Multicache-based Content Management for Web Caching *

Kai Cheng and Yahiko Kambayashi
Graduate School of Informatics, Kyoto University
Yoshida Sakyo, Kyoto 606-8501, Japan
E-mail: {chengk, yahiko}@isse.kuis.kyoto-u.ac.jp

Abstract

Large scale web caches are in fact localized sources of web contents. Besides replacement policies, which decide the contents of a cache, the management of such contents is an issue of special significance to web caching. This paper introduces our ongoing work on the research of cache content management. We propose a multicache based content management scheme in which web contents are distributed to several subcaches rather than maintained in a monolithic priority queue. This scheme is suitable for managing large cache space and capable of implementing sophisticated control logic. The efficiency of this scheme is demonstrated by design and implementation of a new replacement policy. Trace-driven simulations are used to evaluate these results.

1. Introduction

Data access from remote web sites is far more expensive than from local sources due to the scarcity of bandwidth resource and the inherent latency imposed by the velocity of light. Caching is such a technique by storing the frequently used data near the clients to reduce the remote accesses. Caching has been introduced into the web ever since it began to get popular [6, 10, 16]. Extensively employed caches can not only reduce network traffic and downloading latency, but can also distribute the workload from server *hot-spots* [1, 2, 4, 5, 12]. Today, caching has proved a highly successful technique to enhance the web infrastructure [9].

However, current researches on web caching are largely constrained by the assumptions such as small cache space, narrow time scales and limited hint information, which are primarily valid for conventional processor caching, virtual memory paging or database buffering [11]. As a result, only limited properties and simple control logic are found in cache management [1].

Large scale web caches are in fact localized sources of web contents rather than a heap of meaningless data as in traditional caching. On the one hand, replacement policies are required to decide the contents, or *what to cache*, to achieve high hit rates or other performance metrics. On the other hand, in order to make best use of such contents, it is important to develop suitable *content management strategies*. This is because: (1) web cache tends to be increasingly large in size and data can be probated in cache for some long time. So cache performance tends to be not so sensitive to replacement policies; (2) web cache is document-based. Semantics related to a document are important indicators to its usage. However, in current caching framework, such semantic information can hardly be utilized.

Content management ought to be studied at different stages. At the first stage, content management can support efficient implementation of sophisticated replacement policies. At the advanced stage, it is expectable to implement cache-based web information integration, in which cache is not just a passive tool in response to changing of access patterns, but it may become an active environment for integrating useful web information.

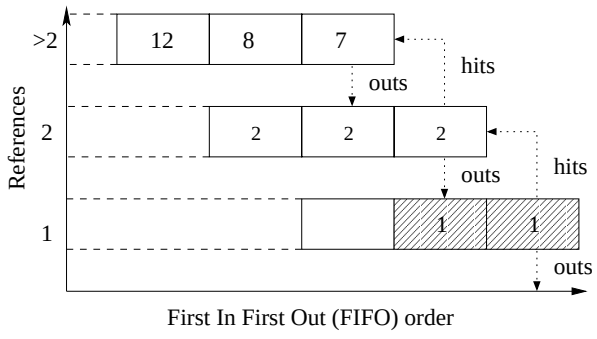

Figure 1. Managing LFU contents in multiple priority queues

In this paper, we focus on the former to explore the architectural aspect of cache content management for supporting

*In *Proc. 1st International Conference on Web Information Systems Engineering(WSIE'00)*, Hong Kong, June 2000

cache implmentation. As a motivating example, consider LFU (Least Frequently Used), a classic cache replacement policy always choosing the least frequently used object to evict. To select the least recently used object, a direct way is to maintain a monolithic priority queue based on reference times. However, the $O(\log(n))$ time complexity is expensive to large web caches. Another scheme as illustrated in Figure 1, implements multiple space-constrained FIFO queues, based on different reference times. Every hit or miss only requires a simple insertion or deletion operation with a overhead ($O(1)$). Long-term inactive objects will enter a lower FIFO queue until being evicted. This happen to be a substantial improvement to origin LFU because it can reduce inactive data fixed in cache ("cache pollution").

In this paper, we propose a *multicache based content management scheme*, in which web contents are allocated to several subcaches based on a so-called *Cache Knowledge Base (CKB)*. Each subcache manages its space and contents independently. Semantic information can be easily incorporated into cache management in the form of CKB knowledge segments or as cache constraints (see Section 2 for details). We have demonstrated the significance of our approach by developing and experimentally assessing the content management strategy for the new caching scheme, *Size-adjusted and Popularity-aware LRU (LRU-SP)*.

The remainder is organized as follows: Section 2 introduces the fundamentals of web cache. Section 3 describes the multicache based content management scheme. Section 4 contains the design of a new algorithm LRU-SP. In Section 5, we experimentally evaluate the introduced algorithms. In Section 6, we briefly review the previous work related with content management. Section 7 concludes this paper and describes the directions of future work.

2. Fundamentals of Web Cache

2.1. Cache Components

There exists some confusion over the definition of cache. In this work and in the web context, we refer to a cache as an autonomous agent, consisting of the following components (Figure 2):

1. **Space**: cache space, a limited storage space for keeping the selected content. No matter how large the cache space is, in the web context, it is impossible and unnecessary to keep all accessed data in cache. Inactive data or valueless data will eventually be flushed out.

2. **Contents**: a collection of objects chosen to stay in cache space. Object is any file with a URL(uniform resource locator) such as HTML documents, pictures, and video/audio files. Metadata about the web objects are critical information for cache management.

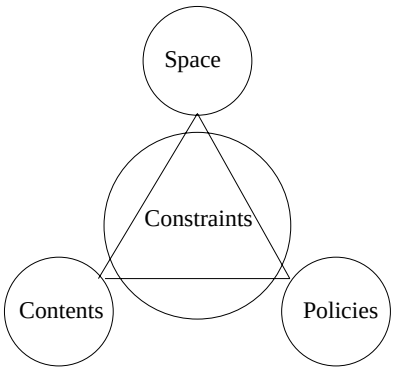

Figure 2. Cache components

3. **Policies**: replacement policies dedicated to maintaining the cache space and object space. A bad replacement policy may be prone to *cache pollution* that inactive objects tend to fix in cache or *early eviction* that real popular objects are purged out too early.

4. **Constraints**: special conditions imposed on the cache. Constraints are important because not all data can always be cached or be dropped out.

2.2. Hint Information for Cache Management

Web caching is based on whole document due to the restriction in HTTP protocols that support whole file transfers only, so a cache hit only happens if entire file is in cache. There are two categories of information are much more important in management of cache contents.

System-oriented information. Web documents vary in size, format, retrieval cost and other physical properties. These properties, together with data about document usage, such as frequency and recency of reference, are easy to quantify and most of them have been utilized in replacement policies. Other system-based properties include URL address, links, estimated bandwidth and the forth.

Semantics-based information. Web document is a complete semantic unit. since most users browse the web in a rational way according to their needs and interest, semantic information such as document content, user profile and access history, could be more direct indicators to future use of this and the related documents. However, such information can hardly be implemented in current cache management schemes.

2.3. Cache constraints

It is necessary to impose various restrictions on each component of a cache and the relationships between them.

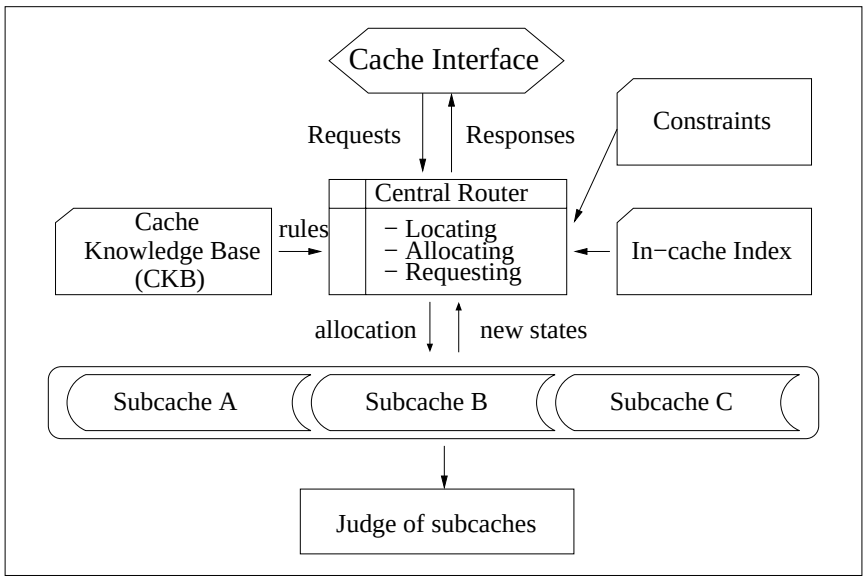

Figure 3. The Multicache-based Content Management Scheme

For example, dynamic pages, too large documents, or objects that are prohibited from caching will not be accepted in cache. Constraints are conditions that impose special restrictions to cache:

Admission constraints. Admission constraints refers to such requirements that a newly arrived object should meet. A new object that does not satisfy such constraints will be rejected for caching. Admission constraints will be validated for new objects, or a cached object with new modifications such as size changed. An example of admission constraint is:

```
Object sizes must be within 2 MB
```

Freshness constraints. A cache copy must be confirmed to be fresh enough before sending the copy to the requesters. A document becomes stale due to modifications. If the copy has already become stale, the cache should contact origin site for a fresh version. For example,

```
Objects without TTL specification can-
not stay longer than 7 days
```

Miscellaneous constraints. For example

```
At the end of a day, to-
tal bytes in cache should not ex-
ceed 95 of total cache space.
```

3. Multicache-based Content Management

Since single priority queue is neither suitable for managing large cache space nor good for incorporating higher semantics information, we propose a general mulitcache based content management scheme. As depicted in Figure 3, there are four major components: *Central Router, Cache Knowledge Base (CKB), subcaches and judge of subcaches*. Central router plays a central role in controlling and coordinating these components.

3.1. Cache Knowledge Base (CKB)

Cache Knowledge Base (CKB) consists of a set of rules designed for classification of web contents. Semantics information can be used here to direct the cache. Sophisticated rules involve more control logic. Based on such rules, we can implement more advanced content management. For example,

R1: allocate(X, $\lfloor log_2(s) \rfloor$):-size(X, s).
R2: allocate(X, 1):-url(X, U), match(U, *.jp),
content(X, baseball), type(X, picture).

Here, R1 tells us object to keep object of a size s in subcache $\lfloor log_2(s) \rfloor$. R2 directs pictural document in subcache 1 if it is about baseball and comes from Japan.

3.2. Subcaches

A subcache is an independent cache that has its own cache space, contents, replacement policy and constraints. Since objects in a same subcache are usually similar in some properties, subcache can use simpler replacement policy. For example, if classification rule is R1 as in previous example, then objects in the same subcache have the similar

sizes. As a result, the subcache need not take size into account. For this reason, simple policies such as LRU, FIFO, CLOCK, and SIZE are often used in this case.

3.3. Judge of Subcaches

The *candidate set of evictions* selected by subcaches are to be assessed by a judge to make final decision. Comprehensive criteria are used in this judge process. The judgments can: *eviction*, *re-caching*, or *probation*. An eviction object will be purged at once. However, if an object is selected to re-cache or to probate, it will remain in cache. The distinction between re-cache and probation is: an object to be re-cached will be cached at once in a certain *subcache*, while a probation object will be held by the judge in its own space and the final decision is expected to make in the next turn.

3.4. Algorithm Framework of Multicache-based Content Management

As a request comes from a client, how is it being serviced? The algorithm is as described in Figure 4: At first, it is looked for in a in-cache index to see whether or not it has been cached. For a cache miss, the request will be forwarded to the origin server, then the downloaded document is returned to client. For a hit, the document will be returned to client if the freshness constraints are validated. Otherwise, a conditional HTTP GET request will be sent to origin server to see if the cached copy is fresh.

The Central Router is responsible for mediating subcaches to work together: a document may be cached in several different subcaches in terms of its states evaluated by CKB-driven classification.

4. Content Management in LRU-SP

To demonstrate the feasibility of the ongoing discussed approach, we will consider a new caching scheme. Since it is built on an existing LRU extension, Size-Adjusted LRU, we will begin with a review of it.

4.1. Review of Size-Adjusted LRU

Size-Adjusted LRU (SLRU) is a generalized LRU replacement algorithm, proposed by Charu Aggarwal et al in a recent paper [2]. To select a victim when there is no enough space for a new object, objects in cache are indexed in order of nondecreasing values of $S_i \cdot \Delta_{it}$:

$$S_1 \cdot \Delta_{1t} \leq S_2 \cdot \Delta_{2t} \leq \cdots \leq S_k \cdot \Delta_{kt}$$

Algorithm
Central Touter services each request ;
Suppose current request is for document p;
(1) Locating p by **In-cache Index**;
(2) If p is not in cache, download p;
1) Validate **Constraints**, if $false$, loop;
2) Fire rules in **CKB**, let **subcache** ID = K;
3) While no enough space in subcache K for p
Subcache K selects an eviction ;
If space sharing, other subcaches do same;
Judge assesses the eviction candidates;
Purge the victim; end While
4) Cache p in subcache K
(3) If p is in subcache , do 1) - 4) re-cache p .

Figure 4. Algorithm Framework of Multicache Based Content Management

(where k is the total number of objects in cache). The highest index objects are greedily picked one by one and purged from the cache until sufficient space being made.

The content management scheme for SLRU is called Pyramidal Selection Scheme (PSS), in which objects are classified into a limited number of groups based on $\lfloor log_2(size) \rfloor$, so that objects within a same group are similar in sizes. Each group is maintained using a LRU mechanism. The basic Size-Adjusted LRU policy only applies to a limited set of least recently used objects from all nonempty groups to make final decision. The object with largest $(S_i \cdot \Delta T_{it})$ will be purged from the cache.

The major problem with Size-Adjusted LRU is its ignorance of long-term popularity and the strong bias towards small objects. Larger objects can hardly compete with smaller ones for surviving in cache, no matter how popular is really being. For example, if requests occur periodically (Figure 5(a)) and each subsequent request occurs before the object ages out, it will stay in the cache. However, if accesses are locally concentrated, the object may have aged out from cache before it gets the next access (Figure 5(b)) even if the average occurrence is the same as the previous case.

4.2. LRU-SP: Incorporating Frequency in Size-Adjusted LRU

To differentiate popularities, it is necessary to incorporate one more property, i.e. *frequency of reference*. We use an accumulative cost-to-size ratio to handle frequency, that

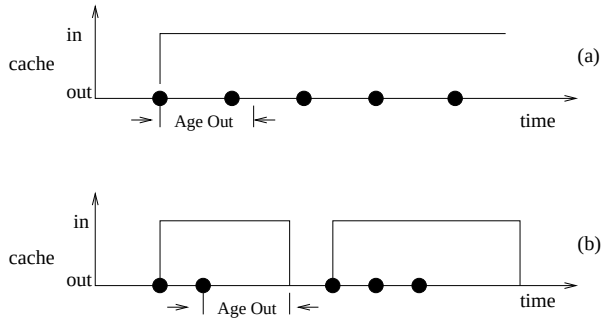

Figure 5. Popular Objects Being Early Evicted

is to say, accumulating the cost-to-size ratio every time an object gets a new access. Given $nref_i$ is the access times since it being cached, then

$$(nref_i/(S_i \cdot \Delta T_{it}))$$

In practice, the objects are indexed in order of nondecreasing values of $\frac{S_i \cdot \Delta_{it}}{nref_i}$:

$$\frac{S_1 \cdot \Delta_{1t}}{nref_1} \leq \frac{S_2 \cdot \Delta_{2t}}{nref_2} \leq \cdots \leq \frac{S_k \cdot \Delta_{kt}}{nref_k}$$

(where k is the total number of objects in cache) then the highest index objects are greedily picked one by one and purged from the cache until sufficient space being made. We call the policy Size-adjusted and Popularity-aware LRU, or LRU-SP.

Based on this benefit-to-cost model, the influence of rereferences is accumulated (represented by sum) so that the objects with more references can stay more time before being driven out. Figure 6(a) and Figure 6(b) show the cases.

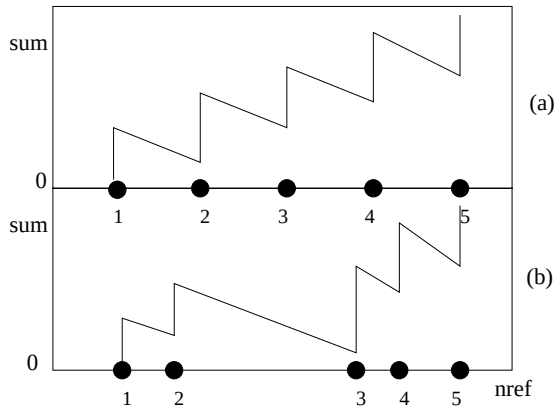

Figure 6. Popular Objects Remain In Cache

4.3. Content Management for LRU-SP

Based on the previous discussion, we devise the following content management scheme for LRU-SP:

1. Subcaches: objects are classified into several subcahes on the basis of $\lfloor log_2(S_i/nref_i) \rfloor$ instead of $\lfloor log_2(S_i) \rfloor$. In ohter words, the cache knowledge base contains a classification rule
   R1: allocate(X, $\lfloor log_2(s) \rfloor$):-size(X, s).

2. Each subcache uses an LRU policy.

3. A hit may make the requested object move to new LRU subcache dependent on its new value of $\lfloor log_2(S_i/nref_i) \rfloor$;

4. The judge of subcaches chooses a victim with largest $(\Delta T_{it} \cdot S_i/nref_i)$ from the candidate evictions based on the new cost-to-size model.

Intuitively, an object with more accesses has been treated as a smaller one $(S_i/nref_i)$, which, to some degree, awards the popular object to become competitive with less popular but small objects. The number of subcaches is reasonable to be no larger than 20, because the objects larger than 1MB ($2^{20}B$) are quite rare.

Maintaining a LRU queue just requires a tail insertion/head taking and incurs no overhead and choosing the final victim only needs constant times (less than 20) of comparisons. In all, the overhead of this scheme is independent of the scales of object space and cache space, in other words, it is $O(1)$ in any cases. Further, LRU-SP has no space boundaries for LRU queues, so it is a parameter-free.

5. Performance Evaluation

5.1. Data Collections

We have two datasets to be used in driving our caching simulator. The dataset NLANR in Table 1 is a one-week top level caching proxy traces publicly available (ftp://ircache.nlanr.net/Traces/). This dataset contains 1,848,319 requests with total 21.0 GB Web data, where unique data is 15.9 GB with a maximum hit rate 0.228 and byte hit rate 0.245. The KAMB dataset is collected from the logfiles of the Squid proxy server in our laboratory. It contains 873,824 requests with total 23.6 GB Web data, where unique data is 21.3 GB with a maximum hit rate 0.251 and byte hit rate 0.098. This low byte hit rate indicates popular objects concentrate in smaller objects.

Dataset	Duration	Total Requests	Total Bytes	Unique Bytes	HR_{max}	BHR_{max}
KAMB	one month	873,824	23.6 GB	21.3 GB	0.251	0.098
NLANR	one week	1,848,319	21.0 GB	15.9 GB	0.228	0.245

Table 1. Profiles of Trace Datasets

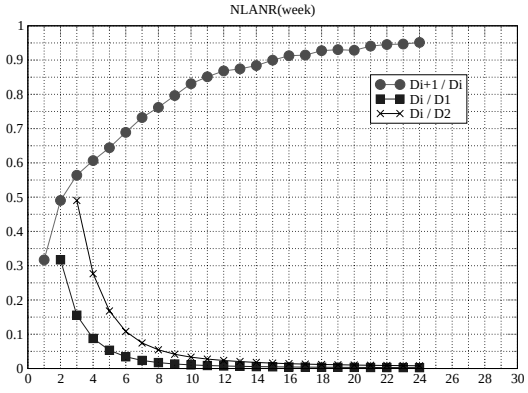

Figure 7. Reference Probability as A Function of Frequency

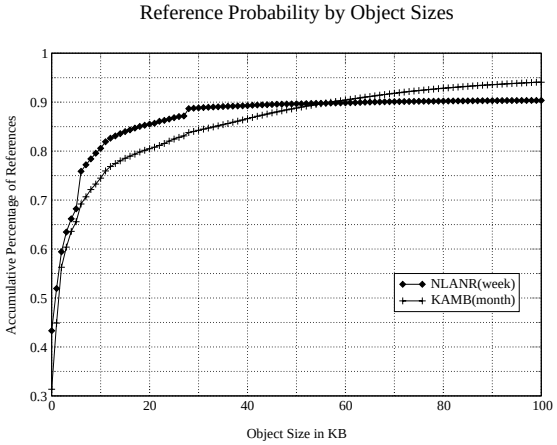

Figure 8. Accumulative Size Distribution

Popularity vs. Access Frequency

We first analyze the relationship between access frequency and the probability of next access. The topmost plot of Figure 7 depicts how probable an object gets one more reference given it has got i accesses. D_j/D_i means when an object already has i references, how probable it will get $j's$ reference. With the increment of i, D_{i+1}/D_i goes up quickly. This indicates that the more frequently an object have been accessed, the more possible it will re-accessed in the future. D_{i+1}/D_i jumps steeply at first few points until 8-10, which implies no need to consider frequency if is exceed a small threshold.

Access Times vs. Object Sizes

The distribution of sizes also appears similar locality: the objects below a small threshold, say 10KB, get quite a majority of accesses as depicted in Figure 8.

5.2. Caching Schemes for Evaluation

Apart from *LRU-SP* and *Size-Adjusted LRU*, the other two caching schemes we choose to evaluate are *Segmented LRU* and *Least Recently Value (LRV)*.

Segmented LRU is a frequency-based extension of basic LRU [8], where objects with different access frequency are maintained in different partitions or segments of cache space. Since Segmented LRU was especially designed for disk caching, it did not take into account variable object sizes. In addition, to compare with other schemes, we extend the basic Segmented LRU by partitioning cache space into more segments than only 2 in basic Segmented.

Least Recently Value (LRV) is a new caching scheme especially designed for Web caching [13]. The designers have developed an elaborate function to handle various characteristics of Web objects. An efficient content management scheme is given to LRV, which also classifies objects into a few groups according to their access frequency. Objects in the first group are maintained using a unit caching policy SIZE, whereas the rest groups are FIFO lists. Final decisions are made by a central cache which is based on LRV-function.

5.3. Simulation Results

We have carried out simulations on both trace datasets of NLANR and KAMB. In addition to evaluation of LRU-SP and its predecessors Size-Adjusted LRU and Segmented LRU, we also compare them to *Least Relative Value (LRV)* [13], a well-know replacement algorithm, which also takes object sizes, recency and frequency of reference into account. LRV is based on extensive Web workload analyses and an elaborate cache design.

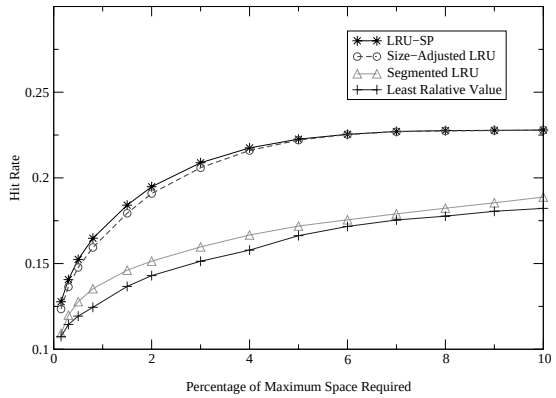

Figure 9. Hit Rates under NLANR Dataset

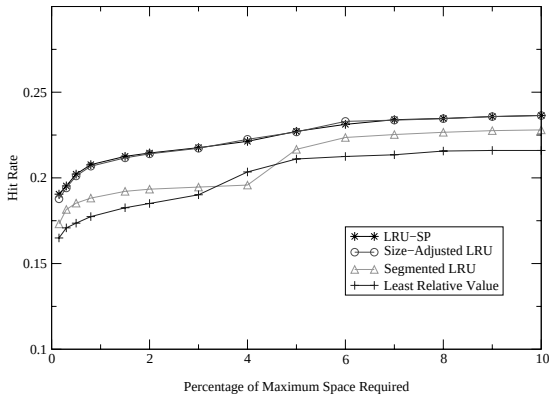

Figure 11. Hit Rates under KAMB Dataset

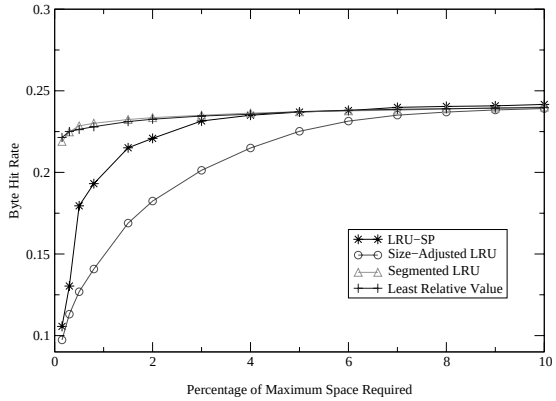

Figure 10. Byte Hit Rates under NLANR Dataset

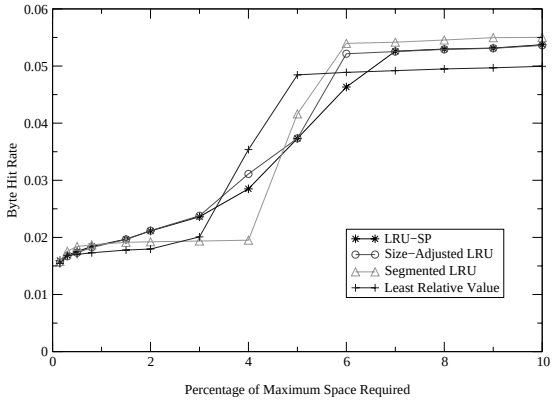

Figure 12. Byte Hit Rates under KAMB Dataset

Outperforming Segmented LRU

LRU-SP achieves much higher hit rates than Segmented LRU under two datasets, (Figure 9 and 11). However, LRU-SP performes almost as well as Segmented LRU in terms of byte hit rates.

This is because Segmented LRU cares nothing about object sizes, thus quite a number of smaller objects are displaced by bigger objects. Whereas LRU-SP balances well between object sizes and popularity, so it preserves a large number of smaller and popular objects in cache, which guarantees high hit rate but no harm to byte hit rates.

Outperforming Size-Adjusted LRU

LRU-SP significantly improves Size-Adjusted LRU in byte hit rate under dataset NLANR (Figure 10) without loss of hit rate. This result is not so good under dataset KAMB because the access pattern of KAMB biases towards so many small objects that the strategy of awarding bigger yet popular objects became not so effective.

LRU-SP performing better than Size-Adjusted LRU demonstrates that LRU-SP has really retained bigger objects that are popular enough to make up the loss of cache space.

Outperforming Least Relative Value

LRU-SP also outperforms LRV in most cases, especially in terms of hit rate. The reason is the pure LRV is unrealistic to implement. The simplified implementation of LRV given by its designers only differentiates object sizes in its first queue (for objects with only one access). While in LRU-SP, objects with different sizes are distributed among several LRU queues. So LRU-SP makes better advantage of information about sizes than LRV. Consequently, LRU-SP can achieve higher hit rates than LRV.

6. Related Work

This work is based on extensive studies on various caching schemes. There are quite a number of specific content management schemes for supporting implementation of high performance low overhead cache replacement policies, especially in the research of operating systems and database systems. For example, *Segmented FIFO* in [17]; *FBR* in [14]; *Segmented LRU* in [8]; *2Q* in [7]. Ozalp Babaoglu et al in [3] analyzed a family of two-level content management schemes. Performance models are developed to measure these them.

In Web caching, most caching schemes have a content management scheme to support efficient implementation. *Least Relative Value (LRV)* proposed by Luigi et al [13] uses 10 FIFO queues and a SIZE priority queue to management cache documents. Only several eviction candidates are evaluated by pure LRV. *LNC-R-W3-U* by Shim et al. [15] is a caching scheme that combines cache replacement policy and cache consistency policy in a unified algorithm. The content management scheme in LNC-R-W3-U uses K priority lists, each list with different reference times. The priority queue is based on the calculation of profits.

However, compared to these ad hoc schemes, the significant improvement of our scheme is that the object classification is rule-based so that various semantics information can be used.

7. Conclusion and Future Work

Owing to the increasingly large space and datasets, content management has special importance to Web caching. In this paper, we have proposed a multicache-based scheme for content management to facilitate the design of efficient implementation of sophisticated replacement policies. We have demonstrated the efficiency by design and implementation of LRU-SP, a new efficient caching policy for Web caching.

As we have mentioned in Section 1, in the advanced stage of our content management scheme, more semantics information and control logic will be utilized to implement cache-based web information integration. To do this, we will first focus on the management of cache knowledge base (CKB).

References

[1] M. Abrams, C. R. Standridge, G. Abdulla, S. Wililams, and E. A. Fox. Caching Proxies: Limitations and Potentials. In *Proceedings of the Fourth International WWW Conference*, 1995.

[2] C. Aggarwal, J. L. Wolf, and P. S. Yu. Caching on the World Wide Web. *IEEE transactions on knowledge and data engineering*, 11(1), 1999.

[3] O. Babaoglu and D. Ferrari. Two-Level Replacement Decision in Paging Stores. *IEEE Transactions on Computers*, c-32(12):1151–1159, December 1983.

[4] J.-C. Bolot and P. Hoschka. Performance Engineering of the World Wide Web: Application to Dimensioning and Cache Design. In *Proceedings of the Fifth International World Wide Web Conference*, May 1996.

[5] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrel. A Hierarchical Internet Object Cache. In *Proceedings of the USENIX Annual Technical Conference*, pages 153–163, January 1996.

[6] S. Glassman. A Caching Relay for the World Wide Web. In *Proceedings of the First International World Wide Web Conference*, Geneva (Switzerland), May 1994.

[7] T. Johnson and D. Shasha. 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm. In *Proceedings of 20th VLDB Conference*, Santiago, Chile, September 1994.

[8] R. Karedla, J. S. Love, and B. G. Wheery. Caching Strategies to Improve Disk System Performance. *IEEE Computer*, 27(3):38–46, March 1994.

[9] B. Krishnamurthy, J. C. Mogul, and D. M. Kristol. Key Differences Between HTTP/1.0 and HTTP/1.1. In *Proceedings of the Eighth International World Wide Web Conference*, May 1999.

[10] A. Luotonen. World-Wide Web Proxies. In *Proceedings of the First International World Wide Web Conference*, Geneva (Switzerland), May 1994.

[11] T. H. Merrett, Y. Kambayashi, and H. Yasuura. Scheduling of page-fetches in join operations. In *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*, pages 488–498. IEEE Computer Society Press, 1981.

[12] J. E. Pitkow and M. M. Recker. A Simple Yet Robust Caching Algorithm Based on Dynamic Access Petterns. Technical Report VU-GIT-94-39, GVU Technical Report, 1994.

[13] L. Rizzo and L. Vicisano. Replacement Policies for a Proxy Cache. Technical report rn/98/13, University College London, Department of Computer Science, Gower Street, London WC1E 6BT, UK, 1998. http://www.iet.unipi.it/ luigi/ caching.ps.gz.

[14] J. T. Robinson and M. V. Devarakonda. Data Cache Management Using Frequency-based Replacement. *Performance Evaluation Review*, 18(1):134–142, May 1990.

[15] J. Shim, P. Scheuermann, and R. Vingralek. Proxy Cache Design: Algorithms, Implementation and Performance. *IEEE Transactions on Knowledge and Data Engineering*, 1999.

[16] N. Smith. What can Archives offer the World Wide Web? In *Proceedings of the First International World Wide Web Conference*, Geneva (Switzerland), May 1994.

[17] R. Turner and H. Levy. Segmented FIFO Page Replacement. In *Proceedings of ACM SIGMETRICS'81*, pages 48–51, 1981.