Software Engineering and Applications 软件工程与应用, 2020, 9(4), 262-271
Published Online August 2020 in Hans. http://www.hanspub.org/journal/sea
https://doi.org/10.12677/sea.2020.94029

Hans汉斯

Research and Application of Dynamic Load Balancing in MQTT Server

Jiayu Li

School of Information, Zhejiang Sci-Tech University, Hangzhou Zhejiang
Email: 365978242@qq.com

Received: Jul. 8th, 2020; accepted: Jul. 21st, 2020; published: Jul. 28th, 2020

Abstract

The unique advantage of MQTT is widely used in the Internet of things technology, but when the server processes a large number of requests, MQTT server may have downtime and other problems. This paper starts with optimizing the load balance, analyzes the proportion coefficient of message queue, CPU utilization and memory utilization, and uses the new weight calculation method, making the system load more evenly distributed to the cluster servers, and the overall average response time is shorter. The experimental data show that the load difference can be reduced from the original 8% to about 4%. It improves the efficiency of MQTT server cluster system and has a certain practical value.

Keywords

MQTT Protocol, Load Balancing, Improved Algorithm, Weight Modification, Nginx Technology

MQTT服务器动态负载均衡的研究与应用

李嘉钰

浙江理工大学信息学院，浙江 杭州
Email: 365978242@qq.com

收稿日期：2020年7月8日；录用日期：2020年7月21日；发布日期：2020年7月28日

摘 要

MQTT其独特的优点在物联网技术得到广泛应用，但服务器处理大量的请求时，MQTT服务器可能会出现宕机等问题，本文从优化负载均衡入手，对消息队列、CPU利用率、内存利用率的比重系数进行分析，利用新的权值计算方法，使系统负载更加均匀地分发至集群各服务器，总体平均响应时间更短，通过实

文章引用: 李嘉钰. MQTT 服务器动态负载均衡的研究与应用[J]. 软件工程与应用, 2020, 9(4): 262-271.
DOI: 10.12677/sea.2020.94029

验数据表明负载差能由原来的8%下降到4%左右，提高了MQTT服务器集群系统的使用效率，具有一定的实用价值。

关键词

MQTT协议，负载均衡，改进算法，权值修改，Nginx技术

1. 引言

随着越来越多的工业设备接入物联网，因处理器能力、网络带宽等具有局限性，所以对其通信技术提出了更高的要求[1]。作为网页标准的 HTTP，已不能满足机器之间的大规模沟通，其请求/回答模式不再合适，取而代之的是发布/订阅模式[1]。这种模式就是轻量级、可扩展的 MQTT 可以施展拳脚的舞台，MQTT 技术在汽车软件、智能家居、智慧医疗等方面有着广泛的应用。近来 MQTT 的优势被深度发掘，当有大量的 MQTT 协议的用户数据和设备实时数据需要存储在服务器端，会使得设备与服务器的交互相当频繁，当服务器接收到大量的数据请求时，MQTT 服务器可能会出现宕机等现象。因此，MQTT 服务器的负载均衡问题变得突出。传统的单台 MQTT 服务器处理能力已无法适应现代物联网大量数据请求[1] [2]，所以需要部署服务器集群系统。对不同性能的 MQTT 服务器，如何给其分配最适合其性能的任务量，协同处理和合理分配外部请求等需要一个负载分配平衡系统。由此可见引入 MQTT 服务器的负载均衡显得尤为重要[3]。

2. 概述

本文的方法在兼顾 MQTT 服务器成本的同时解决了 MQTT 服务器系统稳定性问题。利用动态负载数据平衡分配策略[4]，对消息队列、CPU 利用率、内存利用率的比重系数进行分析，用权值修改函数 update Weight()，对权值修改模块优化更新，把用户请求均衡地重定向到其它站点，让系统在不增加硬件投入的情况下发挥最大的效能。当出现高并发量但单台 MQTT 服务器无法满足需求时，传统方法需要高性能的服务器才可以获得额外的服务资源[1] [5]。本文的方法是通过增加 MQTT 服务器的数量并增加 MQTT 服务器均衡算法就能解决问题。当一开始不知道到底需要多高性能的服务器才能够处理所有需要挂载在其下的设备的请求，每当有设备量增加时，本文通过计算与测试，准确的添加所需要的最高性价比的服务器。

3. 负载均衡器和消息队列

3.1. Nginx 负载均衡器

本文通过对服务器连接算法进行优化，推出一种新的 MQTT 服务器动态负载均衡算法。新算法通过提取当前 MQTT 服务器的相关运行性能及状态参数，改变其权值计算公式计算出为对应的权值，能够实时根据后端服务器的运行状态和负载状况，实时优化用户数据的重新定向[6]。各 MQTT 服务器发送权值至 Nginx 负载均衡器后，负载均衡模块中写入新权值，当有新的用户请求任务时，负载均衡器会根据各服务器的写入的新权值比例关系进行分发。动态负载均衡算法见图 1 所示：

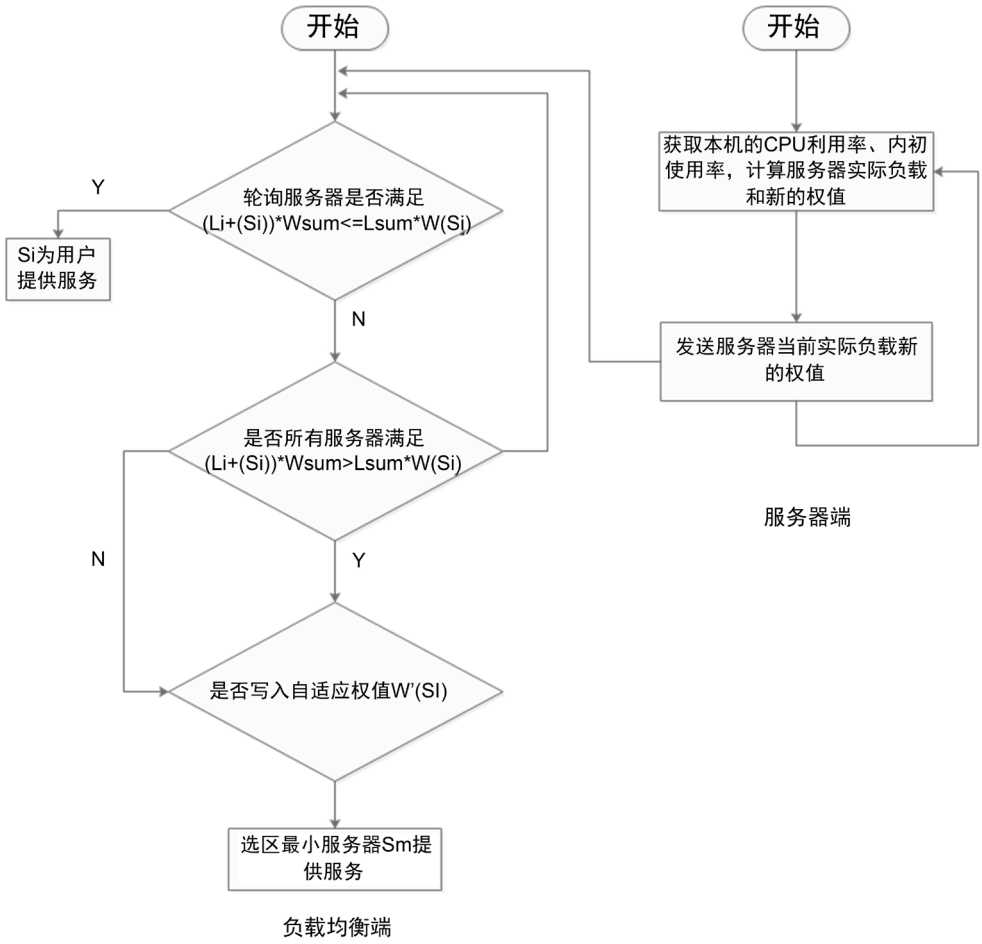

Figure 1. Dynamic load balancing algorithm
图 1. 动态负载均衡算法

3.2. 权值修改模块

权值修改模块运行在集群系统前端的 Nginx 负载均衡器之上，权值修改模块被嵌入到 Nginx 中[5]，为了能够较好的分发用户连接请求，与 MQTT 各服务器集群进行通信[5]，该模块需实现接收后台各服务器发送的权值，对此前保存的权值进行更新，此算法与的静态权值相比，更能够实时并精确反映各服务器真实的负载情况[5]；根据各服务器权值间比例关系将用户连接请求分发至合理的服务器。

Nginx 负载均衡器需与下游客户端基于 MQTT 协议进行通信，但它与上游后端服务器的通讯是基于 TCP 的事件驱动架构的[5]。Nginx 负载均衡器向后端服务器发送用户请求时，需要 stream 模块中的具体负载均衡算法的配合，在后端服务器集群中选择出一台合适的服务器用来处理用户请求[5] [7]。

对后端服务器动态权值的实时获取并更新，主要用到权值修改函数 updateWeight()，在修改权值之前需对包含权值信息的 peers 指针加锁，完成修改后再解锁[8]。

当负载均衡器分发用户请求时，会调用后端各台服务器的函数，重新对后端服务器进行选择。当负载均衡器启动以后，就绑定 Socket 通信端口建立 UDP 服务，对指定端口进行监听，同时用 recv()函数接收后端服务器发送的权值信息。调用 fork()函数创建子进程，建立并发服务器，以接收到的权值更新原来保存在后端的服务器权值信息，各服务器的权值是作为分发用户连接请求的依据。负载均衡器其工作的流程见图 2 所示：

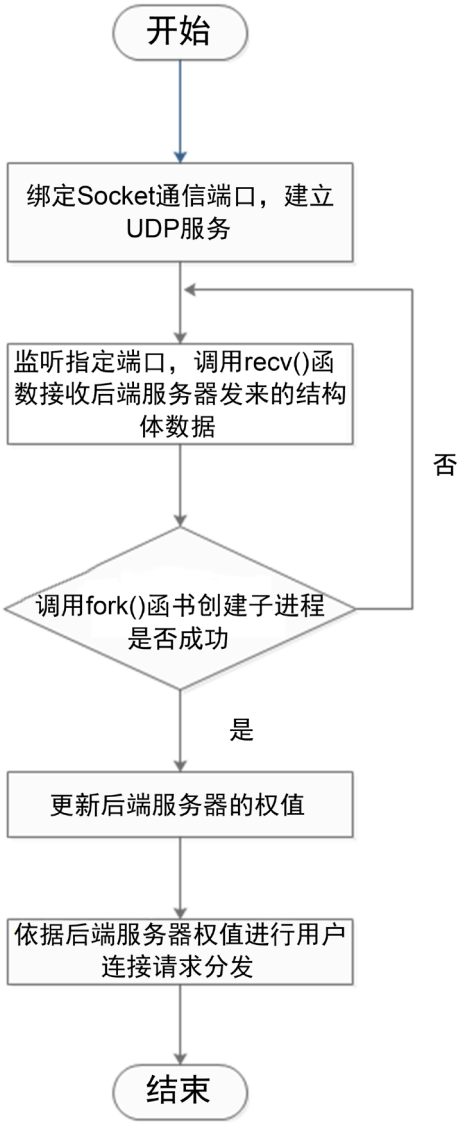

Figure 2. Work flow chart of load balancer
图 2. 负载均衡器的工作流程图

4. 主要设计

4.1. MQTT 服务器权值的算法设计

在改进的负载均衡算法中，MQTT 服务器集群中的各台服务器周期性的收集自身负载信息和连接数，得出权值后发送给 Nginx 负载均衡器，根据各服务器的权值和负载通过筛选[9]，计算出合理的服务器去应答新的用户数据连接请求。此新算法主要包含计算 MQTT 服务器的权值，以及服务器集群周期性计算与上传权值。

当前 MQTT 服务器加权最小连接算法，无法对服务器实际负载进行合理的分配，且权值是由人为凭经验设置，在负载分发时固定不会变，在本文提出的改进算法中，各 MQTT 服务器节点根据自身 CPU 利用率和内存使用率两个性能指标，从而反映出了服务器的实际负载情况，再结合当前用户连接数计算得出综合负载，最后根据综合负载得出各个服务器节点的权值。在改进的负载均衡算法中，为了防止频

繁的计算和发送权值会影响 MQTT 服务器和负载均衡器的性能，各 MQTT 服务器需周期性的去计算自身权值，计算得出权值与上一周期的权值进行比较，若权值变化率不大于 Δt 的权值将不会发送至负载均衡器。选取 T 和 Δt 的值需经过实验验证得到。改进算法工作模型见图 3 所示：

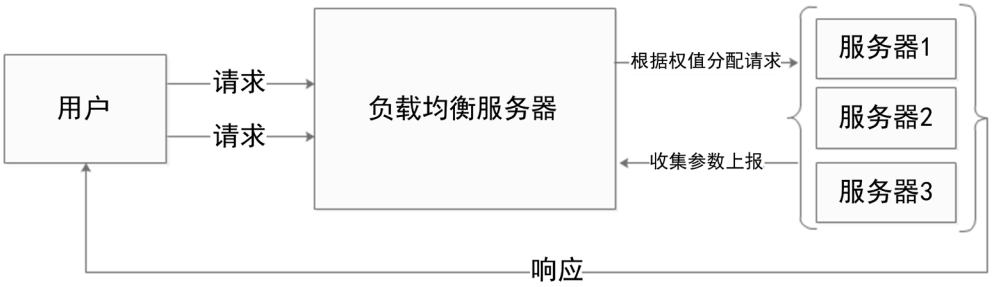

Figure 3. Working model of improved algorithm
图 3. 改进算法工作模型

4.2. 权值计算

权值计算公式如下：

$$W_i = W_i + A * \sqrt[3]{SU - AL(S_i)}$$

其中 SU 是系统利用率，A 为可动态调整的系数(不为 0)，$AL(S_i)$是服务器的综合负载值[10]。

选择两类指标作为确定负载情况的因素[11]，负载计算公式如下：

$$load_i = R_1 * S_i + R_2 * D_i + R_3 * C_i + R_4 * M_i + R_5 * P_i + R_6 * T_i$$

S_i 为连接数指标，表示单个服务器的连接数与平均值的比值。D_i 为磁盘利用率，C_i 为 CPU 利用率，M_i 为内存利用率，P_i 为工作进程数，T_i 为响应时间。R 表示各负载参数重要程度的一组系数。

从服务器的实时性能参数以及状态参数两类参数。服务器实时性能值计算如下：

$$F(S_i) = [R_1, R_2] * \begin{bmatrix} 1 - CPU(S_i) \\ 1 - MEMORY(S_i) \end{bmatrix}$$

R_1, R_2 为 CPU 利用率，内存利用率的比重系数，且 $R_1 + R_2 = 1$。$CPU(S_i)$ 和 $MEMORY(S_i)$ 分别表示服务器 S_i 的 CPU 利用率，内存利用率。其中 CPU 利用率更能反映当前节点的实时性能，从而取 R_1, R_2 的值分别为 0.6 和 0.4 [12] [13]。

4.3. 权值上报设计

当前负载均衡调度分为集中式调度和分布式调度两种，这两种调度方式都需要一个负载均衡器来作为核心节点进行任务分配[9]。随着服务器集群节点的增加，会增加核心节点的参数收集和计算工作，负载分发及系统效率会大幅下降。新的权值计算实时采集各服务器权值并按设定的周期发至负载均衡器，经过计算后得出新的任务分配列表，新的分布式权值调度方式可以减少负载均衡器和各服务器之间的通信次数及核心节点的工作量，负载均衡策略采用分布式调度方式。在数据请求较多时，如果大量上传这类权值也会增加核心节点不必要的工作量，减少这类权值的发送可以提升核心节点负载分发效率，可以利用集群中各服务器在周期 T 内的变化程度，来决定当前权值是否发送至核心节点。权值变化率计算如公式所示。

$$\Delta t = \left| \frac{Weight(t_2) - Weight(t_1)}{t_2 - t_1} \right|$$

1) $Weight(t_1)$：当前服务器在t_1时刻的权值；
2) $Weight(t_2)$：当前服务器在t_2时刻的权值；
3) Δt：当前服务器在一个发送周期内的权值变化率，其中$t_2 > t_1$。

权值用最少上报策略，周期 T 内的权值变化率与预先设定的 Δt 值比较，当 T 内的权值变化率小于 Δt 值时，服务器将不会发送权值至核心节点，只在本地修改保存的权值信息。如果当前服务器权值变化率大于 Δt，则将新权值发送至核心节点。在周期 T 内如果核心节点没有收到权值，则按前面保存的该服务器的权值分发负载。

5. 实验结果和分析

5.1. 实验环境

本文使用了三台八核高性能服务器搭载系统，服务器配置信息参数为：CPU 主频为 4.00 GHz，采用 Intel(R) Core(TM) i7-4790K 处理器，32 G 内存，2 T 硬盘，操作系统版本为 Ubuntu 16.04.2。本文实验测试程序使用 Java 编写，JDK 版本为 1.8，Nginx 使用 1.13.9 版本。通过开源压力测试工具 Jmeter 模拟终端设备大量的 TCP 连接，使用多台压测机器在局域网中对系统进行性能测试。压测机器使用 3 台四核 CPU 的计算机，其配置信息参数为：Intel(R) Core(TM) i5-7200U 处理器，8 G 内存，256 G 固态硬盘，操作系统为 Win10。实验在局域网中进行，机器间通过百兆交换机相连，因此本文实验不考虑带宽的影响。

5.2. 实验设计与分析

通过获取当前时刻的 CPU 的活动信息，分别为用户状态的运行时间、系统内核相关的运行时间、空闲状态的运行时间。CPU 性能指标各项数据见表 1 所示：

Table 1. Data of CPU performance index
表 1. CPU 性能指标各项数据

各项数据名称	描述(从系统启动累积到当前时刻)
user	用户态的运行时间，不包含 nice 值为负进程
nice	nice 值为负的进程所占用的 CPU 时间
system	处于核心态的运行时间
idle	除 IO 等待时间以外的其它等待时间
iowait	IO 等待时间
irq	硬中断时间
softirq	软中断时间

Nginx 收到连接请求后，通过添加代码来获取的后端服务器动态权值进行更新，对修改权值之前需对权值的指针加锁，修改完后解锁。主要加解锁函数代码见图 4 所示：

```
void updateWeight(ngx_stream_upstream_rr_peers_t *peers)
{.....
    ngx_stream_upstream_rr_peers_wlocak(peers);//加锁操作
    ngx_stream_upstream_rr_peer_t *peer;//声明一个指向后台服务器数
    //据的指针
    peer=peers->peer;//指针指向存放服务器权值的地址
    while(weight!=NULL){
        peer->weight=(ngx_int_t)atoi(weight);//设置配置时的权值
        peer->effective_weight=(ngx_int_t)atoi(weight);//设置有效权值
        peer->currect_weight=(ngx_int_t)atoi(weight);//设置当前权值
        peer=peer->next;//指向下一个服务器
        .....
    }
    ngx_stream_upstream_rr_peers_unlock(peers);//解锁操作
    ......
}
```

Figure 4. Add unlock function code
图 4. 加解锁函数代码

本文实验将基于 WLC、基于 WRR 以及 WRR 改进的算法三种方案进行比较。通过测定各服务器的 CPU 负载率，内存利用率接近程度可以比较直观地看出算法的负载均衡效果。

本文根据这三种算法方案进行测试。三种算法的响应时间见表 2 所示，并发请求相应时间对比图见图 5 所示：

Table 2. Response time of three algorithms
表 2. 三种算法的响应时间

并发请求连接数	WLC 算法	WRR 算法	改进算法
150	265	590	270
300	545	715	605
450	800	1115	835
600	1355	1495	1280
750	1610	2120	1505
900	2380	2410	2025
1050	2570	2690	2445
1200	3005	3095	2790
1350	3245	3475	2965
1500	3395	3490	3025

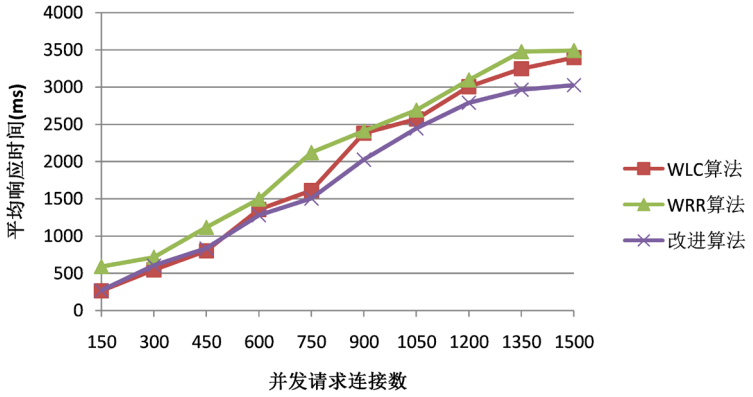

Figure 5. Corresponding time comparison of concurrent requests
图 5. 并发请求相应时间对比图

当并发请求连接数为 1500 时，观察服务器的内存利用率和 CPU 利用率的接近度。CPU 利用率图见图 6，内存利用率见图 7 所示：

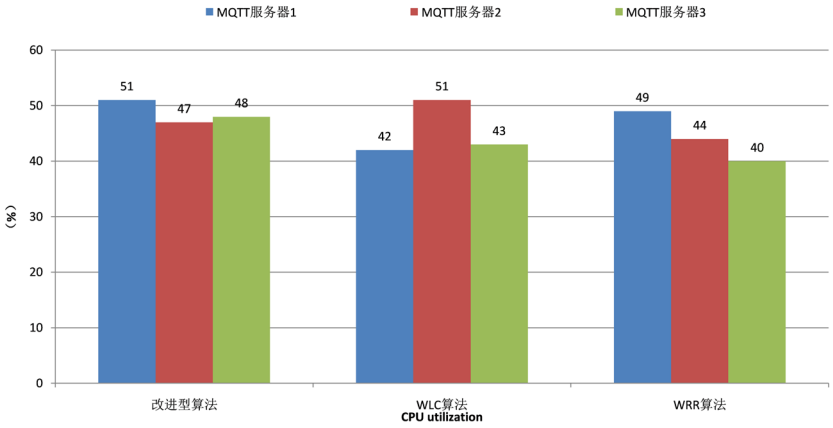

Figure 6. CPU utilization
图 6. CPU 利用率图

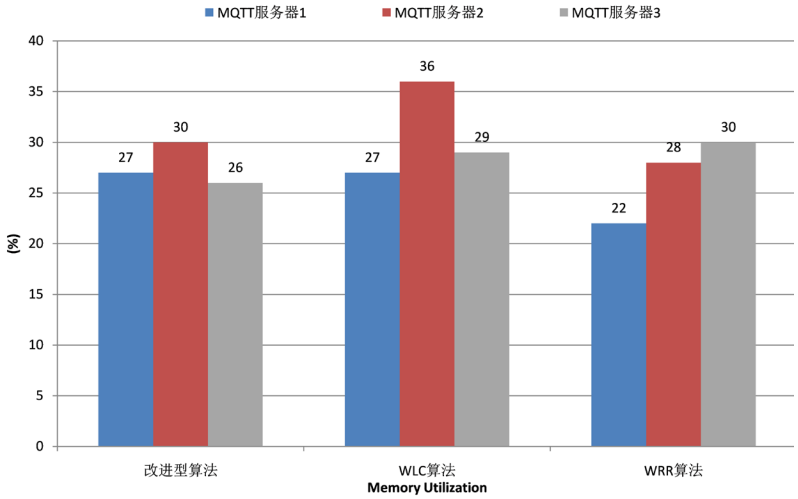

Figure 7. Memory utilization
图 7. 内存利用率图

由此可得，在总体负载水平较轻时，改进算法相比加权最小连接算法的负载均衡效果提高并不明显[14] [15]，当 MQTT 服务器集群出现较重负载时，新的加权算法会按照集群中服务器的实时负载情况合理的分发用户连接请求，新的算法优化能使总体平均响应时间变短，根据上图实验数据可知，能够将负载更加均匀的分发至集群各服务器，负载差由原来的 8%下降到 4%左右，使 MQTT 服务器集群系统的使用效率提高了，因此在用户并发请求连接数较多时，本文新的权值算法提出的改进算法具有较好的实用价值。

6. 结论

本文主要提出了一种基于 MQTT 负载均衡算法的改进设计方案，从优化负载均衡入手，分析 MQTT 服务器权值的算法设计及权值上报设计，MQTT 服务器集群中的各台服务器周期性地收集自身负载信息和连接数，得出权值后发送给 Nginx 负载均衡器，根据各服务器的权值和负载通过筛选，得出合适的服务器去应答新的用户数据连接请求。对服务器负载进行实时分析及负载的重新权值计算及分配，使系统负载更加均匀地分发至集群各服务器。

从多次实验及实际应用数据来看，对消息队列、CPU 利用率、内存利用率的比重系数进行分析，利用权值修改函数 updateWeight()，在修改权值之前对包含权值信息的 peers 指针加锁对权值修改模块优化更新，这种新的 MQTT 负载均衡算法能使各服务器之间的负载差下降，MQTT 服务器集群系统的使用效率提高；因此在用户并发请求连接数较多时，总体平均响应时间将会有一程度的缩短，使 MQTT 服务器集群在一个较为稳定的工况下运行，防止出现数据堵塞而宕机等情况发生。

根据本方案设计的系统已投入智慧校园项目的使用，在实际场景的应用中未出现数据交互不稳定情况，且系统运行效果较好。

基金项目

本课题得到浙江理工大学本科生科研创新计划重点项目(2019ZD-28)、浙江理工大学本科生科研创新计划一般项目(2019YB-24)资助。

参考文献

[1] 李新. 基于 Web 服务器集群的动态负载均衡算法改进及实现研究[D]: [硕士学位论文]. 长沙: 湖南大学, 2012.

[2] 覃川. 基于 Nginx 的 Web 服务器负载均衡策略改进与实现[D]: [硕士学位论文]. 成都: 西南交通大学, 2017.

[3] Gascon-Samson, J., Garcia, F.P. and Kemme, B. (2015) Dynamoth. A Scalable Pub/Sub Middleware for Latency-Constrained Applications in the Cloud. *Distributed Computing Systems* (*ICDCS*), *IEEE* 35*th International Conference*, Columbus, 29 June-2 July 2015, 15-18. https://doi.org/10.1109/ICDCS.2015.56

[4] 赵晔. 基于 Nginx 的 Web 后端服务器集群负载均衡技术的研究与改进[D]: [硕士学位论文]. 昆明: 昆明理工大学, 2017.

[5] 尹友磊. MQTT 服务器负载均衡技术的研究与应用[D]: [硕士学位论文]. 上海: 上海师范大学, 2018.

[6] 王永辉. 基于 Nginx 高性能 Web 服务器性能优化与负载均衡的改进和实现[D]: [硕士学位论文]. 成都: 电子科技大学, 2015.

[7] Hu, S.P., Wen, F. and Lu, S.J. (2019) Design and Optimization of Nginx Sever Based on LNMP.

[8] 包晓安, 曹云棣, 张娜, 等. 基于格分布方差的多目标云工作流调度算法[J]. 电信科学, 2019, 35(2): 1-13.

[9] Liu, X.T., Zhang, T.L., Hu, N., Zhang, P. and Zhang, Y. (2020) The Method of Internet of Things Access and Network Communication Based on MQTT. *Computer Communications*, **153**, 169-176.

[10] 马强. 虚拟化环境下基于 OpenFlow 的服务器集群动态负载均衡架构设计与实现[D]: [硕士学位论文]. 兰州: 兰州大学, 2014.

[11] 章文嵩. Linux 服务器集群系统(四) [EB/OL].

http://www.ibm.com/developerworks/cn/linux/cluster/lvs/part4/index.html

[12] Ansari, N. (2017) Traffic Load Balancing among Brokers at the IoT Application Layer. *IEEE Transactions on Network and Service Management*, **15**, 489-502. https://doi.org/10.1109/TNSM.2017.2787859

[13] Toumura, K. and Nemoto, N. (2013) A Scalable Server Load Balancing Method Using IP Address Stealing. *Computer Software and Applications Conference*, 22-26 July 2013, 1-5.

[14] 姚兆凡. 轻量级 Web 服务器 Nginx 的研究与优化[D]: [硕士学位论文]. 南京: 南京邮电大学, 2017.

[15] 金双喜, 李永, 吴骅, 等. 基于 Kafka 消息队列的新一代分布式电量采集方法研究[J]. 智慧电力, 2018, 46(2): 77-82.