ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.7, July 2007, pp.1592−1602
DOI: 10.1360/jos181592

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于序列模式的Servlet容器缓存替换*

李 洋[1,2+], 张文博[1,2], 魏 峻[1], 钟 华[1], 黄 涛[1]

[1](中国科学院 软件研究所 软件工程技术中心,北京 100080)

[2](中国科学院 研究生院,北京 100049)

Sequential Patterns-Based Cache Replacement in Servlet Container

LI Yang[1,2+], ZHANG Wen-Bo[1,2], WEI Jun[1], ZHONG Hua[1], HUANG Tao[1]

[1](Technology Center of Software Engineering, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: Phn: +86-10-62661581 ext 630, Fax: +86-10-62661580, E-mail: fallingboat@otcaix.iscas.ac.cn

Li Y, Zhang WB, Wei J, Zhong H, Huang T. Sequential patterns-based cache replacement in Servlet container. *Journal of Software*, 2007,18(7):1592−1602. http://www.jos.org.cn/1000-9825/18/1592.htm

Abstract: Servlet cache can effectively improve the throughput of Servlet container and reduce the response time experienced by the users. But the cache effect is dependent on the hit rate determined by the cache replacement algorithms. Servlets represent some business functions, so mining the business association among Servlets can improve the hit rate of cache replacement algorithms which in turn exhances the performance of Servlet container consequently. However existing literatures such as LRU (least recently used), LFU (least frequently used), GDSF (greedy dual size frequency) rarely take into account the relationships between the Servlets. This paper denotes the business associations as sequential patterns of Servlet container, and presents a *k*-steps transfer probability graph to denote the access sequential patterns of Servlet container and designs a sequential patterns discovery algorithm KCTPG_Discovery. Two cache replacement algorithms KP-LRU and KP-GDSF are introduced based on the research of the sequential patterns of the Servlets. Comparing with the traditional algorithms such as LRU and GDSF, the experimental results confirm that the hit radio of the cache can be enhanced by using the above algorithms, the two algorithms effectively improve the performance of Servlet container.

Key words: Servlet caching; sequential pattern; sequential pattern discovery; cache replacement algorithm

摘 要: Servlet 缓存能够有效地提高 Servlet 容器的吞吐量,缩短用户请求的响应时间.然而,Servlet 缓存的性能受到缓存替换算法的影响.Servlet 容器中的 Servlet 对应着一定的业务功能,挖掘 Servlet 之间的业务关联来指导缓存替换算法的设计可以提高 Servlet 缓存的命中率,进而提高 Servlet 容器的性能.然而,目前常见的 LRU(least recently used),LFU(least frequently used),GDSF(greedy dual size frequency)等缓存替换算法均没有考虑上述问题.将 Servlet 对应的业务关联定义为 Servlet 容器序列模式,并提出 *k* 步可缓存转移概率图的概念加以表示,给出了序列模式发现算法 KCTPG_Discovery.最后,基于 Servlet 容器序列模式设计了缓存替换算法 KP-LRU(*k*-steps prediction least recently

* Supported by the National Natural Science Foundation of China under Grant No.60573126 (国家自然科学基金); the National Basic Research Program of China under Grant No.2002CB312005 (国家重点基础研究发展计划(973))

Received 2006-04-21; Accepted 2006-06-09

used)和 KP-GDSF(*k*-steps prediction least frequently used).实验结果表明,KP-LRU 与 KP-GDSF 算法比对应的 LRU 算法和 GDSF 算法具有更高的缓存命中率,有效地提高了 Servlet 容器的性能.

关键词: Servlet 缓存;序列模式;序列模式发现;缓存替换算法

中图法分类号: TP316 **文献标识码:** A

随着Internet的飞速发展,遵循J2EE规范实现、基于请求/应答模式的Web容器已经成为Web服务器的主流,它为Servlet 和JSP(Java server page)组件提供了运行时环境[1].然而,Web Performance公司 2004 年的Servlet性能报告结果显示,提高Servlet容器的性能仍是急需解决的问题[2].缓存是提高服务器性能的有效技术,它在Web服务器与代理服务器领域的成功应用,已经极大地提高了服务器的性能[3,4],应用Servlet缓存同样可以提高Servlet容器的性能.Turner等人注意到,Servlet容器中存在一些运行结果很少发生改变却被频繁访问的Servlet(如商品信息、股票交易历史等),缓存其运行结果可以提高Servlet容器的性能[5].WebSphere中已经采用基本的LRU(least recently used)替换算法实现了对Servlet运行结果的缓存机制[6].

图 1 解释了 Servlet 缓存的原理.在 Servlet 容器中,请求管理器(request manager)管理来自客户的请求(request),当客户请求到来时,请求管理器根据配置文件(configuration)判断请求的 Servlet 类型进行如下处理:(1) 若请求的是一个不可缓存 Servlet,那么它直接被交给对应的 Servlet 进行处理,处理后的结果由请求管理器包装成响应对象(response)返回给客户;(2) 如果一个可缓存 Servlet 被请求且对应的运行结果已经被缓存在存储(store)模块中,请求管理器直接将其包装成响应对象返回给客户;如果对应的运行结果没有被缓存,则请求管理器将其交给对应的 Servlet 进行处理,然后将最后的运行结果存储到存储模块中,同时将运行结果包装为响应对象返回给客户.

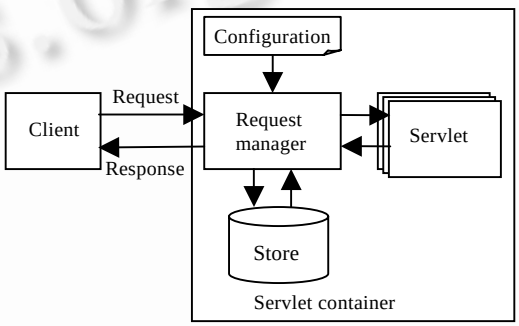


Fig.1 Servlet cache schema
图 1 Servlet 缓存机制

然而,受到存储空间的限制,需要将一些Servlet缓存对象替换出缓存,不同的缓存替换算法会影响缓存的命中率,进而影响到Servlet容器的性能.目前采用的主要缓存替换算法主要包括LRU,LFU(least frequently used),GDSF(greedy dual size frequency)等[7].

应用开发者设计的Servlet对应着一定的业务功能,用户使用这些Servlet的顺序会受到业务逻辑的影响,因此,挖掘Servlet之间的业务关联来指导Servlet缓存替换算法的设计可以提高缓存命中率.访问模式是一种发掘数据关联的有效技术,它首先被引入数据挖掘领域,通过分析数据库的访问模式来指导数据库的设计,提高数据库的性能[8].在Web挖掘领域的应用主要是通过分析服务器日志得到用户的访问模式以指导缓存对象的预取.主要的发现技术包括路径分析、关联规则、序列模式等[9].其中,序列模式得到了广泛的应用[10,11].本文将上述Servlet之间的业务关联定义为Servlet容器的序列模式,并提出*k*步可缓存转移概率图加以表示,然后给出了相应的序列模式发现算法KCTPG_Discovery(*k*-steps cacheable transition probability graph discovery),最后基于Servlet容器序列模式设计了缓存替换算法KP-LRU(*k*-steps prediction least recently used)和KP-GDSF(*k*-steps prediction least frequently used).实验结果表明,KP-LRU和KP-GDSF算法比对应的LRU算法和GDSF算法有更高的缓存命中率,有效地降低了Servlet容器的负载,提高了Servlet容器的性能.

本文的创新性主要在于:(1) 根据 Servlet 之间的业务关联,定义 *k* 步可缓存转移概率图表示 Servlet 容器的序列模式;(2) 设计了基于 *k* 步可缓存转移概率图的序列模式发现算法 KCTPG_Discovery;(3) 设计了基于 Servlet 容器序列模式的缓存替换算法 KP-LRU 和 KP-GDSF.

本文第 1 节介绍缓存替换算法以及序列模式的相关工作.第 2 节给出 Servlet 容器的序列模式定义以及序列模式发现算法.第 3 节描述基于序列模式的 KP-LRU 和 KP-GDSF 缓存替换算法.第 4 节是实验结果及其分析.

最后是本文的结论及对未来工作的展望.

1 相关工作

代理服务器中已经有多种缓存替换算法被成功应用.LRU算法选择最近最久未被使用的缓存对象替换出缓存[12].LFU算法选择被访问次数最少的对象进行替换[7].LRU-Threshold算法是LRU算法的扩展,它仅缓存文件尺寸小于threshold的缓存对象[13].GDSF算法根据替换代价函数(根据文件大小、访问次数等因素定义)从缓存中替换具有最小代价的缓存对象[14].然而,Servlet容器的缓存对象是动态的Servlet运行结果,具有业务逻辑相关的顺序访问关系,上述针对静态文件设计的算法未考虑Servlet的这种特点,难以满足Servlet容器的要求.本文的工作考虑了上述Servlet特点,并引入了预测概率函数来指导缓存替换,提高了缓存命中率.

序列模式作为一种典型的用户访问模式被成功应用于数据挖掘领域[15].Cooley等人在Web服务器中通过分析用户的序列访问模式来设计缓存系统[16],预测将要被访问的对象并提前将其载入缓存以提高缓存命中率.但动态生成的Servlet运行结果依参数不同而不同,无法实现提前载入.Qiang将Web序列访问模式分析作为设计缓存替换算法的依据,并设计了*n*-gram缓存替换算法来提高缓存命中率[14],但该算法仍旧面向静态文件设计,同时仅考虑了单步的顺序关系.与上述工作相比,本文的工作考虑了Servlet之间的多步顺序访问关系,能够更精确地反映Servlet之间的业务逻辑关联.

综上所述,在 Web 服务器和代理服务器中的相关缓存算法仍然面向静态文件设计,无法直接用于依参数变化的 Servlet 运行结果缓存.而一些考虑了序列模式的缓存替换算法仍局限于单步关系,不能反映多步的业务关联.本文首先提出了 *k* 步可缓存转移概率图表示 Servlet 容器的序列模式,在考虑了 Servlet 之间的多步业务关联的基础上给出了序列模式发现算法 KCTPG_Discovery.然后根据发现算法获得的 Servlet 容器序列模式设计了 KP-LRU 和 KP-GDSF 缓存替换算法.实验结果证明,上述缓存替换算法有效地提高了 Servlet 缓存命中率,提高了 Servlet 容器的性能.

2 Servlet 容器序列模式发现

如前所述,由于Servlet对应一定的业务功能,这使得客户对不同Servlet的访问存在一定的顺序关系,因此可以使用序列模式来表示这种关系.本节针对 Servlet 容器的特点给出 Servlet 容器序列模式发现的相关定义.

2.1 基本定义

Srikant举例描述了数据库中的序列模式:“5%的购买了《基础》的用户之后会购买《国家基础》和《第二基础》”[8].在Servlet容器中,我们可以用同样的方式来描述序列模式,即“有 15%的访问Servlet *A*的用户会继续访问Servlet *B*”.本小节给出Servlet容器序列模式的一些基本定义.

在 Servlet 容器中,仅有部分 Servlet 的运行结果能够被缓存,这些 Servlet 的运行结果仅与参数值有关,不依请求者变化.它们由用户定义,通常是一些运行结果很少发生改变却被频繁访问的 Servlet(如商品信息、股票交易历史等).缓存这部分 Servlet 的运行结果为多个请求者服务,使得不必每次都重新运行它们生成相同的结果.

定义 1. 可缓存Servlet.应用开发者根据业务逻辑在配置文件中定义的运行结果可以被缓存的Servlet,我们称之为可缓存Servlet.可缓存Servlet可以表示为$cs=\langle name,parameters\rangle$,其中,*name*是Servlet的名称,*parameters*是Servlet的一组参数名称的集合.在本文中,我们定义S^C为所有可缓存Servlet的集合,S^{UC}为所有不可缓存Servlet的集合.

定义 2. Servlet 缓存对象.Servlet 缓存对象可以定义为 $scache=\langle id,cs,values,result,factors\rangle$,其中,*id* 是缓存对象的唯一标识,*cs* 代表了一个可缓存 Servlet,*values* 是对应可缓存 Servlet 的一组参数值,*result* 代表了在 *values* 下 *cs* 的运行结果,*factors* 是缓存替换算法需要的一组参数.

Servlet 缓存对象不仅与具体的可缓存 Servlet 有关,还决定于 Servlet 的参数值 *values*.另外,*factors* 对于不同算法有着不同的含义.对于 LRU 算法,$factors=\{age\}$,其中,*age* 代表访问时间参数.而对于 GDSF 算法,

factors={*age*,*freq*,*cost*,*size*},其中,*age* 与 LRU 算法有相同的定义,*freq* 代表 Servlet 缓存对象 *scache* 被访问的次数,*cost* 代表在 *scache.values* 参数取值下执行 *scache.cs* 所需的时间,*size* 是 *scache.result* 的大小.

定义 3. 访问序列.访问序列是一个Servlet序列,代表了一个用户顺序访问Servlet容器的过程.可以表示为 $aseq=\langle s_{a1},s_{a2},\ldots,s_{an}\rangle$,其中,$s_{ai}$是被用户顺序访问的一个Servlet.如果有$s=s_{ai}$,那么我们说$aseq$包含$s$,访问序列$aseq$包含的所有Servlet的集合记为$S(aseq)$.

定义 4. k步转移.如果在访问序列$aseq$中存在序列 $...\overbrace{S_i,S_m,...,S_{m+k-1}}^{k-1},S_j,,,$那么我们说在$aseq$中存在从$S_i$到$S_j$的$k$步.$k$步转移可以表示为$S_i \xrightarrow{k} S_j$,这里,$S_i$与$S_j$分别为转移的起点和终点,$k$为转移步数.

一般地,我们也将一步转移称为直接转移,表示为$S_i \rightarrow S_j$.如果直接转移的起点和终点都是同一个Servlet,我们称其为自转移.例如,假设Servlet S可以按照参数productid返回具体产品的基本信息,当用户用*productid*=200与*productid*=208 连续访问S时,一个$S \rightarrow S$的自转移就发生了.

定义 5. Servlet 容器序列模式.根据以上定义,我们定义 Servlet 容器序列模式为两个可缓存 Servlet 被先后访问的概率.记为$S_i \xrightarrow[\omega]{m\le k} S_j$,代表从$S_i$到$S_j$存在$k$步之内的转移,$m$是转移步数,$\omega$为转移概率.

Servlet 容器序列模式表现了可缓存 Servlet 之间的顺序关系.根据序列模式和缓存系统的状态,我们可以预测可缓存 Servlet 被再次访问的概率,选择概率较小的 Servlet 运行结果进行替换.下一节我们将给出转移图的定义,并以之描述 Servlet 容器序列模式.

2.2 转移图

定义 6. 直接转移图.给定一组访问序列 *aseqs*,与之对应的直接转移图是一个有向带权图 $G(aseqs[])=\{V,E\}$,其中,节点代表Servlet,且 $V=\bigcup_{i=1}^{n} S(aseqs[i])$,边代表Servlet之间的直接转移,边的权值$W_{DTG}(i,j)$代表在$aseqs$中直接转移$S_i \rightarrow S_j$发生的次数.直接转移图的矩阵表示为$M_{DTG}$,其中,$M_{DTG}[i,j]=W_{DTG}(i,j)$.

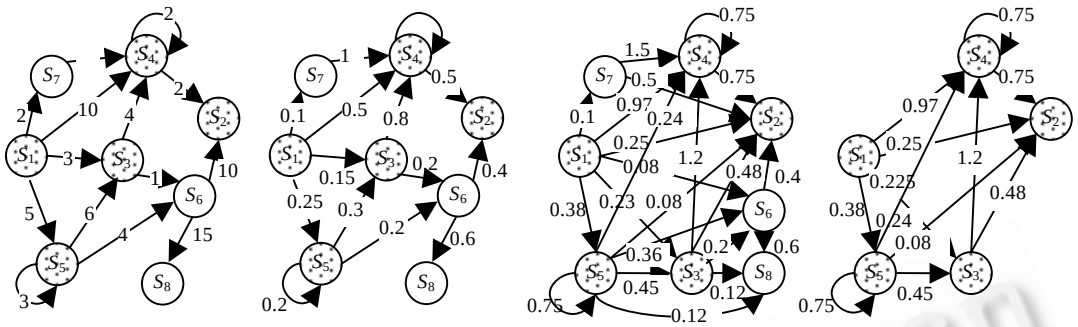
令$P(i,j,k)$代表从S_i经k步转移到达S_j的概率.显然,$P(i,j,1)=W_{DTG}(i,j)/\sum_{m\in EndS} W_{DTG}(i,m)$,其中,$EndS$代表所有以$S_i$为起点的转移$S_i \rightarrow S_m$的终点$S_m$的集合.

定义 7. k 步转移概率图.给定一组访问序列 $aseqs$,与之对应的 k 步转移概率图是一个有向带权图 $G(aseqs[])=\{V,E\}$.G的节点代表Servlet,且有 $V=\bigcup_{i=1}^{n} S(aseqs[i])$,边$e\langle i,j\rangle$代表存在从$S_i$到$S_j$的$k$步之内的转移,边的权值代表$S_i$在$k$步之内转移到$S_j$的概率.$k$步转移概率图的矩阵表示为$M_{K\text{-}TPG}$,根据$P(i,j,k)$的定义,显然有 $M_{K\text{-}TPG}[i,j]=\sum_{m=1}^{k} P(i,j,k)$.一步转移概率图又称为直接转移概率图,其矩阵表示为M_{DTPG}.由$P(i,j,1)$定义,有

$$M_{DTPG}[i,j]=P(i,j,1)=M_{DTG}[i,j]\Big/\sum_{j=1}^{n} M_{DTG}[i,j].$$

定义 8. k步可缓存转移概率图.仅包含了可缓存Servlet的k步转移概率图称为k步可缓存转移概率图,其矩阵表示为$M_{K\text{-}CTPG}$,它表示了可缓存Servlet之间的转移关系,即Servlet容器序列模式.

直接转移图、k步转移概率图、直接转移概率图与k步可缓存转移概率图统称为转移图.假设转移图中包含N个Servlet,L个可缓存Servlet,即$|S^C|=L$,$|S^C \cup S^{UC}|=N$,在生成转移图时,我们首先编号可缓存Servlet,即当 $1\le i\le L$有$S_i \in S^C$,并且当$L<i\le N$时有$S_i \in S^{UC}$.图 2 给出了上述转移图的例子,其中灰色节点代表可缓存Servlet,白色节点为普通Servlet,并且有N=8,L=5.图 2(a)代表了根据一组访问序列得到的直接转移图,图 2(b)~图 2(d)分别是相应的直接转移概率图、2 步转移概率图与 2 步可缓存转移概率图.



(a) DTG (b) DTPG (c) 2-TPG (d) 2-CTPG

(a) 直接转移图 (b) 直接转移概率图 (c) 2 步转移概率图 (d) 2 步可缓存转移概率图

Fig.2 Transfer graph

图 2 转移图

2.3 序列模式发现算法

从上一节的定义可以看出,Servlet容器访问序列模式发现问题可描述为:给定一组访问序列,如何得到对应的k步可缓存转移概率图$M_{K\text{-}CTPG}$的问题.

由上一节转移图的说明可知,k步可缓存转移概率矩阵其实就是k步转移概率矩阵的前L行、前L列组成的子矩阵,因此,只需得到k步转移概率矩阵$M_{K\text{-}TPG}$就可以得到$M_{K\text{-}CTPG}$.另外,根据直接转移图和直接转移概率图的定义,我们可以从访问序列得到对应的M_{DTG}和M_{DTPG}.为从M_{DTPG}得到$M_{K\text{-}TPG}$,我们给出下列定理:

定理 1. 对于 $k\geq 1$,有 $P(i,j,k)=M^{k}_{DTPG}[i,j]$.

证明:我们采用数学归纳法证明定理 1.

(1) 由直接转移概率图定义可知,$P(i,j,1)=M_{DTPG}[i,j]$.

(2) 假设当$k=m$时,有$P(i,j,m)=M^{m}_{DTPG}[i,j]$,其中$m\geq 1$.当$k=m+1$时,有$M^{m+1}_{DTPG}=M^{m}_{DTPG}\times M_{DTPG}$,那么$M^{m+1}_{DTPG}[i,j]=\sum_{k=1}^{n}M^{m}_{DTPG}[i,k]\times M_{DTPG}[k,j]$,由假设,$M^{m}_{DTPG}[i,k]$代表从$S_i$出发,经过$m$步转移到节点$S_k$的概率,而$M_{DTPG}[k,j]$代表了从$S_k$到$S_j$的直接转移概率,因此,$M^{m}_{DTPG}[i,k]\times M_{DTPG}[k,j]$代表从$S_i$出发经过$m+1$步到达$S_j$的概率,其中最后一步是经过节点$S_k$.因此,$\sum_{k=1}^{n}M^{m}_{DTPG}[i,k]\times M_{DTPG}[k,j]$为所有从$S_i$到$S_j$的$m+1$步转移概率之和,即$P(i,j,m+1)=M^{m+1}_{DTPG}[i,j]$.

由(1),(2)可知,$P(i,j,k)=M^{k}_{DTPG}[i,j]$. □

定理 2. $M_{K\text{-}TPG}[i,j]=\sum_{m=1}^{K}M^{m}_{DTPG}[i,j]$.

证明:由k步转移概率图定义与定理 1 可知,$M_{K\text{-}TPG}[i,j]=\sum_{m=1}^{K}P(i,j,m)=\sum_{m=1}^{K}M^{m}_{DTPG}[i,j]$. □

由定理 1、定理 2,我们给出序列模式发现算法 KCTPG_Discovery.

算法 1. 序列模式发现算法 KCTPG_Discovery.

1. $M_{L\times L}$ *KCTPG_Discovery*(*aseqs*[]){
2. M_{DTG}=*DTG_Generate*(*aseqs*);
3. M_{DTPG}=*DTPG_Generate*(M_{DTG});
4. *ClearNoise*(M_{DTPG},*threshold*);
5. $M_{K\text{-}TPG}=M_{DTPG}$;
6. Create $M'=M_{DTPG}$;

```
7.   //计算 K 步转移概率矩阵
8.   for (int i=0;i<k;i++){
9.     M_K-TPG=M_K-TPG+M′;
10.    M′=M′*M_DTPG;
11.  }
12.  ClearNoise (M_K-TPG,threshold);
13.  //M_DTG,M_DTPG,M_K-TPG为N×N的矩阵
14.  Create M_K-CTPG=M_L×L(0);
15.   //M_m×n(0)为m×n阶的零矩阵,以下同
16.    M_K-CTPG[i,j]=M_K-TPG[i,j],(1≤i≤L,1≤j≤L);
17.    return M_K-CTPG;
18. }
19. M_N×N DTPG_Generate(M_DTG){
20.       Create M_DTPG=M_N×N(0);
21.       for (int i=0;i<n;i++){
22.           for (int j=0;j<n;j++){
23.               M_DTPG[i,j]= M_DTG[i, j]/∑_{j=1}^{N} M_DTG[i, j];
24.           }
25.       }
26.       return M_DTPG;
27. }
```

序列模式发现算法中考虑多步转移是因为用户可能交错访问两类Servlet,这使得可缓存Servlet之间的业务关联通过不可缓存Servlet建立,仅考虑单步的转移无法发现这种访问序列模式.比如,Servlet容器中包含两个Servlet“浏览产品信息”和“添加产品到购物筐”,前者是可缓存Servlet,而后者是不可缓存Servlet.一个用户按照“浏览产品信息”、“添加产品到购物筐”、“浏览产品信息”的顺序访问时,一步可缓存转移概率图无法发现可缓存Servlet“浏览产品信息”的 2 步自转移关系,这使得所反映的Servlet容器访问序列模式不够确切.不过,Berkhin的工作也显示出,当转移的步数大于 6 时,就可以认为两个Servlet之间不再存在相互关联[17],因此,我们应选择不大于 6 的k值.

3 缓存替换算法

得到 Servlet 容器的访问序列模式之后,我们可以根据它来指导缓存替换算法的设计.本节根据第 2 节得到的 k 步可缓存转移概率图给出基于序列模式的缓存替换算法 KP-LRU 和 KP-GDSF.

3.1 预测概率函数

定义 9. 可缓存Servlet状态向量,$State=[state(S_1),state(S_2),\ldots,state(S_L)]$,其中,$state(S_i)\in\{0,1\},1\le i\le L$. 当$state(S_i)=1$时,代表缓存中包含$S_i$的输出,否则表示没有$S_i$的输出被缓存.$State$代表了所有可缓存Servlet的状态.

定义 10. k步可缓存转移概率向量,$KProb=[kprob(S_1),kprob(S_2),\ldots,kprob(S_L)]$,其中,$kprob(S_i)$代表根据缓存当前的$State$推断,在$k$步之内可缓存Servlet S_i被访问的概率.

定理 3. $KProb=State\times M_{K\text{-}CTPG}$.

证明:设$K=State\times M_{K\text{-}CTPG}$,则$K[j]=\sum_{i=1}^{L} State[i]\times M_{K-CTPG}[i,j]$,由$State$定义,当$S_i$不在缓存中时有$State[i]=0$,因此,$K[j]$为所有仍在缓存中的Servlet S_i经k步之内的转移到达S_j的概率之和,根据$KProb$的定义,有$kprob(S_j)=k[j]$,即$KProb=State\times M_{K\text{-}CTPG}$. □

由定理 3 可知,我们可以根据上一节得到的k步可缓存转移图与$State$来计算$KProb$,并且计算$KProb$具有$O(L^2)$的算法复杂度.另外,k步可缓存转移概率向量仅与可缓存Servlet状态向量有关,因此得到$M_{K\text{-}CTPG}$之后,我们

可以计算出所有状态向量对应的k步可缓存转移概率向量,以查表的方式用于缓存替换算法.根据k步可缓存转移概率向量,下面我们给出预测概率函数的定义.

定义 11. 预测概率函数,$\sigma_K(S_i)=(K\times L-kprob(S_i))$.

为分析预测概率函数的性质,我们给出定理 4.

定理 4. $M_{K\text{-}TPG}[i,j]\le K$.

证明:由$P(i,j,k)$定义可知,显然$P(i,j,k)\le 1$,则$M_{K\text{-}TPG}[i,j]=\sum_{m=1}^{K}M_{DTPG}^{m}[i,j]\le\sum_{m=1}^{K}1=K$. □

定理 4 说明,尽管 k 步可缓存转移概率图中存在一些转移概率大于 1 的转移关系,但转移概率存在上限值K.例如,当$P(a,b,1)=0.8$,$P(b,b,1)=0.6$ 时,有$M_{2\text{-}TPG}[a,b]=\sum_{m=1}^{2}P(a,b,m)=0.8+0.8\times 0.6=1.28<2$.

预测概率函数通过转移概率的分析预测了一个可缓存Servlet被再次访问的概率.显然,预测概率函数$\sigma_K(S_i)$是$kprob(S_i)$的单调递减函数,即S_i将来被访问的概率越大,其预测概率函数值就越小.另外,由定理 4 与$KProb$定义可知,$kprob(S_i)\le\sum_{i=1}^{L}M_{K-CTPG}[i,j]\le L\times K$,因此,预测概率函数$\sigma_K(S_i)=(K\times L-kprob(S_i))\ge 0$.下面我们基于预测概率函数给出 KP-LRU 算法和 KP-GDSF 算法.

3.2 KP-LRU算法

LRU 替换算法将最近最久未使用的缓存对象替换出缓存.通过扩展 LRU 算法得到的 KP-LRU 算法,同时考虑了时间因素与预测概率函数进行缓存替换.对于 KP-LRU 算法,有 $scache.factors=\{age\}$,age 代表了访问时间参数.KP-LRU 缓存替换算法的优先级函数为

$$Rank_{KP\text{-}LRU}(scache)=scache.age\times\sigma_K(scache.cs).$$

缓存系统维持了一个最小的age值minAge,初始值为 –1,每当一个缓存对象$scache$被命中时,$scache.age$=minAge,同时,minAge减 1,这样使得最近被访问的缓存对象总有着最小的age值.我们每次选择$Rank_{KP\text{-}LRU}(scache)$最小的$scache$替换出缓存系统.根据以上分析,给出KP-LRU算法如下:

算法 2. KP-LRU 缓存替换算法.

```
1. KP-LRU(cs,values){
2.   scache=Cache.get(cs,values);
3.   If (scache!=null){
4.     scache.age=Cache.minAge;
5.     Cache.minAge––;
6.     } else{
7.     nsc=executeAndSetFactors(cs,values);
8.     while (Cache.leftSpace()<nsc.data.size()){
9.       get scache with minimal Rank_KP-LRU(scache)
10.          Cache.evict(scache)
11.          }
12.          nsc.age=Cache.minAge;
13.          Cache.minAge––;
14.          Cache.push(nsc);
15.          }
16.    )
```

3.3 KP-GDSF算法

GDSF 算法综合考虑了可能影响文件与文件访问的各种因素,给出了缓存文件的优先级函数.在 Servlet 容器中,同样有着与文件系统类似的参数,KP-GDSF 算法综合考虑了上述因素与预测概率函数进行缓存替换.对于 KP-GDSF 算法,有 $scache.factors=\{age,freq,cost,size\}$,其中,$age$ 与 KP-LRU 算法中有着相同的定义与设定,$freq$

代表 *scache* 被访问的次数,而 *cost* 代表在 *scache.values* 参数取值下执行 *scache.cs* 所需的时间,*size* 是 *scache.result* 的大小.对应传统 GDSF 算法,KP-GDSF 缓存替换算法的优先级函数为

$$Rank_{KP\text{-}GDSF}(scache)=(scache.age+scache.freq\times scache.cost/scache.size)\times\sigma_K(scache.cs).$$

根据以上分析,给出 KP-GDSF 算法如下:

算法 3. KP-GDSF 缓存替换算法.

```
1. KP-GDSF(cs,values){
2. scache=Cache.get(cs,values );
3. If (scache!=null){
4.     scache.age=Cache.minAge;
5.     Cache.minAge––;
6.     scache.freq++;
7. } else {
8.   nsc=executeAndSetFactors(cs,values);
9.   while (Cache.leftSpace()<nsc.data.size()){
10. get scache with minimal Rank_KP-GDSF(scache)
11.          Cache.evict(scache)
12.          }
13.          nsc.freq=1;
14.          nsc.age=Cache.minAge;
15.          Cache.minAge––;
16.          Cache.push(nsc);
17.        }
18. }
```

在算法 2、算法 3 中,Cache 代表缓存系统,*executeAndSetFactors*(*cs*,*values*)函数执行参数为 *values* 的可缓存 Servlet *cs* 后得到运行结果 *nsc*,同时设置一些运行时确定的缓存参数(如 KP-GDSF 算法中的 *size* 与 *cost*).在算法 2、算法 3 中,*kprob*(*scache.cs*)可以通过第 3.1 节描述的查表方式获取.假设缓存系统中存在 *n* 个缓存对象,那么仅需计算一遍所有缓存对象的优先级函数就可找到被替换的缓存对象,因此,KP-LRU 算法和 KP-GDSF 算法均具有 $O(n)$的复杂度.

4 实 验

KP-LRU算法与KP-GDSF算法已经在中国科学院软件研究所的Web应用服务器产品OnceAS[18]中进行了原型实现,本节通过与传统的LRU算法与GDSF算法比较来验证KP-LRU算法与KP-GDSF算法的有效性.

实验环境为PC机,硬件为CPU Pentium4 2.4G,内存 512M,软件环境为Windows 2000 Professional Service Pack 4.我们采用J2EE标准应用PetStore[19]作为部署于OnceAS中的应用,事先添加 10 000 种宠物(其中包括猫、狗、鱼、鸟和爬虫类宠物各 2 000 种)作为实验数据.

Daniel提出用户行为模型图(customer behavior model graph,简称CBMG)概念,并在TPC-W中给出 3 类用户的访问行为模式用于模拟实际客户的行为[20].采用Daniel的方法,可以计算出PetStore的用户行为模型图(如图 3(a)所示),图 3(a)中每个节点代表对应的Servlet(如cart对应cart.jsp),其中,灰色节点代表可缓存Servlet.将图 3(a)作为模拟用户对Servlet容器的访问模式,每个模拟用户按照图 3(a)的转移概率顺序访问不同的Servlet.

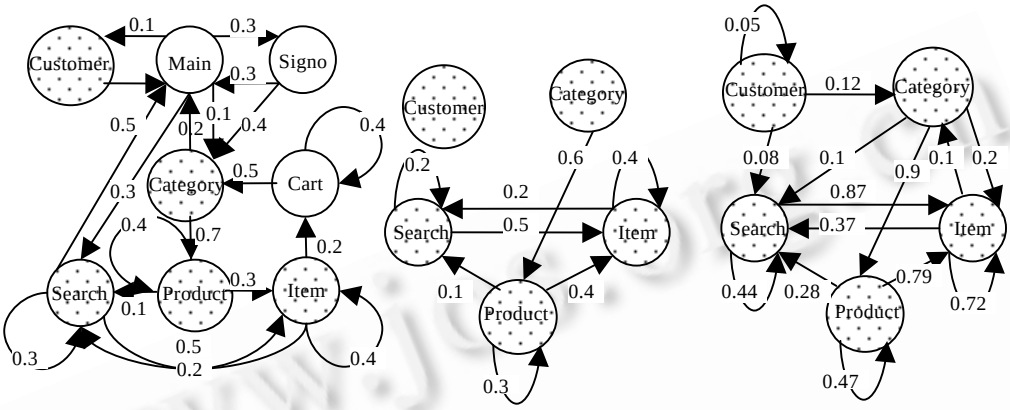
运行于OnceAS中的缓存系统已经提供了缓存一致性解决方案,即如果一个Servlet的运行结果已经不再有效,缓存系统将马上将其从缓存中删除.在一段时间内,除非管理员进行维护,否则,Pet Store 中的宠物信息、宠物分类、产品分类以及产品具体信息相对稳定.因此,我们定义 Pet Store 的可缓存 Servlet 见表 1.

Table 1 Cacheable servlet and parameters of Pet Store

表 1 Pet Store 的可缓存 Servlet 及其参数

Servlet name	category.jsp	product.jsp	item.jsp	customer.jsp	search.jsp
Parameter name	category_id	product_id	item_id	customer_id	keywords

首先模拟 100 个用户按照图 3(a)的访问模式持续访问 2 小时,收集访问序列并运行序列模式发现算法.分别取 k=1,k=2,得到对应的 1 步、2 步可缓存转移概率图如图 3(b)、图 3(c)所示.可以看出,根据收集结果得到的一步可缓存转移概率图基本符合模拟用户的访问模式,但其仅考虑了一步转移,未能发现一些多步序列模式(例如 $category \xrightarrow[0.28]{m\le 2} item$),而图 3(b)的 2 步可缓存转移概率图考虑了 2 步转移的序列模式.更准确地反映了应用的业务逻辑(图 3(b)、图 3(c)中小于 *threshold* 的转移概率已经被去除,这里,*threshold*=0.05).



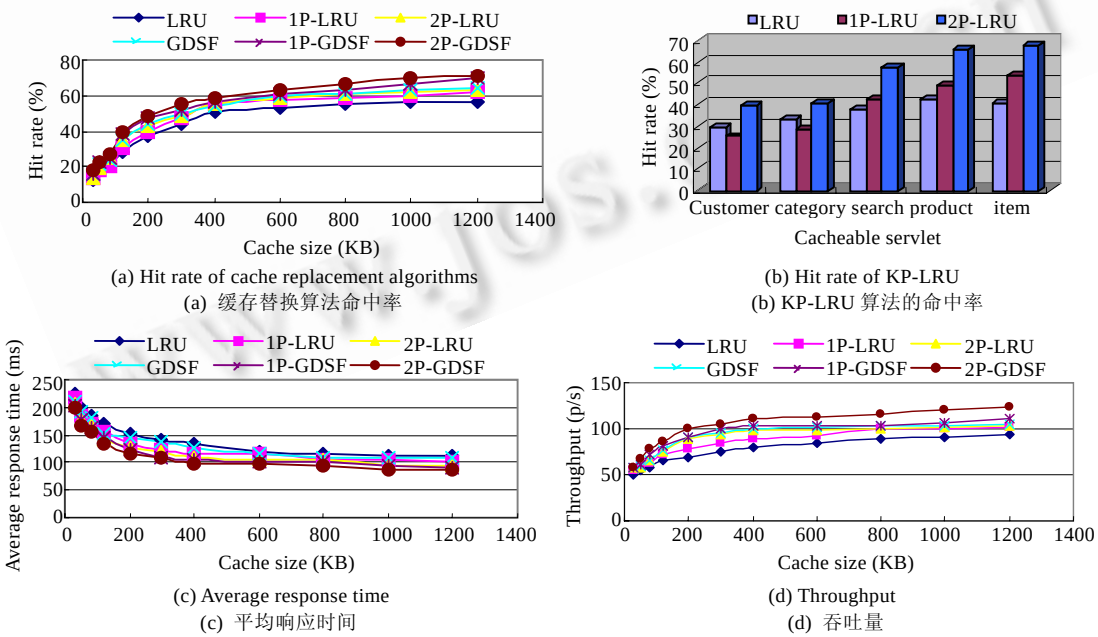
(a) CBMG of simulating customers (b) 1-CTPG (c) 2-CTPG

(a) 模拟用户行为模型图 (b) 1 步可缓存转移概率图 (c) 2 步可缓存转移概率图

Fig.3 CBMG and result of discovery algorithm

图 3 模拟用户的行为模型图和发现算法的结果

分别采用 LRU,1P-LRU,2P-LRU,GDSF,1P-GDSF,2P-GDSF 缓存替换算法(其中,1P-LRU,2P-LRU 指代 k 取 1,2 时的 KP-LRU 算法),对应不同缓存大小,仍采用图 3(a)的访问模式,模拟 100 个用户运行半小时的实验结果如图 4 所示.



(a) Hit rate of cache replacement algorithms (b) Hit rate of KP-LRU

(a) 缓存替换算法命中率 (b) KP-LRU 算法的命中率

(c) Average response time (d) Throughput

(c) 平均响应时间 (d) 吞吐量

Fig.4 Result of evaluation

图 4 模拟实验结果

从图 4(a)可以看出,LRU 的缓存命中率最差,即使在缓存大小提高到 1.2MB 时也仅达到 56%,GDSF 的缓存命中率始终要高于 LRU 算法,但当缓存大小提高到 1.2MB 时也仅有 64.2%.而基于序列模式的 KP-LRU 算法和 KP-GDSF 算法由于考虑了 Servlet 之间的转移关系,其缓存命中率比基本的 LRU 算法和 GDSF 算法都有所提高,其中,2P-LRU 算法的缓存命中率基本已经接近 GDSF 算法,而 2P-GDSF 算法的命中率最高已经达到了 71.2%.

由图 4(b)可以看出,大部分情况下,KP-LRU 算法比传统的 LRU 算法具有更高的缓存命中率.而且,由于考虑了 2 步关系,2P-LRU 算法的表现始终优于 1P-LRU 算法.但是,对于 customer 和 category 可缓存 Servlet 却有所不同,当采用 1P-LRU 算法的时候,它们的缓存命中率反而不如传统 LRU 算法.这是因为 1-CTPG_Discovery 算法并没有发现它们与其他可缓存 Servlet 之间的序列模式(如图 3(b)所示).因此,当采用 1P-LRU 算法时,其他可缓存 Servlet 的优先级函数值相对高于 customer 和 category,因此它们被更多次地替换出缓存系统,其缓存命中率反而有所下降.但是在 2P-LRU 算法中,由于考虑了 2 步转移,customer 和 category 与其他可缓存 Servlet 之间的序列模式能够被发现(如图 3(c)所示).从图 4(b)中可以看出,采用 2P-LRU 之后,customer 和 category 缓存对象的缓存命中率已经高于传统 LRU 算法.

在传统的缓存替换算法中,LRU 算法仅需要替换队首的缓存对象,即具有 $O(1)$的计算复杂度,GDSF 算法因为需要计算所有缓存对象的替换代价,所以需要 $O(n)$的计算复杂度.而对应的 KP-LRU 算法和 KP-GDSF 算法均具有 $O(n)$的计算复杂度,同时需要计算预测概率函数,比传统的替换算法多出了一定的时间开销.然而,从第 3.1 节的分析我们可以看出,预测概率函数可以通过查表的方式获取,因此仅具有 $O(1)$的计算复杂度.同时,对于缓存系统而言,缓存替换算法的运行结果对其性能的影响比缓存替换算法本身的开销要大得多.这是因为当缓存替换算法选择具有较大的预测概率函数值的缓存对象替换时,将会增加运行缓存替换算法的次数,反而会降低缓存系统的缓存命中率(如 1P-LRU 算法中的 customer 与 category).另外,由图 4(c)、图 4(d)也可以看出,采用了 KP-LRU 算法和 KP-GDSF 算法之后,Servlet 容器的平均响应时间和吞吐量的性能均有所改善,这也证明了采用 KP-LRU 算法和 KP-GDSF 算法的额外开销并没有影响 Servlet 容器的性能,算法的采用在提高缓存命中率的同时也提高了 Servlet 容器的性能.

5 结束语

提高 Servlet 容器的性能已经成为亟待解决的问题.本文将 Servlet 之间的业务逻辑关联定义为 Servlet 容器序列模式,并定义了 k 步可缓存转移概率图加以表示,然后给出了相应的序列模式发现算法 KCTPG_Discovery,最后基于 Servlet 容器的序列模式设计了缓存替换算法 KP-LRU 和 KP-GDSF.实验结果表明,KP-LRU 和 KP-GDSF 算法比对应的 LRU 算法和 GDSF 算法有更高的缓存命中率,有效地降低了 Servlet 容器的负载,提高了 Servlet 容器的性能.

然而,本文的序列模式发现算法具有较高的计算复杂度,因此要合理选择运行序列模式发现算法的时机.同时,缓存一致性问题完全由缓存系统提供,并未在替换算法中加以考虑.这些问题将在以后的工作中加以解决.

References:

[1] Shannon B. JavaTM 2 Platform Enterprise Edition Specification, Vo1. 4. Sun Microsystems Inc., 2003.

[2] Merrill CL. Servlet performance report: Comparing the performance of J2EE servers. 2004. http://www.webperformanceinc.com/library/reports/ServletReport/index.html?wmi=6,1

[3] Aggarwal C, Wolf JL, Yu PS. Caching on the World Wide Web. IEEE Trans. on Knowledge and Data Engineering, 1999,11(1): 94−107.

[4] Li K, Shen H, Tajima K. Cache replacement for transcoding proxy caching. In: Proc. of the the 2005 IEEE/WIC/ACM Int'l Conf. on Web Intelligence (WI'05). IEEE Computer Society, 2005. 500−507.

[5] Turner D. Web page caching in Java Web applications. In: Proc. of the Int'l Conf. on Information Technology Coding and Computing (ITCC 2005). Las Vegas: IEEE Computer Society, 2005. 805−808.

[6] Shupp R, Andy C, Chuck F. Web sphere dynamic cache: Improving J2EE application performance. IBM System Journal, 2004, 43(2):351−370.

[7] Podlipnig S, Böszörmenyi L. A survey of Web cache replacement strategies. ACM Computing Surveys, 2003,35(4):374−398.

[8] Srikant R, Agrawal R. Mining sequential patterns: Generalizations and performance improvements. In: Proc. of the 5th Int'l Conf. on Extending Database Technology. Avignon: Springer-Verlag, 1996. 3−17.

[9] Cooley R, Mobasher B, Srivastava J. Web mining: information and pattern discovery on the world wide web. In: Proc. of the 9th IEEE Int'l Conf. on Tools with Artifical Intelligence (ICTAI'97). Newport Beach: IEEE Computer Society, 1997. 558−567.

[10] Garofalakis M, Rastogi R, Shim K. Spirit: Sequential pattern mining with regular expression constraints. In: Atkinson MP, Orlowska ME, *et al*., eds. Proc. of the Int'l Conf. on Very Large Databases. Edinburgh: Morgan Kaufmann Publishers, 1999. 223−234.

[11] Han J, Pei J, Mortazavi-Asl B, Pinto H, Chen QM, Dayal U, Hsu MC. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In: Proc. of the 2001 Int'l Conf. Data Engineering (ICDE 2001). Heidelberg, 2001. 215−224.

[12] Cherkasova L. Improving WWW proxies performance with greedy-dual-size-frequency caching policy. Technical Report, HPL-98-69R1, HP Labs: Computer Systems Laboratory, 1998.

[13] Abrams M, Stanbridge C, Abdulla G, Williams S, Fox E. Caching proxies: Limitation and potentials. In: Proc. of the 4th Int'l WWW Conf. Boston: O'Reilly & Assoc, 1995. 119−133. http://ei.cs.vt.edu/~succeed/WWW4/WWW4.html

[14] Yang Q, Zhang HH. Web-Log mining for predictive Web caching. IEEE Trans. on Knowledge and Data Engineering, 2003,15(4): 1050−1054.

[15] Agrawal R, Srikant R. Mining sequential patterns. In: Yu PS, Chen ASP, eds. Proc. of the 11th Int'l Conf. on Data Engineering. Washington: IEEE Computer Society Press, 1995. 3−14.

[16] Mobasher CB, Srivastava J. Data preparation for mining World Wide Web browsing patterns. Journal of Knowledge and Information Systems, 1999,1(1):5−32.

[17] Berkhin P, Becher JD, Randall DJ. Interactive path analysis of Web site traffic. In: Proc. of the ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining (KDD 2001). San Francisco: ACM Press, 2001. 414−419.

[18] Huang T, Chen NJ, Wei J, Zhang WB, Zhang Y. OnceAS/Q: A QoS-enabled Web application server. Journal of Software, 2004,15(12):1787−1799 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1787.htm

[19] JavaTM Pet Store Demo 1.3.2. http://java.sun.com/developer/releases/petstore/

[20] Menascé DA. TPC-W: A benchmark for e-commerce. IEEE Internet Computing, 2002,6(3):83−87.

附中文参考文献:

[18] 黄涛,陈宁江,魏峻,张文博,张勇.OnceAS/Q:一个面向 QoS 的 Web 应用服务器.软件学报,2004,15(12):1787−1799. http://www.jos.org.cn/1000-9825/15/1787.htm

李洋(1979－),男,辽宁抚顺人,博士生,主要研究领域为网络分布计算,软件工程.

钟华(1971－),男,博士,研究员,CCF 高级会员,主要研究领域为软件工程,分布计算,对象技术,形式化方法.

张文博(1976－),男,博士,助理研究员,主要研究领域为网络分布计算.

黄涛(1965－),男,博士,研究员,博士生导师,主要研究领域为软件工程,网络分布式计算.

魏峻(1970－),男,博士,研究员,主要研究领域为网络分布式计算,软件形式化描述与验证方法.