据动态负载指标模型进行负载信息的计算,并进行归一化成负载率(*LoadRate*),并将负载率填入响应头;f)返回响应,上游服务器在完成客户端动态请求后,将响应转发给 Nginx;g)负载信息解析,Nginx 接收上游服务器的请求响应,进行响应包头解析,当解析到负载率时执行步骤 h),否则执行步骤 i);h)当 *LoadRate*>0.7 时,进行当前上游服务器实时权值更新(即减去该节点的静态权值),移除 LoadRate 响应包头;i)将上游服务器响应返回给客户端。动态反馈负载均衡算法通过以上步骤进行采集指令的发送,上游服务器节点负载率的计算,并完成节点当前权值的更改。具体流程如图 5 所示。

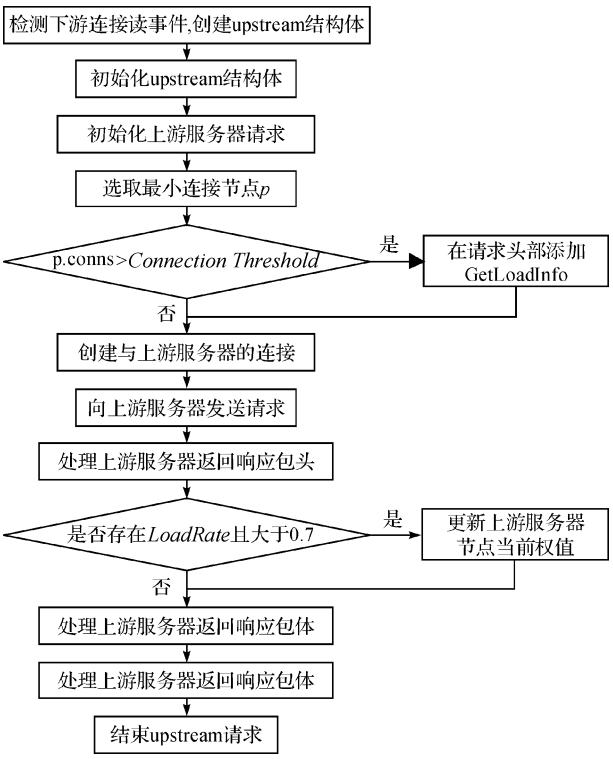


图 5　动态反馈负载均衡算法实现流程

3　算法性能测试验证

3.1　测试环境与测试方案

为验证动态反馈负载均衡算法的性能优化效果,将 WLC 与 DFLB 算法分别在 Nginx 上进行部署,进行服务器性能对比试验,利用 Tsung[11] 压力测试工具对服务器集群进行压力测试。测试服务器集群由 3 台服务器构成,Server{S1,S2,S3},具体配置如表 1 所示。

表 1　服务器软硬件配置

服务器节点	CPU	内存/GB	磁盘/GB	网卡/(Mb/s)	操作系统
S1	双核:2.6GHz	3.9	500	100	Ubuntu 14.04 (LTS)
S2	双核:2.0GHz	3.0	320	100	Ubuntu 14.04 (LTS)
S3	四核:3.7GHz	3.7	980	100	Ubuntu 14.04 (LTS)

测试服务器集群架构如图 6 所示,Nginx 作为反向代理服务器,接收客户端请求,并利用部署的负载均衡算法,将请求向集群中的服务器进行均衡分配。根据表 1 服务器软硬件配置信息与式(3),计算服务器静态性能指标,并通过 Nginx 设置服务器节点静态权值 *Weight*{3,2,5},对应设置服务器连接阈值 *ConnectionThreshold*{150,100,250}.

测试方案:通过测试工具 Tsung 在 50s 内生成 10000 个用户访问服务器集群,每个用户发起一个获取验证码请求,该请求需经 Nginx 转发至上游服务器处理,并返回验证码图片。

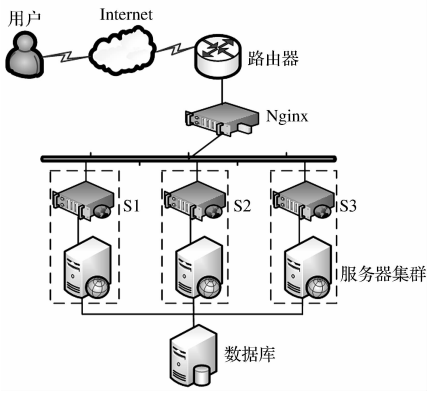


图 6　测试服务器架构

设置测试服务器集群域名:dataplatform. cn,端口:443,协议:ssl。

```
<servers>
    <server host="dataplatform. cn" port="443" type="ssl"/>
</servers>
```

设置负载量:压力分 10 个阶段到达,每秒 200 个用户,持续 5s。

```
<load>
    <arrivalphase phase="1" duration="5" unit="second">
        <users maxnumber="1000" arrivalrate="200" unit="second"/>
    </arrivalphase>
    …
    <arrivalphase phase="10" duration="5" unit="second">
        <users maxnumber="1000" arrivalrate="200" unit="second"/>
    </arrivalphase>
</load>
```

设置传输协议及请求用户类型,其中 Galeon 占 80%,Firefox 占 20%,设置 TCP 端口范围:1025—65535。

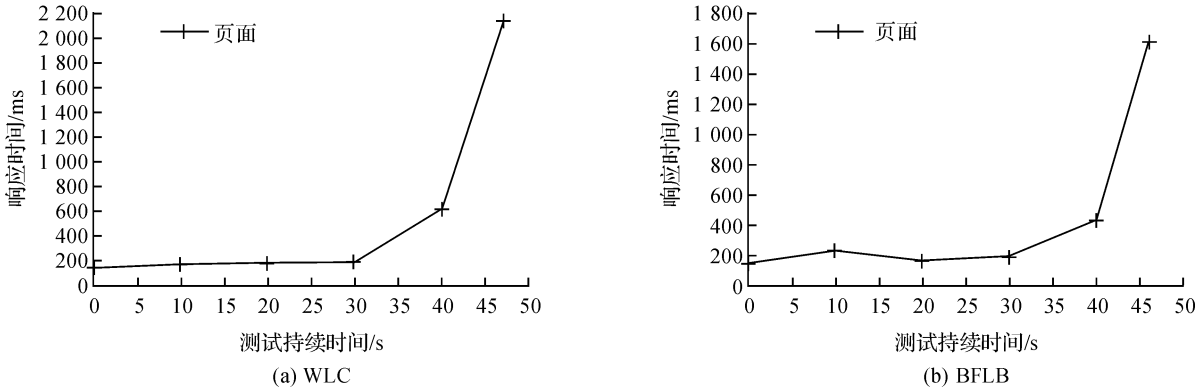
```
<options>
    <option type="ts_http" name="user_agent">
        <user_agent probability="80">Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.8) Gecko/20050513 Galeon/1.3.21</user_agent>
        <user_agent probability="20">Mozilla/5.0 (Windows; U; Windows NT 5.2; fr-FR; rv:1.7.8) Gecko/20050511 Firefox/1.0.4</user_agent>
    </option>
    <option name="ports_range" min="1025" max="65535"/>
</options>
```

设置会话信息,每个用户发起一个会话,每个会话产生一个获取验证码请求,对上游服务器产生负载压力。

```
<sessions>
    <session name="http-example" probability="100" type="ts_http">
        <request><http url="/verifycode/?nocached=1451304777.03" method="GET" version="1.1"/></request>
    </session>
</sessions>
```

3.2 性能测试结果

通过请求事务执行平均响应时间、网络吞吐量以及请求正确响应率这 3 个方面对比分析 WLC 算法与本文提出的 DFLB 算法所作用的服务器性能。图 7 为服务器请求平均响应时间。对比分析两个算法的平均响应时间,随着测试的进行,服务器在前 30s 均表现稳定,平均响应时间在 200ms 左右,从 30s 开始,服务器平均响应时间随着并发用户的累积而增大,直到测试结束,WLC 算法的最大响应时间即将达到 2200ms,而 DFLB 算法的最大平均响应时间为 1600ms 左右。



(a) WLC (b) BFLB

图 7 服务器请求平均响应时间

图 8 为服务器网络吞吐率,其中包含接收吞吐率和发送吞吐率。部署 DFLB 算法的服务器接收吞吐率相对 WLC 较平稳。

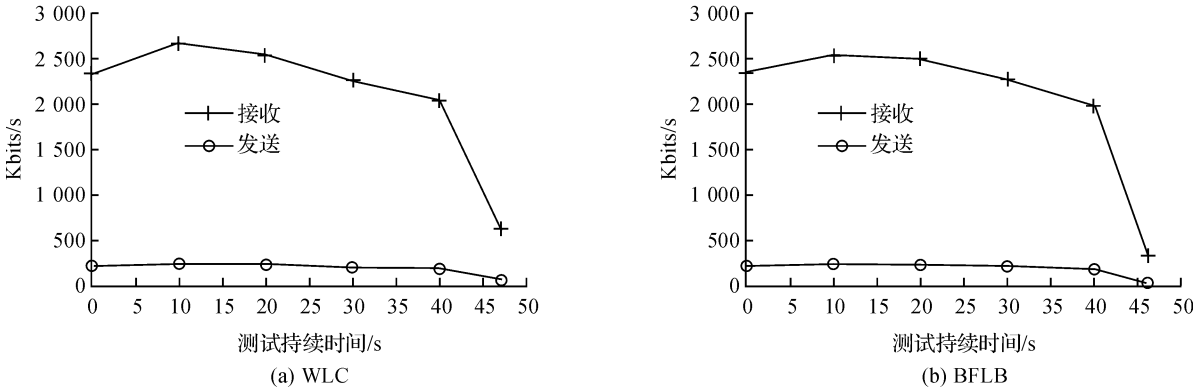


图 8　服务器网络吞吐率

图 9 为客户端请求正确响应率,即服务器返回客户端 HTTP 状态码 200,表示正确接收客户端请求,并响应,图中可以看出两种算法均能正确响应客户端请求,未出现错误状态码。

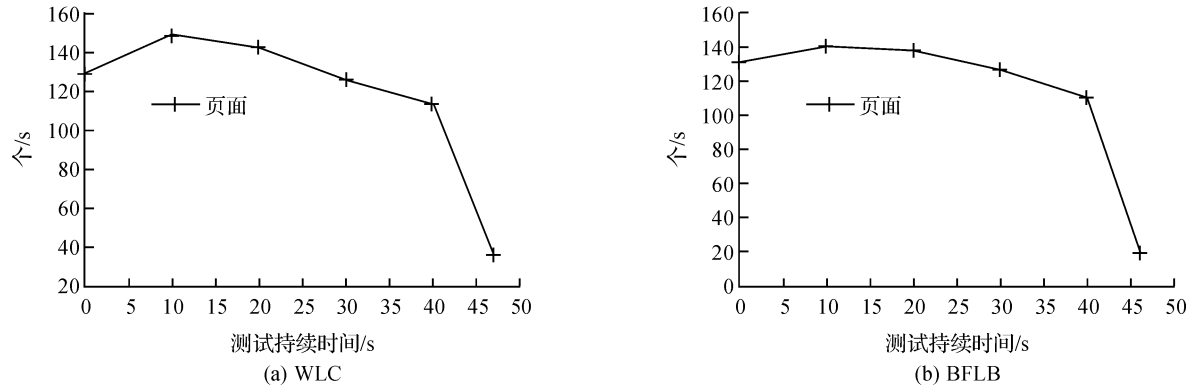


图 9　客户端请求正确响应率

3.3　性能测试结果分析

动态反馈负载均衡算法在加权最小连接的基础上,根据服务器性能设置连接阈值,并结合 upstream 机制,在原有的请求连接基础上,进行负载信息的动态反馈,为 Nginx 在高并发场景下进行负载分配提供可靠依据,提升请求分配的准确性,从而提升 Web 服务器性能。表 2 为测试的详细数据结果。DFLB 相对 WLC 对服务器平均响应时间提升 13%,在并发量、吞吐率及正确响应率方面均有所提高。

表 2　服务器性能指标

负载均衡算法	平均响应时间/s	平均并发量/(请求/s)	平均网络吞吐率/(Mbits/s)	平均正确响应率/(个/s)
WLC	0.25	109.39	2.49	109.39
DFLB	0.22	113.85	2.62	111.19

4　结　语

基于 Nginx 反向代理功能,构建高性能 Web 服务器,利用负载均衡技术对服务器性能进行优化,满足其在高并发场景下集群中存在性能差异节点正常工作需求。压力测试结果表明,动态反馈负载均衡算法能提升 Web 服务器的性能,使请求分配更加准确,提升服务器平均响应时间。

参考文献:

[1] ALI M F, KHAN R Z. The study on load balancing strategies in distributed computing system [J]. International Journal of Computer Science & Engineering Survey,2012,3(2):19-30.

[2] 龚梅,王鹏,吴跃. 一种集群系统的透明动态反馈负载均衡算法[J]. 计算机应用,2007,27(11):2662-2665.

[3] 王玥,蔡皖东,段琪. 一种自适应动态负载均衡算法[J]. 计算机工程与应用,2006,42(21):121-123.

[4] REN X, LIN R, ZOU H. A dynamic load balancing strategy for cloud computing platform based on exponential smoothing forecast[C]//Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on. IEEE,2011:220-224.

[5] ZHANG W S. Weighted least-connection scheduling

[EB/OL]. (2006-12-17)[2016-01-11]. http://kb.linuxvirtualserver.org/wiki/Weighted_Least-Connection_Scheduling.

[6] 陶辉. 深入理解:Nginx 模块开发与架构解析[M]. 北京:机械工程出版社,2013:3.

[7] 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.

[8] 张慧芳. 基于动态反馈的加权最小连接数服务器负载均衡算法研究[D]. 上海:华东理工大学,2013.

[9] 章文嵩. Linux 服务器集群系统:四[EB/OL]. (2002-05-20)[2016-01-11]. http://www.ibm.com/developerworks/cn/linux/cluster/lvs/part4/index.html.

[10] WILLEBEEK-LEMAIR M H, REEVES A P. Strategies for dynamic load balancing on highly parallel computers[J]. Parallel and Distributed Systems, IEEE Transactions on, 1993, 4(9): 979-993.

[11] NICLAUSSE N. Tsung[EB/OL]. (2015-12-10)[2016-01-11]. http://tsung.erlang-projects.org/.

Research on Web Server Performance Optimization Based on Nginx

HUANG Jing, LI Bing

(School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou 310018, China)

Abstract: Dynamic load balancing algorithm can minimize response time or execution time of application services in parallel cluster, and it is essential for solving unpredictable load estimation problem in highly parallel system. Based on combining weighted least connection algorithm with upstream mechanism, a dynamic feedback load balancing algorithm was proposed to solve dynamic feedback problem of Nginx load balancing and improve the accuracy of load balancing. Stress test results show that the proposed algorithm can avoid the influence on traffic load overhead due to the introduction of dynamic load information collection improve the algorithm feasibility, reduce the average response time of web server, enhance the correct response rate, and provide a new solution for server performance optimization.

Key words: weighted least connection; upstream; Nginx; performance optimization; dynamic feedback

(责任编辑:陈和榜)