基于实时性能动态反馈的负载均衡算法①

梁 彪 黄 战 (暨南大学 计算机系 广东 广州 510632)

摘 要: 通过分析与研究当前 Web 服务器集群的负载均衡技术和调度算法，提出了一种新的基于实时性能动态反馈的负载均衡算法设计，算法引入了请求量化方法、实时性能指标和准入控制机制，实验结果表明本算法具有较好的低响应延迟和高吞吐率性能。

关键字: Web 服务器集群；负载均衡技术；调度算法；实时性能

A Load Balancing Algorithm Based on Real-Time Performance Dynamic Feedback

LIANG Biao, HUANG Zhan

(Department of Computer Science, Jinan University, Guangzhou 510632, China)

Abstract: This paper first analyzes the current load balancing technologies and dispatch algorithm, and then presents a new dynamic load balancing algorithm based on real-time performance dynamic feedback. In this algorithm, a method of measuring the request, a real-time performance index and the admission control mechanism are utilized. The experimental result indicates the this algorithm has preferable performance in low response delay and high output rate.

Keywords: cluster Web servers; load balancing; dispatch algorithm; real-time performance

1 前言

如今随着电子商务、网上银行、网络游戏、网上购物等 Web 应用的不断扩展，使得服务站点必须应对更多的动态请求，服务器资源开销剧增。当单台服务器不再能应对日益增加的服务请求的时候，可以用价格更昂贵性能更好的服务器代替，但从可伸缩性、高可用性和性价比方面考虑，集群技术是解决这个问题的首选方案[1]。

负载均衡问题是集群系统的核心，本文将主要对此问题进行研究。文章的组织结构如下：第一部分介绍 Web 服务器集群的负载均衡技术和调度算法，第二部分提出一种新的动态负载均衡算法，第三部分对该算法进行性能分析，最后在结论中对文章进行总结。

2 相关研究

根据调度器工作在 OSI 第几层，可将其分为 4 层调度器和 7 层调度器。

2.1 基于 4 层调度器的负载均衡技术和调度算法

4 层调度器的负载均衡技术工作机制为：当请求到达的时候，根据一定调度算法选择一台服务器与请求客户端建立 TCP 连接，该服务器将直接处理该客户的后续请求，请求的内容对调度器透明。具体实现技术有[2]：

(1) 网络地址转换(NAT)

(2) IP 隧道(IP tunneling)

(3) 直接路由(Direct routing)

根据调度算法考虑系统状态或请求信息与否，可将其划分为静态调度算法和动态调度算法。4 层调度器可使用静态调度算法和内容屏蔽的动态调度算法。

(1) 静态调度算法

静态调度算法一般只工作在 4 层调度器，不考虑任何系统状态信息。静态算法典型代表有 Round Robin(RR)、Best-Fit(BF)、Round Robin Next-Fit (NF)，Weight Round Bobin(WRR)。静态调度算法有

① 收稿时间:2009-06-15

简单性、高效性和可靠性的优点，但由于网络请求的差异太大，静态算法很容易导致服务器负载失衡。

(2) 内容屏蔽的动态调度算法

4 层调度器其动态调度算法可以参考客户端的信息局限在 IP 数据包源地址和 TCP 端口号；更多的研究集中在收集服务器端负载信息。收集的信息可分为三类：输入指标、服务器指标、响应指标。典型代表有动态 WRR 算法，最小连接数(LCS)，加权最小连接数(WLCS),基于局部性的最小链接等。

采用内容隐蔽的动态调度算法调度器开销较小，调度效率高。但 Web 服务器负载波动性大，前一时刻收集的负载信息在下一时刻也许不再能够准确反映服务器负载状态[1]，从而使基于服务器端负载状态信息的动态调度算法失效。

2.2 基于 7 层调度器的负载均衡技术和调度算法

当请求到达的时候，7 层调度器与请求客户端建立 TCP 连接，然后分析请求内容，最后根据调度算法将 TCP 连接交接到目标服务器。其负载均衡技术的典型代表有[3]：

(1) TCP 网关(TCP gateway)

(2) TCP 接合(TCP splicing)

(3) TCP 切换(TCP handoff)

(4) TCP 连接跳跃(TCP connection hop)

基于 7 层调度器的动态调度算法可以参考请求内容的信息，再结合客户端和服务器端信息可以制定出复杂的调度算法。基于内容感知的动态调度算法典型代表有 CAP，LARD。CAP 根据请求主要开销的服务器资源的不同对其进行分类，防止主要消耗某类系统资源的请求集中被分发到某台服务器，使各服务器的各类资源得到充分利用从而实现负载均衡[4]。LARD 将不同服务内容存放于不同服务器，把请求内容相同的所有请求转发到相同服务器以增加 Cache 命中率[5]。

7 层调度器建立连接、转交连接和请求分析等工作资源开销很大，使其容易成为系统的瓶颈。调度器的瓶颈问题将严重影响整个系统的吞吐率和可伸缩性。另一方面，请求分类需要人工预先处理，而且当服务变更的时候这种工作又得从头再来，灵活性差。

3 算法设计

基于以上对 Web 集群负载均衡策略的研究和分析，我们结合 4 层调度机制和 7 层调度机制的优点设计了一种新的基于内容感知的负载均衡算法，我们称之为基于实时性能动态反馈的负载均衡算法(Real-time performance dynamic feedback based Request Dispatcher, RRD)，信息流如图 1 所示。

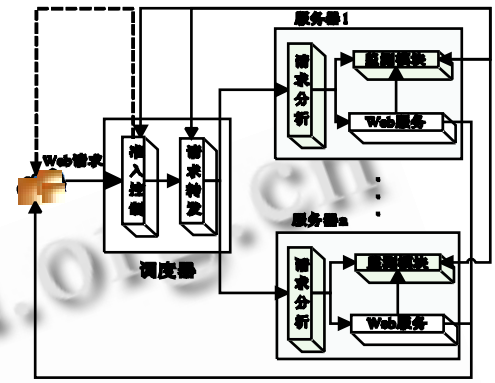

图 1　请求与响应信息流

3.1 模块设计

请求分析模块：对到来的请求进行分析，按请求内容的不同将其分为 m 类，每一类对应一个权值 Wi，并以此来量化请求，大小表示其对服务器资源占用的多少和请求服务时间的长短，其值可单独运行测试设定。

监测模块：收集并计算服务器中的等待响应请求总量和实时性能指标信息。设 Nij 表示服务器 Si 中第 j 类等待响应请求的数量，SUMi 表示服务器 Si 的等待响应请求总量，n 表示服务器的数量，令

$$SUMi = \sum_{j=1}^{m} Nij * Wj$$

$$SUM = \sum_{i}^{n} SUMi$$

设 Ni 表示服务器 Si 等待响应请求的数量，Ri 表示服务器 Si 等待响应请求的总量；Ci 表示服务器 Si 等待响应的请求占系统总的等待响应请求的百分比，令

$$Ri = \sum_{i}^{m} Ni * Wi$$

$$Ci = Ri / SUM$$

设 Pi 表示服务器 Si 硬件性能权值，其大小可根据服务器运行服务器性能测试软件(如 Loadrunner 等)设置，CLi 表示服务器 Si 实时性能指标，令：

$$\sum_{i}^{n} Pi = 1$$

监测模块将SUM和CLi周期性的分别提交到调度器的准入控制模块和请求转发模块并将 SUMi 通知其他服务器。

准入控制模块：根据当前系统中的请求总量值SUM，当其达到预定的上限值时拒绝请求，否则接受请求并将其转交负载均衡模块；其上限值可以根据实际应用测定然后再设置。

负载均衡模块：周期性的收集各个服务器中实时性能指标数据 CLi，据此作为各个服务器的连接权值。系统初始时 CLi 均设为服务器硬件性能权值，CLi 为 0 表示服务器不再接受请求或是已停机，直到下次信息反馈其值不为 0。系统采用 WRR 算法选择服务器转发请求。

3.2 主要数据结构设计

首先给出几个相关的定义：服务器集群集合 S={S1,S2,…,Sn}；用户请求集合 R={R1,R2,…,Rm}；服务类型集合 T={T1,T2,…,Tp}；Max 表示系统访问量阈值。

用户请求等待队列(RQ)：用于存放进入系统的 IP 请求报文。

后端服务器链表(SL)：每一个结构代表一个服务器，结构成员包括 IP 地址、MAC 地址、实时性能指标数据 Cli。

请求类别二维数组(RTA)：记录应用对应的各类 URL，以及不同类别对应的开销量化权值。

等待请求信息 Hash 表(WRT)：记录不同请求类别的等待总数。

3.3 算法的伪代码描述

程序的主流程：

```
While (true) {
    Fetch a request Ri from RQ;
    If (SUM>=Max) then {
      Abandon Ri;
      Continue;
    } else {
          Select target server Si from SL for Ri;
          Route Ri to the Si;
          Classify Ri as Ti according RTA;
          Plus WRT according Ti;
          Deal with Ri;
          Minus WRT according Ti;
          Response to client;
          Continue;
          }
}
```

守护进程：

```
While (true) {
    If (WRT come from other servers) then
    Calculate and set SUM;
    If (period is lapsed) {
    Sent WRT to other servers;
    Calculate and set CLi;
    }
}
```

4 系统实验

4.1 实验设置及结果

我们采用动态 WRR 和本文的 RRD 算法做比较。实验硬件环境为 5 台通过百兆以太网连接的 PC，其中作为后端服务器的 4 台 PC 配置为：Intel 奔 4 2.8GHZ，512M DDR 内存，硬件性能权值均为 0.25；另一台配置为：Intel 奔 4 3.0GHZ，2G DDR 内存，作为调度器；操作系统为 Linux kernel 2.6，运行 Apache 2.0 服务 web 请求，一台客户机运行 RUBis 模拟请求的发送；把请求分为静态页面请求、单动态请求和复杂动态请求三类，权值分别设为 0.1，0.3 和 0.6。实验结果如图 2 图 3 所示。

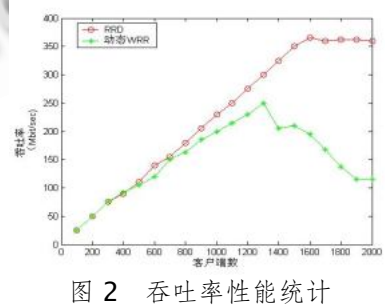

图 2 吞吐率性能统计

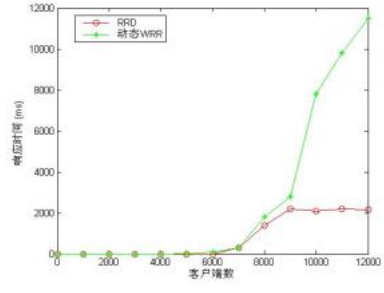

图 3 响应时间性能统计

4.2 实验分析

从图 2 发现，当客户端数量到达 300 个的时候 RRD 开始表现出吞吐率的优势，动态 WRR 最高吞吐率出现在 1300 个客户请求的时候，而 RRD 对应值为 1600，并且得益于准入控制机制，当请求量大于此最大值时吞吐率并没有像动态 WRR 一样显著下降。

由图 3 可知，当客户端数量到达 5000 个的时候动态 WRR 的响应时间开始显著上升，而因为 RRD 额外的系统开销较小当客户端请求数到达 6000 时响应时间才开始缓慢上升，并在 9000 处处于稳定。

由此可知，算法 RRD 确实在 Web 服务器集群中在吞吐率和响应延迟方面优于动态 WRR。

5 总结

Web服务器集群是目前高性能网络服务器的主要架构方法，而集群的负载均衡策略是提高集群整体性能的关键。本文提出了一种集群负载均衡的创新算法，算法采用 4 层调度器调度请求，后端服务器对请求分析，结合了 4 层调度机制的调度高效性和 7 层调度机制的请求内容可知性；服务器负载的计算采用硬件性能和请求总量相结合的实时性能指标，降低了网络负载，减少了负载计算的复杂性，提高了负载信息的准确度并且结合了动态反馈和准入控制机制，实验表明算法具有较好的低响应延迟、高可伸缩性和高吞吐量性能表现。

参考文献

1 Sharifiana S, Motamedi SA, Akbari. MK. A content-based load balancing algorithm with admission control for cluster web servers. Future Generation Computer Systems, 2008,24:775－787.

2. Zhang WS. Linux virtual server Web site. [2002-04-10] http: //www.linux virtual server.org, 2002.

3. Cardellini V, Casalicchio E, Colajanni M, Yu PS, The state of the art in locally distributed Web-server systems. ACM Computing Surveys(CSUR),2001,2(34): 263－311.

4 Casalicchio E, Tucci S, Static and dynamic scheduling algorithms for scalable web server farm. Ninth Euromicro Workshop on Parallel and Distributed Processing. Feb. 2001:369－376.

5 Pai VS, Aron M, Banga G, Svendsen M, Druschel P, Zwaenepoel W, Nahum E, Locality-aware request distribution in cluster-based network servers. Proc. of 8th ACM Conf. on Architecture Support for Programming Languages. San Jose, CA, Oct. 1998:568－596.