석 사 학 위 논 문

프락시 캐쉬를 위한 바이트 단위의 최소 인기도 우선 대체 알고리즘

김 경 백 (金 俓 伯)

전자전산학과 전기및전자공학전공

한국과학기술원

2001

프락시 캐쉬를 위한 바이트 단위의 최소 인기도 우선 대체 알고리즘

Least Popularity per Byte Replacement Algorithm for a Proxy Cache

Least Popularity per Byte Replacement Algorithm for a Proxy Cache

Advisor : Professor Daeyeon Park

by

Kyungbaek Kim

Department of Electrical Engineering & Computer Science
Division of Electrical Engineering
Korea Advanced Institute of Science and Technology

A thesis submitted to the faculty of the Korea Advanced Institute of Science and Technology in partial fulfillment of the requirements for the degree of Master of Engineering in the Department of Electrical Engineering & Computer Science Division of Electrical Engineering

Taejon, Korea
2000. 12. 19
Approved by

Professor Daeyeon Park
Major Advisor

프락시 캐쉬를 위한 바이트 단위의 최소 인기도 우선 대체 알고리즘

김 경 백

위 논문은 한국과학기술원 석사 학위논문으로 학위논문 심사위원회에서 심사 통과하였음.

2000년 12월 19일

심사위원장 박 대 연 (인)

심사위원 박 규 호 (인)

심사위원 백 윤 흥 (인)

MEE 993050

김 경 백. Kyungbaek Kim. Least Popularity per Byte Replacement Algorithm for a Proxy Cache. 프락시 캐쉬를 위한 바이트 단위의 최소 인기도 우선 대체 알고리즘. Department of Electrical Engineering & Computer Science Division of Electrical Engineering. 2001. 44p. Advisor: Prof. Daeyeon Park. Text in English.

With the recent explosion in usage of the world wide web, the problem of caching web objects has gained considerable importance. Web caches cannot only reduce server load, network traffic and downloading latency by replicating popular web objects on proxy caches. The performance of these web caches is highly affected by the replacement algorithm. Today, many replacement algorithms have been proposed for web caching and these algorithms use the other on-line fashion parameters like size, temporal locality and latency to define the object popularity rather than the object popularity value directly from the cache, especially in the Size Adjust LRU which uses size and temporal locality. But, recent studies suggest that the correlation between the on-line fashion parameters , especially temporal locality and the object popularity in the *proxy cache* is weakening due to the efficient *client caches*. In this paper, we suggest a new algorithm, called *Least Popularity Per Byte Replacement*(LPPB-R). This LPPB-R algorithm is the extension of the Size Adjust LRU. We use the popularity value as the long-term measurements of request frequency to complement the weak point in the temporal locality and vary the popularity value by changing the impact factor easily to adjust the performance to needs of the proxy cache. In addition, we apply the multi queue by managing the objects and the meta information and suggest a technique for managing objects to avoid the cache pollution phenomenon. And we examine the performance of this and other replacement algorithm via trace driven simulation.

Contents

1 Introduction **1**

2 Related Work **6**

2.1 Traditional Replacement algorithm 6

2.2 Key-based Replacement algorithm 7

2.3 Function-based Replacement algorithm 9

2.4 Parameters 11

3 Lowest Popularity Per Byte Replacement Algorithm **13**

3.1 Overview of LPPB-R 13

3.2 Getting the Popularity Value 14

3.3 Managing the objects 17

3.4 Avoiding the cache pollution phenomenon 20

4 Performance evaluation **23**

4.1 Traces used 23

4.2 Performance metrics and algorithms 25

4.3 Implementation issues on LPPB-R 26

4.4 Performance Measurement 28

4.4.1 Hit rate 29

4.4.2 Byte hit rate 29

4.4.3 Reduced Latency 35

5 Conclusion **39**

6 Future Work **40**

Summary (in Korean) **41**

List of Figures

1.1 Various caching points 2

2.1 The conceptional view of the replacement algorithms 9

3.1 Popularity values for variable β values 16

3.2 Multiqueue structure used by LPPB-R 19

3.3 This figure show that objects follow the theory, $2 \cdot U(i) > U(p)$ . . 20

3.4 Cache pollution avoidance mechanism 22

4.1 Performance comparison between ideal and practical algorithm . . . 27

4.2 Hit Rates of LPPB-R 2 for each β value 30

4.3 Hit Rates of many algorithms for each traces 31

4.4 Byte Hit Rate of LPPB-R 2 for each β value 33

4.5 Byte Hit Rate of many algorithms for each traces 34

4.6 Reduce Latency of LPPB-R 2 for each β value 37

4.7 Reduce Latency of many algorithms for each traces 38

List of Tables

2.1 Examples of key-based algorithms 8

4.1 Traces used in our simulation (Cache size = infinity) 24

1. Introduction

The recent increase in popularity of the World Wide Web has led to a considerable increase in the amount of traffic over the Internet. As a result, the Web has now become one of the primary bottlenecks to network performance. When objects are requested by a user who is connected to a server on a slow network link, there is generally considerable latency noticeable at the client end. Further, transferring the object over the network leads to an increase in the level of traffic. This has the effect of reducing the bandwidth available for competing requests, and thus increasing latencies for other users. In order to reduce access latencies, it is desirable to store copies of popular objects closer to the user.

Consequently, web caching has become an increasingly important topic [1] [10]. Web caching aims to reduce network traffic, server load, and user-perceived retrieval delay by replicating popular content on caches that are strategically placed within the network. Caching can be implemented at various points in the network. In the server side, there is typically a cache in the *Web server* itself and there is a cache which behaves like a *web server accelerator*. In the opposite side, there are *client caches* in the Web browsers. And it is increasingly common for a university or corporation to implement specialized servers in the network called *caching proxies* [2]. Figure 1.1 shows the various caching points. In this paper, we shall discuss the cache replacement policies designed specifically for use by proxy web caches.

In the recent studies, we can find several important reasons why many people try to enhance the performance of proxy caches by using the efficient replacement policy. First, the growth rate of Web capacity is much higher than the rate with which memory sizes for Web caches are likely to grow [2] [3]. This tell us that we need much more storage cost to get more Hit Rate by using the same replacement

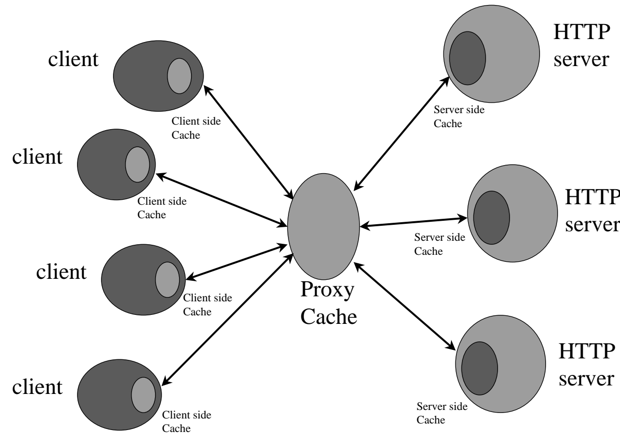


Figure 1.1: Various caching points

policy. Second, recent studies have shown that Web cache Hit Ratio and Byte Hit Ratio grow in a log-like fashion as a function of the cache size [4] [5]. Thus, a better algorithm that increases Hit Ratio and Byte Hit Rate by only several percentage points would be equivalent to a several fold increase in cache size. Third, for an infinite sized cache, the Hit Ratio for a web proxy grows in a log-like fashion as a function of the client population of the proxy and of the number of requests seen by the proxy [4] [5]. In this case, we get the same result as the result from second reason. Then, when we use a better replacement policy, the proxy can serve much more clients and requests than before. Finally, the benefit of even a slight improvement in cache performance may have an appreciable effect on network traffic.

Today many replacement algorithms have been proposed for web caching. We have studied several algorithms from simple to complex. According to these researches, we classify the replacement algorithms into three categories. (Traditional algorithm, Key-based Algorithm, Function-based Algorithm). In the beginning to implement proxy cache, most proxy caches use traditional algorithms like LRU,

LFU. Because these algorithms are easy to implement and to manage, and the using in main memory paging proves that these are robust. But these algorithms don't consider the web characteristic.

Key-based algorithms are proposed by using several parameters like an object size, a latency, a last access, references, long-term frequency. Key-based algorithms evict objects based on a primary key, break ties based on a secondary key, break remaining ties based on a tertiary key, etc. There are LFF [3], LRU-min [3], Hyper-G [3], Lowest-Latency First [6]. But these algorithms are good for a specific metric. For example, size parameter is good for Hit Rate, but it is not good for other metrics. So when we use LFF for the proxy cache, we get very high Hit Rate, but get poor Byte Hit Rate.

To cover this limitation, recent researches propose Function-based algorithms. These algorithms use the object utilization that is calculated by many parameters which relate to web characteristics. There are many policies, like Hybrid [6], Greedy Dual-Size [5], Size-adjust LRU [7], to make the performance of proxy caches better than before. The object utilization which used by these algorithms infer the popularity of the object. But in the most algorithms, this utilization is calculated by estimate value like a size, a latency, a last access rather than the direct popularity value of object like the relative access frequency or reference counts through a proxy. Especially in the case of Size-adjust LRU, this algorithm use the size and the last access number to get the object utilization. Each of these parameters means the negative correlation between the popularity and the object size, and temporal locality in a request stream. Both of these are assumed to be indicative of the future popularity of the object, and hence reflective of the merit of keeping such and object in the cache. But recent studies [9] suggest that such relationships are weakening and hence may not be effective in capturing the popularity of Web objects.

In this paper, we suggest a new algorithm, called *Least Popularity Per Byte Replacement*(LPPB-R). This LPPB-R algorithm is an function-based algorithm.

The purpose of the LPPB-R is to make the popularity per byte of the outgoing objects to be minimum. When an object is to be inserted into the cache, the LPPB-R calculates the utilization of each object, then evict the object which has smallest utilization. This utilization value is calculated by using the popularity and the size. In this point, we have a difference between LPPB-R algorithm and the others. The LPPB-R algorithm is the extension of SLRU that uses temporal locality and size by the main parameters to calculate the utilization. The LPPB-R algorithm uses the object popularity parameter, which is not considered by SLRU algorithm, from proxy cache directly, to make up for the weak points in the SLRU algorithm.

According to the recent studies [9], we know the relationship of the temporal locality and the popular object is weakening in these days. One reliable reason for getting this result is the efficient client caching. Therefore, the efficient client caching deals with the short-term measurement of request frequency like temporal locality property. Then, to get better performance of the proxy cache, the proxy cache should deal with the long-term measurement of request frequency. Consequently, we use the object popularity as the long-term measurement of request frequency. And because how to set the popularity value determines the performance of this LPPB-R algorithm, we suggest the 2 type of calculating the popularity value. One is the simple mechanism by using the reference count, the other is the advanced mechanism by using the reference count as the power factor of the impact factor. Then, we want to accommodate a variety of performance goals by changing the β value which is the impact factor.

In addition, we use the multi queue by managing the objects and the meta information in the cache, to make the LPPB-R algorithm more practical and to reduce the overhead which is needed by calculating the utilization values and processing the requests. And because our LPPB-R algorithm has the LFU property by using the popularity value, we must consider the cache pollution phenomenon. Therefore, we suggest a technique for managing objects to avoid the cache pollution phenomenon.

The paper is organized as follows. In section 2, we describe several related works about the replacement algorithm. Section 3 introduce our new replacement algorithm , LPPB-R. The simulation environment and the performance evaluation are given in section 4. We conclude the paper in section 5. Finally, we suggest future works in section 6.

2. Related Work

A key aspect of the effectiveness of proxy caches is a document replacement algorithm that can yield high hit rate. There are many replacement algorithms to obtain better performance of proxy caching, from simple to complex. They attempt to minimize various cost metrics, such as hit rate, byte hit rate, and average latency. We classify these algorithms into three categories; Traditional algorithm, Key-based algorithm, and Function-based algorithm. Bellow we give a description of all of them. In describing the various algorithms, it is convenient to view each request for a document as being satisfied in the following way. The algorithm brings the newly requested document into the cache and then checks that the document is in the cache. If it is in the cache, the cache just updates the hit information, otherwise, evicts other documents for the new document until the capacity of the cache is no longer exceeded. Algorithms are then distinguished by how they choose which documents to evict.

2.1 Traditional Replacement algorithm

There are Least Recently Used (LRU), Least Frequently Use (LFU), First In First Out (FIFO), and etc. in this category. These algorithms are well-known cache replacement strategies for paging scenarios. These algorithms are simple and well-managed, so in the beginning to implement proxy cache, most proxy caches apply these to replacement algorithms. We briefly survey several algorithms below.

- Least Recently Used (LRU)

  LRU evicts the object which was requested the least recently. LRU leverages temporal locality of reference. It means recently accessed objects are likely to

be accessed again. It can be implemented easily with $O(1)$ overhead per one request.

- Least Frequently Used (LFU)

  LFU evicts the object, which is accessed least frequently. LFU leverages the skewed popularity of objects in a reference stream. It means that objects frequently accessed in the past are likely to be accessed again in the future.

- First In First Out (FIFO)

  FIFO evicts the object, which is the last content of the queue. The newly requested document is brought into the head of the queue. FIFO leverages the fairness of the request stream. But FIFO is rarely used in the pure form.

These traditional algorithms are used widely, because of their simplicity and robust feature. But at first time, these algorithms are suggested for homogeneous paging caching. So these are not satisfied the web environment, whose traffics have the nonhomogeneous feature, the variable latency, the dynamic relative frequency of request, and etc. Namely, the difficulty with such algorithms in this category is that they fail to pay sufficient attention to the characteristic of Web.

2.2 Key-based Replacement algorithm

The algorithms in this category enhance the performance of proxy caching by applying the web characteristics to the traditional algorithms. The main idea in key-based policies is to sort objects based upon a primary key, break ties based on a secondary key, break remaining ties based on a tertiary key, and so on. There are size, latency, reference count, last access and etc which are used as the keys. Figure 2.1(a) show the conceptional process of the generation from traditional algorithms to key-based algorithms. And Some examples of key-based algorithms are illustrated in Table 2.1.

- Large File First (LFF) [3]

  LFF evicts the largest object. It can be implemented by maintaining a priority

Name	Primary Key	Secondary Key	Tertiary Key
LFF	Size	Time Since Last Access	
LOG2SIZE	$\lfloor log_2(size) \rfloor$	Time Since Last Access	
LLF	Latency		
HYPER-G	Frequency of Access	Time Since Last Access	Size

Table 2.1: Examples of key-based algorithms

queue of the objects in the memory biased on their size. It has $O(1)$ overhead per one request, but it has sorting overhead $O(logk)$ per one request.

- LRU-MIN [1]

  LRU-MIN is biased in favor of smaller objects. If there are any objects in the cache which have size at least S, LRU-MIN evicts the least recently used such object from the cache. If there are no objects with size at least S, then LRU-MIN starts evicting objects in LRU order of size at lease S/2. That is the object who has the largest log(size) and is the least recently used object among all objects with the same log(size) will be evicted first.

- LOG2SIZE [1]

  LOG2SIZE is biased in favor of smaller objects more than LRU-MIN. In LOG2SIZE, how to manage the objects is similar to LRU-MIN. But the difference between of them is how to evict the objects. LOG2SIZE evicts the largest and lease recently used object. Namely, LOG2SIZE is the mixture of LFF and LRU-MIN.

- Lowest Latency First [6]

  LLF wants to minimize the average latency by evicting the object with the lowest download latency first. [6] show LAT algorithm what estimates download time and evict objects with shortest download time. This also shows HYB algorithm, which combines estimates of the connection time of a server and

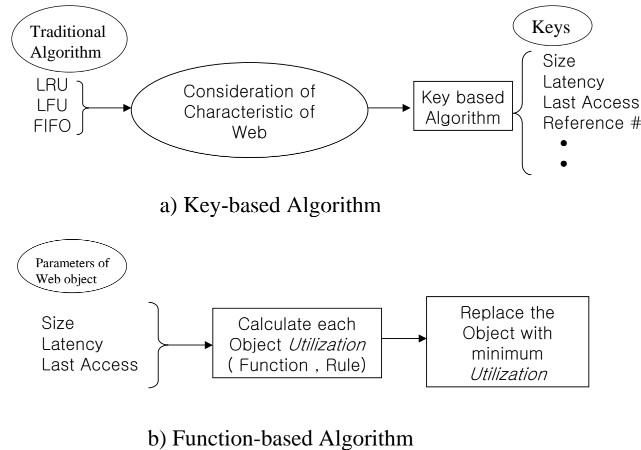


Figure 2.1: The conceptional view of the replacement algorithms

the network bandwidth that would be achieved to the server with object size and number of document accesses since the document entered the cache.

- Hyper-G [3]

  Hyper-G algorithm uses frequency of access as the primary key, breaks ties using the recency of last use, and then finally uses size as the tertiary key.

The idea in using the key-based algorithms is to prioritize some replacement factors over others. However, such prioritization may not always be ideal. So these algorithms are only good for a specific metric. For example, size parameter is good for Hit Rate, but it is not good for other metrics. So when we use LFF for the proxy cache, we get very high Hit Rate, but get poor Byte Hit Rate.

2.3 Function-based Replacement algorithm

The idea in function-based replacement algorithms is to employ a potentially general function of the different factors such as time since last access, entry time of the

object , transfer time cost, object expiration time, object size and so on. Namely, these algorithms use the object utilization that is calculated by many parameters which is related to web characteristics. This utilization infers the object popularity indirectly. Because these algorithms consider many parameters, these offer the reasonable performance in many metrics to proxy caches. Figure 2.1(b) show the general process for the function-based algorithms.

- Greedy Dual-Size [5]

  The basic Greedy Dual algorithm sorts objects biased on their measured retrieed cost H. The object with the minimum value for H is the candidate for replacement, and when a replacement occurs all the objects get aged by the current value, H, of the purged object. In this algorithm, parameter H is slightly different being set to the ratio of the object (cost/ size), to account for the variable size of Web objects. Updating the cache state at each reference has cost $O(logk)$, requiring a list search.

- Hybrid [6]

  Hybrid evicts the object with the smallest function value. Hybrid aimed at reducing the total latency. A function is calculated for each object that is designed to capture the utility of retaining a given document in the cache.

- Lowest Relative Value [11]

  LRV includes the cost and size of a object in the calculation of a value that estimates the utility of keeping a object in the cache. LRV evicts the objects with the lowest value. The calculation of the value is based on extensive empirical analysis of trace data. If the cost in LRV is proportional to size, the authors of this algorithm suggests an efficient method that can find the replacement in $O(1)$ time, though the cost can be large. If the cost is arbitrary, then $O(k)$ time is needed to find a replacement, where k is the object number.

- Least Normalized Cost Replacement [13]

LNC-R employs a rational function of the access frequency, the transfer time cost and the size.

- Bolot/Hoschka [12]

  This algorithm employs a weighted rational function of the transfer time cost, the size, and the time since last access.

- Size-Adjust LRU [7]

  Size-Adjust LRU orders the object by ratio of cost to size and choose objects with the best cost-to-size ratio. If we use PPS algorithm, which is the practical version of Size-Adjust LRU, we need the $O(log2(cachesize))$ time for the replacement.

The object utilization which used by these algorithms infer the popularity of the object. But in the most algorithms, this utilization is calculated by estimate value like a size, a latency, a last access rather than the direct popularity value of object like the relative access frequency or reference counts through a proxy. Especially in the case of Size-adjust LRU, this algorithm use the size and the last access number to get the object utilization. Each of these parameters means the negative correlation between the popularity and the object size, and temporal locality in a request stream. Both of these are assumed to be indicative of the future popularity of the object, and hence reflective of the merit of keeping such and object in the cache. But recent studies [9] suggest that such relationships are weakening and hence may not be effective in capturing the popularity of Web objects.

2.4 Parameters

In the key-based algorithm or function-based algorithm, there are many keys that affect the performance of cache replacement policies. Among others, these factors include object size, miss penalty, temporary locality, and popularity.

- Object Size

  Unlike traditional caching in memory systems, Web caches are required to manage objects of variable sizes. we found the characteristic of size, that is the preference for small objects in Web access. [8] But this preference seems to be weakening. [9]

- Miss Penalty

  This means the retrieval cost of missed objects from server to proxy, like form file to cache in memory systems. But in the web environment, the miss penalty varies significantly. Thus, giving a preference to objects with a high retrieval latency can achieve high saving. [6]

- temporary locality

  This is the base concept of the cache, i.e., recently accessed objects are more likely to be accessed again in the near future. This has led to the use of LRU cache replacement policy. We found the web traffic patterns to exhibit temporal locality. [7] [14] But, recent studies have suggested a weakening in temporal locality. [9]

- popularity

  The popularity infers the popular object directly unlike other parameters. But the popularity of Web objects was found to be highly variable(i.e. bursty) over short time scales, so many replacement algorithms don't use this parameter due to the worry about the cache pollution phenomenon. However, we found that the popularity is much smoother over long time scales. [16] This suggests the significance of long-term measurement of access frequency in cache replacement algorithms.

3. Lowest Popularity Per Byte Replacement Algorithm

According to these related works, we found that we must study the function-based algorithm for proxy cache replacement algorithm. Because Key-based algorithms give more priority one parameter and can not have the ideal performance of proxy caches in many metrics. And to suggest a function-based algorithm which has the good performance, we consider the recent research result that the correlation between temporal locality or size and popular objects is weakening. In this section, we present Least Popularity Per Byte Replacement algorithm(LPPB-R), which is the extension of SLRU that uses temporal locality and size by the main parameters to calculate the utilization. This LPPB-R algorithm use object popularity parameter, which is not considered by SLRU algorithm, from proxy cache directly.

3.1 Overview of LPPB-R

When an object is to be inserted into the cache, more than one object may need to be removed in order to create sufficient space. In this case, LPPB-R follows the behavior of function-based algorithm. Like all other function-based algorithm, LPPB-R calculates the utilization of each object, then evict the object, which has smallest utilization. This utilization $U(j)$ of an object j is calculated by the following model:

$$U(j) = P(j)/S(j)$$

where $P(j)$ is the popularity of the object j and $S(j)$ is the size of the object j. We describe the detail of the popularity in section 3.2. In this model, the utilization of an object, $U(j)$ represents the popularity per byte. So when the proxy cache

wants to the replacement for a new object, the cache evicts the object, which has the smallest popularity per byte value.

According to this utilization and the cache policy, we can have the following model for the LPPB-R replacement algorithm in general.

$$Minimize \sum_{j \in C} D(j) \times U(j)$$

$$such\ that \sum_{j \in C} D(j) \times S(j) \ \geq \ R$$

$$and\ D(j) \ \in \ \{0, 1\}$$

where C means the group of object in the cache when the replacement event occur, R is the required size for the new object which is to be inserted. Let $R \geq 0$ denotes the amount of additional space in the cache, which must be created in order to accommodate the new object. And $D(j)$ is the decision variable for object j defined to be 1 if we wish to purge it, and to be 0 if we want to retain it. We can analyze above formulas by using these notations. If the replacement event occur (i.e $R \geq 0$), the proxy cache selects the object which is evicted. In this case, the proxy cache makes the sum of objects which are removed to be larger than the R and makes the sum of popularity per byte to be as small as possible. Briefly, the purpose of the LPPB-R is to make the popularity per byte of the outgoing objects to be minimum.

3.2 Getting the Popularity Value

In this LPPB-R algorithm, the popularity value is the most important parameter to get the utilization value. And this popularity value considers the long term measurement of request frequency, which is neglected in the other algorithms, especially in the SLRU. So how to set the popularity value determines the performance of this LPPB-R algorithm. We set the popularity value by using the information from the cached object directly, not by using the value from the request stream indirectly. Namely, the popularity value of the object j changes not by the arbitrary request,

but by only the request for the object j. So, This property makes the popularity value to deal with the long term measurement of request frequency.

In the LPPB-R algorithm, the popularity value can have one of the two-type values. One is the simple value, and the other is the enhanced value. We describe the detail of these two-type values. In the following models, $P(j)$ is the popularity of an object j.

$$(1) \quad P(j) = R(j)/T$$

where $R(j)$ is the reference count number of object j through the proxy cache (i.e. $R(j)$ shows how popular the object j) and T is the total request number through the proxy cache at the time when the new object is to be inserted into the cache. In this model, we don't use the reference count number directly, but we use the normalized reference count number, to get the reasonable range of the utilization value. The reason for this is found in the process which compare the utilization values. When we compare the utilization values in the implementation, we evict the object which has not the minimum utilization value, but the maximum unutilization value. If we use $R(j)$ to get the $P(j)$ directly, the utilization value varies heavily by changing the popularity value slightly. This behavior is similar to the behavior of the Key-based algorithm, which weight the one parameter, in this case, popularity value. Then the popularity value has more priority than the others, and this property prevents the LPPB-R algorithm getting the reasonable performance in the all metrics. So to prevent the popularity value being weighted, we use T, which is the total request number.

$$(2) \quad P(j) = 1/(\beta)^{R(j)} \ , \ (\ 0 \ < \ \beta \ < \ 1 \ )$$

where $R(j)$ is the reference count number of object j, and β is the constant for the impact factor. β can have the value from 0 to 1. This model is in a exponent -like fashion as a function of the $R(j)$. And the β value adjusts the raising point of the

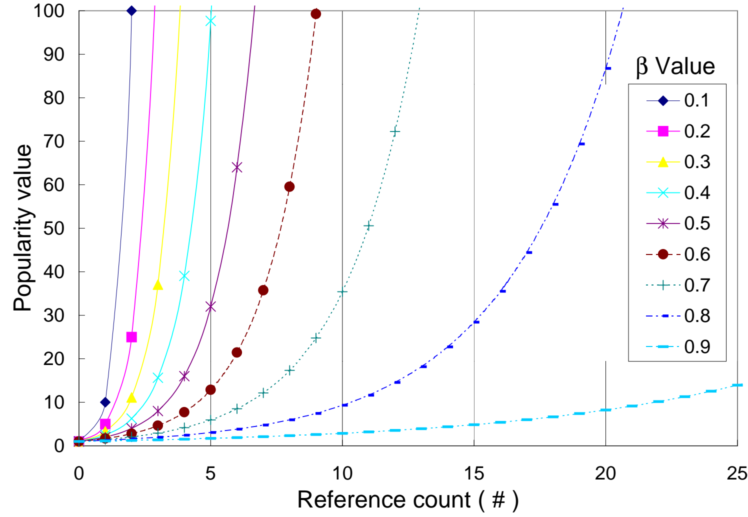


Figure 3.1: Popularity values for variable β values

graph. If the β value is near zero, the raising point is near zero (i.e. y axis). And if the β value is near one, the raising point is near infinity. By using this model for the popularity parameter, we weight the popularity parameter partially. This means that we have the popularity threshold. If the popularity of the object is larger than the threshold, the popularity value is weighted, then this object is evicted very seldom.

Consequently, if β value is near zero, the increasing point where the priority of the popularity value increases is near zero and smaller than 5, otherwise if β value is near one, the increasing point where the priority of the popularity value increases is in the region which is larger than 100. For example, if $\beta = 0.1$ and the difference between $R(j)$ and $R(i)$ is 1, the difference of the utilizations become 90. But if $\beta = 0.9$ and the difference between $R(j)$ and $R(i)$ is 1,the difference of the utilizations become 0.1. If $\beta = 0.9$ and we want to make the difference of the utilizations 90, the difference between $R(j)$ and $R(i)$ must be more than 65. Figure 3.1 show this property of β value. So when we use this type, because β value

affects the popularity value heavily, how set the β value determines the performance of this LPPB-R algorithm. We show the detail experiment about the β value in the section 4.3. In this paper, we set the β value on the range from 0.3 to 0.5.

In both the models, we use $R(j)$, the reference count number of object j. This value means the long-term meauserment of request frequency, not shot-term measurement. According to the recent studies [9], we know the relationship of the temporal locality and the popular object is weakening in these days. One reliable reason for getting this result is the efficient client caching. By using the client caching, when the client has a request for one object, first the client checks the client cache whether the object is in the client cache, then if the object is in the cache, uses that object, but if not, the client sends the request to the proxy cache. Nowadays, the client like a web browser has the client cache which has sufficient size to behave efficiently. So, the efficient client caching deals with short-term measurement of request frequency like temporal locality property. Then, to get better performance of the proxy cache, the proxy cache should deal with the long-term measurement of request frequency. So, we use the reference count number as the long-term measurement of request frequency.

3.3 Managing the objects

When we use the function-based algorithm, we must consider how to minimize the overhead which is made from calculating the utilization value. Generally, traditional algorithms like LRU or LFU need $O(1)$ time to process the one request. And key-based algorithms like LFF or LRU-MIN need $O(1)$ time for $O(logk)$ time to process the one request, and these algorithms need the data update time. So, in the traditional or key-based algorithms, the overhead which is need to process the requests is small. But function- based algorithms need the time to process the request and to calculate the utilization value of the each object. So to design the good function-based algorithm we must reduce the overhead which is needed to process

the requests and to calculate the utilization value by managing the objects and the meta information effectively.

Strictly following the LPPB-R algorithm makes the overhead serious. When the new object is inserted into the cache and the cache needs the replacement event, the LPPB-R calculates the utilization values of the all objects and evicts the object which has the smallest utilization value until the cache has sufficient space. This operation needs $O(k)$ time, where k is the object number in the cache. So, the overhead is proportional to the cache size and this method is not practical.

To turn this ideal theory to practical algorithm, we use the multi queue to manage the objects, not use only one queue. Each queues determine the objects which are managed by the size of the objects. we use $\lceil log_2(size) \rceil$ to determine the range of the size. For example, ith queue manages the objects whose size is from 2^{i-1} to $2^i - 1$. Thus, there will be $N = \lceil log(M+1) \rceil$ different queues of objects, where M is the cache size. If we divide the queues by $\lceil log_2(size) \rceil$, when the cache manages the objects, the cache gets more advantages in determining object size, because the bitwize operation which uses the size of object is possible. For example, if the size of the object j is 10byte we present this size 000a by using 32bit. Then, we find the 1 in the 4th bit and we insert this object to the 4th queue. For the cache, the objects in each queue i are maintained as a separate LFU list. And the Meta information like reference count, size, latency, and etc are attached to each object in every queue and are used to calculate the utilization value of each object. Figure 3.2 shows the general structure of the multi queues. Using these queues, when cache finds the objects which are evicted, cache does not search the whole queue, but compares the utilization values for the least frequently used objects of each list. Finally, the time which is needed to process one request decreases from $O(k)$ to $O(\lceil log(M+1) \rceil)$, where k is object number and M is the total size of the cache and LPPB-R algorithm becomes practical.

If we use this practical algorithm, we have some disadvantage to find unpopular

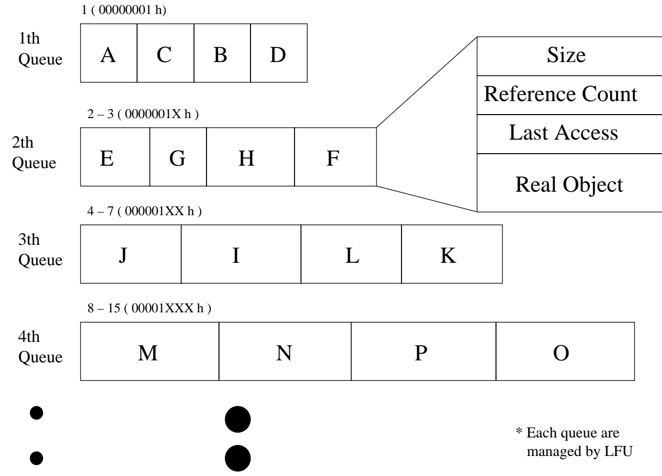


Figure 3.2: Multiqueue structure used by LPPB-R

objects which are evicted. But this disadvantage has limit. If $U(i)$ is the minimum of the utilization value by the ideal LPPB-R theory and $U(p)$ is the minimum of the utilization by the practical LPPB-R algorithm, $U(p)$ has the same value or the less value than $2 \cdot U(i)$. Namely, this means the following model:

$$U(p) < 2 \cdot U(i)$$

Figure 3.3 show a summary of this model. The detail description for this model is following: First, we assume that the any one queue has the object i which has the minimum of the utilization value by the ideal LPPB-R theory and the LFU list header object for this queue is object j. In this case, $S(i) < 2 \cdot S(j)$ and $2 \cdot U(i) > U(j)$ is true. (if this condition is not true, the i and j can not exist.) Then, if the object p has the minimum of the utilization value by comparing the LFU list header objects for the whole queue, $U(j) > U(p)$ is true. According to these facts, $2 \cdot U(i) > U(j) > U(p)$. This happens in the worst case. But in reality, the value $U(p)$ is so close to $U(i)$ that the performance difference between the ideal

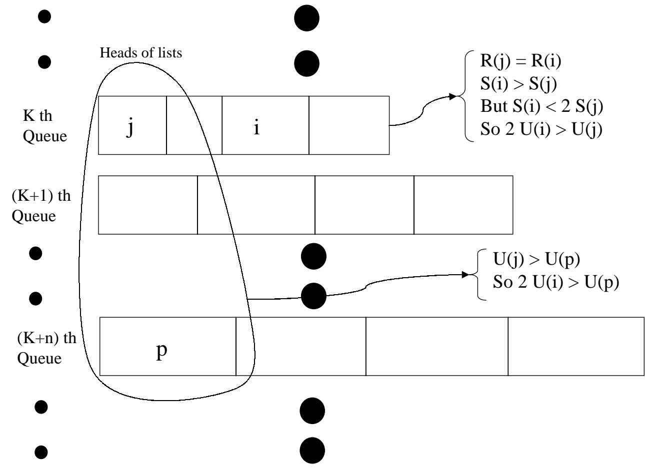


Figure 3.3: This figure show that objects follow the theory, $2 \cdot U(i) > U(p)$

theory and the practical LPPB-R algorithm. We shall illustrate a comparison of the hit ratios of these two cases (ideal theory and practical policy) in the section 4.3.

Notice that the LOG2-SIZE discussed in [3] and SLRU discussed in [7] bears at least some resemblance to the independently derived LPPB-R scheme. The LOG2-SIZE scheme always chooses the least recently used items in the nonempty stacks corresponding to the largest size range. And in the SLRU scheme, the objects in the queue are maintained as a LRU list. In contrast, the LPPB-R scheme applies LFU algorithm to each queue and looks at the least frequently used objects of each stack, and among these picks the object which has the least utilization.

3.4 Avoiding the cache pollution phenomenon

In LPPB-R algorithm, the most important parameter among the many factors, which are used to calculate the utilization value, is popularity and this value is determined by the object reference count, which can infer the long-term frequency of an object. This reference count value has an advantage, that the cache get the

popularity information of the objects directly by the requests through the cache. But this parameter, object reference, has a bad characteristic which decreases the performance. This is that the request is bursty. This property decreases the performance of the cache which uses the LFU algorithm. So, in the short-term period, the LFU has less performance than the LRU. For example, in the short-term period, the reference count value of the popular objects is much higher than the others which are not popular by the bursty property. In this case, these objects have the lower probability to be evicted than the others. So, though these objects are not popular any more and these must be evicted, cache can't evict these objects because their reference count values have been very high. This phenomenon is called *cache pollution phenomenon*. The cache pollution phenomenon occurs not only in the short -term period, but also in the long-term period. Consequently, to make our algorithm, LPPB-R more efficient than other algorithms, we must avoid the cache pollution phenomenon.

We show that the objects in the cache are managed by the LFU list. In addition, we use the LRU list to avoid the cache pollution phenomenon. Namely, the proxy cache uses two lists, LRU list and LFU list, to manage one queue. The LFU list is used to calculate the utilization value and to select the objects which is removed, and the LRU list is used to avoid the cache pollution phenomenon.

In this paragraph, we describe the detail about the avoiding the cache pollution phenomenon. First of all, the proxy cache checks the least recently used objects which are in the whole LRU list periodically. If the difference between the last access time of the object and the current time is greater than the threshold value, the LPPB-R algorithm sets the reference count of the object on new value which refers that this object is not popular. In our LPPB-R algorithm, when the proxy cache finds this object which fits above situation for the first time, we change that reference count value to 2. When that object meets the similar situation again for the second time, we change that reference count value to 1 which is the minimum

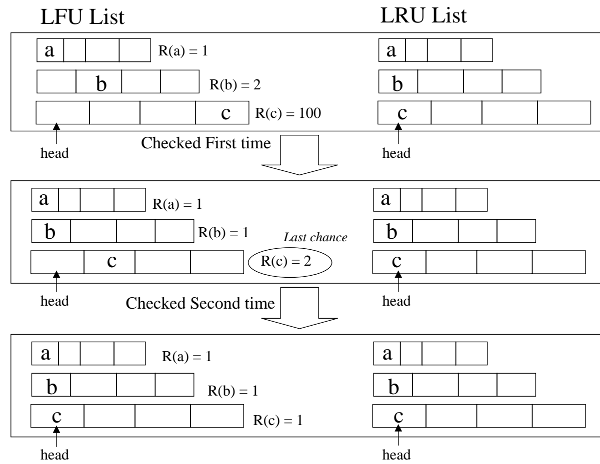


Figure 3.4: Cache pollution avoidance mechanism

value of an reference count value. We use these two steps for the operation, because we want to give the last chance to the objects which were popular. Figure 3.4 show the whole mechanism for avoiding the cache pollution phenomenon. We put the period by which the cache checks the LRU list to 10000 requests and put the threshold value to 1000000 requests.

4. Performance evaluation

In this section, we present the result of extensive trace driven simulations that we have conducted to evaluate the performance of LPPB-R. We design our proxy cache simulator for doing the performance evaluation. This simulator illustrates the behavior of a proxy cache. That is, when the simulator gets the request from any client, the simulator parses the request and get the object name. If the object is in the proxy cache, the simulator deals with the request and calls this case 'hit', otherwise the simulator sends the request to the original server which has that object and calls this case 'miss'. In our simulation, we use the traces from NLANR [15]. And to compare the performances, we simulate the LPPB-R algorithm with the representative algorithms of traditional, key-based, and function-based algorithms. We select LRU, LFU, LOG2SIZE and Size Adjust LRU for representative algorithms. We use hit rate, byte hit rate, and reduced latency as the performance metrics.

4.1 Traces used

In our trace-driven simulations we use traces from NLANR [15]. We have run our simulations with traces from the *pb* proxy server and the *bo2* proxy server of NLANR since September, 2000. The *pb* proxy server manages the objects which are from .ch, .fr, .se, .uk and .com domain and has a 24GB harddisk and a 512MB main memory. The *bo2* proxy server manages the objects which are from .edu, .gov, .org, .mil and .us domain and has a 30GB harddisk and a 512MB main memory. Both proxy cache servers have 100MB/s traffic bandwidth.

We show some of the characteristics of these traces in Table 4.1. Note, these characteristics are the results when the cache size is infinite. Namely, since our simulations assume limited cache storage, the ratios like hit rate , byte hit rate and

Traces	PB server	BO2 server
Measuring days	2000.10.03 - 04	2000.09.14 - 20
All requests size	18595227993 B	25607415375 B
Unique objects size	10016257877 B	15363371177 B
All requests number	2366457	2215404
Unique objects number	1183791	969892
Hit Rate	49.976 %	56.221 %
Byte Hit Rate	46.135 %	40.004 %
Reduced Latency	5887966.423 sec	7106974.665 sec

Table 4.1: Traces used in our simulation (Cache size = infinity)

reduced latency cannot be higher than the infinite cache ratios. And we also analys the traffics by the requested object size, then we find that almost the requests are in the range from 256 byte to 2048 byte. And we also find that the hit rate in the pb trace is lower than in bo2 trace, but the byte hit rate in the pb trace is higher than in bo2 trace. This means that pb trace has more large objects which are requested frequently than bo2 trace.

We have some assumption to simulate the behavior of a proxy cache effectively. The whole structure of the proxy cache, clients and servers are same as figure 1.1. The size of a proxy cache is in the range from 0.01MByte to 500MByte. We assume that The used traces are from the proxy cache which is located in the position between user's clients and web servers and that there are not any ICP packet and other cache interactions. Namely, when an cache miss occur, the proxy cache does not send the request to the nearest cache by ICP, but send the request directly to the original server which creates that object. We assume not only that there are not any other proxy caches between the clients and target proxy cache, but also that there are not any problems, like congestions and overflow buffers in the network which links the proxy cache to the servers. But the transfer rate of each server is different.

4.2 Performance metrics and algorithms

We consider three aspects of web caching benefits: hit rate, byte hit rate and reduced latency. By hit rate, we mean the number of requests that hit in the proxy cache as a percentage of total requests. Higher the hit rate is, more requests the proxy cache can treat and less requests the original server must deal with. So the overhead of the server which is needed to process the requests decreases and the number of clients which is served by this server increases. By byte hit rate, we mean the number of bytes that hit in the proxy cache as the percentage of the total number of bytes requested. Hit rate is the parameter to measure the performance of an proxy cache above the network layer of the OSI 7 layer, but byte hit rate is the parameter to measure the performance of an proxy cache below the link layer. Higher byte hit rate is , more network traffic decreases in the server side. Namely, if byte hit rage is high, the traffic overhead of the whole network, except the autonomous system whose gateway is the proxy cache, decreases. By reduced latency, we mean the response time, which is reduced by the cache. More latency is reduced, less time is need for client to get the response for an request. In the recent study, the user don't want to visit the web site which is loaded for 10 seconds or more time. So more latency is reduced, more user visit the server. And besides, long latency means that the packet travel in the network for long time. This also means that the network has overhead. So more latency is reduced, less overheads the network has. In section 4.4, we measure this metrics of different algorithms.

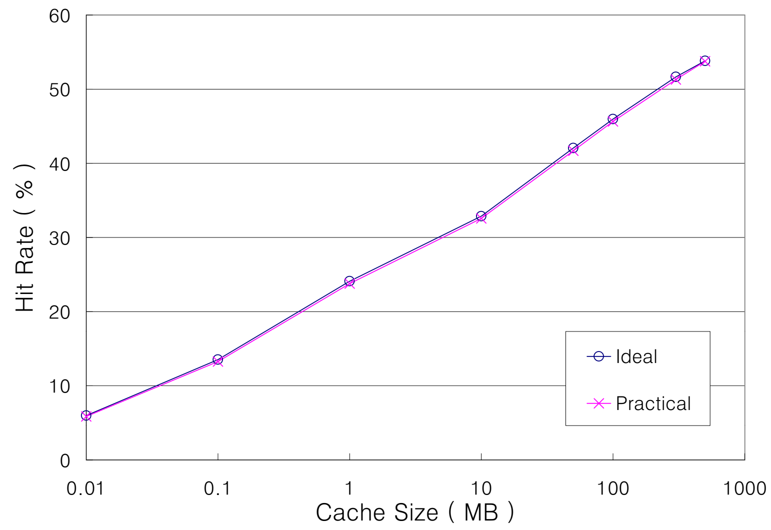
We compare the performance of LPPB-R with LRU, LFU, LOG2SIZE [1] and Size Adjust LRU [7]. LRU and LFU are the representative algorithms of the traditional replacement algorithm. Both two algorithms use only one simple queue to manage the objects. Because these algorithms are implemented very simply, the simulation is very easy and take less time than others, but LFU can not have a good performance by the cache pollution phenomenon. LOG2SIZE algorithm is applied

as the representative of the key-based replacement algorithm. This algorithm acts like LRU-MIN algorithm and uses the object size as the primary key and the last access time as the secondary key. And this algorithm manages the object by the multi queue according to the object size. When the replacement event occurs, this evicts the least recently used object among the objects in the largest size object queue. The performance of LOG2SIZE algorithm is good at the hit rate, but is not good at other metrics. To compare between LPPB-R and other function-based replacement algorithms, we implement the Size Adjust LRU algorithm. Our algorithm is the extension of this algorithm. This algorithm uses the object size and the last access time as the parameters which is used to calculate the utilization. And this uses a multi queue according to the object size. The performance of Size Adjust LRU is good at every metric, because this acts like function -based algorithm. Our algorithm, LPPB-R, is also a function-based algorithm like Size adjust LRU. We use the object size and the object popularity as the parameters and use the multi queue to manage the objects. More detail process for LPPB-R is described in section 3.

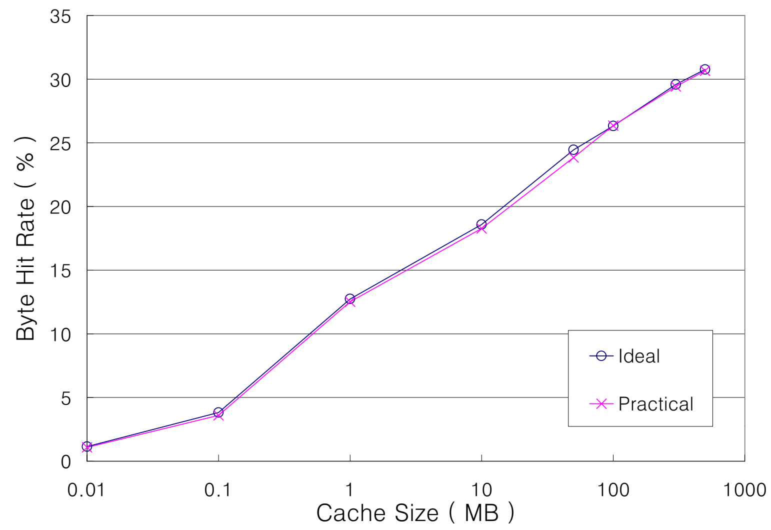
4.3 Implementation issues on LPPB-R

There are two considerations to implement the LPPB-R algorithm. One is how the performance of the practical policy differ from the performance of the ideal theory. The other is how the β value set to get the type-2 popularity value.

First, we present the result of the comparison between the practical policy and the ideal theory in figure 4.1. Finally, we find that both of them get the similar performance in all metrics. In other words, like the description in section 3.3, most evicted objects which are selected by the practical policy are same objects as the ideal theory selects by comparing the utilization values of all objects or have similar utilization values which the ideal theory finds as the minimum values. But when we use the ideal theory, if the cache size increases, the calculation overhead increases exponentially. According to these results, we use the practical policy to implement



(a) Hit rate comparison between ideal and practical algorithm



(b) Byte hit rate comparison between ideal and practical algorithm

Figure 4.1: Performance comparison between ideal and practical algorithm

LPPB-R algorithm.

Next, we measure the performance of the all metrics with variable β values to select the β value for good LPPB-R 2. We use 0.1, 0.3, 0.5, 0.7 and 0.9 as the β value and apply LPPB-R 2 with these values to both of pb and bo2 traces. This result are presented in section 4.4. In figure 4.2, we show the change of the hit rate by the β value. In bo2 traces, the hit rate is maximum when the β value is 0.7 and is minimum when the β value is 0.9. But in pb traces, the hit rate is maximum when the β value is 0.5 and is minimum when the β value is 0.9. So if we set the β value to the range from 0.5 to 0.7, LPPB-R 2 achieves the best hit rate. On the other hand, in the byte hit rate and the reduced latency, closer to zero the β value is, better the performance of the cache is. In both of the pb and bo2 traces, two metrics follow this feature. Figure 4.4 and figure 4.6 show this result.

According to these results, we can know the fact, that there are some trade off between the metrics for using the β value. Namely, if we set the β value to 0.7 to achieve the best hit rate, the performance in the byte hit rate and the reduced latency decreases. And if we set the β value to 0.1 to achieve the best byte hit rate and reduced latency, the performance in the hit rate decreases. So, when we set the β value, we must consider the balance of the performances in all metrics. We set the β value to 0.5 in simulation for bo2 traces, and to 0.3 for pb traces.

Last, because of the difference of mechanisms for calculating the popularity value, we divide the LPPB-R algorithm into LPPB-R 1 and LPPB-R 2. LPPB-R 1 algorithm uses the first way to get the popularity value, which use the total reference number. And LPPB-R 2 algorithm uses the second way to calculate the popularity value, which uses the β value.

4.4 Performance Measurement

We measure the performance of each algorithms in the three metrics by using the assumptions and our simulator. We present the results of hit rate, byte hit rate, and

reduced latency sequentially. In the result of each metric, first of all, we compare the performances of the result of various β value. We set the β value to 0.5 for bo2 traces and to 0.3 for pb traces. Figure 4.2, Figure 4.4 and Figure 4.6 the result of the LPPB-R 2 performance with various β value.

4.4.1 Hit rate

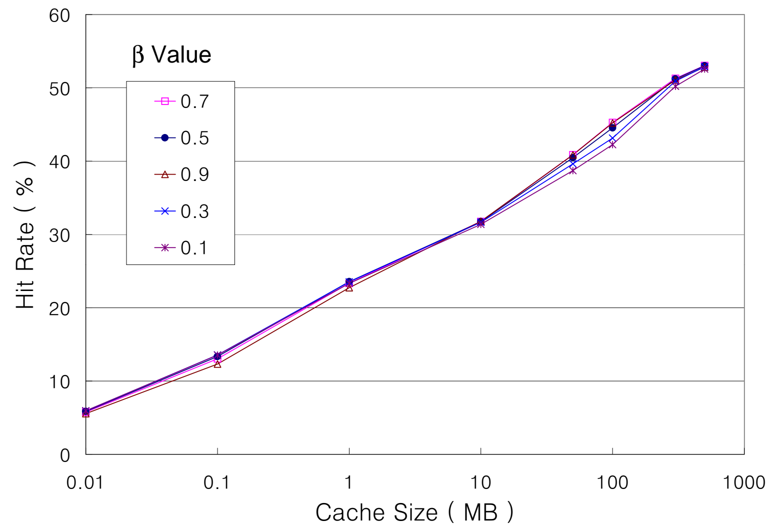
The results show that clearly, LPPB-R achieves the best hit rate among all algorithms across traces and cache sizes. Both of the LPPB-R 1 and the LPPB-R 2 (0.5)have the best hit rate. In the result of bo2 traces, if the cache size is 500MByte hit rate of the LPPB-R is about 53%. And LPPB-R has more hit rate by the value from 2% to 4% than Size Adjust LRU independently of the cache size. This result presents that the popularity value makes up for the weak point, that is the weakening temporal locality, in the Size Adjust LRU.

The traditional algorithms like LRU and LFU have the smallest hit rate among all algorithms. This result is quite natural since the traditional algorithms don't consider the characteristics of the Web traffics. In the result of bo2 traces, if the cache size is 500MByte, LFU has less hit rate by 12% than LPPB-R algorithm. On the other hands, LOG2SIZE, which is the one of the key-based algorithm has high hit rate. This result is similar to the result of Size Adjust LRU. This show the characteristic of the key based algorithm which use the object size as the primary key. In other words, if we use key based algorithms, hit rate is very high, but other metrics, like byte hit rate and reduced latency, are low. This fact is shown in the following section.

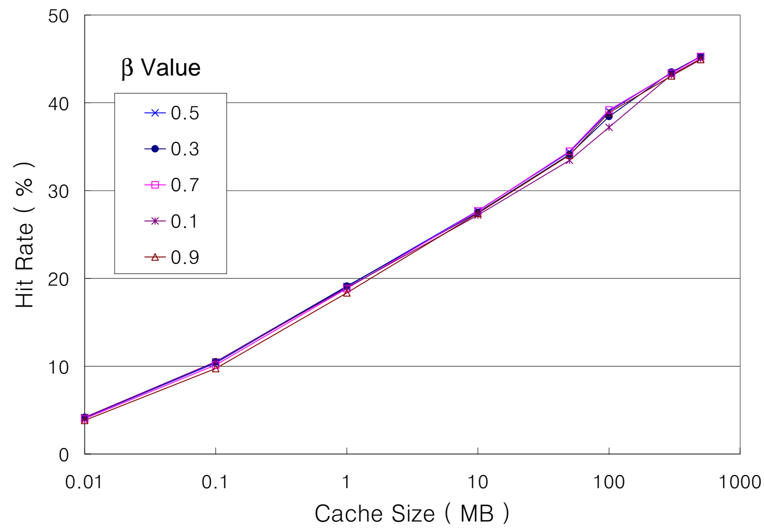
Figure 4.2 and figure 4.3 show the result of hit rate.

4.4.2 Byte hit rate

LPPB-R achieves better byte hit rate than Size Adjust LRU, not generally but partially. But, LPPB-R achieves better byte hit rate than LOG2SIZE which is key based algorithm or LFU in general. Namely, if the cache size is smaller than

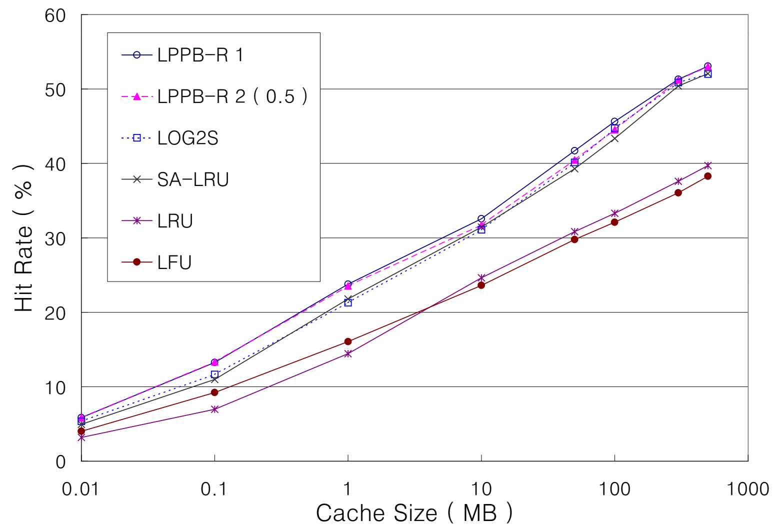


(a) BO2 Traces

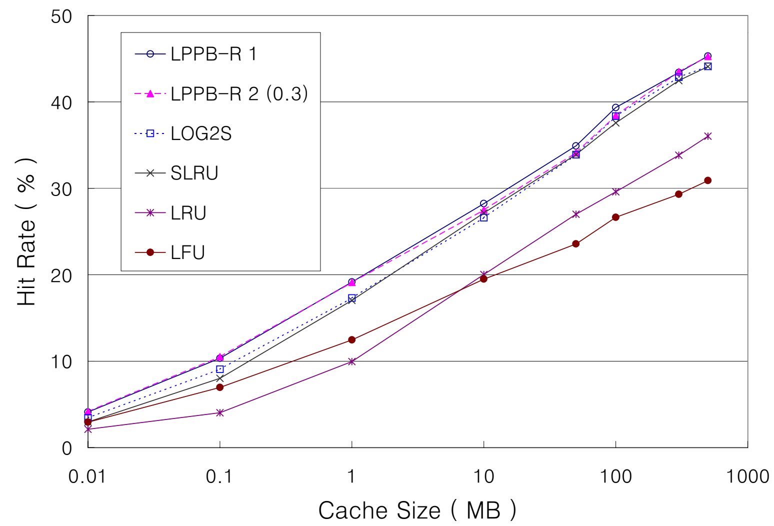


(b) PB Traces

Figure 4.2: Hit Rates of LPPB-R 2 for each β value



(a) BO2 Traces



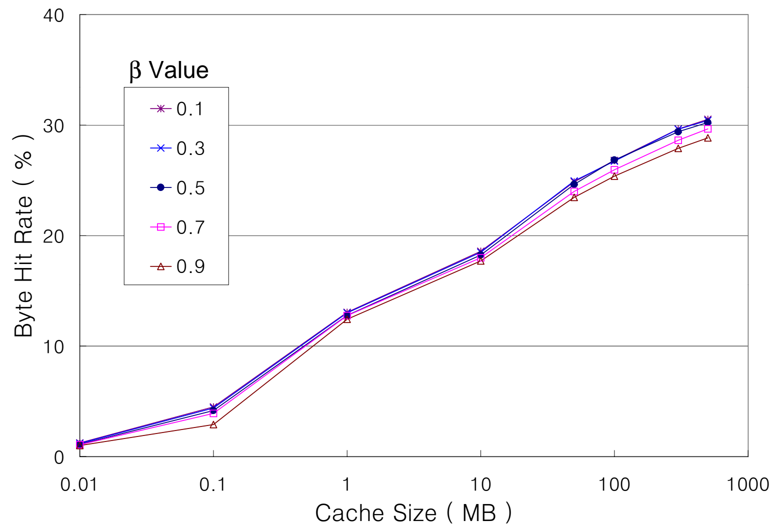
(b) PB Traces

Figure 4.3: Hit Rates of many algorithms for each traces

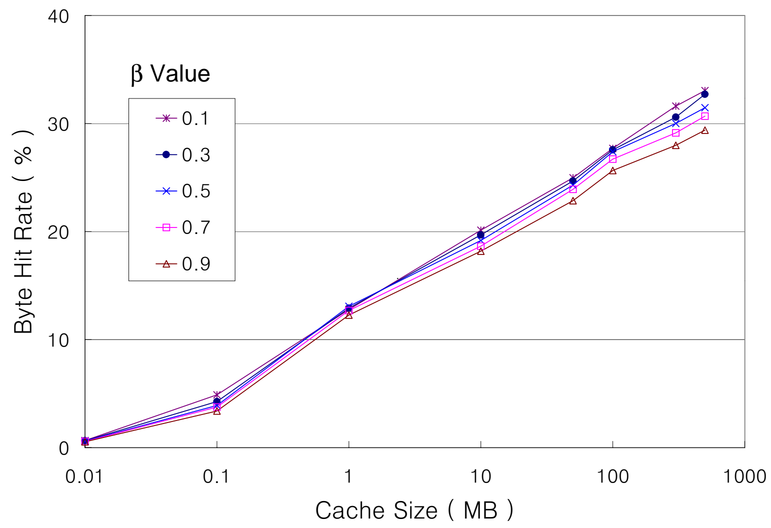
40MByte LPPB-R achieves the best byte hit rate among all algorithms, but if the cache size is larger than 40MByte LRU achieves the best byte hit rate among all algorithms. This result is found in both of bo2 traces and pb traces. The reasonable reason for this result is that the traces have the large objects which are requested frequently. Truly, in the bo2 traces the objects whose size is more than 1MByte are requested three thousands times, and in the pb traces the objects whose size is more than 1MByte are requested twelve hundreds times. Though the object number which is more than 1MByte is more in bo2 traces than in pb traces, the large objects are requested more frequently in pb traces than in bo2 traces. So, in byte hit rate if the cache size is more than 40MByte, LRU has the best byte hit rate. And according to the description of section 4.1, because pb traces follow that characteristic more strongly than bo2 traces, the difference of byte hit rate between the algorithms are larger in pb traces than in bo2 traces.

Differently form LRU, LFU can not achieve the good byte hit rate because of the cache pollution phenomenon. And LOG2Size which use the object size as the primary key also has low byte hit rate since this algorithm is proper to store the small objects. This is the characteristic of the key based algorithm. Size Adjust LRU follows the LRU characteristic because this algorithm use the temporal locality which is the feature of LRU. The difference of the byte hit rate between Size Adjust LRU and LRU is 1%.

LPPB-R uses the reference count rather than the temporal locality. In other words, this uses the feature of LFU rather than LRU. Though LPPB-R has that feature, LPPB-R doesn't follow the LFU characteristic but achieves the reasonable byte hit rate. Especially, if the β value is near zero, LPPB-R 2 achieves high byte hit rate. Figure 4.4 show that the byte hit rate of LPPB-R 2 with 0.1 is similar to that of Size adjust LRU. LPPB-R achieve the better byte hit rate than other algorithms, in the small-size cache, like the main memory cache for hottest objects. So if we want to achieve the high byte hit rate rather than the high hit rate, we set

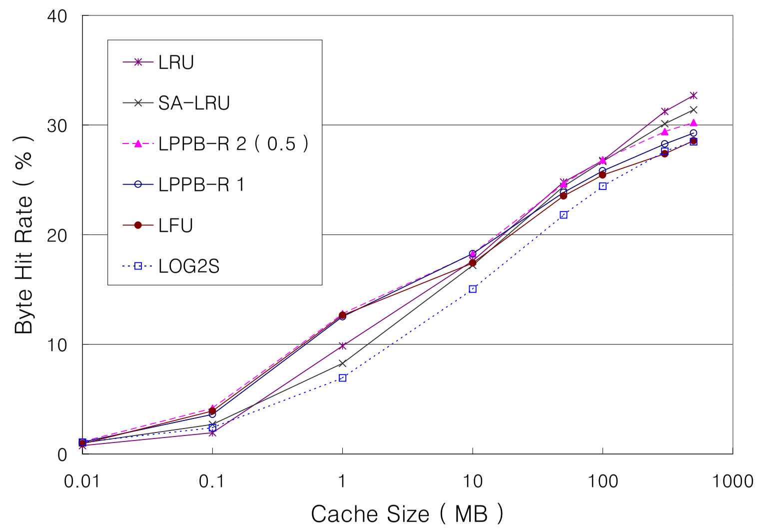


(a) BO2 Traces

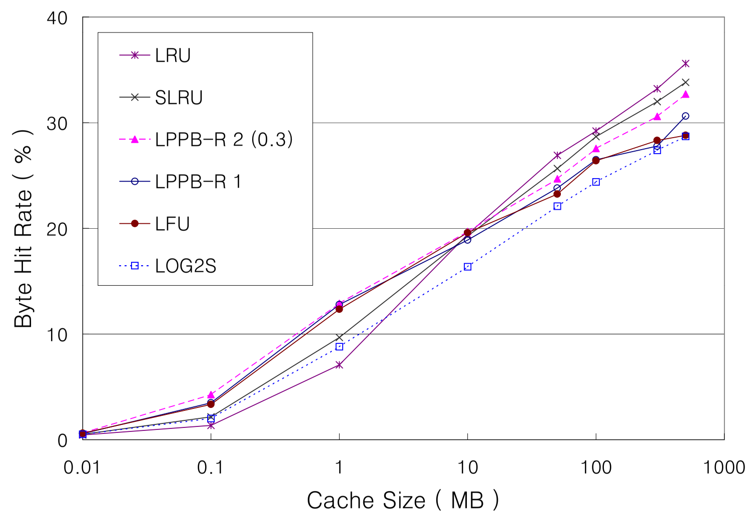


(b) PB Traces

Figure 4.4: Byte Hit Rate of LPPB-R 2 for each β value



(a) BO2 Traces



(b) PB Traces

Figure 4.5: Byte Hit Rate of many algorithms for each traces

the β value to the near value from zero and get the high byte hit rate.

Figure 4.4 and figure 4.5 show the result of byte hit rate.

4.4.3 Reduced Latency

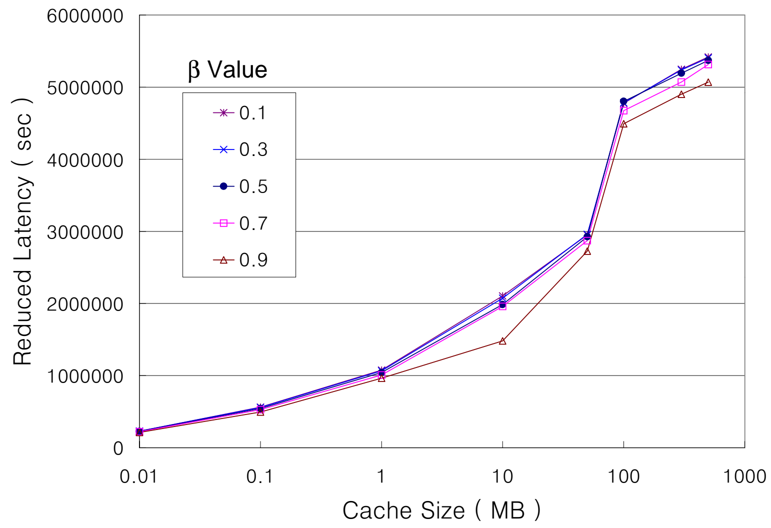
Another major concern for proxies is to reduce the latency of HTTP requests through caching, as numerous studies have shown that the waiting time has become the primary concern of Web users. So we compare the algorithms by the reduced latency. To measure the performance of this metric, we use the assuntions which are introduced in section 4.1. Namely, if the cache hit occurs, the request is processed in cache and not sent to the original server, so, we save the time which is needed by transferring data from server. We call this time the reduced latency.

In this metric, LPPB-R has much better performance than Size Adjust LRU. In the bo2 traces, if the cache size is 500MByte, the difference of the reduced latency between LPPB-R and Size- Adjust LRU is 2 million seconds. The effect of this value is equal to saving about one day for one thousand users. This is very important factor for users to select the web site. Size Adjust LRU follows the LRU property in byte hit rate, but in reduced latency this doesn't follow that property. The reason of this result is the high priority of the size parameter. So, in this case, Size Adjust LRU follows LOG2Size property. On the other hand, LPPB-R which has the LFU property gets better performance than LFU. Especially, if we set β value to the value which is near to zero, the reduced latency of LPPB-R 2 can be closer to that of LRU. In the real case like PB traces(figure 4.7), we find that LPPB-R 2 has the similar reduced latency value to that of LRU.

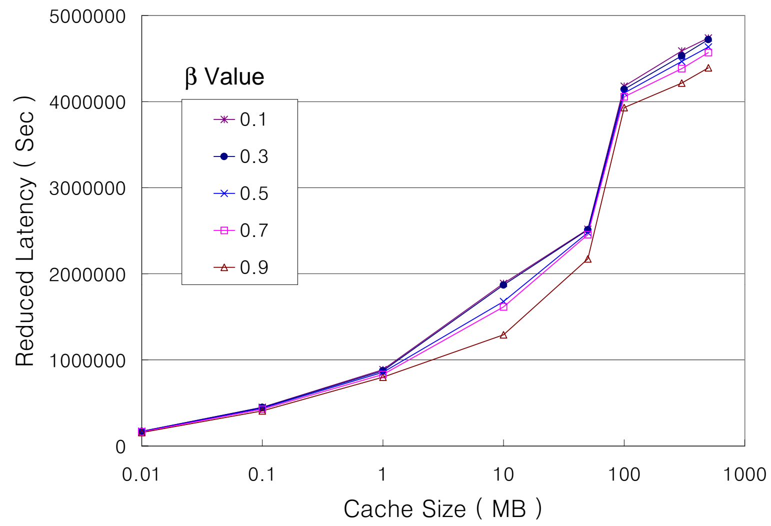
But, like the result of the byte hit rate, in the reduced latency LRU has the best reduced latency among all algorithms. The characteristic of the traces that is described in section 4.4.2 can be the one reason for this result. In addition, more objects which have the large latency the proxy cache has, more the reduced latency value increases. Because of these features of traces and caches, the traditional

algorithms like LRU and LFU, which are simple and pair for all objects, achieve better reduced latency than other algorithms except our LPPB-R algorithm. On the other hand, LOG2Size which is key based algorithm also has the least reduced latency.

Figure 4.6 and figure 4.7 show the result of reduced latency.

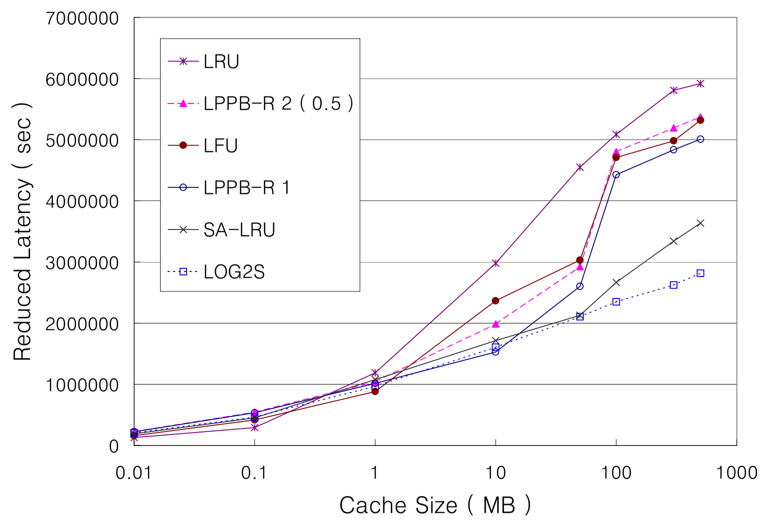


(a) BO2 Traces

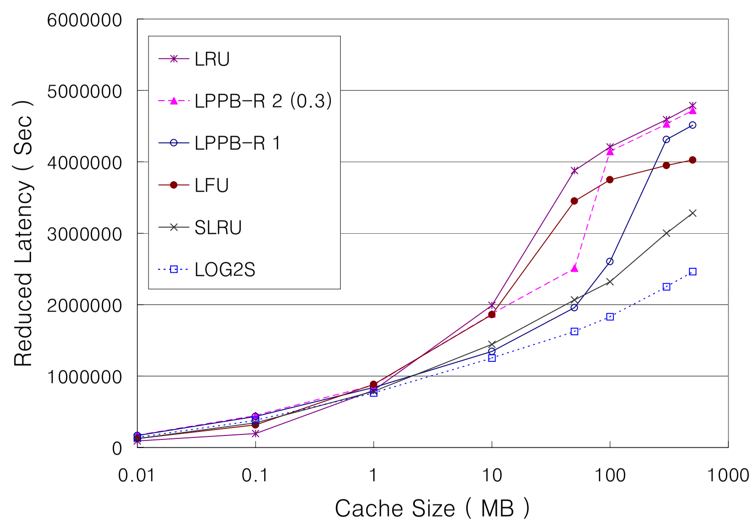


(b) PB Traces

Figure 4.6: Reduce Latency of LPPB-R 2 for each β value



(a) BO2 Traces



(b) PB Traces

Figure 4.7: Reduce Latency of many algorithms for each traces

5. Conclusion

This paper introduces the new function-based replacement algorithm, LPPB-R for the web proxy cache and shows that it outperforms existing replacement algorithms in many performance aspects, including the hit rate and the reduced latency.

LPPB-R algorithm uses the object popularity value and the object size to calculate the utilization value. The purpose of the LPPB-R is to make the popularity per byte of the outgoing objects to be minimum. This popularity value infers the long-term measurement of request frequency to complement the weak point in the temporal locality. LPPB-R uses a multi queue, and each queue is managed by LFU. It is simple to implement and accommodates a variety of performance goals by changing the β value. Through trace-driven simulations, we show that LPPB-R algorithm outperforms other replacement algorithms and is practical. Especially, in the comparison between LPPB-R and Size Adjust LRU, we show that the popularity value covers the weak point in the Size Adjust LRU which uses the temporal locality. LPPB-R achieves the best hit rate among all replacement algorithms including the Size Adjust LRU and the better reduced latency than Size Adjust LRU by one million second with the 500MByte cache size. And LPPB-R has reasonable byte hit rate.

In addition, we show that LPPB-R is easy to adjust the performance to needs of the proxy cache. This feature is possible by the characteristic of β value. In the simulations, if the β value is near to zero, hit rate is low but byte hit rate and reduced latency is high. Otherwise, if the β value is near to one, byte hit rate and reduced latency is low. The hit rate is highest between 0.5 to 0.7.

Consequently, we conclude that using the LPPB-R is better than other algorithms in the proxy cache which treats the traffics for the long-time period.

6. Future Work

Our LPPB-R has the feature of LFU. It means that the LPPB-R can meet the cache pollution phenomenon though the LPPB-R uses the measurement of long-term frequency. According to this we suggest the mechanism to avoid the cache pollution in section 3.4. But the effect of this mechanism is not presented in our simulation greatly, because it is applied to the objects individually and the period of the used traces are short. So to simulate with the long-time trace, for a month or a year is our ongoing work. And we are trying to apply this mechanism not to the individual objects, but to the bulk objects like domain-based objects.

In addition, to adjust the cache performance to the web traffic, we are designing the dynamic popularity calculation algorithm. This work needs more web traffic evaluations and simulations with the various β value.

요 약 문

프락시 캐쉬를 위한 바이트 단위의 최소 인기도 우선 대체 알고리즘

최근의 월드 와이드 웹의 폭발적인 사용양과 더불어서, 웹 오브젝트들을 캐싱하는 문제가 중요시 되고 있다. 웹 캐쉬는 서버로드를 줄이고, 네트워크 트래픽양을 줄이고, 다운로드하는 시간을 줄일 수 있다. 이러한 웹 캐쉬의 성능은 대체 알고리즘에 의해서 영향을 많이 받는다. 오늘날, 많은 대체 알고리즘들이 웹 캐싱을 위해서 제안되었고, 이 알고리즘들은 크기, 시간적 위치성, 레이턴시와 같은 온라인 상에서의 특성들을 사용하는데, 이들은 오브젝트의 인기도를 정하기 위해서 사용된다. 특히 Size Adjust LRU와 같이 크기나 시간적 위치성을 사용하는 알고리즘 에서와 같이 직접적인 인기도는 사용하지 않게 된다. 그러나 최근의 연구에 따르면 이러한 온라인 상에서의 특성들과 인기도간의 관계가 프락시 캐쉬에서 관찰하면 약해짐을 알 수 있다. 이 결과는 클라이언트 캐쉬가 효율적으로 동작하면서 생기게 된 것이다. 이 논문에서는 이러한 약점을 보완하기 위해 새로운 알고리즘인 *바이트 단위의 최소 인기도 우선 대체 알고리즘*(LPPB-R)을 제안 한다. 우리는 약점을 보완하기 위해서 긴 시간단위의 주기성 측정을 위한 변수를 사용하였다. 그리고 프락시 캐쉬의 용도에 맞게 성능을 조절 할 수 있도록 변수를 설정할 있다. 이에 덧붙여서, 우리는 다중 대기배열을 사용하였고 캐쉬 오염을 막기 위한 기법을 적용하였다. 그리고 우리는 트레이스를 분석하는 실험을 통해서 알고리즘들 간의 성능을 비교 하였다.

References

[1] M. Abrams, C. Standridge, G. Abdula, S. Williams and E. Fox, "Caching Proxies: Limitations and Potentials." Proc.Fourth International World Wide Web Conference, Boston, 1995.

[2] Jia Wang, "A Survey of Web Caching Schemes for the Internet." ACM Computer Communication Review, 29(5):36-46, October 1999.

[3] S. Williams, M. Abrams, C. R. Standbridge, G. Abdulla and E. A. Fox, "Removal Policies in Network Caches for World-Wide Web Documents." In Proceedings of the ACM Sigcomm96, August, 1996.

[4] Lee Breslau, Pei Cao, Li Fa, Graham Phillips, and Scott Shenker. "Web caching and Zipf-like distributions: evidence and implications." In Proceedings of Infocom'99, April, 1999.

[5] Pei Cao and Sandy Irani, "Cost-aware WWW proxy caching algorithms." In Proceedings of Usenix symposium on Internet Technology and Systems, December, 1997.

[6] R. P. Wooster and M. Abrams, "Proxy caching that estimates page load delays", Proceedings of the 6th International WWW Conference, April 1997.

[7] Charu Aggarwal, Hoel L. Wolf and Philip S. Yu, "Caching on the World Wide Web", IEEE Transaction on Knowledge and data Engineering, Vol.11, No.1 January 1999.

[8] Carlos Cunha, Azer Bestavros, and Mark Crovella. "Characteristics of WWW client-based traces." Technical Report BUCS-95-010, April 1995.

[9] Parul Barford, Azer Bestarvros, Adam Bradley, and Mark Crovella. "Changes in Web client access patterns : characteristics and caching implications." WWW Journal Vol.2 No.1, 1999.

[10] A. Luotonen and K. Altis, "World Wide Web proxies", Computer Networks and ISDN Systems, First International Conference on WWW, April 1994.

[11] L. Rizzo, and L. Vicisano, "Replacement policies for a proxy cache", Research Note RN/98/13, Department of Computer Science, University College London, 1998.

[12] J. C. Bolt and P. Hoschka, "Performance engineering of the World-Wide Web: Application to dimensioning and cache design", Proceedings of the 5th International WWW Conference, Paris France, May 1996.

[13] P. Scheuermann, J. Shim, and R. Vingralek, "A case for delay-conscious caching of Web documents", Proceedings of the 6th International WWW Conference, Santa Clara, April 1997.

[14] Virgilio Almeida, Azer Bestavros, Mark Crovella, and Adriana de Oliveira. "Characterizing Reference Locality in the WWW." In Proceedings of International Conference on Parallel and Distributed Information Systems, December, 1996.

[15] National Laboratory for Applied Network Research. Weekly access logs at NLANR's proxy caches. Available via ftp:// ircache.nlanr.net/Traces/.

[16] Martin Arlitt and Carey Williamson. "Internet Web servers: Workload characterization and implications." IEEE/ACM Transactions on Networking, 5(5): 631-644, 1997.

[17] R. Caceres, F. Douglis, A. Feldmann, G. Glass, and M. Rabinocich, "Web proxy

caching: the devil is in the details", ACM Performance Evaluation Review, 26(3):11-15, December 1998.

감사의 말

이 논문이 나오기까지 많은 분들의 도움이 있었습니다. 모든 분들께 감사 드립니다. 먼저 부족한 저에게 많은 격려와 충고로 항상 배려해 주시고 석사 과정 동안 여러모로 보살펴 주신 박대연 교수님께 감사드립니다. 교수님의 열정과 가르침은 저에게 많은 표본이 되었고 항상 용기를 주셨습니다. 모든일을 철저히 살피시고, 인자한 웃음으로 항상 저희를 맞아주시던 박규호 교수님의 따끔한 충고와 조언에 감사드립니다. 또한 언제나 부담없는 대화와 웃음으로 대해 주신 백윤홍 교수님의 충고와 지도에 감사드립니다.

2년동안 동고 동락하였던 실험실의 선배님들, 동기들 그리고 후배들에게도 감사드립니다. 항상 날카로운 충고와 많은 이해심으로 저에게 많은 도움을 주셨던 철호형, 실험실의 대장으로써 책임을 다하느라 소리를 질러 대지만 그래도 항상 많이 생각 해주는 우현이형, 그리고 조용하면서도 할일은 다하는 그리고 다른사람들에게 도움을 많이 주는 상호형, 항상 즐거운 듯한 그러나 때로는 진지한 양우형, 후배 생각을 많이 해주고 실험실에 대해서 항상 많은 신경을 쓰면서도 묵묵히 자신의 일은 제대로 해내는 재선이형, 우리 실험실의 근면에 있어서 둘째라면 서러울 진수형, 고등학교 때부터 친구였고 무슨일이든 잘 해 낼 수 있는 용진이, 운동도 좋아하고 공부도 열심히 그리고 항상 밝은 모습의 근태, 실험실의 전화는 모두 받았주고 막내로써 일을 잘 해낸 용주, 모든일에 열심인 상엽이형, 나와 비슷한 분야를 시작해서 얘기를 많이 나눴던 재섭이, 그리고 지금은 학생은 아니지만 논문을 쓰는데 많은 도움을 주었던 승원형, 항상 후덕하게 웃고 다니는 재웅이형, 밝은 모습으로 꿋꿋하게 살아가는 동은이. 이 모두에게 다시한번 감사의 말을 전합니다. 그리고 같은 실험실은 아니지만 항상 저에게 많은 도움을 주셨던 경호형, 종현이형, 주영이형 형진이형, 창규형, 상석이에게도 감사드립니다.

나의 친구들, 경석, 성진, 승은, 용무, 한태, 계태등 같은 길을 걸어온 전남과고 친구들 모두에게 감사의 말을 전합니다. 그리고 이런 선배를 믿어준 세완, 창효, 동

희등의 모든 후배들아 고맙다.

보준아, 넌 나에게 있어서 정말 소중한 동생이다. 비록 떨어져 있어서 많은건 해주지 못했지만 항상 옆에서 지켜 봐 줄게, 너도 너의 꿈을 펼쳐 보렴. 그리고 사랑하는 어머니 아버지, 고등학교 때부터 쭉 떨어져 지내느라 고맙다는 말한번, 표현한번 못해 드렸네요. 이 논문을 빌어서 사랑한다는 말을 드립니다.

마지막으로 항상 내가 외로울때 그리고 힘들때 나에게 용기를 북독아 주고, 현재의 내가 될 수 있도록 도와주었던 사랑하는 하나에게 감사의 말을 전합니다.

이 력 서

성　　　명 : 김 경 백

생 년 월 일 : 1976년 5월 15일

출　생　지 : 전라남도 목포시

본　　　적 : 전라남도 목포시

학 력

1994.3–1999.2　한국과학기술원 전기및전자공학과 (B.S.)

1999.3–2001.2　한국과학기술원 전자전산학과 전기및전자공학전공 (M.S.)