一种针对 websearch 应用的缓存替换算法

司成祥[1,2],孟晓▮[1],许 鲁[1]
(1.中国科学院计算技术研究所,北京 100190;2.中国科学院研究生院,北京 100039)

摘 要: 本文通过对 websearch 负载的分析,总结出负载访问模式的特点,在此基础上提出了一种新的缓存替换算法——ERDP-LRU.与传统的 LRU 算法的区别是它采用基于重用距离的放置策略.通过模拟实验和实际系统验证,在各种不同的典型负载和缓存大小下,ERDP-LRU 的效果均好于其它替换算法.

关键词: web 搜索; 缓存; 替换算法
中图分类号: TP316 **文献标识码:** A **文章编号:** 0372-2112 (2011) 05-1205-05

A Novel Replacement Algorithm Designed for Websearch Applications

SI Cheng-xiang[1,2], MENG Xiao-xuan[1], XÜ Lu[1]
(1. *Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190, *China*;
2. *Graduate University of Chinese Academy of Sciences, Beijing* 100039, *China*)

Abstract: We studied the access pattern of web search workloads and proposed a new replacement algorithm called ERDP-LRU based on the observed access properties. The most obvious difference with other algorithms is that it adopts the placement policy based on the reused distance. Through simulation and real validation, ERDP-LRU uniformly outperforms the others for all the workloads and cache size.

Key words: web search; cache; replacement algorithm

1 引言

随着信息量的爆炸式增长,如何获得有效的信息成为人们关注的问题.各种搜索引擎的出现极大的方便了人们对信息的获取;而且随着数据量的迅猛增长,人们对搜索引擎的要求也越来越高.为了满足需求,作为搜索系统的存储子系统也应具有更高的性能[1].缓存是有效加速系统性能的常用技术之一,对存储系统性能有非常重要的影响.对于搜索引擎的后端存储系统,通常采用内存作为缓存,存放频繁访问的内容.然而,我们在研究中发现,对于 websearch 应用,经典的缓存替换算法表现出很差的适用性.图 1 显示了一个 web 搜索负载[8]的存储系统分别使用 LRU 和 LFU 的命中率曲线.从中可知,对于 websearch 应用,两种算法表现出很差的性能;在容量小于 1GB 时,两种算法几乎没有命中;当容量达到 2GB 时,命中率也只有 15% 左右.显然,低效的缓存替换算法并不能满足搜索应用对存储系统的要求.在本文中,我们总结出负载访问模式的特点,在此基础上设计了一种新的替换算法——ERDP-LRU;并通过模拟和实际系统验证其有效性.

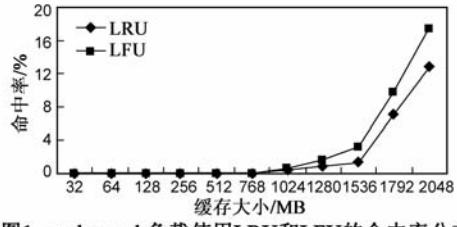

图1 websearch负载使用LRU和LFU的命中率分布

2 相关研究

大量文献已广泛研究了缓存替换算法,如 LRU、LFU、MRU、2Q[2]、ARC[3]等.对于 web 应用方面,Cao[4]使用基于费用的贪心算法,为每一个缓存中的 web 文档赋予一个 H 值,缓存中的每个文档都和一个价位 H 关联;当发生替换时,选择 H 值最小的 web 文档替换出去,并将所有的文档的 H 值减去这个最小的 H 值.Williams[5]利用了 web 静态页面的大小分布特性,即用户倾向于访问较小的 web 文档,在替换发生时将最大的文档替换出去.Rizzo[6]通过考虑获取文档的代价、文档的大小,并利用对 web 代理访问日志的分析,计算文档下一次或一段时间后被访问的概率,并利用这个概率值

收稿日期:2010-08-19;修回日期:2010-11-24
基金项目:国家 863 高技术研究发展计划(No.2007AA01Z402)

作为替换的依据.Wooster[7]使用访问 web 文档的延迟作为替换的标准,并兼顾了文档的访问频率和大小.尤[14]采用分块缓存技术管理动态网页,降低了缓存分配和替换的粒度,提高了数据的命中率.李[15]通过在网络节点上采用区分服务的方法,把服务分成有保证和无保证两类,在发生数据替换时,首先替换无保证类服务的数据,从而提高了网络服务质量.Konanki[9]已经指出单一的替换算法不能满足各种不同应用的需要;本文中,借鉴此结论,提出了适合 websearch 应用的缓存替换算法.

3 负载访问模式分析

我们从时间局部性和访问频率两个角度来分析应用的访问模式.表 1 给出了负载文件的参数.

表 1 3 种 websearch 负载的参数

负载	大小 (GB)	时长 (hr)	数据集 (GB)	*IO* 数 (M)	读比例 (%)	平均请求大 (KB)
websearch1	94	0.88	6.08	1.0	99.98	15
websearch2	94	4.28	8.48	4.4	99.97	15
websearch3	97	75.73	8.44	4.1	99.98	15

3.1 时间局部性分析

使用重用距离(reused distance)[10]来分析时间局部性.重用距离是指给定访问序列中对同一块的相邻两次访问间距,访问序列是指按访问时间先后次序排列的请求队列.例如,在访问序列 $A_1B_2C_3D_4E_5C_6F_7$ 中,对块 C 的两次关联访问(C_3 和 C_6)的重用距离为 2,这其中我们称对同一块的相邻两次访问为关联访问.图 2 给出了 3 种负载的重用距离分布,展示了重用距离相同的关联访问总量分布.从图中我们可以看出,三种负载的重用距离分布表现为"小山形".websearch1 的关联访问的重用距离峰值出现在 100K 处左右;而另外两种负载的峰值出现在 16K 处左右.据统计,有超过 70% 的关联访问出现在"小山形"区域.我们定义"小山形"的起始位置为最小距离(*MinDist*)[10].由于大量的关联访问出现在了"小山形"区域内,理想替换算法应该使得"小山形"区域内的块在缓存中至少驻留 *MinDist* 的虚拟缓存时间.对于 LRU 算法:当容量小于 *MinDist* 时,绝大多数"小山形"区域内的块在缓存中驻留时间不足以等到其下一次被访问,从而表现出很低的性能.

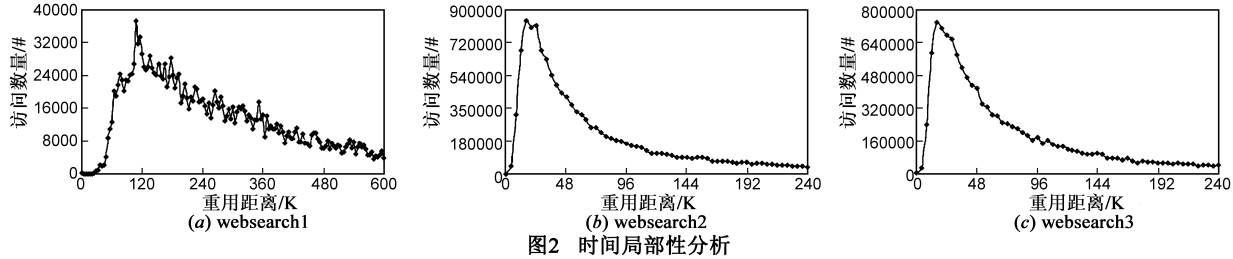

图2 时间局部性分析

3.2 访问频率分析

从访问频率角度来分析负载.块的访问频率是在给定访问序列中,对同一块的总的访问次数.图 3 显示了 3 种负载的块访问频率累积分布,其中 x 轴为块访问频率,而 y 轴为具有相同频率的块总和.可以看出,负载中的数据块访问频率普遍较低;且相同频率的数据块总和随访问频率的增加而显著下降.对于每种负载,通过统计我们发现超过 80% 的块的访问频率小于 10;而对这些块的访问占了总访问的 70%.这表明访问频率分布是比较均匀的.因此,由于访问频率不能有效区分热块和冷块,LFU 也不能提供很好的性能.此外,值得注意的是每种负载中约有 30% 的块只被访问过 1 次.

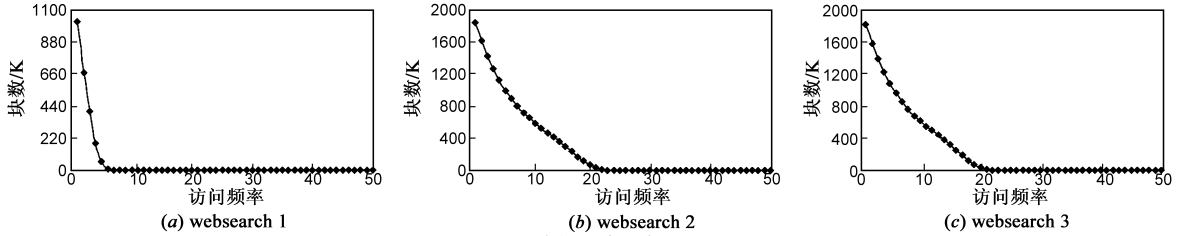

图3 访问频率分析

定义重引用频率(re-reference frequency)为给定访问序列中对同一块的重复访问次数.研究重用距离与重引用频率的关系.例如,对于序列 $A_1C_2B_3A_4C_5A_6D_7C_8$,A_4 是对 A 的第 1 次重访问,其重引用频率是 1,重用距离是 2.A_6 是对 A 的第 2 次重访问,其重引用频率是 2,重用距离是 1.在重引用频率固定条件下,计算平均重用距离(average reused distance).例如,当重引用频率是 1(A_4和 C_5)时,其平均重用距离是 2,即((2+2)/2).图 4 显示了平均重用距离和重引用频率的关系.可知,随重引用频率的增长,平均重用距离迅速降低.这表明,重用频率相对较高的访问具有相对较小的重用距离.假设对 A 的第 n 次访问的重用距离是 $dist_n$,可以预测对 A

的第 $n+1$ 次访问的重用距离 $dist_{n+1} < dist_n$. 根据以上分析,总结负载访问模式的特点:(1)关联访问的重用距离大;(2)块的访问频率小,并存在大量只访问 1 次的块;(3)块的访问频率呈均匀分布;(4)重用距离和重引用频率之间具有反比关系. 依据这 4 个特点设计一个更有效的替换算法.

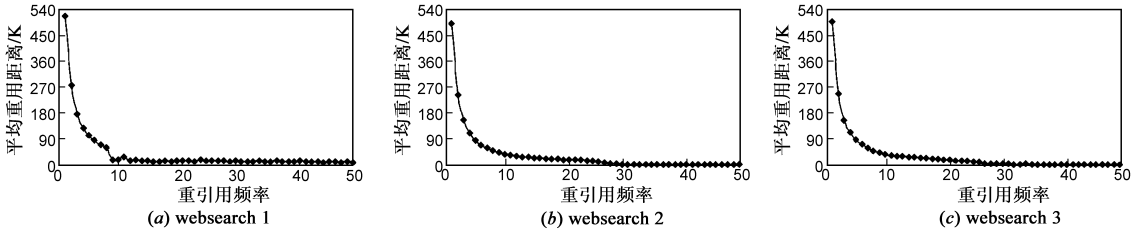

图4　平均重用距离与重引用频率关系

4　ERDP-LRU 算法

基于负载的访问模式,设计了一种新的替换算法——ERDP-LRU(LRU extended with reused distance based placement). 其基本思想如下:区分冷热块,并根据块的冷热采取不同的放置策略. 具体说,缓存块被组织为一个 LRU 队列(标记为 Q)统一集中管理. 当一个块被访问时,根据重用距离来识别块的冷热,并采用不同的放置策略. 该算法与 LRU 的替换策略相同,主要区别在于:它采用了基于重用距离的缓存放置策略,而不是传统的按需访问放置策略. 图 5 给出了算法的伪码描述,其中 G 为虚缓存队列(见下一节),K 为预置虚缓存最大值.

```
// If there is space, we give it to x.
// If there is no space, we free a block slot to
// make room for block x.
reclaimfor(block x)
begin
    if there are free blocks then
        put x into a free block slot
    else
        remove the tail of Q, call it y
        add identifier of y to the head of G
        if(|G| >K)
        remove identifier of z from the tail of G
    end if
    put x into a free block slot
end
-----------------------------------------------
On accessing a block x :
begin
    if x is in Q then
        move x to the head of Q
    else if (x is in G) then   // x is regard as "hot"
        reclaimfor(x)
        move x to the head of Q
    else // x isn't in cache
        reclaimfor(x)
        add x to the tail of Q
    end if
end
```

图5　算法描述

4.1　区分冷热块

对块访问的冷热进行定义.

冷访问:此前从未被访问或重用距离超过实际缓存容量大小的访问.

热访问:重用距离小于实际缓存容量大小的访问.

为了有效的区分失效访问类型,ERDP-LRU 利用了虚拟缓存技术(ghost buffer)[3]. 将虚缓存标记为 G 队列,其记录了最近被替换丢弃块的相对访问次序,其管理方式为 FIFO. G 队列中的每个队列元素仅记录了块的 ID(存储空间中的地址偏移量),并不存储真实数据. 具体来说:(1)若失效访问块的 ID 在 G 中,则我们认为该失效访问为热访问,被访问的失效块为热块;(2)反之,则认为该失效访问为冷访问,被访问的失效块为冷块.

4.2　放置策略

当一个块被访问,通过区分冷热,采用不同的放置策略. 具体如下:

缓存预热阶段:失效块被添至 Q 队列的 LRU 端.

缓存稳定阶段:失效块被识别为热块,添至 Q 队列 MRU 端;失效块被识别为冷块,添至 Q 队列的 LRU 端.

基于对访问模式分析可知,算法的放置策略有以下两个优点:(1)通过把冷块放置在队列的 LRU 端,可以加快冷块被替换的概率和速率;从而可避免大量一次访问块对有限缓存空间的污染(据特点(1)+(2));(2)通过区分负载访问中的冷热块,并只允许热块加至队列的 MRU 端,可以有效提高缓存空间利用率(据特点(1)+(4)). 首先,负载中关联访问的重用距离较大,将下一次访问重用距离超过缓存容量的块添加至缓存将不会带来命中;其次,对于同一块的关联访问,其下一次访问的重用距离通常小于其前一次. 将识别出的热块加入缓存是有益的:若失效块的重用距离小于缓存容量,可推测对此块的下一次访问间距很可能也小于缓存容量.

除了不同的放置策略外,ERDP-LRU 仍沿用 LRU 替换策略. 其依据为:缓存进入稳定状态后,冷块聚集在队列的 LRU 端附近,可以加速冷块的替换;而热块聚集在队列的 MRU 端附近,且重用距离很可能小于缓存容量,有利于减少热块被替换的概率.

4.3　预取策略结合

在实际系统中,替换算法通常与预取策略结合以优化应用的性能. 前者利

```
// If there is space, we give it to x.
// If there is no space, we free a block slot to
// make room for block x.
reclaimfor(block x)
begin
    if there are free blocks then
        put x into a free block slot
    else
        remove the tail of Q, call it y
        add identifier of y to the head of G
        if(|G| >K)
        remove identifier of z from the tail of G
    end if
    put x into a free block slot
end
-----------------------------------------------
On accessing a block x :
Begin
    if x is in Q then
        if x.pref is TRUE
            move x to the tail of Q
            set x.pref to FALSE
        else
            move x to the head of Q
        end if
    else if (x is in G) then
        reclaimfor(x)
        if x.pref is TRUE
            move x to the tail of Q
            set x.pref to FALSE
        else
            move x to the head of Q
        end if
    else // x isn't in cache
        reclaimfor(x)
        add x to the tail of Q
    end if
end
```

图6　与预取结合的算法描述

用负载的局部性来提高缓存效率;后者利用请求间隙来屏蔽慢速设备对性能的影响.这里并不关注具体的预取策略,而仅讨论算法如何与预取策略有机结合.通过 *x.pref* 标记块 *x* 是否是预取块,具体流程如图 6 所示.

5 实验评价

5.1 模拟结果

本节给出了 ERDP-LRU 和其它算法(LRU、MRU、LFU、2Q 和 ARC)的模拟结果.如图 7 所示,ERDP-LRU 在各种缓存容量下的性能均优于其它算法,其命中率在测试范围内随容量的增长而稳步上升.

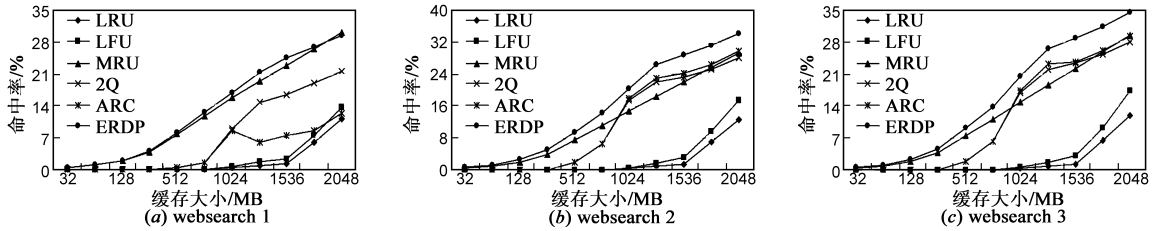

图7 各种替换算法的命中率曲线

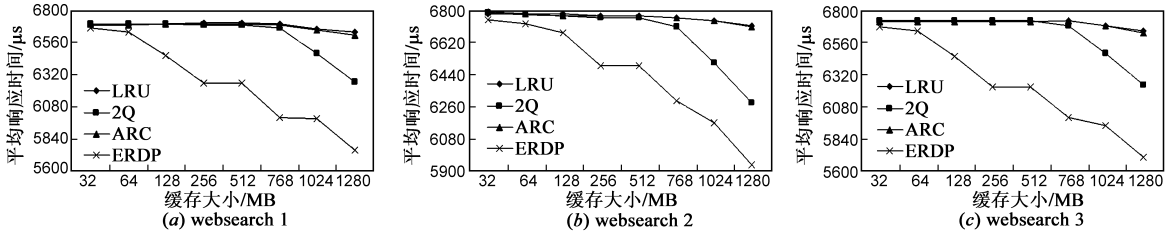

图8 各种替换算法结果

测试配置见表 2,2 块磁盘以 RAID0 方式向系统提供一个条带卷*.使用的 trace 是从某门户网络搜索引擎的后端存储系统中搜集得到,其格式遵循 SPC-1 标准.其中每一条记录指出了访问偏移块号、访问长度、访问模式(读或写)、访问间隔时间等.使用回放工具——*Kernio*[11],进行回放 trace 文件.

5.2.2 测试结果

以平均响应时间为评价指标.如图 8 所示,相对于其它算法,ERDP-LRU 有效的减少了平均响应时间.当容量为 1.25GB 时,ERDP-LRU 相对于 LRU 最大提高 17%(websearch2),平均提高 14%.根据模拟结果,随着容量的增长,其它算法和 ERDP-LRU 的性能差距将增大.实验结果验证了算法在真实系统中的有效性.

5.2.3 预取效果评测

在 Flexi-Cache 中,采用自适应线性预取策略[12].本节检验算法是否能有效地与预取策略相结合以进一步提升性能.图 9 给出了 websearch2 的结果(其它负载结果类似),其中图 9(*a*)对比了算法在有无预取两种情形下的性能;在结合预取策略后,平均响应延迟在各种容量下均得到降低,约为 7.7%.图 9(*b*)给出了预取窗口随时间变化分布,其大小在回放过程中稳定在 28 ~ 68KB 左右(相当于 717 个缓存块);根据表 1,负载的平均请求大小仅为 15KB 左右,由此可知,算法在结合预取策略后通过提前从磁盘中读取已探测到的顺序访问序列,并利用请求间隙尽可能屏蔽慢速磁盘对负载回放性能的影响.结果表明,ERDP-LRU 在与预取策略结合后可以进一步提高应用的性能.

5.2 真实系统结果

将 ERDP-LRU 集成到一个真实的缓存系统 Flexi-Cache[11]中,评价其在真实系统中的性能.首先,验证 ERDP-LRU 的模拟结果,为了便于比较,也实现了 LRU,2Q 和 ARC 算法.其次,测试算法与预取结合的效果.

5.2.1 实验配置

表 2 测试配置

CPU	Intel(R) Pentium(R) D CPU 2.80GHz
memory	DDR2 2GB
os	Linux2.6.18
storage	RAID0:2× 250GB(sata disk,7200RPM)

6 结论

本文首先研究了 websearch 负载的访问模式,并根据负载特点提出了一种新的缓存替换算法 ERDP-LRU.

* 需要说明的是,当前搜索引擎系统的后端存储系统通常是由多个存储节点组成的存储集群,其中每个成员节点的缓存资源通常由操作系统采用默认的缓存替换算法(类 lru 算法)独立管理.在我们的实验中采用单机系统进行验证,测试结果表明了在单个节点上 ERDP-LRU 具有很大优势.由于集群节点缓存管理的独立性,显然,如果每个存储节点上皆采用高性能替换算法,必然会提高整个存储集群的性能,从而提高整个搜索引擎系统的性能.

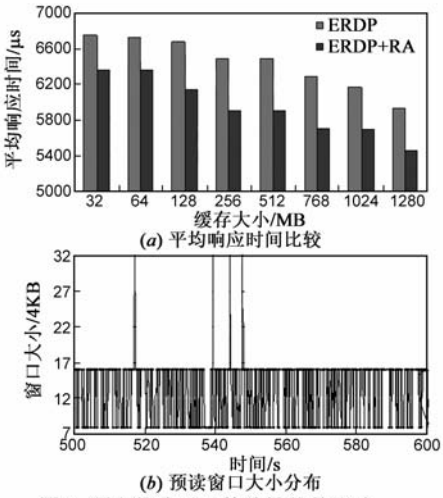

图9　预取策略对于算法性能的影响

与传统的 LRU 算法的区别是它采用基于重用距离的放置策略.模拟实验结果表明我们提出的算法在各种不同的负载和缓存容量下,其效果要好于当前流行的其他替换算法.为了验证算法在真实系统中的有效性,我们把 ERDP-LRU 集成到真实的缓存系统中并与自适应顺序预取策略有机结合;实验结果表明, ERDP-LRU 算法是非常有效的.

参考文献

[1] R Yates, et al. The impact of caching on search engines [A]. Proceedings of the annual international ACM SIGIR conference on Research and development in information retrieval [C]. New York: ACM, 2007. 183 – 190.

[2] T Johnson, et al. 2Q: A low overhead high performance buffer management replacement algorithm [A]. Proceedings of Very Large Databases Conference [C]. New York: ACM, 1995. 439 – 450.

[3] N Megiddo, et al. ARC: A Self-tuning, low overhead replacement cache [A]. Proceedings of USENIX Conference on File and Storage Technologies[C]. Berkeley: USENIX, 2003. 115 – 130.

[4] P Cao, et al. Cost-aware WWW proxy caching algorithms[A]. Proceedings of the USENIX Symposium on Internet Technologies and Systems[C]. Monterey: USENIX, 1997. 18 – 18.

[5] S Williams, et al. Removal policies in network caches for world wide web documents [A]. Proceedings of ACM SIGCOMM [C]. New York: ACM, 1996. 293 – 305.

[6] L Rizzo, et al. Replacement policies for a proxy cache [J]. IEEE/ACM Transactions on Networking (TON). 2000, 8(2): 158 – 170.

[7] P Wooster, et al. Proxy caching that estimates page load delays [A]. Proceedings of the international conference on World Wide Web[C]. Essex : Elsevier, 1997. 977 – 986.

[8] K Bates, et al. Search Engine I/O[DB/OL]. http://traces.cs.umass.edu/index.php/Storage/Storage, 2007 – 07 – 01.

[9] P Konanki, et al. FlexiCache: A flexible interface for customizing linux file system buffer cache replacement policies [A]. Proceedings of USENIX conference on File and Storage Technologies [C]. San Jose: USENIX, 2007. 16 – 16.

[10] Y Zhou, et al. Second-level buffer cache management [J]. IEEE Transactions on Parallel and Distributed Systems. 2004, 15(6): 505 – 519.

[11] X Meng, et al. A flexible two-layer buffer caching scheme for shared storage cache [A]. Proceedings of International Conference on High Performance Computing and Communications [C]. Seoul: IEEE, 2009. 424 – 431.

[12] B Gill, et al. AMP: adaptive multi-stream prefetching in a shared cache [A]. Proceedings of USENIX Conference on File and Storage Technologies [C]. San Jose: USENIX, 2007. 26 – 26.

[13] R Lempel, et al. Predictive caching and prefetching of query results in search engines [A]. Proceedings of the 12th international conference on World Wide Web[C]. New York: ACM, 2003. 19 – 28.

[14] 尤朝,等.一种在线的动态网页分块缓存方法[J].电子学报,2009,37(5):1087 – 1091.
You Chao, et al. An online approach for fragment-based caching of dynamic web pages [J]. Acta Electronical Sinica, 2009, 37(5): 1087 – 1091. (in Chinese)

[15] 李锁钢,等.一种综合缓存管理和主动队列管理的区分服务节点机制[J].电子学报,2005,33(5):847 – 851.
Li Suo-gang, et al. A node mechanism of differentiated services combining buffer managementand active queue management [J]. Acta Electronical Sinica, 2005, 33(5): 847 – 851. (in Chinese)

作者简介

司成祥　男,1982 年生于山东. 中国科学院计算技术研究所博士研究生. 主要研究方向:缓存管理、存储系统.
E-mail: chengxiangsi@gmail.com

孟晓　男,1980 年生于江苏. 工学博士. 主要研究方向:并行处理、网络存储、缓存管理.