计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(4):138−144 [doi: 10.15888/j.cnki.csa.006284]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

# 基于权重轮询负载均衡算法的优化①

汪佳文, 王书培, 徐立波, 郭家军, 俞成海

(浙江理工大学 信息学院, 杭州 310018)
通讯作者: 汪佳文, E-mail: 2396351132@qq.com

**摘　要**: 随着电商网站用户规模不断增长, 高并发问题成为在搭建大规模电商网站系统时面临的一项重大挑战, 通过负载均衡算法来实现 Web 服务集群中各节点均衡负载是解决高并发的手段之一. 然而, 目前通用的负载均衡算法都存在一些不足之处, 针对这一问题, 提出了一种动态自适应权重轮询随机负载均衡算法 (Dynamic Adaptive Weight Round-Robin Random Load-Balancing, DAWRRRLB), 该算法考虑到影响 Web 服务集群中服务器节点性能的多重因素, 根据节点在运行过程中的实时负载情况动态的改变集群中节点的负载性能, 并结合改进的 Pick-K 算法对权重轮询负载均衡算法进行优化, 始终保证性能最优的服务器节点在提供服务. 通过多次实验对比, 改进的 DAWRRRLB 算法可以有效的提高负载均衡效率.

**关键词**: 高并发; Web 服务集群; 动态负载均衡; Pick-K 算法; 权重轮询

引用格式: 汪佳文,王书培,徐立波,郭家军,俞成海.基于权重轮询负载均衡算法的优化.计算机系统应用,2018,27(4):138–144. http://www.c-s-a.org.cn/1003-3254/6284.html

## Optimization of Load-Balancing Algorithm Based on Weight Round-Robin

WANG Jia-Wen, WANG Shu-Pei, XU Li-Bo, GUO Jia-Jun, YU Cheng-Hai

(School of Informatics and Electronics, Zhejiang Sci-Tech University, Hangzhou 310018, China)

**Abstract**: With the continuous growth of Internet users, the high concurrency becomes a major challenge in building large-scale electricity-business website system. To solve the problem, the load-balancing algorithm is used to realize the balanced load of each node in the Web service cluster. However, the current load-balancing algorithms generally have some shortcomings. In view of this problem, this paper proposes a dynamic adaptive weight round-robin random load-balancing algorithm (DAWRRRLB). This algorithm takes into account the multiple factors that affect the performance of the server nodes in the Web service cluster, and changes the load performance of nodes in the cluster according to the node in the operation process of the real-time dynamic load. It combines with the Pick-K algorithm to improve the dynamic adaptive weight round-robin random load-balancing algorithm, ensuring the best performance of the server node is providing services. By many experiments, the DAWRRRLB algorithm is proved to be able to effectively improve the load-balancing efficiency.

**Key words**: high concurrency; Web service cluster; dynamic load-balancing; Pick-K algorithm; weight round-robin

随着电子商务的高速发展, 网购已经变的越来越普遍, 用户规模也在不断增长, 因此许多大的电商网站系统应运而生, 伴随网站架构而来的是高并发[1]问题, 例如淘宝的“双十一”购物节, 并发量每秒高达几百万次, 而传统的单一系统模式并不能满足这种要求, 支撑庞大的高并发必须基于并行分布式技术[2]的服务集群.

① 基金项目: 浙江省公益技术研究工业项目 (2016C31072)
收稿时间: 2017-07-15; 修改时间: 2017-07-28; 采用时间: 2017-08-07; csa 在线出版时间: 2018-03-31

因此在面对大流量、高并发的冲击下, 如何更高效的展现出分布式服务集群中每台服务器的性能成为技术关键. 当前由于硬件性能发展的瓶颈, 无法满足人们对于高并发的要求, 软负载就成为人们取得突破的重点. 对于高并发高负载的大型网站架构实现软负载一般采用负载均衡技术[3]和缓存技术, 由于 Nginx[4]代理服务器采用了负载均衡技术和缓存技术并且作为一款轻量级的 Web 服务器广泛的受到网站开发者用户的喜爱, 而且 Nginx 的并发能力在同类型的 Web 服务集群中表现优良, 网站架构中采用 Nginx 作为代理服务器的比例也逐年上升. 目前国内外很多网站架构都采用 Nginx 作为负载均衡和反向代理[5]的 Web 服务器, 其中比较典型有淘宝、京东、新浪、网易等.

由于负载均衡算法在解决大型网站架构中高并发问题的重要性, 研究人员对于分布式计算系统中的负载均衡算法从动态和静态两个方面进行了深入的研究. 静态负载均衡算法是事先给定分配策略, 在以后的运行的过程中不再改变服务器节点负载性能, 一般的静态负载均衡算法主要有: 随机、权重轮询、一致 Hash 等. 但在实际情况下分布式服务集群中的服务器性能在不断消耗变化, 并且服务器节点数也在动态的变化之中 (服务器宕机或增加新的节点等), 所以预分配方案并不能实时保证服务集群处于最优的负载均衡状态. 因此动态负载均衡应运而生, 动态负载均衡算法最直观的是将客户请求实时分配给当前负载能力最优的服务器. 国内外目前有很多学者在负载均衡算法也有过研究, 并且提出了很多改进之处. 比如国外的 Singh 等[6]研究发现了一种基于动态调度的负载平衡 Web 服务器系统, Kaur 等[7]研究发现了一种通过虚拟机用蚁群算法实现负载平衡的混合方法, Ding 等[8]研究发现了一种在异构网络中的动态负载平衡算法, 张尧等[9]研究发现了一种改进的基于权值动态负载均衡算法, Li 等[10]研究发现了一种基于动态时间步长反馈的负载均衡方法, 这些算法都在一定程度上提升了集群的负载性能, 然而其中基于权重轮询算法在负载均衡算法是比较常见的. 由于现在一般的分布式服务集群都是基于异构网络[11], 影响服务器节点性能的因素比较多, 不能只从单一方面考虑, 需要综合去考虑服务集中影响节点性能的各个因素, 根据节点的真实负载情况去动态的调整节点的性能.

基于以上原因本文对已有的权重轮询负载均衡算法进行了优化, 提出了一种动态自适应权重轮询随机负载均衡算法 (DAWRRRLB). 该算法主要考虑到影响节点的多重因素, 根据服务集群中节点的实时负载性能动态的、自适应的去调整当前节点的负载性能, 并且引进了改进的 Pick-K 算法, 确保始终是最优的服务器在提供服务, 同时避免一些节点由于接收到大量的请求或者因为某个进程过于庞大而出现过载的情况. 而且在在集群系统中要想快速响应客户端的请求, 只有始终保持是负载能力最优的服务器在提供服务, 才能使整个集群负载处于最优状态, 从而提升负载均衡效率.

## 1 基于权重的负载均衡算法分析

### 1.1 加权最小连接算法

加权最小连接负载均衡[12]算法 (WLC) 是根据当前节点连接数与当前节点权重比来实现负载均衡, 将请求分配给比值最小的节点, 并且连接数加 1. 即:

$$P_i = \min\left(\frac{C(N_i)}{W(N_i)}\right) \tag{1}$$

其中, $C(N_i)$ 是表示该节点的链接总数, $W(N_i)$ 表示当前节点的权重值, $P_i$ 表示响应请求的服务器节点.

缺点: 由于集群中各节点的异构, 连接数和权重比很难真实反映各节点的负载真实情况, 并且节点的负载性能在运行的过程中是不断变化的, 且权重需要根据管理员经验去设置, 不能动态自适应去调整节点, 难以准确确定.

### 1.2 加权轮询算法

加权轮询[13]算法 (WRR) 适用性较强, 不依赖于客户端的任何相关信息, 仅仅依赖后端服务器节点的负载情况, 根据后端服务器节点的权重比例轮流分配客户端的请求. 该算法比较简洁, 不需要记录当前客户端的所有连接, 是一种无状态的调度.

缺点: WRR 没有考虑到集群中每台服务器的当前连接数以及响应速度, 而且在集群运行过程中, 服务器节点的性能也是在不断消耗变化, 不能及时对服务器端节点的动态负载做出准确的判断, 当节点负载不均时, 极容易出现个别服务器负载过高等, 从而影响集群的整体负载性能. 也有可能因为某个连接请求响应时延比较长, 严重影响该节点对其他的连接请求响应, 导致集群性能下降.

## 2 动态自适应权重轮询随机负载均衡算法(DAWRRRLB)实现

本文提出的动态负载均衡算法是实时获取影响服务器性能的参数, 根据各节点性能参数的使用率实时的、动态的改变节点的负载性能. 因在实现动态负载均衡的过程中需要收集各个服务器的当前负载信息需要一定的额外开销, 为了避免过于频繁的去收集各个服务器的性能参数而导致一定的时延, 通常是周期性的去收集当前负载信息以尽可能的降低这些额外的开销. 并且如果收集的信息不及时或误差比较大在一定程度上也会影响负载均衡, 为了尽可能降低此类情况的发生, 本文均采用平均值的思想来减少这种误差.

### 2.1 服务器负载均衡影响参数

由于 Web 服务器的性能受到服务器节点 KPI 数据的影响, 所以可以根据这些影响因素变化的情况, 动态的修改服务器的权重. 本文主要根据节点的 CPU、内存、磁盘 IO 以及网络带宽的使用情况对节点性能动态的进行调整. 又因为 Nginx 是根据集群中各节点的权重比, 权重越大分配的请求连接数越多, 权重越小分配的客户端请求连接数就越少, 因此可动态修改服务器的权重大小来实现动态负载均衡.

现假设在周期时间 $T$ 内单个节点 CPU 的使用率为 $C(N_i)$, 内存的使用率为 $M(Ni)$, 磁盘 IO 的使用率为 $D(Ni)$, 网络带宽的使用率为 $B(Ni)$. 用 $S_c(\text{Total})$、$S_m(\text{Total})$、$S_d(\text{Total})$、$S_b(\text{Total})$ 分别表示集群中 CPU、内存、磁盘 IO、网络带宽的性能总和.

$$S_c(\text{Total})=\sum_{i=1}^{n}C(N_i) \tag{2}$$

$$S_m(\text{Total})=\sum_{i=1}^{n}M(N_i) \tag{3}$$

$$S_d(\text{Total})=\sum_{i=1}^{n}D(N_i) \tag{4}$$

$$S_b(\text{Total})=\sum_{i=1}^{n}B(N_i) \tag{5}$$

又因为各个因素对服务器性能依赖的程度不一样, 现假设 CPU、内存、磁盘 IO、网络带宽的依赖比重分别为 $K_c$、$K_m$、$K_d$、$K_b$, 并且

$$K_c+K_m+K_d+K_b=1 \tag{6}$$

集群中每个服务器节点的性能真实比重为:

$$\begin{aligned}W_p(N_i)=&K_c*(C(N_i)/S_c(\text{Total}))+\\&K_m*(M(N_i)/S_m(\text{Total}))+\\&K_d*(D(N_i)/S_d(\text{Total}))+\\&K_b*(B(N_i)/S_b(\text{Total}))\end{aligned} \tag{7}$$

其中, $W_p(N_i)$ 表示服务集群中每个节点性能的真实比重.

由于服务集群中每个服务器的各个性能参数指标不一样, 因此集群中每个服务器节点的性能比重也不一样.

### 2.2 负载均衡权重

在实际运行的过程中, 集群中各节点性能是在不断变化, 权值也在动态的改变, 因此每个节点在集群中性能比也在不断变化. 现假设 $W(N_i)$ 表示第 $i$, $i\in\{1,\cdots,n\}$ 个服务器节点的权重, 则每个节点在集群中权重比例 (即当前节点在集群中的负载均衡性能) 为:

$$E_w(N_i)=W(N_i)/\sum_{i=1}^{n}W(N_i) \tag{8}$$

其中, $E_w(N_i)$ 表示第 $i$ 个节点的权重比例, 其中 $i\in\{1,\cdots,n\}$.

### 2.3 响应时延

网站的性能优劣一般通过响应时延[14]来衡量, 平均响应时延越短, 则表明服务器性能越好. 假设第 $i$ 个节点的平均响应时间为 $T(N_i)$, 假设集群的每个服务器节点平均响应时间 (其中每个节点平均响应时间为该节点在周期 $T$ 内 $n$ 次响应的平均时延) 为:

$$T_i=[T(N_1),\cdots,T(N_i),\cdots,T(N_n)]^{\mathrm{T}} \tag{9}$$

在周期 $T$ 内整个集群的平均响应时延为:

$$\overline{T_n}=\frac{1}{n}\sum_{i=1}^{n}T(N_i) \tag{10}$$

其中, $T(N_i)$ 表示第 $i$ 个服务器节点的平均响应时间, $\overline{T_n}$ 表示整个集群平均响应时延.

### 2.4 动态利用率

在实际运行的过程中需要根据各节点的性能参数实时计算各节点的负载性能, 因此需要获取在运行过程中影响各节点性能因素的真实使用率. 设 $U_c(N_i)$、$U_m(N_i)$、$U_d(N_i)$、$U_b(N_i)$ 分别表示第 $i$ 个服务器节点的 CPU、内存、磁盘 IO、网络带宽的实时使用率. 则第 $i$ 个服务器节点实时性能为:

$$U(N_i)=K_c*U_c(N_i)+K_m*U_m(N_i)+K_d*U_d(N_i)+K_b*U_b(N_i) \quad (11)$$

其中, $U(N_i)$ 表示第 $i$ 个服务器节点实时性能利用率.

### 2.5 集群中服务器节点性能模型的建立与分析

因为在周期时间 $T$ 内, 每个节点由于权重的不同获取到的请求连接数也不同, 权重越大, 说明请求到该节点的连接数越多, 服务器节点使用率也就越大. 所以根据单个连接请求在集群中某个服务器节点所占平均性能比来反映当前节点的负载均衡能力. 为了更好的反映集群中节点的实时性能, 另引入单个节点平均响应时延与整个集群的平均响应时延的比来反映节点的负载均衡能力, 因为响应时延最能直观的反应服务器节点的负载性能, 可以避免由于某个连接请求进程过大导致当前节点成为超载节点. 根据上述分析以及反映 Web 集群负载均衡的条件建立以下分析模型:

$$P(N_i)=\frac{W_p(N_i)*U(N_i)}{E_w(N_i)}*\frac{T(N_i)}{\overline{T_n}} \quad (12)$$

其中, $P(N_i)$ 表示节点 $i$ 实时负载均衡情况, $T(N_i)$ 为单个服务器平均响应时延, $\overline{T_n}$ 为整个集群平均响应时延. $E_w(N_i)$ 为第 $i$ 个节点的在集群中负载比重, $W_p(N_i)$ 表示第 $i$ 个节点的性能比重, $U(N_i)$ 为第 $i$ 个节点的性能实时利用率.

由于单位时间内请求服务器的连接数是随机的, 因此在这里引入泊松分布, 即根据客户端随机请求连接数来确定所需要的服务器个数, 通过 Pick-K 算法[15] 在 Web 服务集群中筛选所需服务器. Pick-K 算法的原理是: 在更新周期 $T$ 内, 当有任务到来时, 从 $n$ 台成员服务器中选择 $k$ 个服务器, 比较每台服务器的负载情况, 从性能最优的服务器依次选择, 将任务发送给它们. 但在实际运行过程中节点的性能是不断变化的, 当某些节点由于运行过程. 本文对该算法进行了一些改进, 在 Pick-K 算法的基础上加入了一些约束性, 引入改进的 Pick-K 算法的目的是确保集群中性能优的服务器始终提供服务, 同时避免由于某个访问请求进程过大而阻塞其它请求访问的情况. 改进的 Pick-K 算法基本原理如下: 现假设有 $n$ 台服务器, 从 $n$ 台服务器中选择 $k$ 台, 总共有 $C_n^k$ 种情况, 选出其中性能变化最平稳 $k$ 台服务器提供服务, 并通过引入标准差来衡量集群负载变化情况, 当选中的 $k$ 台服务器节点的负载性能标准差高于某个阈值 (在上一个周期所求得的标准差), 则动态的调整集群中相应服务器节点的负载性能. 由公式 (12) 可知 $P(N_i)$ 表示第 $i$ 个节点当前负载情况, 现假设 $P$ 表示集群中所有节点实时负载性能的向量集合, $P_d$ 表示集群中 $k$ 个服务器节点性能变化情况, 则:

$$P=[P(N_i),\cdots,P(N_k),\cdots,P(N_n)] \quad (13)$$

$$P_d=\sqrt{\frac{(P(N_1)-\overline{P})^2+\cdots+(P(N_i)-\overline{P})^2+\cdots+(P(N_k)-\overline{P})^2}{k}} \quad (14)$$

$$\overline{P}=\frac{1}{K}\sum_{i=1}^{k}P(N_i) \quad (15)$$

其中, $P$ 表示集群中所有节点实时负载性能的向量集合, $P_d$ 表示集群中参与负载的服务器节点性能变化情况, $\overline{p}$ 表示选中的 $k$ 个服务器实时负载性能的平均值.

分析: 从公式 (14) 可以看出, 当前标准差 $P_d$ 反映的是集群中参与了负载均衡相关节点的负载性能的离散程度, 当标准差越小则说明 Web 服务集群负载越均衡, 反之负载越不均衡. 根据引入的 Pick-K 算法计算得出在集群中筛选参与负载均衡所需服务器个数时总共有 $C_n^k$ 种情况, 其中标准差最小的即为满足条件的 $k$ 台服务器, 现假设 $P_d(\min)$ 表示所求最小标准差, $D_p$ 表示给定的阈值. 则:

$$P_d(\min)=[P_d(1),P_d(2),P_d(3),P_d(4),\cdots] \quad (16)$$

下面通过对比最小标准差与阈值来判断是否需要动态调整集群中节点的性能:

(1) 如果 $P_d(\min)<D_p$, 则此时直接选择标准差最小情况下筛选出来的 $k$ 台服务器即可;

(2) 如果 $P_d(\min)>D_p$, 则对集群服务器节点的性能进行调整, 通过修改权重. 修改的策略如下: 求出集合 $P$ 中离散程度大的节点, 然后对离散程度大的节点在集群中的负载权重进行动态调整, 具体调整策略根据公式 (12):

1) 当 $P(N_i)$ 大于某个给定的值 (上一次更新周期 $T$ 所求的平均值 $\overline{p}$ ), 则说明该节点负载过重, 如果权重比较大, 可能是由于服务器节点的实时使用率过大, 可以减小当前节点的权重. 如果当前权重比较小, 响应时延比较长, 此时有两种情况: 一是当前节点负载能力差; 二是因为某些访问请求进程过大而阻塞其它请求访问, 从而导致平均时延变长. 以上两种情况都应当减小当

前节点的负载权重, 以此来减少请求到当前节点的访问连接数.

2) 当 $P(N_i)$ 小于给定的值 (上一次更新周期 $T$ 所求的平均值 $\overline{p}$ ) 说明服务器实时使用率比较低, 此时可以动态增加相应的权重, 提高其负载均衡能力.

3) 当 $P(N_i)$ 过大 (即在周期 $T$ 内请求该节点绝大多数请求超时), 大于某个给定的值 (初始给定), 则认为该服务器节点宕机不参与集群负载均衡, 将该服务器节点置为不可用.

根据上诉分析可知该算法考虑到了反映服务器节点负载性能的双重因素 (响应时间比和性能参数使用率), 通过动态的调整节点在集群中负载均衡的权重来使集群达到均衡负载. 并且该算法确保始终是负载均衡能力最好的服务器节点在提供服务, 当服务器节点宕机时也可以减少请求失败的次数, 避免了由于请求宕机的服务器节点而长时间等待, 降低了平均响应时延. 而且不必频繁的去修改权重, 只有当不满足一定的条件时才会修改权重, 同时也可以避免由于某个访问请求进程过大导致该节点成为超载节点.

## 3 实验

本实验采用 Nginx 搭建 Web 服务集群[16](如图 1 所示), 并使用成熟的 Web 测试工具 Httperf+Autobench 对 Web 服务集群进行性能测试, 该实验采用的是 Http 请求方式, 实验环境配置如表 1, 其中一台服务器用作负载均衡服务器, 另五台为后端服务器. 在搭建 Nginx 服务器时引入了 Nginx+ngx_lua 模块[17], 并且利用 Lua 脚本对 nginx.conf 配置文件进行修改, 动态改变集群中服务器节点的负载权重. 由于影响服务器节点的各个因素对服务器性能依赖的程度不一样, 本次实验主要依赖 CPU 和内存, 令 CPU、内存、磁盘 IO 以及网络带宽的依赖比重分别为: 0.4、0.4、0.1、0.1, $K$=[0.4, 0.4, 0.1, 0.1]. 通过调用 Httperf 执行 Autobench 脚本, 设置并发连接数增长, 本次测试通过每次增加 200(其中包含 20 庞大进程并发数) 并发连接数进行实验, 并根据并发响应时延对加权连接最小数、内置加权轮询算法、和本论文改进的 DAWRRRLB 三种算法的负载均衡性能进行分析.

表 1 实验环境配置

| IP | 处理器 | 内存 | 系统 | 用途 |
|---|---|---|---|---|
| 192.168.0.113 | 双核 | 2 G | Centos6.5 | 负载均衡服务器 |
| 192.168.0.116 | 双核 | 2 G | Centos6.5 | 后端服务器 |
| 192.168.0.105 | 双核 | 4 G | Centos6.5 | 后端服务器 |
| 192.168.0.101 | 四核 | 2 G | Centos6.5 | 后端服务器 |
| 192.168.0.107 | 四核 | 4 G | Centos6.5 | 后端服务器 |
| 192.168.0.98 | 四核 | 4 G | Centos6.5 | 后端服务器 |

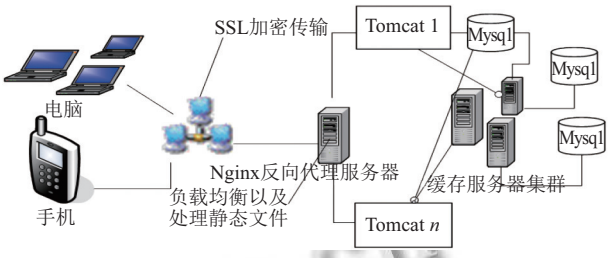


图 1 测试环境 Web 服务集群架构示意图

实验一. 假设在运行的过程中集群中没有服务器节点宕机或断网等特殊情况. 根据实验一得到表 2.

表 2 三种算法的平均响应时延如下

| QPS(并发连接数) | WRR(加权轮询) | | WLC(加权最小连接数) | | DAWRRRLB(动态自适应权重轮询随机负载均衡算法) | |
|---|---|---|---|---|---|---|
| | 响应时延(ms) | QPS(成功响应连接数) | 响应时延(ms) | QPS(成功响应连接数) | 响应时延(ms) | QPS(成功响应连接数) |
| 100 | 3.62 | 99 | 2.59 | 99 | 2.68 | 100 |
| 300 | 3.74 | 297 | 2.63 | 298 | 2.66 | 298 |
| 500 | 4.79 | 489 | 3.27 | 487 | 3.31 | 491 |
| 700 | 19.87 | 686 | 7.34 | 684 | 7.68 | 687 |
| 900 | 48.61 | 881 | 12.4 | 879 | 12.1 | 878 |
| 1 100 | 188.7 | 1074 | 166.5 | 1076 | 159.6 | 1065 |
| 1 300 | 358.32 | 1249 | 346.3 | 1253 | 287.4 | 1259 |

分析: 根据表 2 得到的平均响应时延并结合图 2 可以看出当并发量比较低的时候三者的平均时延相差不多, 此时对服务器的性能影响比较小, 3 种情况下服务器都能快速响应客户的请求. 当并发量逐渐变大并大于 500 的时候, 加权最小连接数和动态自适应权重轮询随机负载均衡算法负载均衡性能明显高于加权

轮询算法, 此时服务器节点的性能影响比较大, 后两种情况是根据服务器的实时性能动态调整服务器节点的权重实现负载均衡, 动态的去调整始终保持性能优的节点响应更多请求. 当并发量继续增大并大于 1100 时, 此时对服务器节点各个性能参数要求比较高, 因此加权最小连接数响应时延要大于改进动态自适应算法 (DAWRRRLB) 的响应时延, 因为改进的动态自适应算法 (DAWRRRLB) 考虑到各个服务器节点的实时性能参数利用率, 是根据集群中服务器节点实时性能进行负载分配的. 由此可以看出当节点不发生特殊情况 (宕机或断电) 下符合我们的预期, 通过平均响应时延计算 WLC 算法比 WRR 算法性能性能提高了 14.99%, DAWRRRLB 负载均衡算法性能比 WLC 算法性能提高了 10.61%.

实验二. 模拟集群在运行过程中部分服务器节点存在短暂的宕机或断网不可用的情况. 本实验当并发用户数为 700 的时候, 让集群中部分服务器断网或宕机不可用后又恢复.

根据实验二得到表 3. 从表 3 实验数据和图 3 可以看出满足实验一的结果. 当出现宕机情况下改进的动态自适应负载均衡算法要优于其他两种情况. 因为当出现宕机不可用时在下一个更新周期 T 内, 不可用的两台服务器节点不参加负载均衡, 从实验数据可以看出提高了成功响应连接数, 并且也有效降低了整个集群的平均响应时延, 符合预期的结果. 通过响应时延计算 WLC 算法比 WRR 算法性能提高了 14.98%, DAWRRRLB 算法负载性能比 WLC 算法负载性能性能提高了 15.8%. 并且通过实验一和实验二数据对比, 优化后的 DAWRRRLB 算法在 Web 服务集群环境不稳定 (集群中服务器节点容易宕机或断网) 的情况下负载性能比其它算法更加优越.

表 3 三种算法的平均响应时延

| QPS(并发连接数) | WRR(加权轮询) | | WLC(加权最小连接数) | | DAWRRRLB(动态自适应权重轮询随机负载均衡算法) | |
|---|---|---|---|---|---|---|
| | 响应时延 (ms) | QPS（成功响应连接数） | 响应时延 (ms) | QPS(成功响应连接数) | 响应时延(ms) | QPS(成功响应连接数) |
| 100 | 3.61 | 99 | 2.57 | 98 | 2.69 | 99 |
| 300 | 3.75 | 295 | 2.64 | 298 | 2.68 | 297 |
| 500 | 4.78 | 483 | 3.26 | 484 | 3.33 | 487 |
| 700 | 19.89 | 582 | 12.34 | 583 | 8.69 | 632 |
| 900 | 98.61 | 743 | 68.1 | 746 | 42.4 | 791 |
| 1 100 | 247.7 | 926 | 196.5 | 931 | 179.6 | 987 |
| 1 300 | 405.47 | 1058 | 396.3 | 1061 | 334.6 | 1109 |

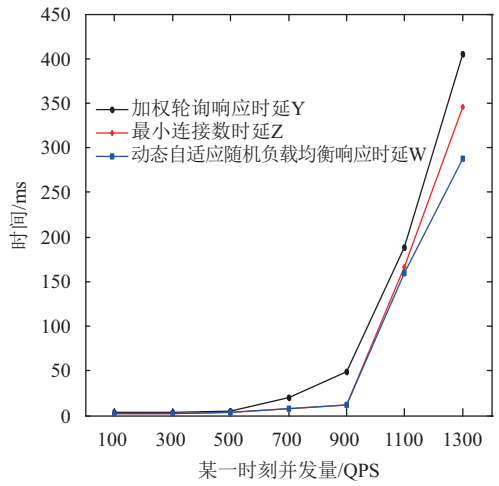


图 2 响应时延 (纵坐标) 和某一时刻并发量 (横坐标) 的关系折线

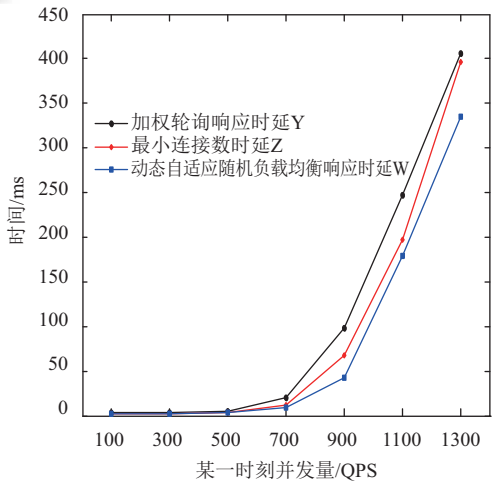


图 3 响应时延 (纵坐标) 和某一时刻并发量 (横坐标) 的关系折线

## 4 总结

本文提出的动态负载均衡算法 (DAWRRRLB), 能够根据服务集群中各个服务器节点的实时性能去动态改变服务器节点负载能力, 使集群中每个节点能够实时处于最优负载状态, 始终保证性能最优的服务器节点提供服务, 而且不必频繁的去修改权重, 只有当不满足一定的条件时才会修改权重, 同时也可以避免由于某个连接请求进程过大导致该节点成为超载节点, 也降低了某些节点由于接收到大量请求出现过载的概率. 并采用 Nginx 作为负载均衡和反向代理服务器, 通过多次实验数据对比, 负载均衡性能得到了较大的提升. 但是此算法也存在一定的缺陷, 比如更新周期时间 $T$, 如果周期 $T$ 太大时, 各服务器节点的负载情况变化太快, 不能实时反映节点的性能; 反之, 如果周期 $T$ 太小则需要过于频繁去获取服务器节点性能参数, 反而会降低服务器节点的性能.

## 参考文献


1 Huang JW, Wang JX, Zhang T, *et al*. Tuning the aggressive TCP behavior for highly concurrent HTTP connections in data center. Proceeedings of the 36th International Conference on Distributed Computing. Nara, Japan. 2016. 98–107.

2 柳伟卫. 分布式系统常用技术及案例分析. 北京: 电子工业出版社, 2017: 2.

3 Wang YZ, Jiang JR, Ye H, *et al*. A distributed load balancing algorithm for climate big data processing over a multi-core CPU cluster. Concurrency and Computation Practice and Experience, 2016, 28(15): 4144–4160. [doi: 10.1002/cpe.v28.v28.15]

4 苗泽. Nginx 高性能 Web 服务器详解. 北京: 电子工业出版社, 2013: 10.

5 Lin H. Research on Nginx reverse proxy visual management scheme and the programming realization. Journal of Guizhou Normal College, 2015, 31(12): 32–35.

6 Singh H, Kumar S. Dispatcher based dynamic load balancing on web server system. International Journal of Grid & Distributed Computing, 2011, 4(3): 89–106.

7 Kaur K, Kaur A. A hybrid approach of load balancing through VMs using ACO, MinMax and genetic algorithm. Proceedings of the 2nd International Conference on Next Generation Computing Technologies (NGCT). Dehradun, India. 2016. 615–620.

8 Ding ZX, Wang XJ, Yang WM. A dynamic load balancing algorithm in heterogeneous network. Proceedings of the 7th International Conference on Intelligent Systems, Modelling and Simulation. Bangkok, Thailand. 2016. 337–342.

9 张尧. 基于 Nginx 高并发 Web 服务器的改进与实现[硕士学位论文]. 长春: 吉林大学, 2016: 5.

10 Li HJ, Zhao YY, Zhu GF, *et al*. A load balancing method based on dynamic time step feedback. Proceedings of the 2nd IEEE International Conference on Computer and Communications (ICCC). Chengdu, China. 2016. 2911–2916.

11 Gu Y, Wu CQ. Performance analysis and optimization of distributed workflows in heterogeneous network environments. IEEE Transactions on Computers, 2016, 65(4): 1266–1282. [doi: 10.1109/TC.2013.62]

12 高振斌, 潘亚辰, 华中, 等. 改进的基于加权最小连接数的负载均衡算法. 科学技术与工程, 2016, 16(6): 81–85.

13 Li T, Baumberger D, Hahn S. Efficient and scalable multiprocessor fair scheduling using distributed weighted round-robin. Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. Raleigh, NC, USA. 2009. 65–74.

14 Elshater Y, Martin P, Hassanein E. Using design patterns to improve web service performance. Proceedings of 2015 IEEE International Conference on Services Computing. New York, NY, USA. 2015. 746–749.

15 王霜, 修保新, 肖卫东. Web 服务器集群的负载均衡算法研究. 计算机工程与应用, 2004, 40(25): 78–80. [doi: 10.3321/j.issn:1002-8331.2004.25.024]

16 张开涛. 亿级流量网站架构核心技术. 北京: 电子工业出版社, 2017.

17 陶辉. 深入理解 Nginx: 模块开发与架构解析. 2 版. 北京: 机械工业出版社, 2016.