第 21 卷 第 3 期　　控 制 与 决 策　　2006 年 3 月
Vol. 21 No. 3　　Control and Decision　　Mar. 2006

文章编号: 1001-0920(2006)03-0319-04

一种基于预取的集群服务器调度算法

燕彩蓉, 沈钧毅, 彭勤科
(西安交通大学 软件所, 西安 710049)

摘　要: 针对集群服务器LARD 调度算法只能利用已有缓存的问题, 提出一种基于预取的算法 Prefetch-LARD. 该算法从Web 访问日志中挖掘页面之间的转移概率, 建立马尔科夫链模型, 在调度请求时利用概率关系提前将下一次可能访问的文档从节点磁盘取到本地 cache 中, 提高了请求的缓存命中率; 算法还采用了加权的节点超载判断方法, 以提高集群节点的负载均衡度. 实验表明, 在同样的测试环境下, Prefetch-LARD 算法比LARD 算法的缓存命中率提高 26.9%, 系统的吞吐量相应提高 18.8%.

关键词: 集群服务器; 调度算法; 预取; 缓存命中率; 负载均衡
中图分类号: TP393　　**文献标识码**: A

A Scheduling Algorithm Based on Web Prefetching for Cluster Servers

YAN Cai-rong, SHEN Jun-yi, PENG Qin-ke
(Institute of Software, Xi'an Jiaotong University, Xi'an 710049, China. Correspondent: YAN Cai-rong, E-mail: cryan@mail.xjtu.edu.cn)

Abstract: To the problem that the scheduling algorithm of locality-aware request distribution (LARD) for cluster server can only make use of the existing node caches, an advanced algorithm based on Web prefetching, Prefetch-LARD, is proposed. By mining the transition probability between pages from Web access logs, the algorithm builds up a prefetching model based on Markov chain to fetch documents for next possible requests ahead from disks to caches. Furthermore, the algorithm adopts a weighted node choosing method to improve the load balancing metric among nodes. Experiments show that, Prefetch-LARD algorithm increases cache hit ratio up to 26.9% and the throughput up to 18.8% compared with LARD algorithm.

Key words: Cluster server; Scheduling algorithm; Web prefetching; Cache hit ratio; Load balancing

1 引　　言

基于集群技术的Web 服务器的典型结构是使用分发器将客户端请求转发给后端服务器(又称为节点), 让整个集群表现得象一个服务于同一 IP 地址的虚拟服务器. 分发器的调度对整个系统的性能起决定因素, 其方法主要分为两种: 1) 传统的基于负载均衡的调度, 包括RR, WRR, LC 和WLC 等, 不能识别请求的内容, 从而不能充分利用节点缓存; 2) 基于内容的调度, 包括LARD, Adaptive-LARD 等[1~5], 可以识别请求的内容, 能够利用缓存提高响应效率, 但只能利用已有的缓存内容, 不能提前获得未缓存内容.

网页预取技术能够很好地弥补缓存命中率不高的缺陷[6~9]. 本文提出将网页预取功能与LARD 调度方法相结合, 不仅充分利用已存在的缓存内容, 而且提前获得未缓存内容, 从而提高了缓存命中率和系统性能.

2 集群服务器调度算法

2.1 调度算法分析

LARD (Locality-aware request distribution)[2]

收稿日期: 2005-01-13; **修回日期**: 2005-03-21.
基金项目: 国家自然科学基金项目(60175015, 60373107).
作者简介: 燕彩蓉(1978—), 女, 湖北仙桃人, 博士生, 从事集群服务器的研究; 沈钧毅(1939—), 男, 江苏常熟人, 教授, 博士生导师, 从事数据挖掘的研究.

调度算法的基本思想是: 如果当前请求曾经被访问, 则将它转发给最近响应过的节点; 如果没有, 则选择一个负载最小的节点并转发, 充分地利用节点缓存. 图 1 是调度的一个示例

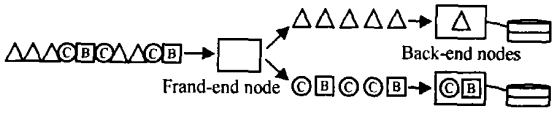

图 1 LARD 调度算法示例图

LARD 算法中, 不管请求的内容是否已被缓存, 均统一使用连接数来评价节点目前的负载情况. 但是, 如果内容已被缓存, 则响应的速度快, 即节点真实负载小, 所以连接数不是一个较好的负载评价尺度.

为了更精确地描述负载信息, Lei 等提出 Adaptive-LARD 调度算法[1], 将有缓存的和没有缓存的负载连接数区分开, 分别用 active-queue-len 和 disk-queue-len 表示, 即节点负载是由访问磁盘与访问 cache 的请求连接数组成, 而访问 cache 的时间要少得多, 因此要减少节点的负载, 则需要减小 disk-queue-len 的值. 节点负载 load 表示如下:

$$\begin{aligned} &\text{load} = \\ &\text{disk-queue-len} * 64 + \\ &\text{active-queue-len}. \end{aligned} \tag{1}$$

2.2 算法的改进

本文借鉴 LARD 算法的调度思想, 采用 Adaptive-LARD 的负载评价尺度, 并在此基础上加入了改进的节点超载判断方式. 对于节点超载的判断, 由于集群节点的处理能力可能相同, 也可能不同, 节点之间的负载均衡并不体现在提供相同的处理能力, 而是提供相同的处理能力比率. 因此, 本文提出加权的负载超载判断方法 $w_i (1 \le i \le n)$ 表示节点 i 的处理能力权值, tradeoff 参数的值可在 1 和 infinity 之间变化. 条件 f 成立则判断节点 n 超载, 重新选择负载最小的节点 m.

$$\begin{aligned} f: &n.\ \text{load} * w_m > \\ &m.\ \text{load} * w_n * \text{tradeoff}. \end{aligned} \tag{2}$$

同时针对由请求以前没有缓存的页面或以前处理此请求的节点超载引起的响应速度慢的问题, 本文采用了 Web 预取技术.

3 基于预取的调度

3.1 预取模型

集群服务器的性能很大程度上体现在分发器的调度上, 所以调度算法必须简单有效. 故本文使用简单而有效的基于马尔科夫链的预取方法[8,9].

一个离散的马尔科夫链模型可描述为 3 元组 $\langle S, A, \lambda \rangle$: S 表示状态空间, A 表示状态之间的转移概率矩阵, λ 表示状态 s 在状态空间 S 中的初始概率分布. 根据马尔科夫性质, 如果 $s[t]$ 表示 t 时刻的概率向量, 则有

$$s[t] = s[t-1]A. \tag{3}$$

马尔科夫链模型可用于 Web 页面访问序列建模, 此时状态就是访问的 URL. 使用最大释然方法评估 A, A 中每个元素 $A(s, s')$ 可计算为

$$A(s, s') = \frac{C(s, s')}{\sum_{s''} C(s, s'')}, \tag{4}$$

其中: $C(s, s')$ 是指训练数据中状态 s' 紧跟 s 被访问的次数, $A(s, s')$ 可解释成从状态 s 到 s' 的一步转移概率.

Web 服务器通常有一个日志文件, 记录每个用户的页面请求信息. 为了分析用户的访问模式, 需要对日志进行预处理. 按如下方法将 Log 文件整理成会话集: 1) 按用户的 IP 地址, 将 Log 文件分割成独立的访问记录集; 2) 将每个访问记录集的请求按时间排序, 设立时间窗口阈值 t_w, 分割访问记录集, 时间间隔小于 t_w 的相邻访问请求同属于一个用户会话. 然后根据会话集计算 $C(s, s')$ 以及转移概率矩阵 A.

对于集群服务器, 每个节点都有各自的状态转移次数, 分别设为 $C_1, C_2, \cdots, C_n$, 整个集群的转移概率矩阵为 A, 则有

$$A(s, s') = \frac{\sum_{i=1}^{n} C_i(s, s')}{\sum_{i=1}^{n} \sum_{s''} C_i(s, s'')}. \tag{5}$$

如果当前的请求为 s, 则选择 $A(s, s')$ 最大的 s 作为预取页面; 如果需要预取多个页面, 则通过计算从大到小获取.

3.2 Prefetch-LARD 算法

针对从磁盘访问文档负载大的特点, 本文在集群调度上添加预取功能, 提出 Prefetch-LARD 调度算法. 此算法的调度思想与 LARD 一致, 负载的评价尺度与 Adaptive-LARD 一致, 但修改了节点超载时的选择方式, 增加了页面预取模块, 以此提高缓存命中率和负载均衡度. 在预取命中率较高的情况下, 节点负载值可近似为 active-queue-len. 算法的调度部分与文献[1]类似, 但文献[1]没有预取功能. 算法整体描述如下:

算法 1 带有预取功能的请求调度 Prefetch-LARD.

输入: 请求与节点的映射集合 NodeSet; 节点磁

盘连接数组 disk- queue; 节点缓存连接数组 active- queue; 请求 r; 负载平衡参数 tradeoff; 节点集修改时间阈值 K; 节点的权值数组 w.

输出: r 的调度节点 n.

```
1) prefetch- thread do (r, NodeSet, tradeoff, w);
2) if (NodeSet[r. target]= NULL) then
3)    n,NodeSet[r. target]←{least loaded node};
4)    disk- queue[n]+ +;
5) else
6)    m ←{least loaded node};
7)    n←{least loaded node in NodeSet[r. target]};
8)    m1←{most loaded node in NodeSet[r. target]};
9)    if(n. load * w [m ]≤m. load * w [n] * tradeoff) then
10)      active- queue[n]+ +;
11)   else
12)      n←m;
13)      disk- queue[n]+ +;
14)   if |NodeSet[r. target]|> 1&&time()-
        NodeSet[r. target]. lastMod> K then
15)      remove m1 from NodeSet[r. target];
16) if NodeSet[r. target]was changed then
17)   NodeSet[r. target]. lastMod←time();
procedure prefetch- thread do (r, NodeSet, tradeoff, w)
1) m ←{least loaded node};
2) p←r. target. next();
3) if (NodeSet[p ]= NULL ||NodeSet[p ]. load * w [m ]> m. load * w [p ] * tradeoff) then
4)    NodeSet[p ]←m;
5) send prefetch command of p to node m.
```

Prefetch- LARD 的转发功能和预取功能是通过两个线程在分发器上并行执行的, 因此在不影响原有转发效率的基础上会提高后端服务器的响应效率. 但值得注意的是, 两个线程都要使用参数 NodeSet, 从而需要同步, 这会对并发效果产生一定的影响. 设 NodeSet 集合的大小为 s, 则调度的时间复杂度为 $O(s)$, 预取的时间复杂度也是 $O(s)$, 满足集群调度算法应简单的基本要求.

当请求结束时, 根据内容是否缓存的标志释放节点的连接数, 数组 disk- queue[n]减一或 active- queue[n]减一.

4　实　　验

4.1　实验设计

为了检验集群调度算法的性能, 从 http://kdd. ics. uci. edu 网站上获得用户在 1998 年 2 月某 1 周时间内对 Microsoft 网站的访问日志, 其中包含 98 654 个 URL 序列, 文件数目为 294. 将此日志分成两部分, 一部分作为训练集, 一部分作为测试集. 以如下几个定义作为算法性能评价指标:

定义 1　请求命中率 Hit ratio[7], 表示成功预取的用户请求数与用户提交的请求总数的比率.

定义 2　负载均衡度 LBM, 表示集群各节点负载的平衡程度. 与文献[1]的定义有所不同, 本文考虑了每个节点的处理能力权重. 取 m 个样本点, n 为集群节点数. 设 $\mathrm{load}_{i,j}(1\le i\le n, 1\le j\le m)$ 是节点 i 在第 j 个样本点的负载, $\mathrm{load}_{i,j}$ 是其单位负载; upeak- load$_j$ 是所有节点在第 j 个样本点的最高单位负载值; w_i 是各节点的处理能力权值. 则有

$$\mathrm{LBM}=\frac{\sum_{j=1}^{m}\sum_{i=1}^{n}\mathrm{uload}_{i,j}\Big/n}{\sum_{j=1}^{n}\left(\frac{\text{upeak- load}_j}{\left(\sum_{i=1}^{n}\mathrm{uload}_{i,j}\right)\Big/n}\times\frac{\sum_{i=1}^{n}\mathrm{uload}_{i,j}}{n}\right)},$$

即

$$\mathrm{LBM}=\frac{\sum_{j=1}^{m}\sum_{i=1}^{n}\mathrm{uload}_{i,j}\Big/n}{\sum_{j=1}^{n}\text{upeak- load}_j},$$

$$\mathrm{uload}_{i,j}=\frac{\mathrm{uload}_{i,j}}{w_i}.$$

定义 3　系统吞吐量 Throughput[2], 表示集群服务器系统每秒响应的字节数.

两台 PC 机作为物理客户机, 每个客户机模拟多个逻辑客户端, 客户机的配置不完全一致. 分发器与 5 个后台服务器配置一致, 均为 Dell 服务器(CPU: PⅢ1 G, RAM: 512 M, Cache: 256 kB), 另一个后台服务器的配置为 Dell 服务器(CPU: PⅢ 1 G, RAM: 1 000 M, Cache: 512 kB). 实验使用 Ziff Davis Media WebBench 4.1, 它通过模拟真实的网络工作环境对 Web 服务器进行测试.

4.2　实验结果

表 1 是 LARD, Adaptive- LARD 和 Prefetch- LARD (p 为 Markov 链模型预取的页面数) 调度算法的请求命中率 Hit ratio, 负载均衡度 LBM 和系统吞吐量 Throughput 的比较.

LARD (其中 $T_{high}=65$, $T_{low}=25$, $k=20$ s 是文献[2]在讨论之后获得的较为合理的参数值) 与

表 1 调度算法的性能比较

Scheduling algorithms	Hit ratio/%	LBM (w = 1, 1, 1, 1, 1.5)	Throughput (bytes/s, 40 clients)
LARD (T_{high} = 65, T_{low} = 25, k = 20s)	45.28	0.801	3 565 492
Adaptive-LARD (tradeoff = 2.5)	46.98	0.870	3 635 521
Prefetch-LARD (p = 1)	54.05	0.911	3 927 924
Prefetch-LARD (p = 2)	57.46	0.915	4 236 805
Prefetch-LARD (p = 3)	57.32	0.915	4 220 732

Adaptive-LARD 算法的调度方式基本相同, 所以请求命中率相差不多. Prefetch-LARD 算法在其基础上又添加了预取功能, Hit ratio 有较大提高, 而且在预取页面数为 2 时, Hit ratio 最高. 表 1 的数据表明, Prefetch-LARD 算法 (p = 2) 的 Hit ratio 比 LARD 高 26.9%, 比 Adaptive-LARD 高 22.3%.

Adaptive-LARD 与 Prefetch-LARD 采用的负载评价尺度相同, 但节点超载判断方式不同, 在服务器权值分别为 1, 1, 1 和 1.5 时, Prefetch-LARD 算法的 LBM 最高.

评价一个调度算法的总体尺度就是系统的吞吐量, 吞吐量越大, 调度效果越好. 从表 1 数据可知, Prefetch-LARD (p = 2) 算法获得的吞吐量最大, 相比 LARD 可提高 18.8%, 相比 Adaptive-LARD 可提高 16.5%.

5 结 论

本文针对 LARD 和 Adaptive-LARD 调度算法在请求命中率上存在的不足, 提出了一种基于预取的集群调度算法 Prefetch-LARD. 该算法采用基于马尔科夫链模型的预取方法, 简单而有效地预测出需预取的页面, 提高了调度的缓存命中率. 同时, 该算法继承了 Adaptive-LARD 的细粒度负载评价尺度, 更新了节点的超载判断方法, 使集群达到更好的负载均衡效果, 最终能获得更高的集群吞吐量. 在相同的测试环境中, Prefetch-LARD 相比 LARD 在吞吐量上可以提高 18.8%, 相比 Adaptive-LARD 可以提高 16.5%.

参考文献(References)

[1] Lei Y C, Gong Y L, Zhang S, et al. Research on Scheduling Algorithm in Web Cluster Servers[J]. *J of Computer Science and Technology*, 2003, 18(6): 703-716.

[2] Pai V S, Aron M, Banga G, et al. Locality-aware Request Distribution in Cluster-based Network Servers [A]. *Proc of the 8th Conf on Architectural Support for Programming Language and Operating Systems* [C]. San Jose, 1998: 205-216.

[3] Bunt R B, Eager D L, Oster G M, et al. Achieving Load Balance and Effective Caching in Clustered Web Servers[A]. *Proc of the 4th Int Web Caching Workshop* [C]. San Diego, 1999: 159-169.

[4] Ahn Woo Hyun, Kim, Woo Jin, et al. Content-aware Cooperative Caching for Cluster-based Web Servers[J]. *J of Systems and Software*, 2004, 69(1-2): 75-86.

[5] Bhinder R, Maheswaran M, Diamond J. Evaluation of Request Distribution Schemes for Web-server Clusters [A]. *Canadian Conf on Electrical and Computer Engineering* [C]. Canada: Institute of Electrical and Electronics Engineers Inc, 2004: 1609-1612.

[6] Crovella M, Barford P. The Network Effects of Prefetching [A]. *The IEEE Conf on Computer and Communications* [C]. San Francisco, 1998: 1232-1240.

[7] 许欢庆, 王永成. 基于用户访问路径分析的网页预取模型[J]. ***软件学报***, 2003, 14(6): 1142-1147.
(Xu H Q, Wang Y C. A Web Pre-fetching Model Based on Analyzing User Access Pattern[J]. *J of Software*, 2003, 14(6): 1142-1147.)

[8] Sarukkai R. Link Prediction and Path Analysis Using Markov Chains[J]. *Computer Networks*, 2001, 33(1-6): 377-386.

[9] Nanopoulos A, Katsaros D, Manolopoulos Y. A Data Mining Algorithm for Generalized Web Prefetching[J]. *IEEE Trans on Knowledge and Data Engineering*, 2003, 15(5): 1155-1169.