计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(9):220−223 [doi: 10.15888/j.cnki.csa.006521]

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

基于预测模型及独立训练节点的负载均衡策略①

陈大才[1,2], 吕　立[2], 高　岑[2], 孙　咏[2]

[1](中国科学院大学, 北京 100049)
[2](中国科学院 沈阳计算技术研究所, 沈阳 110168)

摘　要: 随着业务量、用户量的增大, 提高服务器集群的效率变得越来越重要. 本文使用机器学习算法, 通过对历史数据进行训练得到响应时间预测模型, 来预测新请求的响应时间, 根据每个服务器节点的预估响应时间将请求分配给具有最少响应时间的服务器节点, 从而提高集群中请求分配的均衡性, 提高集群的效率. 本文通过对三种机器学习算法的实验, 均表明本策略能降低小集群高并发场景中系统的平均响应时间.

关键词: 负载均衡; 机器学习; 最少响应时间

引用格式: 陈大才,吕立,高岑,孙咏.基于预测模型及独立训练节点的负载均衡策略.计算机系统应用,2018,27(9):220–223. http://www.c-s-a.org.cn/1003-3254/6521.html

Load Balancing Strategy Based on Predictive Model and Independent Training Nodes

CHEN Da-Cai[1,2], LYU Li[2], GAO Cen[2], SUN Yong[2]

[1](University of Chinese Academy of Sciences, Beijing 100049, China)
[2](Shenyang Institute of Computer Technology, Chinese Academy of Sciences, Shenyang 110168, China)

Abstract: As business and users increase, it becomes more and more important to improve the efficiency of server clusters. In this study, the machine learning algorithm is used to predict the response time of new requests by training the historical data. According to the estimated response time of each server node, the request is allocated to the server node with the least response time. The balanced allocation of requests in a cluster has been improved and improves the efficiency of the cluster. In this study, experiments on three kinds of machine learning algorithms show that this strategy can reduce the average response time of system in small-scale high-concurrency clusters.

Key words: load balancing; machine learning; least response time

1　引言

互联网的高速发展使得上网用户剧增, 很多互联网服务的日活跃量成指数增长. 这就给服务器带来了巨大的压力. 服务器必须升级以服务更多的用户. 同时, 近几年互联网越来越多的新“玩法”给服务器的并发访问能力带来了严峻的挑战. 比如微信抢红包、淘宝双十一、春运抢票等. 2016 年除夕夜发的红包数量就达到了几十亿个. 对承载这些秒级数据和高并发响应的的服务器的性能的需求可想而知[1].

如何提高服务器的并发访问能力还是一个值得深入研究的课题. 本文通过提出一种新型的负载均衡策略来提高服务器集群的整体处理效率, 从而提高 Web 服务器的并发访问能力.

2　研究现状

多年来, 负载均衡一直是服务端一个热门的话题, 新的策略和解决方案不断的被提出, 已经从最初的只能通过静态设置, 到后来的可以根据集群运行情况作出动态分配, 再到最后的自适应策略等. 已经有很多的相关技术人员作出了不可磨灭的贡献[1]. 其中, 国外的许多企业和研发团队推出了很多较好的解决方案. Microsoft 公司针对于集群的负载情况, 提出了具有较

① 收稿时间: 2018-01-07; 修改时间: 2018-02-09; 采用时间: 2018-02-26; csa 在线出版时间: 2018-08-16

高可伸缩性和可用性的网络负载均衡技术 (NLB) 以及基于作用在多层集群网络的中间件与网络负载均衡的组件负载均衡技术 (CLB)[2]. IBM 公司推出的 Web Sphere[3]相关的一套强大的 Web 应用服务器, 其提供了优秀的集群解决方案以及卓越的负载均衡功能. Intel 公司网擎系列中的负载均衡器采用的快速响应算法[4]. 在国内, 诸多高校也在致力于研究负载均衡策略, 例如清华大学、国防科学技术大学、浙江大学等, 其中具有代表性的有清华大学研发的可扩展的 Web 服务器集群系统, 国防科技大学章文嵩博士主持开发的 Linux 虚拟服务器项目等[5].

3 最少响应时间负载均衡策略

本文提出的算法根据当前请求的请求类型计算出当前每个服务器节点预计的响应时间, 并从中选择一个拥有最少响应时间的节点, 用该节点处理请求.

本算法适用于具有负载调度器、业务服务器池和共享存储三层结构的负载均衡集群结构[6].

为了简化模型, 本文假设集群中的所有服务器节点都具有相同的硬件和软件配置.

3.1 相关概念

负载调度器: 负载均衡集群结构中用来接收用户请求并向后端服务器节点分发请求的节点.

业务服务器: 用于实际业务操作的服务器.

请求: 用户发送的 HTTP Request, 如登录请求、注册请求等.

请求类型: 请求的动作, 如登录、注册、购买商品等.

响应: 服务器处理完用户请求后, 反馈的结果.

请求处理时间: 服务器从请求队列中抓取请求进行处理, 并处理完成的时间.

请求响应时间: 请求从负载调度器发出后, 直至接收到响应所需要时间.

3.2 影响响应时间因素的分析

在集群结构中, 影响单节点中响应时间的因素大致有以下几种:

(1) 当前服务器节点的性能情况[1]

如 CPU 、内存、磁盘 IO、网络带宽情况.

当前节点的性能决定了请求的处理时长.

(2) 请求类型

不同类型的请求所需要做的操作步骤不同 (如有的请求查询数据库较多, 有的数学计算较多), 所需要的资源也不同 (如 CPU 密集型任务需要更多的处理机资源, IO 密集型任务需要更多的 IO 资源), 进而所需要的处理时间也不同.

例如登录请求所做的操作一般比获取用户资料多, 也更耗费时间.

(3) 请求队列的总响应时间

即当前请求队列中的所有请求都被处理完所需要的时间.

当前请求队列的总响应时间决定了新请求多长时间之后才能被处理.

经过上面的分析, 本文提出了一种算法, 通过使用以往的实际请求响应时间及节点的负载情况等历史数据进行建模, 然后预测新请求的响应时间, 从而选择具有最少响应时间的服务器节点来处理请求, 达到所有请求都近似拥有最少响应时间的目标, 最终集群的总响应时间就减少了.

3.3 算法描述

给每个服务器节点设置以下变量:

(1) 当前性能情况 P

(2) 预估处理等待时间 T_w

用于标记新请求可能需要等待多长时间才能被处理.

(3) 预估响应时间 R_p

给每个请求设置以下变量:

(1) 请求类型 Action 参数 A

(2) 请求的开始响应时间 R_{start}

(3) 请求的实际响应时间 R_r

(1) 提取请求中的请求类型[7]

负载调度器收到用户发送的请求, 提取出其中的请求类型参数 A.

(2) 预测各节点的响应时间

将请求类型参数 A、各节点的当前性能情况信息 P 和各节点的预估处理等待时间 T_w 输入响应时间预测模型中得出每个服务器节点预估的响应时间.

(3) 选择具有最少响应时间的服务器节点

负载调度器从上一步计算出的各节点预估响应时间中选择一个具有最少响应时间的服务器节点, 并更新该节点的 R_p 值, 将请求发送给该服务器节点, 并记录该请求的开始响应时间 R_{start}.

(4) 处理请求并记录实际响应时间

服务器节点收到请求后, 将请求放入请求队列中. 服务器节点从请求队列中选取一条请求进行处理. 处理完成后, 在响应头中添加当前服务器节点的性能情况信息 P' 作为响应头, 并将响应回传给负载调度器.

(5) 使用实际响应时间矫正响应时间预测模型

负载调度器用收到响应时的当前时间 T_n 减去第 2 步记录的开始响应时间 R_{start}, 得到该请求的实际响应

时间 R_r. 负载调度器将该实际响应时间 R_r 和第 2 步记录的 A、P、T_w 放入响应时间预测模型中, 对预测模型进行校正.

(6) 更新预估处理等待时间 T_w

更新预估处理等待时间为 T_w=R_p–R_r, 其中 R_p 等于服务器节点中当前请求队列中最末尾的一条记录的预估响应时间; R_r 等于最近收到响应的请求的实际响应时间, 也就是请求队列中刚处理完的请求的实际响应时间. 两者之差就是当前请求队列的总响应时间.

(7) 记录结点当前负载情况

负载调度器从收到的响应头中提取结点的当前负载情况 P', 更新变量 P.

(8) 给用户发送响应

负载均衡器从响应头中删去负载均衡算法使用的信息, 并将响应发送给用户.

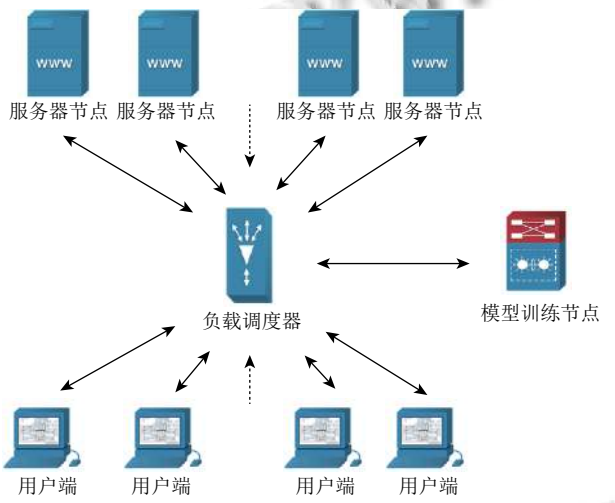


图 1 集群结构

3.4 提取请求中的请求类型

当负载调度器接收到由用户发送过来的请求时, 负载调度器会从请求中提取请求类型.

例如, 从请求 http://www.example.com/login?userID=123456&pwd=s2a51s&timestamp=1400&verify=s45f83f 中可以提取其请求类型为 login; 从请求 http://www.example.com?action=signup&userID=123456 中可以提示其请求类型为 signup.

3.5 响应时间预测模型

本算法通过使用机器学习算法建立响应时间预测模型来预测请求的响应时间.

通过 3.2 节的分析, 本算法使用请求类型 A、服务器节点的性能情况 P、请求队列的总响应时间 T_w 作为输入参数, 用请求响应时间 R_p 作为输出参数. 即有如下模型:

$$R_p = f(A, P, T_w) \tag{1}$$

为了方便实验, 本文只选用了 CPU 占用率、内存剩余量、磁盘 IO 读写速度、网络读写速度等简单的性能参数. 为了方便机器学习, 以上参数都使用标准化算法将他们转换到一个数值范围内.

为了减少负载调度器的计算开销, 将模型的训练阶段的计算任务放到一个单独的计算机中进行, 称之为模型训练节点. 模型训练节点将模型训练好之后通过网络将结果发送给负载调度器, 负载调度器用它来预测响应时间.

3.6 减少负载调度器的计算开销

本算法为了提高任务调度的均衡性, 给负载调度器增加了不少的额外计算开销, 而如果系统的额外开销大于系统的优化, 那么很容易进一步增加集群的负载. 本算法的细节设计部分采用了以下几种方法进行优化, 以更大程度的减少负载调度器的计算开销:

(1) 将模型建立过程的计算任务放到一个独立于负载调度器之外的计算机中执行, 这样降低了负载调度器的计算量.

(2) 当处理完成的请求数达到一定数量后才矫正一次模型, 而不是每一次处理完请求后都要矫正响应时间预测模型, 减少了频繁训练模型带来的计算量[8].

(3) 随着系统运行时间的增加逐渐增大矫正模型的请求量阈值.

当系统运行足够长时间之后, 模型就变得非常稳定了, 所以就不需要频繁矫正模型了.

4 实验验证

4.1 实验环境

本文所使用的实验环境如下:

(1) 服务器节点

数量: 16; 硬件: 6G 内存/2 核 4 线程 3.2 GHz CPU; 软件: CentOS 7 + Tomcat 8 + Redis + MySQL + kafka + 商品秒杀系统 (业务系统).

(2) 负载调度器

数量: 1; 硬件: 8G 内存/4 核 8 线程 3.3 GHz CPU; 软件: CentOS 7 + Nginx[8,9] 1.12 + 本文所讲述的负载平衡算法模块.

(3) 模型训练节点

数量: 1; 硬件: 4G 内存/2 核 4 线程 2.2 GHz CPU; 软件: CentOS 7 + Python3 + Scikit-learn.

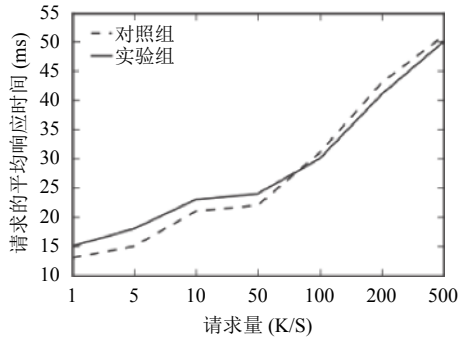
(4) 用户端

数量: 2; 硬件: 4G 内存/2 核 4 线程 2.2 GHz CPU; 软件: CentOS 7 + Java8 + JMeter.

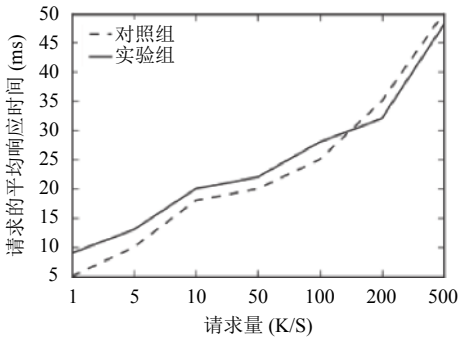
4.2 实验过程

本实验使用 JMeter[3]模拟用户的高并发访问请求; 用商品秒杀系统作为实验中的基准测试程序, 该系统拥有完整的用户登录、注册、管理模块和商品浏览、抢购模块; 使用 Scikit-learn[3]运行机器学习程序.

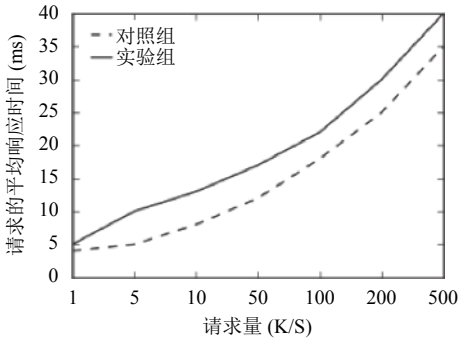
本实验通过设置不同的并发量、不同的机器学习算法 (决策树 CART、向量机、KNN 等算法)[10]、不同的服务器节点 (4 节点集群、8 节点集群和 16 节点集群) 来进行不同组的实验, 采用传统的 URL 散列算法作为对照组, 得出实验结果如图 2.



(a) 4节点集群



(b) 8节点集群



(c) 16节点集群

图 2 实验结果

从实验结果可以看出, 本算法适用于小规模集群、高并发请求的场景. 对于小集群、低并发请求和大集群高并发的情况, 负载调度器的额外开销超过了系统的优化, 增加了系统的平均响应时间; 而小集群、高并发请求的情况下, 由于集群中服务器节点的性能是集群中的性能瓶颈, 本算法优化了任务调度来均衡各服务器节点的性能, 提高了集群的效率, 降低了系统的平均响应时间.

5 总结

本文通过使用机器学习算法建立了响应时间预测模型对请求的响应时间进行预估, 并使用最少响应时间策略进行负载调度, 提高了小集群高并发场景下系统的效率. 通过实验验证了本算法对减少集群的平均响应时间具有一定的效果.

参考文献

1 吴延庆. 一种基于 Nginx、Memcached、Tomcat 负载均衡网状架构. H04L29/08 CN104618508A. [2015-05-13].

2 刘琨. 云计算负载均衡策略的研究[博士学位论文]. 长春: 吉林大学, 2016.

3 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.

4 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与实现[硕士学位论文]. 成都: 电子科技大学, 2015.

5 蒲晓阳. 基于 Nginx 和 Redis 高并发 Web 服务负载均衡的研究. 内江科技, 2016, (1): 40–33. [doi: 10.3969/j.issn.1006-1436.2016.01.032]

6 王利萍. 基于 Nginx 服务器集群负载均衡技术的研究与改进[硕士学位论文]. 济南: 山东大学, 2015.

7 王艳, 陈卫卫. 基于 Nginx 替代 Apache 在高并发 WEB 负载均衡系统中的应用. 电子测试, 2015, (6): 88–92. [doi: 10.3969/j.issn.1000-8519.2015.06.027]

8 Soni R. Nginx Core Architecture. New York: Apress, 2016: 97–106.

9 Soni R. Load Balancing with Nginx. New York: Apress, 2016: 153–171.

10 Hsiao HC, Chung HY, Shen HY, *et al*. Load rebalancing for distributed file systems in clouds. IEEE Transactions on Parallel & Distributed Systems, 2013, 24(5): 951–962. [doi: 10.1109/TPDS.2012.196]