doi:10.3772/j.issn.1002-0470.2021.04.003

面向用户体验的动态负载均衡算法研究①

郑晓辉②*** 史 骁* 金 岩* 宋永浩* 唐宏伟* 赵晓芳③*

(*中国科学院计算技术研究所 北京 100190)

(**中国科学院大学 北京 100049)

摘 要 随着互联网用户数量的迅速增长,Web 服务面临着巨大的访问压力,对 Web 服务器持续提供服务提出了更高的要求。当负载过重时可能会导致服务器宕机等严重状况,从而影响 Web 服务器对用户 HTTP 请求的响应,进而影响服务质量,因此 Web 服务器集群需要更好地响应用户请求。本文提出了一种基于用户体验的动态负载均衡算法(QRFS),在保证用户满意度的前提下,兼顾服务器的负载和请求调度的公平性。实验表明,QRFS 算法能够获得更好的用户满意度。此外,在请求的平均响应时间上,QRFS 算法较加权轮询(WRR)算法提高了 12%,较加权最小连接数(WLC)算法提高了 11%;在服务器负载的均衡性上,QRFS 算法取得了更好的效果,节点间的负载方差更小。

关键词 负载均衡; Web 服务器集群; 公平性; 调度; 用户满意度; 平均响应时间

0 引 言

随着大数据时代的到来和科技的进步,Internet 迅猛发展,Internet 上服务的数量日益增多,逐渐成为用户使用互联网的主要方式。用户访问数量和客户 HTTP 请求数量的提高使得 Web 服务器面临着巨大的访问压力,面对可能引起的负载过重甚至宕机等情况,要求服务器能够更好地应对。Web 服务器集群通过并行处理来进行扩展从而缓解了单台服务器节点的不足,能够将大量的并发访问请求分担到多台服务器节点上,避免单节点的超负荷,提供了更高的可靠性和可用性。负载均衡策略用来调整服务器集群内多台服务器间的负载均衡,并且提高 Web 服务器集群的吞吐量和效率[1-3]。

随着用户对 Internet 服务质量要求的提高,没有好的服务质量则难以满足用户的需求。满足大量不同用户的请求、保证用户的满意度、减少用户的流失对于 Internet 服务越来越重要[4-5]。因此,负载均衡策略不仅要尽可能平均服务器集群中多个服务器节点间的负载,也要更好地传递用户的请求。

负载均衡是服务器集群中的重要问题。负载均衡的目标是尽可能分配与服务器处理能力相匹配的请求任务量、避免服务器超载(overload)或低负载(underload)、优化响应时间、提高服务器资源利用率及系统性能。

目前已经有很多关于 Web 服务器集群负载均衡问题的研究,现有算法可以分为静态和动态两类[6-7]。静态负载均衡算法直接将请求分配给服务器节点,而不考虑服务器的状态,如轮询(round robin)算法、加权轮询(weighted round robin, WRR)算法[8]和随机(random)算法。动态负载均衡算法分配请求时考虑服务器的实时负载信息,如最小连接数(least connection)算法和加权最小连接数(weighted least connection,WLC)算法[9]。然而现有算法从服务器本身出发,更多考虑服务器的负载情况,较少考虑用户的多样性需求,对用户的 HTTP 请求不加

① 国家重点研发计划(2018YFB0904503)资助项目。

② 女,1990 年生,博士生;研究方向:云计算,机器学习;E-mail:zhengxiaohui@ict.ac.cn

③ 通信作者,E-mail:zhaoxf@ict.ac.cn

(收稿日期:2020-04-21)

以区分。当服务器负载过高时甚至会丢弃一部分用户请求,这严重影响了用户体验,因此,仅靠网络的服务质量(quality of service, QoS)机制并不能满足 Web 客户端到服务端的 QoS 问题[10-11]。

用户体验质量(quality of experience,QoE)作为衡量用户使用服务体验情况的重要指标,从主观角度描述用户的满意情况。由于不同用户的习惯偏好不同,对请求的容忍度不同,因此对服务质量的要求也有所不同[5]。如果服务器响应任务请求在考虑服务器负载的同时,也能够考虑请求本身的特征以及不同用户发起请求的不同需求,则有利于更好地提高用户满意度,从而提高服务的整体满意度。

本文提出了一种基于用户体验的动态负载均衡(QoE-based request fair scheduling, QRFS)算法。其基本思想是在分配请求时优先满足请求的 QoE 需求,同时兼顾服务器的负载和请求调度的公平性。本文主要工作如下。

(1) 考虑不同类型的用户对请求响应的满意度情况以及不同请求的特征,根据请求的不同特性预测请求的运行时间,根据用户的阈值设定请求的最长响应时间。

(2) 为了保证大多数用户的请求都能够得到满足,提出了服务器集群的 QoE 安全状态的概念。

(3) 为了避免低优先级的请求总是得不到处理,兼顾了调度的公平性。

1 相关工作

负载均衡是集群技术的重要研究内容,采用负载均衡算法可以将负载(请求)分摊到多个服务器节点上处理,能够更好地提高服务器的响应速度。关于负载均衡问题有很多经典算法,如轮询算法、最小连接数算法和随机数算法[1-3,7]。最简单直接的算法就是轮询算法,该算法将所有请求轮流分配给各个服务器。该算法实现简单,但是当请求数量较大时,容易发生负载倾斜,即某一台服务器的负载严重过高,从而导致服务时间也过长。另一种可选的算法是随机算法,它可以减轻服务时间不平均的问题。这些算法都是静态负载均衡算法,没有考虑服务器的负载情况。因此,基于这些经典算法出现了一些改进的动态负载均衡算法,如加权轮询算法和加权最小连接数算法[9]。

文献[12]提出了一种负载预测模型来判断服务器的负载情况是否过重或过轻,该算法实现采用机器学习中的随机森林算法。文献[13]提出了一种动态的负载均衡算法,考虑服务器的计算能力和负载情况两个指标,旨在最小化服务器的负载,但是该算法导致更高的响应时间并且很多重要指标并未考虑。这两种算法都没有考虑用户以及请求的优先级。

HP 实验室在 Apache 服务器上引入了 Web QoS 的机制[14],主要是通过专门的管理进程对到达的请求进行处理,将 HTTP 请求按照类别设置不同的优先级,然后优先处理优先级高的请求,从而获得更好的服务质量。但是这种机制丧失了请求处理的公平性,容易导致低优先级的请求迟迟得不到处理,而总是处理优先级高的请求[15-16]。

文献[10]根据不同请求的要求,在网络 QoS 机制的基础上,提出 Web 服务器也应具备 QoS 机制,讨论了请求的 QoS 需求和服务的公平性,并提出 3 种方案在两者之间进行平衡。文献[8]采用了混合的负载均衡算法,该算法结合了轮询算法和动态方法,分别为短请求和长请求维护两个等待队列并单独执行,同时平衡队列中请求的等待时间,对长请求更加友好,有利于调度的公平性。然而该算法的效率和等待时间都有待提升。

以上算法虽然考虑了服务器负载和请求特性,但较少从用户的角度出发,考虑用户的不同需求和用户的满意度。因此,亟需一个更有效的负载均衡算法,兼顾请求调度的公平性和服务器效率的同时,更好地满足用户的满意度。

2 算 法

2.1 符号说明和相关定义

表 1 给出了本文需要的符号及其含义说明。

2.2 相关定义

(1) 用户满意度

表1 符号说明及定义

符号	定义
M	请求数目
N	服务器节点数目
P	用户类型
$ReqSeq$	请求序列,$ReqSeq = \{R_1, R_2, \cdots, R_M\}$
$NodeList$	服务器节点序列,$NodeList = \{node_1, node_2, \cdots, node_N\}$
$UnServedReq$	尚未分配的请求序列

采用 QoE 来衡量用户对请求响应情况的满意程度,并用平均分(mean opinion score,MOS)进行量化,用户 u_i 的满意度表示为 s_i。

(2) 整体满意度

表示所有/全体用户的满意度情况为

$$S = \frac{\sum_1^{|U|} s_i}{|U|} \tag{1}$$

(3) 请求的收益值

根据调研与数据分析发现,不同用户对请求响应的满意度有所差异,根据用户的特征和喜好特点以及对响应时间的要求高低进行归类,与之对应的是该类用户发起的请求。假设根据用户 QoE 需求的不同将用户归为 P 类,分别对应 P 个不同的响应级别的请求。除此之外,不同的 Web 请求所需的资源情况也有所不同,如资源的类型有文本、图片、视频等。每个请求的权重与它本身的响应级别(需求)和它请求的资源类型有关。因此,本文提出权益值概念用来表示处理每个请求所获得的收益。

通过给每个请求赋予一定的收益值,服务器节点处理不同的请求将会获得不同的收益,每个请求的初始收益不仅与请求的优先级和请求类型等相关,还与用户的 QoE 需求相关。并且每个请求的收益都会随着时间衰减。调度的目的则是将最终处理所有请求获得的收益最大化,即在一定时间内满足更多用户请求的 QoE 需求。其中,$priority$、$type$ 和 $QoERequire$ 分别代表请求 R_i 的优先级、请求类型和请求 dQoE 需求。$profit_0^i$ 表示请求 R_i 的初始收益值,$profit^i(T)$ 表示在 T 时刻请求 R_i 的收益值。

$$profit_0^i = \alpha \cdot \frac{priority \cdot QoERequire}{type} \tag{2}$$

$$profit^i(T) = profit_0^i e^{-\lambda T} \tag{3}$$

(4) 服务器的负载能力

由于每台服务器节点都有其处理的极限,在达到极限之前服务器处理请求的能力逐渐趋于饱和,在达到极限之后服务器为避免死机会产生“假死”现象,将毫无区分地挂起所有任务请求。根据对每个服务器节点的承压测试,测定每台服务器节点的负载能力,确定其饱和状态。$Capacity$ 表示服务器节点的负载能力,$CurrentLoad$ 为服务器节点的当前负载,$loadRatio$ 表示服务器节点的负载率,其计算公式为

$$Node.loadRatio = \frac{Node.CurrentLoad}{Node.Capacity} \tag{4}$$

(5) QoE 安全状态

为了保障用户的体验质量和整体满意度,本文提出 QoE 安全状态,反映当前 Web 服务器集群对请求的响应情况,也反映了对用户请求的 QoE 需求的满足情况。

如果存在由当前大多数请求构成的安全序列 $\{R_1, R_2, \cdots, R_k\}(k <= N)$,且其中请求 R_i 的 QoE 需求被满足,并且 $k \geqslant N \cdot \xi$,则认为服务器集群处于安全状态。安全状态下,不存在请求被丢弃的状况,并且大多数请求的 QoE 需求都会被满足。如果不存在这样的安全序列,则处于不安全状态。

当请求的数量不断增多,服务器节点的负载逐渐升高,出现不能满足用户的 QoE 需求的情况时,则需要扩充服务器节点;当随着请求的不断处理,服务器节点的负载逐渐降低,服务器集群再次达到 QoE 安全状态时,则可以适当减少服务器节点。

在安全状态下,大多数用户(如 80%)的请求都能被高质量地满足。这样不仅可以满足用户请求的 QoE 需求,而且可以有效地增减节点。

(6) 请求调度的公平性

在对请求进行调度时,不能永远优先调度优先级最高的用户/请求,这样会导致优先级低的用户请求得不到处理与响应,影响调度的公平性。因此,在处理请求时应该兼顾公平性,若不同类型请求处理量与请求到达数量成正比时,公平性较好。如 t_1 类请求到达数量为 $number_1$,请求处理量为 $through_1$,t_2 类请求到达数量为 $number_2$,请求处理量为 $through_2$,

若 $\frac{through_1}{through_2} \approx \frac{number_1}{number_2}$，则认为公平性较好。

2.3 QRFS 算法

QRFS 算法的基本思想是尽可能满足更多用户的 QoE 需求，而不是强调某一位用户的 QoE 需求，以此来提升整体满意度并兼顾公平性。

QRFS 算法的基本原理如下。

（1）每位用户对请求的响应要求不同，因此不同类型用户发起的请求具有不同的优先级。

（2）确定请求的收益值，按照收益值对请求降序排列。

（3）对于每个服务器节点，在保证能够满足请求的 QoE 需求的前提下，都尽可能地分配更多的请求，使得最终收益最大化。

（4）当现有的服务器节点都不能满足所有请求的 QoE 需求时，此时出现/达到不安全状态，则适当增加服务器节点；当现有服务器节点出现空闲状态时/满足所有请求的 QoE 需求后仍有富余时，则再次回到安全状态，此时适当减少服务器节点。QRFS 算法描述如算法 1 所示。

算法 1 QRFS 负载均衡算法

输入：Web 请求序列 *ReqSeq*，服务器节点列表 *NodeList*

输出：请求在服务器节点的分配序列 *Node*

```
1.  initializeProfit;//初始化请求收益值
2.  R = sortRequst(ReqSeq);//按收益值、QoE 需求降序排列
3.  Node = sortNode(NodeList);//按负载升序排列
4.  int k = 0;
5.  for Node[i] in Nodelist:
6.      int r = k;
7.      for (j = k + 1; j < M; j + +)
8.          while (r > =0&&R[r]. deadline >
9.              R[j]. deadline)
10.         end while
11.         r--;
12.     end for
13.     if ( (R[r]. deadline < =R[j]. deadline) &&
            (R[j]. deadline > R[r]. deadline +
             R[j]. pretime) )
14.             for(temp = k; temp > =r + 1; temp--)
                    R[temp + 1] = R[temp];
15.             end for
16.             R[r + 1] = R[j];
17.             Node[i]. insert(R[j]);
18.     else
19.         if( (i + 1 < N) &&
20.           (Node[i + 1]. loadRatio < δ))
21.               Node[i + 1]. insert(R[j]);
22.         else
23.               UnServedReq. add(R[j]);
24.         end if
25.     end if
26.     k + +;
27. end for
28. SafeState(Node,R,i, j);
29. FairSchedule(R, j);
```

其中，sortRequstDesc（*ReqSeq*）是对请求序列 *ReqSeq* 按照请求收益值进行降序排序，请求的收益值越高则优先级越高。sortNodeAsce(*NodeList*) 是对服务器节点列表 *NodeList* 按照当前负载值进行升序排序，当前负载值越低则优先级越高。QRFS 算法根据每个请求的收益值将请求尽可能多地在满足 QoE 需求的情况下交由合适的服务器节点处理。服务器集群安全状态算法 SafeState(*Node*, *R*, *i*, *j*) 用于调整服务器节点的数目以保证服务器集群时刻处于安全状态。公平调度算法 FairSchedule(*R*, *j*) 则用于保证请求调度的公平性。QRFS 算法的复杂度为 $O(MN)$。

为了满足大多数用户的 QoE 需求，在一定时间内处理更多的请求，保证服务器的 QoE 安全状态，本文提出服务器集群的安全状态算法 SafeState(*Node*, *R*, *i*, *j*)。根据服务器节点当前处理请求时对请求的满足情况，*R*[*i*]. deadline 表示请求 *R*[*i*] 的响应时间阈值，*R*[*i*]. runtime 表示请求 *R*[*i*] 的预计运行时间。如果请求的真实响应时间在请求的时间阈值之前，则认为满足请求的响应时间需求。若服务器节点集群能够满足大多数请求的响应需求，则认为处于安全状态；否则处于不安全状态。当不能够满足多数用户的响应需求，将对整体的用户满意度产生较大影响。ξ 表示当对请求响应需求的不满足情况达到一定比例时，则添加服务器节点。该算法的时间复杂度为 $O(1)$，详细描述见算法 2。

算法 2 节点动态调整算法 SafeState(*Node*, *R*, *i*, *j*)

输入:服务器节点 *Node*,请求 *R*,当前服务器节点 *i*,当前处理请求 *j*

输出:服务器节点列表 *NodeList*

```
1.  int k = UnServedReq.length;
2.  if (k > N · (1 - ξ))
3.      NodeList.addNode();//待处理请求列表
4.  end if
5.  if (k < N · (1 - ξ))
6.      NodeList.deleteNode(i + 1, N);//节点富余
7.  end if
```

本文提出 FairSchedule(*R*, *j*)算法用来保证请求调度的公平性,当每一类请求处理的吞吐量和请求到达速率的比值相等时,认为请求的调度近似公平。其中,$through_i$ 表示第 *i* 类请求处理的吞吐量,$number_i$ 表示第 *i* 类请求的到达数量,该算法的时间复杂度为 $O(P)$, *P* 为请求类型的数目。公平调度算法见算法 3。

算法 3 公平调度算法 FairSchedule(*R*)

输入:请求序列 *R*

输出:更新了优先级的请求序列 *R*

```
1.  for (int i = 1; i <= P.length; i++)
2.      if (through_i < M · number_I / Σ_1^M number_k)
3.          R[P[i]].priority++;
4.      end if
5.  end for
```

3 实验结果与分析

为了评价 QRFS 算法的性能和有效性,根据现有的条件,对所实现的负载均衡算法进行了测试。采用 WRR 和 WLC 与本文算法进行性能比较。分别在用户满意度、平均响应时间、CPU 利用率、负载均衡性性和调度公平性 4 个指标上进行分析对比。

3.1 实验环境

通过仿真实验对算法进行验证,代码的实现采用 Python 语言,实验运行在配置为 2.5 GHz CPU 和 4 GB RAM 的便携式计算机上,操作系统为 OS,处理器为 Intel Core i5,主频为 2.9 GHz,运行内存为8 GB,250 GB 磁盘空间。

仿真器功能类似于 CloudSim,由于本实验需要根据不同用户需求生成不同类型的请求,因此开发实现了仿真器。仿真实验包含两部分,分别是请求发生器和负载均衡器。请求发生器用来模拟用户请求,根据文献[5],通过用户的评分行为以及不同用户对请求响应满意度的要求情况,将用户类型分为 3 类,即用户生成 Web 请求的优先级分为 3 类,并且按照正态分布生成 3 类请求。由于 Web 请求之间的时间间隔的时间序列服从指数分布[10],因此,Web 请求根据泊松分布按照不同的速率到达,分别是每秒 5000 ~ 35 000 个请求。负载均衡器用来实现不同的负载均衡算法,并产生调度结果。负载均衡器共包含一个负载均衡节点和 5 个服务器节点。节点的配置情况如表 2 所示。

表 2 测试环境配置参数

	CPU	内存	操作系统	负载临界值
负载均衡节点	1 core	2 GB	Centos 6.5	
服务器节点 1	1 core	1 GB	Centos 6.5	90%
服务器节点 2	1 core	2 GB	Centos 6.5	80%
服务器节点 3	1 core	2 GB	Centos 6.5	85%
服务器节点 4	1 core	4 GB	Centos 6.5	85%
服务器节点 5	1 core	4 GB	Centos 6.5	90%

3.2 实验结果及分析

本节对提出的 QRFS 算法与传统的 WRR 算法和 WLC 算法进行详细对比评测。主要讨论随着数据规模的增长,算法在不同性能指标上的对比与分析,以及参数 ξ 对用户满意度的影响。

图 1 展示了在各负载均衡算法下用户整体的满意度情况,计算方式采用文献[5]中的主客观评价

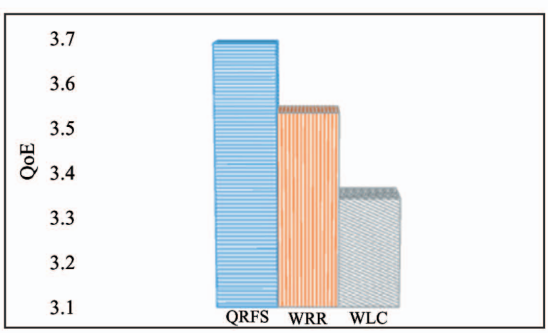


图 1 用户整体满意度

方法，用 QoE 表示用户的满意度，采集请求响应过程中的各项参数并完成评价。从图中可以看出，采用本文算法的情况下，QoE 评分明显较高，即用户整体满意度较高。

图 2 展示了各负载均衡算法在不同的请求数目下平均响应时间的情况。分别展示了 3 种算法在请求数目从 5000 到 35 000 的变化过程中的平均响应时间。从图中可以看出，在请求较少时，WRR 算法优于 WLC 算法；但是随着请求的增多，WRR 算法的响应时间逐渐高于 WLC 算法，但是两者的差别不大。而 QRFS 算法的平均响应时间则明显优于另外两种算法，并且随着请求数的提高，差异也显著提高。QRFS 算法平均响应时间较 WRR 算法提升了约 12%，较 WLC 算法提升了约 11%。

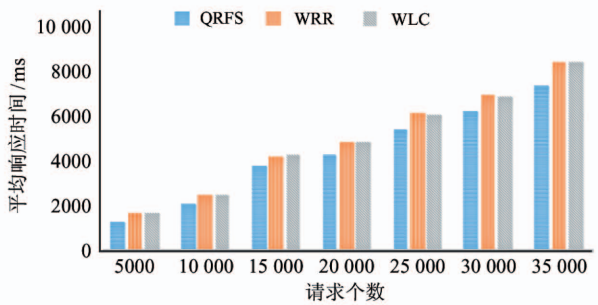
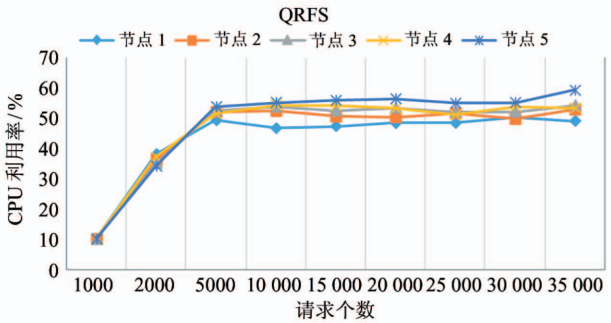


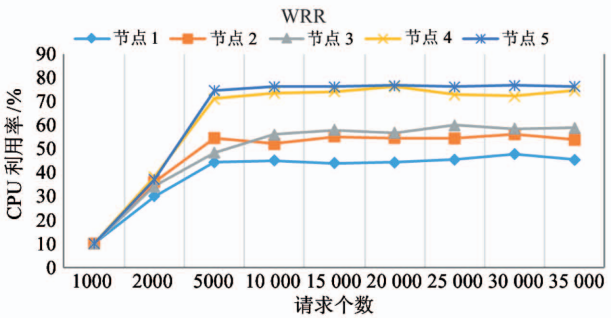
图 2　平均响应时间

不同算法的 CPU 利用率对比情况如图 3 所示，分别记录了 5 个服务器节点随请求个数增加的 CPU 利用率情况。图 3(a)表示 QRFS 算法的 CPU 利用率情况，可以看出，节点间的 CPU 利用率情况比较接近；图 3(b)表示 WRR 算法的 CPU 利用率情况，可以看出，节点间的 CPU 利用率情况差距较大；图 3(c)表示 WLC 算法的 CPU 利用率情况，可以看出，节点的 CPU 利用率比 WRR 算法小，但仍然比 QRFS 算法的波动大。

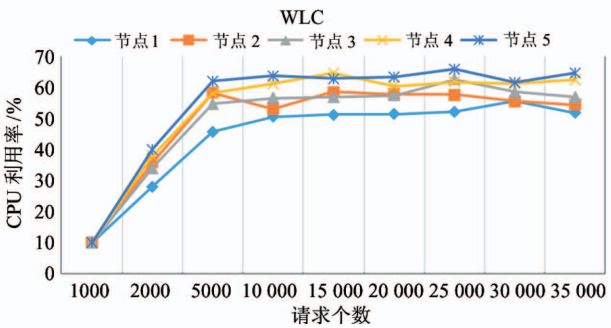
图 4 表示了各负载均衡算法在不同的请求数目下服务器节点的平均负载情况，请求数量从 5000 个到 35 000 个。从图 4(a)中可以看出，采用 WRR 算法的情况下服务器节点的负载一直处于较高水平。而当请求数量较少时，采用 WLC 算法的服务器节点负载较低，效果较优；随着请求数量的不断增加，采用本文算法的服务器节点负载情况逐渐变好，证明



(a) QRFS 算法



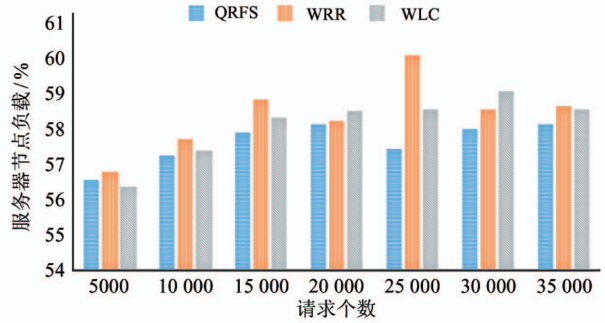
(b) WRR 算法



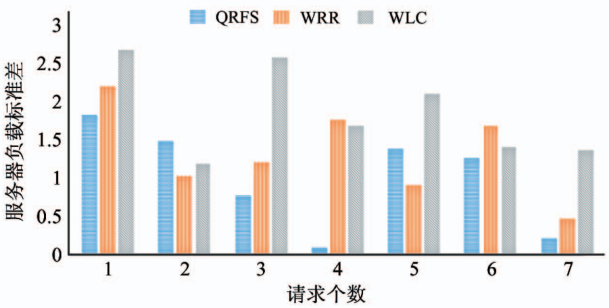
(c) WLC 算法

图 3　不同算法 CPU 利用率对比

QRFS 算法在请求数量越多的情况下越能显现优势。图 4(b)表示了在采用不同算法的情况下 5 个服务器节点负载的均匀情况，通过计算 5 个服务器节点负载的标准差，更能表示 5 个服务器节点承受请求的分配平均情况。可以看出，采用 QRFS 算法和 WRR 算法的情况下，服务器节点的负载标准差较低，表明 5 个服务器节点的负载更加均衡，因为 WRR 算法本身是在多个服务器节点间进行轮询，因此更加证明了本文算法的有效性；而采用 WLC 算法的情况下，服务器节点的负载标准差高出很多，表明 5 个服务器节点容易出现某一个或某两个节点的负载明显高于其他服务器节点的情况。



(a) 服务器节点负载对比



(b) 服务器节点负载标准差对比

图 4　服务器节点负载情况

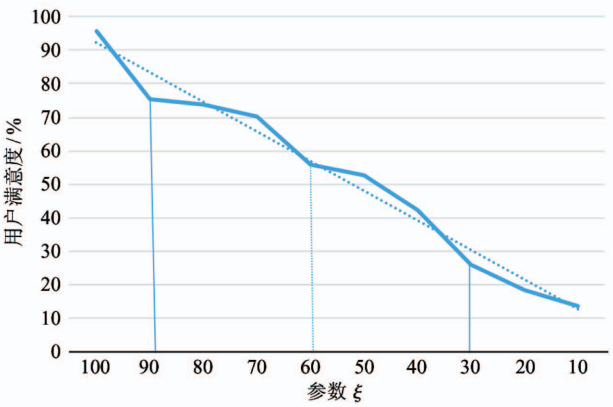


图 5　参数 ξ 的确定

为了保持服务器集群的安全状态,对服务器节点进行动态调整,需要确定参数 ξ 的值,以便寻找服务器集群性能变化对用户满意度影响的临界点。通过图 5 可以看出,当 ξ =90% 时,用户满意度出现第一次明显下降;当 ξ =60% 时,用户满意度出现第二次明显下降,此时用户满意度仍然在 50% 以上,如果 ξ 值继续降低,说明不能满足一半以上用户的满意度。因此,当 60% ≤ξ≤90% 时,能够满足大多数用户的需求。在服务器资源充足的情况下,可以将 ξ 设置高一些;当服务器资源比较缺乏的情况下,可以将 ξ 设为临界值 60%。

4　结　论

为了让服务器能够更好地响应 HTTP 请求,满足用户需求并提升用户满意度,本文提出了一种面向用户体验的负载均衡算法 QRFS,考虑了不同用户请求的需求,优先处理优先级更高的请求,并在兼顾服务器负载的前提下处理尽可能多的请求,同时保证请求调度的公平性。该算法能够自适应服务器集群的状态,通过动态的调整服务器节点数目,在保证满足需求的基础上,提高了服务器的资源利用率,有效地减少了请求的平均响应时间,并且通过大量实验证明该算法与 WRR 算法和 WLC 算法相比,能够获得更好的用户满意度;在平均响应时间上较 WRR 和 WLC 算法分别提升了约 12% 和 11%;并且取得了较好的负载均衡效果,节点间的负载均衡标准差降低了 3% 左右,说明服务器节点间的负载更加平均,达到了更好的负载均衡性。

参考文献

[1] Kumar P, Kumar R. Issues and challenges of load balancing techniques in cloud computing[J]. *ACM Computing Surveys*, 2019, 51(6):1-35

[2] Einollah J G, Amir M R, Nooruldeen N Q. Load-balancing algorithms in cloud computing: a survey[J]. *Journal of Network and Computer Applications*, 2017, 88: 50-71

[3] Poonam S, Maitreyee D, Naveen A. A review of task scheduling based on meta-heuristics approach in cloud computing[J]. *Knowledge and Information Systems*, 2017, 52: 1-51

[4] 苏命峰, 王国军,李仁发. 基于利益相关视角的多维 QoS 云资源调度方法[J]. 通信学报,2019,40(6):102-115

[5] Zheng X H, Jin Y, Shi X, et al. UCWE: a user-centric approach for Web quality of experience measurement[C]//Proceedings of 2019 IEEE 33th International Symposium on Parallel and Distributed Processing, Xiamen, China, 2019: 928-935

[6] 朱虹宇,李挺,闫健恩,等. 基于动态负载均衡的分布式任务调度算法研究[J]. 高技术通讯,2014,24(12):1261-1269

[7] 董一. 关于计算机集群中的负载均衡的探讨[J]. 信息系统工程,2019(7): 58-58

[8] Samir E, Shahenda S, Manar J. A novel hybrid of shortest job first and round robin with dynamic variable quantum time task scheduling technique[J]. *Journal of Cloud Computing*, 2017, 6(1):1-12

[9] Zhu P, Zhang J X. Load balancing algorithm for Web server based on weighted minimal connections[J]. *Journal of Web Systems and Applications*, 2017, 1(1): 1-8

[10] 单志广,戴琼海,林闯,等. Web 请求分配和选择的综合方案与性能分析[J]. 软件学报,2001,12(3):355-366

[11] 胡文斌,邱振宇,聂聪,等. 面向请求预处理的实时按需数据广播调度方法[J]. 计算机学报,2018,41(9): 2060-2076

[12] Anju Band Inderveer C. Prediction-based proactive load balancing approach through VM migration[J]. *Engineering with Computers*, 2016, 32(4): 1-12

[13] Chen S L, Chen Y Y, Kuo S H. CLB: a novel load balancing architecture and algorithm for cloud services[J]. *Computers and Electrical Engineering*, 2017, 58:154-160

[14] 张俊星, 马建红, 周松松. 基于区分 Web QoS 的负载均衡集群模型[J]. 计算机系统应用,2014, 23(2): 189-194

[15] 郭成城,晏蒲柳. 一种异构 Web 服务器集群动态负载均衡算法[J]. 计算机学报,2005,28(2):37-42

[16] Maria A R, Rajkumar B. Scheduling dynamic workloads in multi-tenant scientific workflow as a service platforms [J]. *Future Generation Computer Systems*, 2018, 79(2):739-750

Research on a dynamic load balancing algorithm based on quality of user experience

Zheng Xiaohui[*,**], Shi Xiao[*], Jin Yan[*], Song Yonghao[*], Tang Hongwei[*], Zhao Xiaofang[*]

([*] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([**] University of Chinese Academy of Sciences, Beijing 100049)

Abstract

With the development of information technology and exploding of the number of Internet users, Web services are facing huge access pressure, which puts forward higher requirements for Web servers to provide services continuously. Web server overloading can lead to serious conditions such as server downtime, which can affect the response to the users' HTTP requests and thus impacting the quality of experience. A quality of experience-based request fair scheduling (QRFS) algorithm is proposed, which gives consideration of different types of requests and users' priority, with the goal of ensuring user satisfaction and achieving good fairness and requests distribution among server nodes. Experiments show that QRFS achieves better user satisfaction than the weighted round robin (WRR) algorithm and the weighted least connection (WLC) algorithm. Besides, QRFS improves the average response time by 12% compared to WRR, and 11% compared to WLC. In terms of load balance of the server, QRFS algorithm achieves smaller load variance.

Key words: load balancing, Web server cluster, fairness, scheduling, user satisfaction, average response time