一种基于优先级队列的集群动态反馈调度算法

柳少锋，董 剑，吴智博

（哈尔滨工业大学 计算机科学与技术学院，哈尔滨 150001）

摘 要：在分析现有面向LVS集群的负载均衡调度算法优缺点的基础上，提出了一种新的调度算法—基于优先级队列的动态反馈调度算法。该算法根据定期采集到的各服务器负载信息动态地调整各服务器的权值，并根据权值建立优先级调度队列借以实现连接的调度。算法可保证良好的负载均衡性，且时间复杂度降低至O（1）。

关键词：集群；负载均衡；LVS；调度算法；动态反馈

中图分类号： TP311 **文献标识码：** A **文章编号：**2095-2163（2012）04-0078-04

A Dynamic-feedback Scheduling Algorithm for Cluster Load Balancing based on Priority Queue

LIU Shaofeng, DONG Jian, WU Zhibo

(**School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China**)

Abstract: This paper, for a start, analyzes the strengths and weaknesses of currently existed scheduling algorithms. Based on that, a new algorithm is presented. The algorithm adjusts the weight of each real server according to load data collected periodically, and it builds priority queues on the basis of the adjusted weights. This algorithm gives a low time complexity of O(1) while keeping load well balanced.

Key words: Cluster; Load Balancing; LVS; Scheduling Algorithm; Dynamic-feedback

0 引 言

互联网技术的快速发展伴随着网络用户的迅速增加，不断给网络服务器的性能带来越发巨大的挑战。集群服务器系统因其具有扩展性好、可用性和性价比高的特点，已经受到学术界和工业界的广泛关注[1]。各种商业的或非商业的集群系统相继出现。其中由国防科技大学的章文嵩博士发起的LVS集群系统[2]，因其开放性及出色的性能在全球范围内得到大量应用[3]。

集群服务器系统通常由前端服务器（任务调度机）与后端真实服务器池组成[4]。前端服务器是客户访问集群服务器的唯一入口，通过特定的调度算法将收到的客户请求数据包分配给内部的真实服务器。选择一个良好的调度算法以保证内部真实服务器负载的均衡性，避免出现个别服务器的偏载甚至过载，同时保证调度的执行效率，对于集群系统至关重要。调度算法的优劣直接影响了系统性能的好坏[5]。事实上，负载均衡调度属于并行系统调度的一个特例，而并行系统的调度问题是NP难解的，因此无法在合理的时间内找到最优解[5]。目前的负载均衡调度算法多是启发式的，不同的算法在不同环境下表现差异很大。如何找到一个尽可能优异的算法是目前学术界的一个研究热点。自集群技术出现至今，已有多种算法被相继提出。以LVS集群为例，目前常用的面向同构集群（各服务器提供相同的服务集）的基本调度算法主要有[5]：轮转调度（Round-Robin Scheduling）、加权轮转调度（Weighted Round-Robin Scheduling）、最小连接调度（Least-Connection Scheduling）和加权最小连接调度（Weighted Least-Connection Scheduling）。这几种算法都属于静态算法，均不考虑各服务器的当前负载。此外，还有文献[6]提出的动态反馈负载均衡算法（Dynamic-feedback load balancing），在调度时考虑各服务器实时的负载状况，属于动态算法。

这些算法都是在内核中实现的，其调度粒度都是面向连接的。本文将针对上述的5种算法分别作出分析，并在此基础上给出一种新的调度算法—基于优先级队列的动态反馈负载均衡调度算法。

1 LVS集群负载均衡调度算法概述与分析

所谓负载均衡调度，包含两层含义：一是调度，即将收到的用户访问请求分配给内部的真实服务器，由其作出应答；二是负载均衡，即在进行调度时要尽力保证各个真实服务器不出现过载。

一个负载均衡调度算法的优劣主要从两方面来评价：一是调度的负载均衡性，即保持各服务器负载均衡的能力。由于用户请求的服务时间变化很大，集群在运行过程中很容易出现负载的不平衡现象。对这一问题处理不好，将很容易导致用户的连接请求延迟甚至无法得到应答。二是调度的执行效率。前端服务器（即负载调度器）作为集群系统的唯一入口，其CPU资源是非常宝贵的，特别是对于VS/NAT型的集群，前端服务器还要处理对系统所有数据包的转发。如果算法的执行效率不够高，将很容易使负载调度器成为

收稿日期：2012-06-12
基金项目：国家自然科学基金（61100029）；哈尔滨工业大学优秀青年教师培养计划（HITQNJS.2009.053）；科技部国防科技合作计划（2010-DFA14400）；科技部国家科技支撑计划（2011BAH04B03）。
作者简介：柳少锋（1985-），男，河北宁晋人，硕士研究生，主要研究方向：计算机容错、集群计算；
董 剑（1978-），男，山东章丘人，博士，副教授，主要研究方向：计算机系统结构、容错计算、分布式计算等；
吴智博（1954-），男，黑龙江哈尔滨人，博士，教授，博士生导师，主要研究方向：容错计算、普适计算与移动计算等。

整个系统的性能瓶颈。

对上文提到的 5 种负载均衡调度算法简要分析如下：

（1）简单轮转调度。简单轮转调度算法就是以轮转的方式将客户连接请求分配给各真实服务器[5]。这种算法实现最为简单，且算法的时间复杂度为 O（1）。但该算法不区分各真实服务器的性能差异，也不考虑各服务器的当前状态。当各真实服务器性能差异较大，并且当请求服务时间变化较大时，这种算法有可能导致各服务器间负载的严重不均衡。

（2）加权轮转调度。运用这种算法，需要由系统管理员根据各内部真实服务器的性能设置一个静态的权值，然后算法以轮转的方式选择出一个合适的真实服务器，并保证各服务器所收到的连接请求数正比于其权值[5]。这种算法的时间复杂度为 O（n），其中，n 代表系统的规模，即真实服务器的数量。这种算法考虑了服务器间的性能差异，负载均衡能力强于简单轮转调度，但却仍然忽略了各服务器的当前状态，并且时间复杂度较高，当系统规模较大时，可能会带来较高的调度延时，增大系统开销以致降低影响系统性能。

（3）最小连接调度。调度器记录各真实服务器的连接数，算法执行时以轮转的方式查找连接数最小的服务器并为其调度当前请求[5]。类似于轮转调度，该调度也没有考虑各服务器的性能差异，并且时间复杂度也是 O（n）。

（4）加权最小连接调度。类似于加权轮转调度，由系统管理员为每个服务器根据其性能设定一静态的权值。该算法在调度新连接时尽可能使服务器的当前连接数与其权值成比例。该算法的负载均衡能力要优于前三种算法，但是时间复杂度也为 O（n）。

（5）动态反馈负载均衡调度。这一算法会根据服务器的实时负载，动态地调整各服务器的权值。在此基础上，调度采用加权轮转调度或者加权最小连接调度。研究表明，基于动态反馈的调度算法在表现上优于其他算法。但是由于调度采用的是加权轮转调度或者加权最小连接调度，算法的时间复杂度仍为 O（n）。因此当系统规模较大时，算法也会带来较高的系统开销。

可见，以上 5 种算法中，除简单轮转调度算法外，其他 4 种算法的时间复杂度都是 O（n）。然而，有研究表明，简单轮转调度算法的综合性能最差，而动态反馈负载均衡算法优于现有的其他不考虑服务器动态负载的算法。本文所提出的负载调度算法将基于现有的动态反馈负载调度算法并结合加权轮转调度，通过加入优先级队列使算法的时间复杂度降至 O（1）以达到更优异的性能表现。

2　基于优先级队列的动态反馈负载均衡调度算法

动态反馈负载均衡算法考虑了各服务器的动态负载状况，使其获得了更好的性能。本文所提出的算法吸收了这一优势，并通过建立优先级调度队列，使得算法的时间复杂度降至 O（1）。

2.1　动态反馈

文中，动态反馈的含义是指各真实服务器动态反馈给负载调度器的负载信息。这些负载信息是通过一个专门的守护进程 Monitor Daemon 进行收集的。其工作环境如图 1 所示。

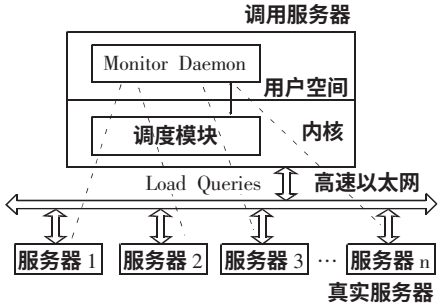

图 1　基于动态反馈的负载均衡算法工作环境

由图 1 所可知，Monitor Daemon 周期性（周期 T 为 5~20 秒）地对各真实服务器进行轮询以采集其负载信息。所采集的负载信息包括 CPU 利用率、内存可用量、进程数和服务响应延时。

除了动态反馈得到的负载，还可以在调度程序内统计两次轮询期间各服务器所收到的连接请求数。

2.2　动态权值的计算

接下来，将通过这些动态的负载信息计算出一个动态负载权值 wd。最终的权值为原权值与动态负载权值的和。

考虑一组服务器 $S = \{S_0, S_1, S_2, \cdots, S_{n-1}\}$，以向量 $\boldsymbol{LOAD}_i = (CONNECTION_i, CPU_i, MEMORY_i, PROCESS_i, RESPONSE_i)$ 表示服务器 S_i 的负载。其中，$CONNECTION_i$ 表示服务器 S_i 在两次轮询期间所收到的连接数 N_i 与各服务器收到的连接数平均值的比值，其公式为：

$$CONNECTION_i = \frac{N_i}{\sum_{m=0}^{n-1} N_m / n} \tag{1}$$

CPU_i 的计算公式为：

$$CPU_i = \frac{RATE_i}{\sum_{m=0}^{n-1} RATE_m / n} \tag{2}$$

其中，$RATE_i$ 表示服务器 S_i 的 CPU 利用率。

$MEMORY_i$ 的计算公式为：

$$MEMORY_i = \frac{AVAILABLE\_MEMORY_i}{\sum_{m=0}^{n-1} AVAILABLE\_MEMORY_m} \tag{3}$$

其中，$AVAILABLE\_MEMORY_i$ 表示 S_i 的可用内存大小。

$PROCESS_i$ 和 $RESPONSE_i$ 的计算方法也类似于 $CONNECTION_i$，公式直接列出如下：

$$PROCESS_i = \frac{PROCESS\_COUNT_i}{\sum_{m=0}^{n-1} PROCESS\_COUNT_m} \tag{4}$$

$$RESPONSE_i = \frac{RESPONSE\_DELAY_i}{\sum_{m=0}^{n-1} RESPONSE\_DELAY_m} \tag{5}$$

现在，引入负载权值向量 $WEIGHT_i = (1-CONNECTION_i, 1-CPU_i, MEMORY_i-1, 1-PROCESS_i, 1-RESPONSE)$。该向量的各个维度表征了相应负载对动态负载权值的贡献。

由于各类负载对服务器的影响程度是不同的，且不同

种类的服务也有其特点，因此再引入权值系数向量 $\boldsymbol{R}=(R_{con}, R_{cpu}, R_{mem}, R_{proc}, R_{resp})$，且有关系 $R_{con}+R_{cpu}+R_{mem}+R_{proc}+R_{resp}=1$。该向量可由系统管理员根据实际情况动态地设定。例如 $\boldsymbol{R}=(0.1, 0.4, 0.1, 0.1, 0.3)$。服务器 S_i 的动态负载权值 wd_i 计算公式如下：

$$wd_i=[1-CONNECTION_i,\ 1-CPU_i,\ MEMORY_i-1,\ 1-PROCESS_i,\ 1-RESPONSE_i]\cdot\begin{bmatrix}R_{con}\\R_{cpu}\\R_{mem}\\R_{proc}\\R_{res}\end{bmatrix} \tag{6}$$

$$=R_{con}(1-CONNECTION)+R_{cpu}(1-CPU_i)+R_{mem}(MEMORY_i-1)+R_{proc}(1-PROCESS_i)+R_{resp}(1-RESPONSE_i)$$

最终的动态权值为：

$$w_i \leftarrow w_i+\lambda\cdot wd_i$$

即，每个周期 T 内根据采集到的服务器负载信息对权值做一次调整，如果某服务器出现偏载（负载高于平均水平），权值就会被削弱，否则被提高。其中，λ 为负载权值系数，也可由系统管理员动态设定。在最初服务器被加入集群时，系统管理员为其设定一个初始权值 $DEFAULT\_WEIGHT_i$（当服务器 S_i 不可用时，$DEFAULT\_WEIGHT_i$ 为零，该服务器不接受调度）。

为避免动态权值变得很大，将 w_i 的取值区间限定为 $[\max(1, DEFAULT\_WEIGHT_i/4),\ DEFAULT\_WEIGHT*4]$。

2.3 优先级调度队列

根据新采集的负载信息算出新的动态权值之后，将根据新权值建立优先级调度队列。

假定服务器 S_i 的调度信息存储在数据结构 SERVER 中，定义下列数据结构作为调度队列的节点：

```
struct QUEUE_NODE
{       int       c;
        SERVER*   server;
        struct QUEUE_NODE* next; };
```

定义调度队列头结点数据结构：

```
struct QUEUE_HEAD
{       int  c;
        struct QUEUE_NODE* first;
        struct QUEUE_HEAD* next; };
```

令 $\lfloor c_i=10*w_i \rfloor$ 作为负载调度的参考权值，同时作为调度队列的优先级。可按照下列过程建立优先级调度队列：

（1）定义数组 QUEUE_HEAD ***hv*** [MAX[c_i]]，并将各元素 c 字段初始化为数组下标值，*first* 和 *next* 字段初始化为 NULL。

（2）定义数组 QUEUE_NODE ***nodes***[n]。依次扫描服务器调度信息表，将 ***nodes***[i]的 c 字段初始化为 c_i，server 字段初始化为指向 S_i 的调度信息数据结构。如果 $DEFAULT\_WEIGHT_i>0$，则将 ***nodes***[i]插入到 ***hv***[c_i]指向的调度队列中。重复第（2）步直至扫描完 S_{n-1}。

（3）扫描数组 ***hv***，将其中建立起的调度队列头部组织为一个链表，该链表的表头为具有最高 c 值即最高优先级的调度队列头。

假设有一组真实服务器 $S=\{S_0, S_1, \cdots, S_8\}$，在某一时刻其动态权值向量为 $\boldsymbol{W}=(3.5, 1.8, 4.7, 2.3, 1.8, 2.3, 1.8, 3.5, 2.6)$，则按照上述过程建立起来的优先级调度队列如图 2 所示。

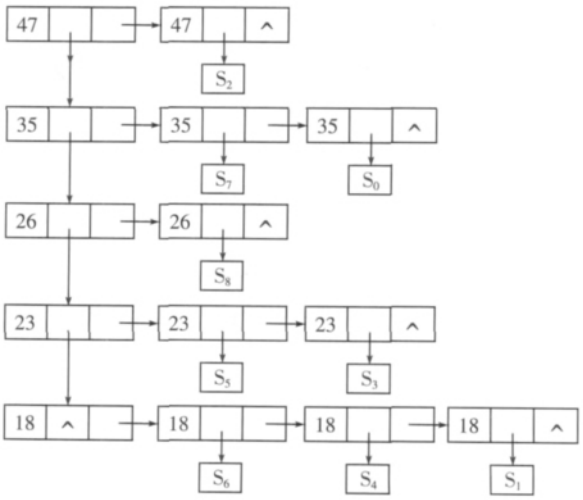

图 2 一个优先级队列的例子

图 2 中，节点 S_i 代表描述服务器 S_i 的调度信息的数据结构。

2.4 调度算法

定义优先级调度队列头链表的头指针为 h，指针 p 指向上次选择的服务器，对其初始化为 h->*first*。调度算法描述伪代码如下。

```
cur = p->next;
p = cur;
    if (h->c==0)
    return NULL;
if (p->next==NULL)
{     h->c--;
    if (h->next!=NULL && h->c==h->next->c)
        {合并 h->first 和 h->next->first 指向的调度队列}
} return cur->server;
```

为降低开销，各节点的内存都是静态分配并存放在数组中，不需要动态地申请或释放。分析以上算法，很容易得出其时间复杂度为 O（1）。

随着算法的执行，高优先级的调度队列会不断降级，直至与其下一级队列优先级相等并与其合并。这样，最终各优先级队列将合并为一个大的调度队列。当优先级降级至 0 时，算法返回 NULL。此时将恢复原来的优先级队列（可通

（下转第 85 页）

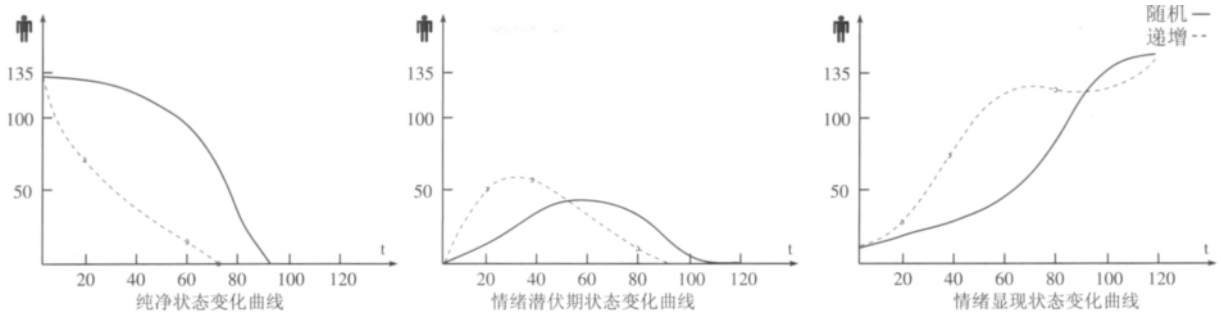

图 8 三种情绪状态的智能体数量

群体行为的影响作用，但是未考虑现实中情绪传染经历的变化，如情绪的形成期、高潮期、衰退期，本系统在未来将针对这个问题做进一步的改进。

参考文献：

[1] GUSTAVE L B. The crowd: a study of the popular mind. USA：Dover publications, Inc., 2002:2-8.

[2] COSTA P T, MCCRAE R R, A I P. Revised NEO personality inventory (NEOPIR) and neo five-factor inventory (NEO-FFI). Psychological Assessment Resources, 1992.

[3] TSAI J, BOWRING E, MARSELLA S, et al. Empirical evaluation of computational emotional contagion models[J]. Intelligent Virtual Agents, 2011:384-397.

[4] LAZARUS R S. Emotion and adaptation[M]. USA: Oxford University Press, 1991:127-129.

[5] REYNOLD C W. Flocks, herds and schools: a distributed behavioral model[C]// Proc. of the 14th annual conference on Computer graphics and interactive techniques, 1987:25-34.

[6] TU X, TERZOPOULOS D. Artificial Fishes: physics, locomotion, perception, behavior[C]// Proc. of the 21st annual conference on Computer graphics and interactive techniques, 1994: 43-50.

[7] HELBING D, MOLNAR P. Social force model for pedestrian dynamics. Physical Review, 1995:4282-4286.

[8] MUSSE S R, THALMANN D. A model of human crowd behavior: Group inter-relationship and collision detection analysis[C]//Proc. of Computer Animation and Simulation of Eurographics, 1997:39-51.

[9] 维基百科-五大性格特质. zh.wikipedia.org/zh/五大性格特质, 2012.

[10] GRANOVETTER M. Threshold models of collective behavior[J]. American Journal of Sociology, 1978:1420-43.

（上接第 80 页）

过创建一份整个优先级队列的拷贝实现，此时调度时间会有一个小峰值，称之为毛刺现象）。在一个这样的过程中，服务器 S_i 所得到的连接请求数与其动态权值是成正比的。这一点可以通过以下的分析证明：

在每个优先级队列内，调度是通过轮转进行的。每当完成一次轮转，队列内各服务器被调度一次，且队列降级一次。因此参考权值为 c_i 的服务器参与的降级次数为 c_i 次，从而其得到的连接请求数 N_i 也为 c_i。而 c_i 与 w_i 成正比，故 N_i 与 c_i 成正比。这样就保证了算法的负载均衡能力。

此外，当下一次 Monitor Daemon 负载采集发生时，调度队列将被重建。在重建完成后（而非完成前）替换原来的优先级队列。这句话的涵义在于，在重建优先级调度队列的过程中，调度程序仍可继续进行。这是因为内核中的负载调度进程具有更高的进程优先级。

3　结束语

调度算法的选择，对集群系统的性能起着至关重要的作用。本文分析了现有通用集群负载均衡调度算法的优缺点，并在此基础上提出了一种新的调度算法，即基于优先级队列的动态反馈负载均衡调度算法。该算法采用动态反馈机制以达到尽可能高的负载均衡效果，并且具有 O（1）的时间复杂度，因而使其同时具有较以往调度算法更高的执行效率。这有助于提高集群系统的吞吐率，增强其性能。

参考文献：

[1] SCHROEDER T, GODDARD S, RAMAMURTHY B. Scalable Web server clustering technologies [J]. IEEE Network, 2000, 14 (3): 38-45.

[2] ZHANG Wensong, JIN Shiyao, WU Quanyuan. LinuxDirector: A connection director for scalable Internet services[J]. Comput.Sci& Technol. 2000,15 (6) :560-571.

[3] MACK J. LVSDocument [EB/OL] . http: //www. linuxvirtualserver.org, 2003.

[4] ZHANG Xiaolan, BARRIENTOS M, CHEN J, et al. HACC:An Architecture for Cluster-based Web Servers[C]// Proceeding of the 3rd USENIX Windows NT Symposium, Seattle, Washington, 1999: 155-164.

[5] GRUDENIC, BOGUNOVIC N. Computer cluster scheduling algorithm based on time bounded dynamic programming[J]. IEEE Network, 2011:23-27.

[6] 周集良，彭小宁，等. 基于集群的负载平衡调度算法研究与实现[J]. 计算机工程，2005，31（12）：108-110.

[7] 王晋鹏，潘龙法，李降龙. LVS 集群中的动态反馈调度算法[J]. 计算机工程，2005，31（19）：40-42.

[8] 唐丹，金海，张永坤. 集群动态负载平衡系统的性能评价[J]. 计算机学报，2004，27（6）：803-811.

[9] 章文嵩. LVS 集群的负载调度 [Online]. http: //linuxvirtualserver.org, 2002，5.