doi:10.3772/j.issn.1002-0470.2013.07.005

一种面向 Web 内容分发的缓存替换策略①

李 乔② 何 慧 方滨兴

(哈尔滨工业大学网络与信息安全研究中心 哈尔滨 150001)

摘 要 研究了 Web 内容分发和对 Web 服务性能有重要影响的缓存替换机制。考虑到当前的缓存替换策略主要采用基于频度与本地局部性的替换基准,而且通过对实际 Web 数据访问情况的分析发现访问间隔的变化率对命中率的影响具有更高的准确性,提出了一种基于访问密度与大小混合的缓存替换策略。该策略通过统计近期缓存对象的平均访问间隔,结合该对象的字节大小进行缓存替换,并分别在固定对象数目空间与对象字节空间上进行对比实验,实验结果表明,该策略比最近最少使用(LRU)和最近频繁使用(LFU)算法提升 3% ~5% 的命中率,比空间与频度混合贪心(GDSF)算法提升 5% ~8% 的字节命中率。

关键词 Web 缓存,缓存替换,访问间隔,命中率,字节命中率

0 引 言

随着 Web 数据的多元化发展,网页内容的分发逐步成为影响 Web 服务性能的关键因素。当前主流的数据分发机制是采用内容分发网络技术,将用户的请求重定向至离其最近的服务器,从而减少访问延时与源服务端负载压力。为了有效提升服务质量,内容分发网络供应商在多个网络边界部署内容代理服务器。例如 Akamai 公司在 70 多个国家与地区的 1000 多个网络内部署超过 25000 个内容服务器[1]。当前,内容分发网络通常关注代理服务器部署位置选择以及内容的路由机制,如文献[2-4],然而缓存的效率是影响内容分发性能的一个关键因素。本文针对 Web 内容管理中的缓存机制,提出一种密度与内容大小混合的缓存替换策略,以此降低内容分发过程中流量的压力以及用户的访问时延。

缓存替换机制主要依赖于以下两个原则:(1)频繁访问的信息应该被缓存;(2)信息的热度变化意味着访问间隔时间的变化。至今已有多种缓存替换策略基于重引用时间[5],但均未考虑整个访问历史。基于此,本文首先在校园网网关上提取实际网络的统一资源定位符(uniform resource locator, URL)并分析用户的访问行为,发现在最近最少使用(least recently used, LRU)的缓存中,高流行度的 URL 常被一些低流行度的替换;其次为了避免这种命中损失,采用访问间隔变化率作为对象的权值因子,并结合对象所占空间大小进行缓存替换。最后通过实验将该策略分别与 LRU、最近频繁使用(least frequently used, LFU)和空间与频度混合贪心(greedy-dual size and frequency, GDSF)算法进行对比,结果表明单纯基于访问间隔变化的替换算法能够提升 3% ~5% 的命中率,而混合的替换算法比 GDSF 提升 5% ~8% 的字节命中率。

1 相关工作

提升 Web 内容分发性能策略涵盖多个技术层面:拓扑层上的服务器位置选择策略[6];网络层上的资源重定向技术;应用层上的副本部署机制[7]及缓存管理机制[8]。缓存替换策略是缓存管理性能的关键因素之一。针对 Web 内容的特性,当前已有的缓存替换机制可分为三类:(1)基于最近使用的策略:以最近使用作为基准,结合缓存对象大小进行替换权值计算,如文献[9]采用 LRU 作为 Web 的缓存替换算法,并进行了详细的测试,认为对于 Web 数据在缓存中的命中率最高只有 30% ~50%;(2)基于访问频度的策略:以访问频度作为基准,集合对象大小进行权值计算,如文献[10]将网页大小、访问频度与访问时延加权混合计算缓存对象的替换权

① 863 计划(2011AA010705),973 计划(2011CB302605)和国家自然科学基金(61173145, 60203021)资助项目。

② 男,1984 年生,博士生;研究方向:网络安全,分布式系统;联系人,E-mail: liqiao84@hit.edu.cn

(收稿日期:2012-06-25)

值;(3)基于其它自定义函数的策略:以缓存对象的价值作为基准,采用自定义的函数计算权值,如文献[11]考虑网页对象的流行度与频度混合的替换机制,文献[12]采用最近出现时间与文件被访问频率作为替换权值的基准参数对缓存中的对象进行替换,文献[13]将缓存对象的替换开销作为替换基准。

缓存对象的最近使用时间、访问频度、空间大小以及替换开销是缓存替换机制考虑的基本因素。上述的缓存替换机制选择一项或多项作为评价缓存对象价值的标准。但是这些算法均未考虑在相对长时间内访问序列空间的变化,本文所提出的缓存策略利用访问历史以及访问间隔变化率计算对象的权值,更加有效地预测缓存未来的变化趋势。

2　Web 数据分析

每个缓存对象在缓存空间中都具有自身的生命周期,如图 1 所示。缓存中的对象 A 的生存周期可分为两个部分:(1)活跃期:从进入缓存直至最后一次被访问;(2)僵死期:从最后一次被访问直至被替换出缓存。图 1(a)和图 1(b)描述了对象 A 在缓存中的两种生命周期,图 1(c)是对象 A 在缓存中的极

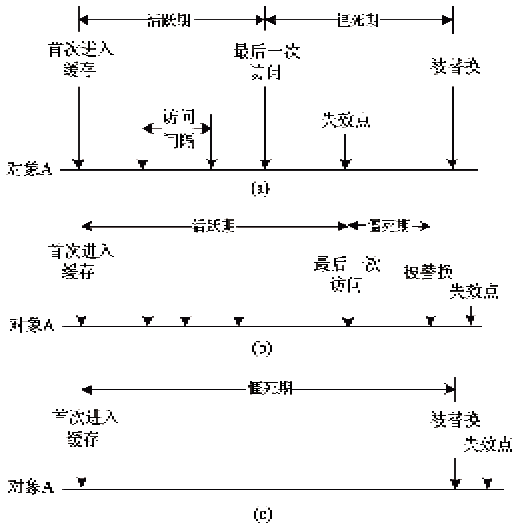

图 1　缓存对象的生命周期

端情况。缓存对象的活跃期越长,缓存性能越高。尽管不能精确预测缓存对象未来行为,然而尽量延长其活跃期或者减少长僵死期对象在缓存中的时间是提升缓存效率的有效方法。为了研究 Web 请求的分布特征,在连续三天的校园网关的网络日志中提取了 427936 个用户请求,包括 167981 个不同的 URL。如图2所示,该数据集符合典型的$zipf_\alpha$分布,

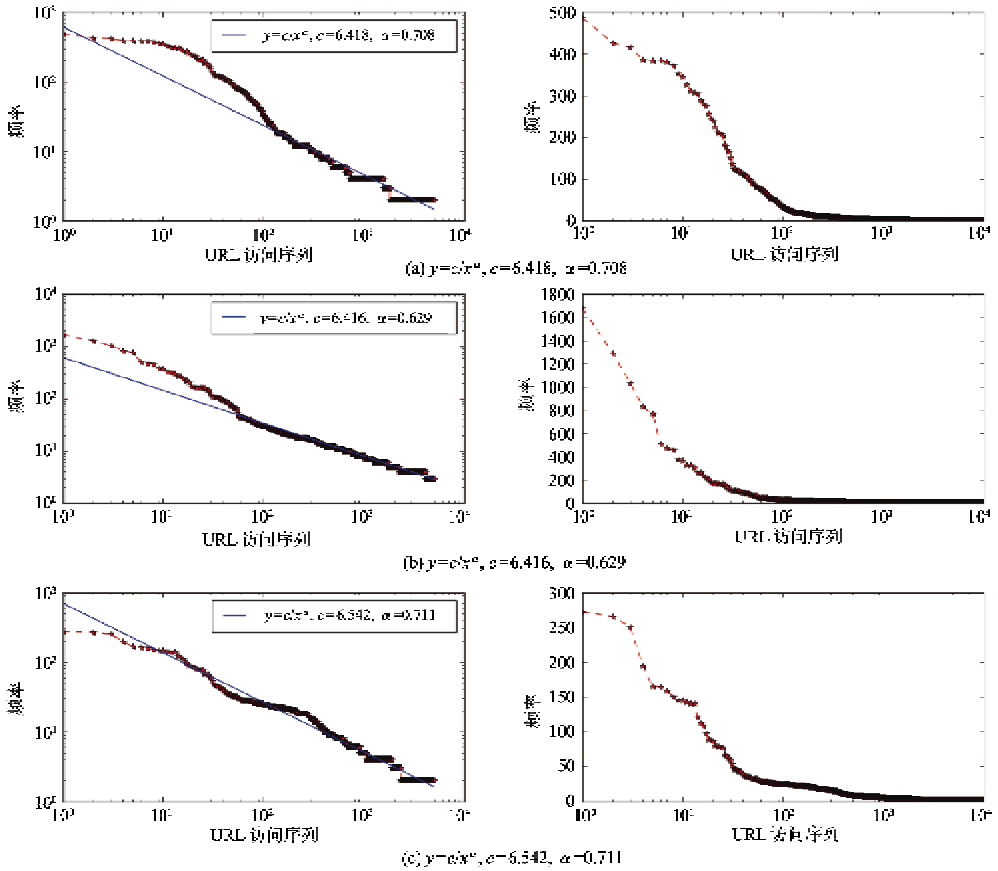

图 2　数据集中的 URL 访问频率分布

其中 $0.6 < \alpha < 0.8$。热门对象意味着在相对短时间内被访问多次。因此延长这些热门对象在缓存中的时间是提升缓存效率的有效方法。访问间隔是客观描述对象热度的关键度量。图 3 展示了在数据集中最热的 8 个 URL 的访问序列，y 轴表示这 8 个 URL，x 轴表示在这三天内每个 URL 出现的次序。从图 3 中可知，热门的 URL 在某段时间内都具有间隔极低的特性，同时一些热门 URL 的访问间隔是逐渐变小的，如 *url*-2 和 *url*-5，同时一些热门 URL 的访问间隔较为均匀，如 *url*-8，该类 URL 在各类缓存替换策略中一般都不会被替换。而对于类似 *url*-1 的这类 URL，由于其访问间隔具有明显的周期性，但周期间隔较长，在 LRU 的缓存策略下，将被替换多次，从而降低了缓存性能。对于类似 *url*-7 的这类 URL，虽然具有一定的周期性，但其周期间隔不断增长，这类 URL 在 LFU 策略下，其权值始终上升，然而其热度呈现下降趋势，从而导致缓存污染。

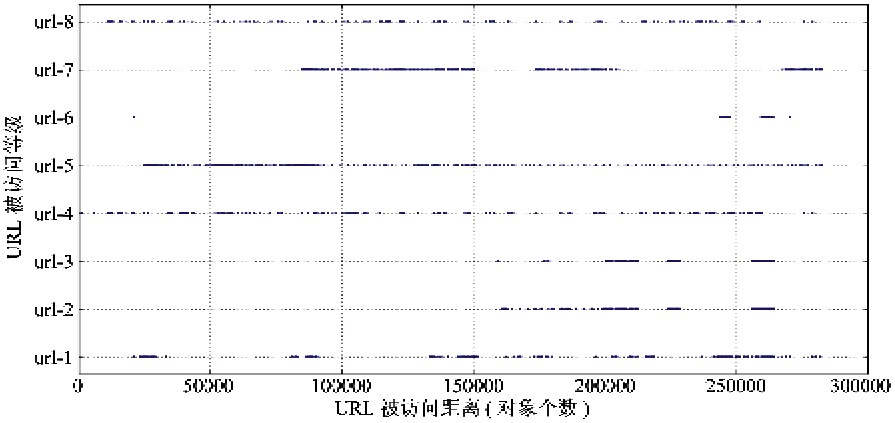

图 3 热门 URL 的访问序列

从以上分析可知热门 URL 的访问行为可分为以下 3 类情形：(1) 均匀且具有周期性，如 *url*-8，*url*-4；(2) 访问间隔从稀疏到稠密，如 *url*-5；(3) 突发性增长，如 *url*-3，*url*-6。

鉴于 LRU 算法的本地局部性与 LFU 算法的权值单调性，本文提出基于访问间隔变化的缓存替换算法，以提升缓存性能。

3 缓存替换策略

为了更直观地描述本文提出的算法，首先进行以下定义。

定义 1 访问间隔：缓存内对象的重引用距离，即本次命中该对象与上次命中该对象之间相差的缓存访问次数。

定义 2 访问密度：在一段时间内该对象被访问间隔变化率，即在一段时间内该对象被访问次数与缓存总访问次数的比值。

由于 LRU 的局部性与 LFU 的缓存污染性质，本文提出基于访问密度的替换策略(cache policy based on access density，CPBAD)在更长的时段内观察缓存对象的变化趋势，并通过减少热度降低对象的权值有效避免缓存污染从而提高缓存命中率。图 4 直观表示了本文的替换算法，CPBAD 算法的命中率高于 LRU 和 LFU。通过图 4 观察到在步骤(7)之前，三种缓存策略在缓存中保存的对象一致，而由当对象‘5’到达时，由于‘2’和‘3’的访问密度一致，但‘3’的新鲜度优于‘2’而低于‘4’，同时‘4’的密度大于‘2’和‘3’，因此在 CPBAD 缓存中对象‘2’被替换，而在 LFU 中‘4’被替换，从而降低了命中率(本文假设在 LFU 和 CPBAD 中，当对象频度或密度一致时，按照出现时间进行替换)。

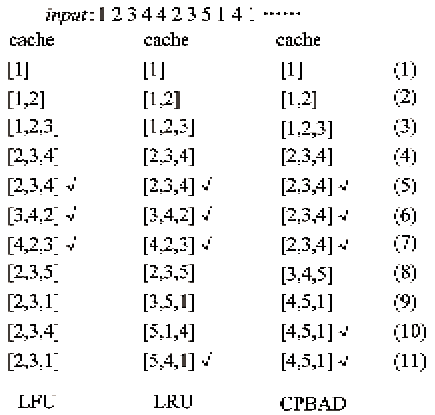

图 4 多种替换算法比较

与文献[14]提出的动态失效的最近频繁使用

(least frequently used with dynamic aging,LFUDA)算法类似,CPBAD 算法为每个缓存对象都设置失效时间以此避免彻底的缓存污染。同时为了解决计数器的存储开销,在周期维护缓存新鲜度的同时进行计数器重置。为了更清晰地描述 CPBAD 算法的替换过程,本文首先进行参数定义,如表 1 所示。当计数器值 C_{total} 超过给定阈值时,动态失效(dynamic aging,DA)模块将降低所有缓存对象的 $last_i$ 与 C_{total},以保证缓存对象密度的一致性。

表 1　CPBAD 算法参数表

参数	描述
C_{total}	某段时间内缓存被访问的总次数
$last_i$	对象 i 上次被访问时在总访问序列上的位置
now_i	对象 i 上次被访问时在总访问序列上的位置
ad_value_i	对象 i 的密度值
$freq_i$	对象 i 被访问的总频度
$avg\_accintvl_i$	对象 i 的平均访问间隔
$now\_accintvl_i$	对象 i 当前的访问间隔

为描述方便,本文假设缓存空间最多可容纳 M 个对象,若空间未满,则替换过程与其他缓存替换策略一致。当空间已满时,对于新到达对象 i,首先计算对象i的当前访问间隔$now_accintvl_i$,若该间隔低于 i 的平均间隔,意味 i 的热度呈下降趋势,因此降低其密度值,反之增加,如算法 1 所示,其中的 *cachepool. pop*( )表示将密度值最低的缓存对象删除,*cachepool. push*(*obj*)表示将新到来的对象放入缓存。整个替换策略的核心在于计算对象的密度值,如公式

$$ad\_value_i^n = \begin{cases} INITIAL\_VALUE, \ if\ obj_i\ is\ a\ new\ object \\ ad\_value_i^{n-1} \cdot \lambda \cdot \dfrac{avg\_accintvl_i}{avg\_accintvl_i + now\_accintvl_i}, \\ if\ avg\_accintvl_i < now\_accintvl_i \\ ad\_value_i^{n-1} \cdot (1 + \lambda \cdot \dfrac{now\_accintvl}{avg\_accintvl_i + now\_accintvl_i}), \\ if\ avg\_accintvl_i > now\_accintvl_i \end{cases} \quad 0 < \lambda < 1 \quad (1)$$

所示。考虑到一些对象的热度存在突发下降情况,此时式(1)中的 $avg\_accintvl_i$ 远低于 $now\_accintvl_i$,因此能够较好地解决该效应的消极结果。

Algorithm 1 Pseudo-code for cache replacement based on access density

```
Function CACHE-REPLACE (obj, cachepool, count_total)
{
  if  obj not in cachepool  {
    if  cachepool not full  then
        cachepool. put(obj)
        initial(obj. ad_value, obj. last, obj. freq, obj. avg_accintvl)
    else
        cachepool. pop( )
        cachepool. put(obj)
        initial(obj. ad_value, obj. last, obj. freq, obj. avg_accintvl)
  }
  else {
    now_accintvl = count_total - obj. last
    if  obj. freq = =1 then
        obj. avg_accintvl = now_accintvl
        obj. freq ++
        obj. ad_value = f(now_accintvl)//means inverse correlation between ad_value and now_accintvl
    else
        if  now_accintvl > avg_accintvl  then
            obj. ad_value = obj. ad_value · λ · obj. avg_accintvl / (obj. avg_accintvl_i + now_accintvl)
            obj. avg_accintvl = now_accintvl
```

```
        else
            obj.ad_value = obj.ad_value · (1 + λ · now_accintvl / (obj.avg_accintvl + now_accintvl))
        obj.avg_accintvl = (obj.avg_accintvl * obj.freq + now_accintvl) / (obj.freq + 1)
        obj.last = count_total
        obj.freq + + }
    count_total + +
}
```

为了加速缓存对象的插入和搜索,可采用树形结构进行对象存储,因此整个 CPBAD 算法的空间复杂度为 $O(\log n)$,其中 n 为缓存对象的个数。总而言之,CPBAD 算法通过访问间隔预测缓存对象的热度变化,有效避免 LRU 算法的局部性与 LFU 算法所造成的缓存污染。

4 实 验

为了有效验证 CPBAD 算法,本文在第 2 节中的实际网络数据集上与 LRU 和 LFU 算法进行比较。该数据集通过日期被分割为三个子数据集,如表 2 所示。为了确定公式(1)中的 λ 值,需要确定 λ 对算法的影响。因此本文首先生成 10000 个 URL,并且设置缓存可容纳 500 个 URL 对象,通过改变 λ 观察算法在不同的 α 分布下的性能。如图 5 所示,可明显看出 λ 在[0.6, 0.8]区间时,命中率较高。在之后的实验中,本文选择 λ = 0.8 作为实验数值。

表 2 网关日志数据集

数据集	URL 总数	相异 URL 总数
Dataset1	110745	41619
Dataset2	167397	73868
Dataset3	149494	52494

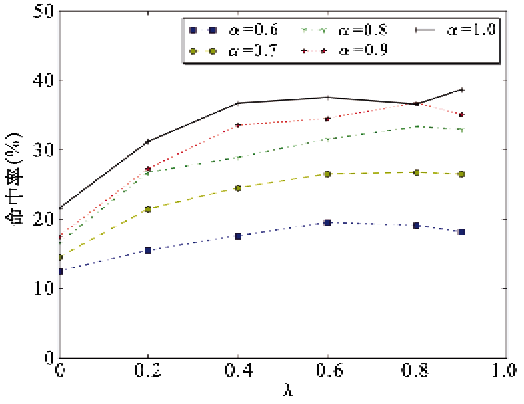

图 5 不同 *zipf* 分布下 λ 对算法的影响

为了更好地对替换策略进行比较,本文在 3 个子数据集和总数据集上进行测试。图 6 显示了不同算法下的命中率,可观察到 CPBAD 算法优于 LRU 和 LFU 算法。从图 6(c)中可知当缓存大小为 500 时,LFU 算法明显优于 LRU。而且在图 2 中可知,数据集 dataset3 的 α 值大于另外 2 个数据集,这意味着当 α 值越大,LFU 的命中率越高。同时当缓存大小为 2000 时,三种算法的命中率几乎一致,因为当缓存大小增加到一定时,即热门 URL 的数目与缓存大小相接近,增大缓存空间已无法提升命中率。对于图 6(d),虽然缓存空间增长到 8000,但是 LRU 算法依然低于其他,这是由于缓存中存储了大量热度较低的 URL。CPBAD 算法的优势在于当热门数据逐渐变冷时能够降低其在缓存空间中的权值,从而提升命中率。虽然图 6(c)中缓存空间为 2000 时,CPBAD 与 LFU 接近,甚至在缓存空间为 500 时,LFU 高于 CPBAD,这是由于持续热门数据的数量接近缓存空间,因此 LFU 中的缓存一直保持高命中状态,而对于 CPBAD 而言,由于密度值的变化弱于 LFU 以及保持缓存的新鲜度,从而将一些热门数据替换出缓存,进而导致命中率下降。但总体而言,CPBAD 算法在多数情况下优于 LFU 和 LRU。

对于实际部署该替换算法时,缓存空间限制应当考虑缓存对象总字节大小,因此字节命中率对于实际系统具有更高价值。考虑到 web 内容大小的多样性,本文假设 URL 页面的大小在(1KB,1MB)之间符合均匀分布。并针对式(1)进行修改,增加文件大小参数,如公式

$$size\_ad\_value_i^n = ad\_value_i^n + \log(obj\_size/cachesize) \quad (2)$$

所示,使用 $size\_ad\_value_i^n$ 作为缓存对象的权值进行计算。

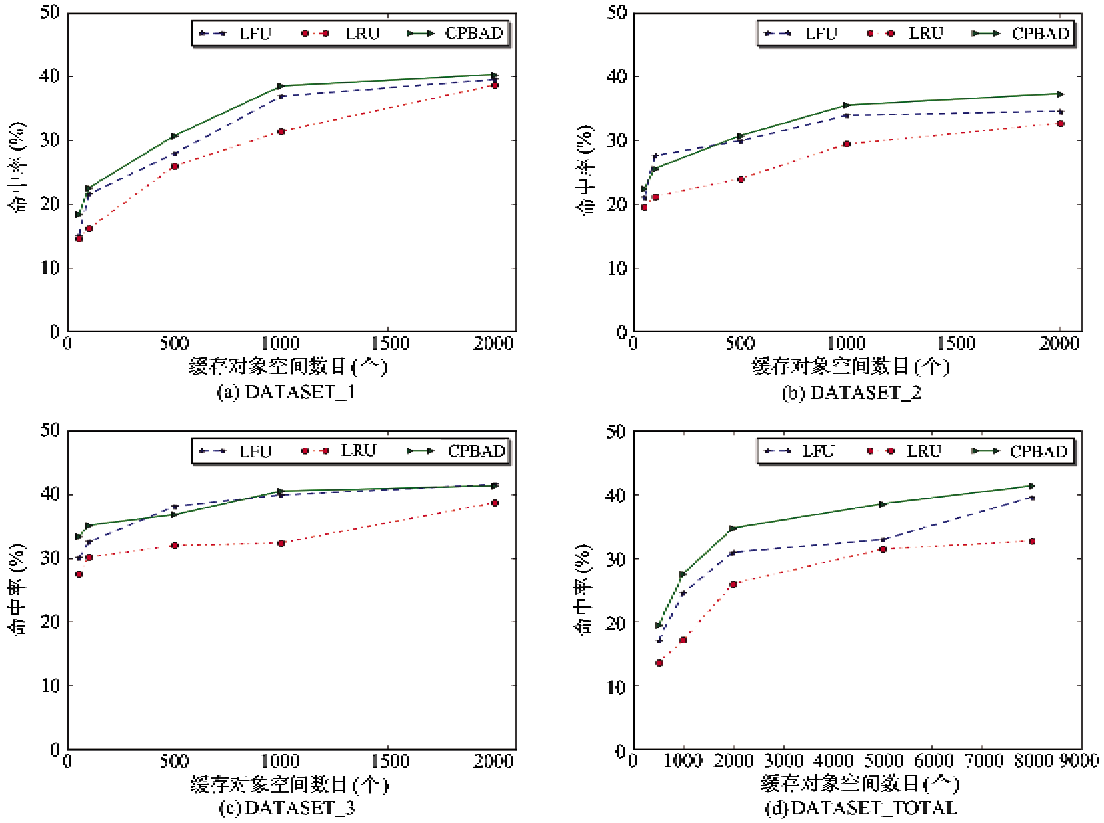

图 6　缓存替换算法命中率比较

在实验中假定缓存空间上限为 2GB,URL 分布使用总数据集 dataset。图 7 显示了每种缓存替换算法在不同缓存空间上的字节命中率。从图 7 中可见,当缓存较低时,GDSF 优于 CPBAD,然而当缓存空间逐渐增大,本文提出的算法明显优于 GDSF。这是由于在对象权值计算中,当空间足够时,一些呈现由热到冷的大文件并不能在 GDSF 中很快被替换,而 CPBAD 能够很快降低 ad_value 值,从而降低 $size\_ad\_value_i^n$,以达到更有效的空间利用率。

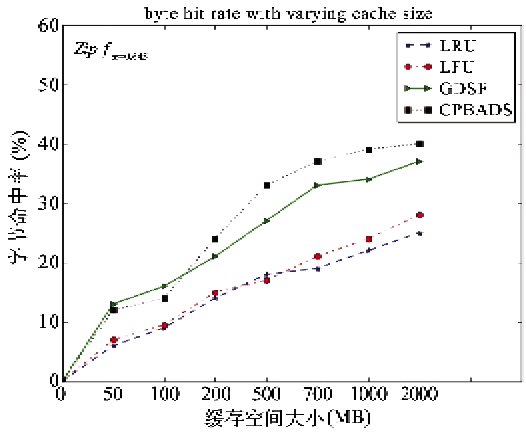

图 7　多种替换算法的字节命中率对比

总而言之,采用基于密度的缓存替换算法对于 Web 内容分发具有比 GDSF 更高的字节命中率,进而降低 Web 服务器的性能损耗。同时更高的字节命中率也意味着在分布式 Web 机群中的同步带宽消耗降低。

5　结　论

本文主要针对 Web 内容分发系统的缓存替换策略进行改进,提出了基于访问密度的缓存替换算法。该算法避免 LFU 的缓存污染与 LRU 的局部性缺陷,尽量在更长的时间内观察缓存对象的变化规律,得出其热度变化趋势,从而对缓存对象进行更准确的预测。单纯的命中率实验表明,CPBAD 算法的命中率比经典的 LRU 和 LFU 提升了 3% ~5%,同时在字节命中率实验中,对 CPBAD 算法增加空间大小因子,并与 GDSF 进行对比,字节命中率提升了 4% ~8%。

但是本文的字节命中率实验仅仅是在随机大小的缓存对象空间上进行,并未对实际网络中的 Web 对象大小分布进行更详细的测试,未来将在这部分进行更细致的工作。

参考文献

[1] Akamai Briefing: Highly Distributed Computing is Key to Quality on the HD Web. http://www.akamai.com/hd-wp.

[2] Jiang H, Wang Z, Wong A K, et al. A replica placement algorithm for hybrid CDN-P2P architecture. In: Proceedings of 15th International Conference on Parallel and Distributed Systems, Shenzhen, China, 2009. 758-763

[3] Pierre G, Vansteen M. Globule: a collaborative content delivery network. *IEEE Communications*, 2006, 44(8): 127-133

[4] Lu Z H, Gao X H, Huang S J, et al. Scalable and reliable live streaming service through coordinating CDN and P2P. In: Proceedings of the IEEE 17th International Conference on Parallel and Distributed Systems, Taiwan, China, 2011. 581-588

[5] Kang S, Yin H. A hybrid CDN-P2P system for video-on-demand. In: Proceedings of the 2010 Second International Conference on Future Networks, Sanya, China , 2010. 309-313

[6] Wu T, Starobinski D. A comparative analysis of server selection in content replication networks. *IEEE/ACM Trans on Networking*, 2008, 16(6): 1461-1474

[7] Zaman S, Grosu D. A distributed algorithm for the replica placement problem. *IEEE Trans on Parallel and Distributed Systems*, 2011, 22(9): 1455-1468

[8] Laoutaris N, Smaragdakis G, Bestavros A, et al. Distributed selfish caching. *IEEE Trans on Parallel and Distributed Systems*, 2007, 18(10): 1361-1376

[9] Abrams M, Standridge C R, et al. Caching proxies: Limitations and potentials. In: Proceedings of 4th WWW Conference. Boston, USA, 1995. 119 - 133

[10] Niclausse N, Liu Z, Nain P. A new efficient caching policy for the World Wide Web. In: Proceedings of the Workshop on Internet Server Performance, Madison, USA, 1998. 119-128

[11] Sathiyamoorthi V, Bhaskaran V M. Web caching through modified cache replacement algorithm. In Proceedings of 2012 International Conference on Recent Trends In Information Technology, Chennai, India, 2012. 483-487

[12] Ponnusamy S P, Kathikeyan E. Cache optimization on Hot-Point Proxy (HPProxy) using dual cache replacement policy. In: Proceedings of International Conference on Communications and Signal Processing, Bangalore, India, 2012. 108-113

[13] Cao P, Irani S. Cost-aware www proxy caching algorithms. In: Proceedings of the USENIX Symposium on Internet Technologies and Systems, Monterey, USA, 1997. 193-206

[14] Arlitt M F, Cherkasova L, Dilley J, et al. Evaluating content management techniques for web proxy caches. *ACM SIGMETRICS Performance Evaluation Review*, 2000, 27(4): 3-11

A novel cache replacement policy for Web content delivery

Li Qiao, He Hui, Fang Binxing

(Research Center of Network and Information Security, Harbin Institute of Technology, Harbin 150001)

Abstract

The Web content delivery and the cache replacement policy greatly affecting the performance of Web server were studied. Considering that the current cache replacement schemes mainly use the frequency and locality as the basis of replacement and it is found the access interval change rate is more valuable in predicting the new objects arrival through analyzing the real network logs, a novel cache replacement policy based on the access density and the object size was proposed. When using this novel method, the cache can achieve a higher byte hit ratio. The experimental results shows that the method improves 3% ~5% of the hit rate than the LRU (least recently used) and LFU (least frequently used), and 5% ~8% of the byte hit rate than the GDSF (greedy dual size and frequency).

Key words: Web cache, cache replacement, access interval, hit rate, byte hit ratio