IEICE TRANS. INF. & SYST., VOL.E90–D, NO.1 JANUARY 2007

PAPER Special Section on Parallel/Distributed Processing and Systems

Efficient and Tailored Resource Management for the P2P Web Caching

Kyungbaek KIM[†a)], ***Student Member and*** **Daeyeon PARK**[†], ***Member***

SUMMARY While web proxy caching is a widely deployed technique, the performance of a proxy cache is limited by the local storage. Some studies have addressed this limitation by using the residual resources of clients via a p2p method and have achieved a very high hit rate. However, these approaches treat web objects as homogeneous objects and there is no consideration of various web characteristics. Consequently, the byte hit rate of the system is limited, external bandwidth is wasted, and perceived user latency is increased. The present paper suggests an efficient p2p based web caching technique that manages objects with different policies so as to exploit the characteristics of web objects, such as size and temporal locality. Small objects are stored alone whereas large objects are stored by dividing them into numerous small blocks, which are distributed in clients. On a proxy cache, header blocks of large objects take the place of objects themselves and smaller objects are cached. This technique increases the hit rate. Unlike a web cache, which evicts large objects as soon as possible in the case where clients fulfill the role of backup storage, large objects are given higher priority than small objects in the proposed approach. This maximizes the effect of hits for large objects and thereby increases the byte hit rate. Furthermore, we construct simple latency models for various p2p based web caching systems and analyze the effects of the proposed policies on these systems. We then examine the performances of the efficient policies via a trace driven simulation. The results demonstrate that the proposed techniques effectively enhance web cache performance, including hit rate, byte hit rate, and response time.

key words: *peer-to-peer, Web caching, replacement algorithm*

1. Introduction

Web caching is a widely deployed technique. Its role is to reduce network traffic, server load, and user-perceived retrieval delay by replicating popular content on caches that are strategically placed within the network. Browser caches reside in client desktops, and proxy caches are deployed on dedicated machines at the boundary of corporate networks and Internet service providers. The performance of a proxy cache depends on the size of its client community [1]. As the user community increases in size, so does the probability that a cached object will soon be requested again. According to this, most proxy caches are deployed at the boundary of an institution such as a university or a corporation where users crowd, and the network is connected by a fast LAN.

The performance of a proxy cache has a limitation caused by the limited cache storage. When the client community increases in size, the number of requested objects per second also increases. General proxy caches store new objects for future requests and evict old objects to make room for the new objects. If the proxy cache storage area is too small to store both the new and old objects, objects will be evicted before they can fulfill user requests, and thus the performance of the proxy cache decreases. That is, when more clients want to use the proxy cache, more proxy cache storage is needed to conserve the level of performance.

Some works [2]–[5] have exploited p2p systems to support web caching. In s web caching system, the main performance limitation is the size of the cache storage. This limitation has been addressed by making use of the residual storage of clients, which is managed by the peer-to-peer method. Every client that wants to use the proxy cache system provides its own storage and this storage is organized into the peer-to-peer based proxy cache system, which is used to store requested objects. Each client uses a DHT based p2p protocol to find the location where an object resides in the cache system and each client only manages its small DHT. When the number of clients increases, the storage for the cache system increases automatically and the cache system also preserves the level of performance without any cost for new storage. These p2p based web caching systems can be categorized into two types: central server based systems and fully decentralized systems. On the Sect. 2, we describe the detailed mechanisms of them.

Although p2p based web caches have many advantages, there is a common problem that limits performance improvement. That is, the relation between the p2p storage and the characteristics of web objects, such as the size and temporal locality, is not considered. Because of the nature of the DHT based p2p protocol, an object is matched to a client that has the numerically closest node ID to the object ID. However, the size of the web object varies from 10 bytes to 10 Mbytes or more, and the client loads are unbalanced. Some clients cannot store large sized objects whose size is greater than the maximum size of their storage. As a result of this deficiency in the management of p2p storage, storage is not used efficiently even when the available storage of the p2p based web cache system is increased. In particular, storage of large sized objects is problematic and the hit ratio for such objects decreases. As a result, more external bandwidth is required to fetch large sized objects from an outside source.

In addition, the lengthy lookup of the p2p based system should be addressed. Unlike normal file sharing systems, the web cache system should respond to each client as soon

Manuscript received March 1, 2006.
Manuscript revised June 30, 2006.

[†]The authors are with the Department of Electrical Engineering and Computer Science [Division of Electrical Engineering], Korea Advanced Institute of Science and Technology, 373–1 Guseong-dong Yuseong-gu, Daejeon, 305–701, Korea.
a) E-mail: kbkim@sslab.kaist.ac.kr

DOI: 10.1093/ietisy/e90–d.1.48

as possible. P2p based systems distribute the routing information to each client and a lookup message follows some hops to the destination. A request for p2p storage requires more lookup traffic and more latency than in a traditional web proxy. In this case, if the p2p storage attempts to store a greater number of small sized objects so as to maximize the effect of temporal locality, even in circumstances of light request traffic and a high hit ratio, the lookup overhead will be relatively long. Accordingly, the previous approaches involve long user-perceived latency.

In this paper, in order to exploit the characteristics of web objects efficiently for p2p based web caching systems, we manage distributed objects with different caching policies according to their size. The primary emphasis of the proposed approach is on storing large sized objects efficiently. If the number of clients that want to use the cache system increases, the total storage of the whole cache system also increases. This feature yields sufficient storage to store requested objects, particularly small sized objects, and leads to a high hit ratio. However, because of the limited storage of clients and low priority of large sized objects, the effect of the hit for large sized objects is not exploited. As such, internet traffic is reduced enormously. If large sized objects are stored efficiently, then both a high hit ratio and a high byte hit ratio could be achieved. To this end, we suggest two policies: a *storage policy* and a *replacement policy*.

For the storage policy, small sized objects are stored alone, but large sized objects are stored by dividing then into many small blocks, as each client does not have adequate storage for the whole of the object. Each client who stores large sized objects obtains the header objects for the objects and data blocks are distributed in other clients. Thus, the storage overhead for each client is reduced and is balanced. In particular, the proxy cache that is used in central server based p2p web caching systems [3], [4] stores only small sized objects and the header objects for large sized objects except data blocks so as to achieve a high hit ratio and reduce the response delay.

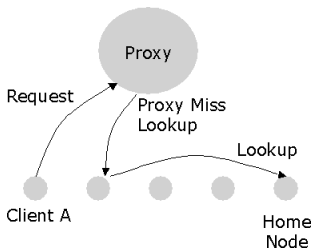
When a cache requires space for new objects, it evicts less useful cached objects, which are selected by the replacement policy. The proxy cache, which handles all requests, adheres to a legacy replacement policy, which evicts large sized objects as soon as possible. However, in the client storage, we evict small sized objects first. That is, we give large sized objects higher priority than small sized objects so as to increase the availability of the large sized objects. In order to prevent the loss of even a single block of the large sized object spoils the whole of the object, we apply the n-chance replacement algorithm to the clients. This policy permits the chance of moving blocks for n times before evicting the blocks and makes the availability higher.

By using these policies, both the hit ratio and the byte hit ratio increase for the p2p based web caching system. In web caching systems, in addition to the hit ratio and the byte hit ratio, the perceived response time is an important parameter. Generally, p2p based web caching systems can achieve a high hit rate, but the response time is longer than that of a traditional web caching system because of the heavy lookup cost. To determine how the proposed policies affect these systems, we constructed simple models for these p2p based web caching systems and analyzed the response times. According to this analysis and the simulation result in Sect. 4, we can find out that our policies help for the p2p based web caching system decreasing the user latency.

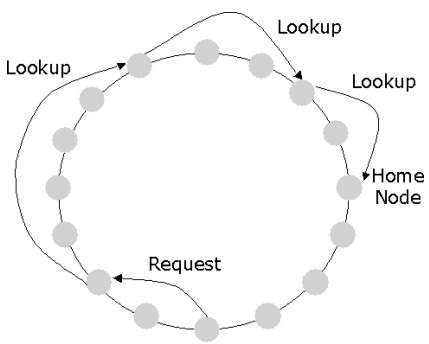
This paper is organized as follow. In Sect. 2, we describe the peer-to-peer web caching briefly. Section 3 introduce the detail of storing large sized objects and cache replacement policies. The analysis of the response time is also appeared in this section. The simulation environment and the performance evaluation are given in Sect. 4. Finally, we conclude in Sect. 5.

2. Background

In recent years, new solutions have been proposed to utilize residual client storage, which is managed by p2p methods to address the issue of scalability. These solutions can be categorized into two types: central server based systems [2]–[4] and fully decentralized systems [5]. The central server based systems use client storages as backup storages. As shown in Fig. 1 (a), if a client request misses in a local browser cache and the proxy cache, the proxy server will find the correct object in the clients' storage. In [3], the proxy server connected to a group of networked clients maintains an index file of data objects of all clients' caches. To distribute the load of the proxy server to the clients, [2] proposes new approach involving superclients which help the proxy server and maintain a divided index file. In [4], the virtual web cache is composed of the proxy server and the connected



(a) Central server based system.



(b) Fully decentralized system.

Fig. 1 Peer-to-peer based web cache systems.

clients. These are in the same AS and the proxy server only maintains a small DHT, which is used to lookup the client storage. In [5], a fully decentralized p2p web cache called *Squirrel* is proposed. Web caching workloads are taken by all clients and the dedicated proxy server is eliminated. Figure 1 (b) shows the basic operation of Squirrel.

To efficiently exploit the resources of clients, these approaches use a self-organizing p2p routing substrate for their object location service in order to identify and route *the home node*, which contains a copy of the requested object [6]–[9]. In a self-organizing and decentralized manner, these protocols provide a DHT that reliably maps a given object key to a unique live node in the network. If a node wants to find an object, the node simply sends a query to a selected node determined by the DHT with the object key corresponding to the object. According to this procedure, an object is mapped to a live node. However, because web objects have very various sizes ranging from 10 bytes to 10 Mbytes or more, the storage usage and requested traffic is unbalanced despite that p2p protocols balance the number of objects for which each client is responsible.

Moreover, because client storage is small and may be limited by owner imposed constraints, large sized objects whose size is greater than the maximum storage of a client cannot be stored. Consequently, a load unbalance occurs and the byte hit rate is limited; this wastes external bandwidth despite solving scalability problems with the p2p method. Moreover, the p2p based web caching system have too much overhead to find objects and the response time also increases. When the based p2p protocols attempt to locate an object, the lookup cost is averagely O(log(N)), where N is the number of live nodes in the system. In the case of CHORD [6], when the number of nodes is 500, the average lookup cost is O(log2(500)), i.e., about 9, which increases the response time excessively. In addition, when a client requests an object, the p2p based web caching systems should use O(log(N)) of lookup messages, which constitute additional traffic overhead.

3. Proposed Idea

3.1 Handle of Large Sized Objects

In both types of p2p based web caches, an object is stored in the corresponding client, called the *home node*, which has numerically closest node key to the object key. Small sized objects can be stored at each home node by itself. However, each client supports the residual resource which is not used by a client and it is too small to store the whole of the large sized object. To solve this problem, we break up the large sized object into many small sized blocks and store these blocks to many clients. Each block has the block key which is obtained by hashing the block itself and the home node for the block key stores it. According to this, all of blocks for a large sized object are distributed in the clients and the storage overhead for each client is reduced and balanced.

The index-based allocation method to store large sized objects is used, as this method is simple, cost-effective functions simply for random access. Figure 2 shows the simple structure of the proposed index-based allocation method. Initially, the *object header block* is required. This contains basic information about a large sized object, such as its URL, size and time of modification, and index pointers, such as single, double and triple indirect index pointers. Direct index pointers are not used in the object header block. In the general indexed method, a direct index pointer is used to store small files and to avoid making unnecessary index blocks. However, the size of the stored objects is large enough so that the overhead of the index blocks can be neglected. The home node for a large sized object stores this object header block instead of storing the object itself.

An index pointer indicates an index block using an index block key, which is the hashed value of the index block itself. The index block is composed of a URL, as well as block pointers which address data using a data block key, and the range of the block pointers. The data block is the leaf block of this method, and stores the actual data chunk. Each data block has a URL as well as a block number, which is assigned continuously from the start of the object to the end.

3.2 Storage Policy

Figure 3 shows how the client or proxy stores the requested objects. We use two lists, a *Small sized Object List* (SOL) and a *Large sized Object List* (LOL), to store objects. The SOL and LOL are basically managed by the LRU (Least-Recently-Used) list. The SOL is simple linked lists of small

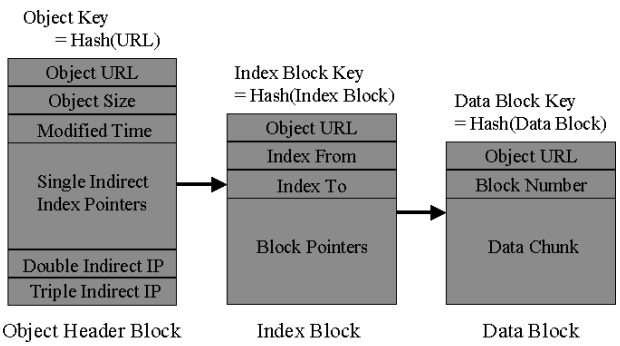


Fig. 2 The structure of a large sized object.

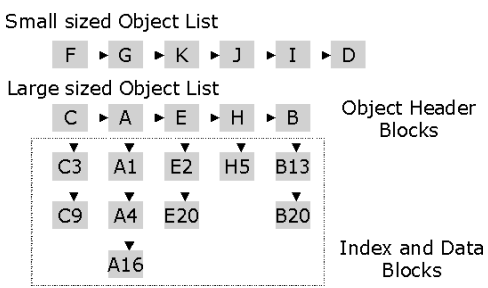


Fig. 3 SOL (Small sized Object List) and LOL (Large sized Object List).

sized objects, because small sized objects are stored by themselves. The LOL is the linked lists of the object header blocks of large sized objects. One header block has a block list whose entries indicate the index and data blocks of the large sized object on this storage. For example, in Fig. 3, the storage that has this LOL contains C, A, E, H, and B, which are large objects, and C3 and C9, which are the index or the data blocks for the large object C.

In the central server based system, the clients use both the SOL and the LOL; otherwise the proxy cache uses the SOL and the LOL without an index or data blocks. The proxy cache should handle all requests regardless of the location of the requested object. If there are more requested objects, the p2p based web caching system saves bandwidth and reduces more perceived user latency. By this approach, a maximal number of objects is stored in the proxy cache. To do this, in the proxy cache, the object header block takes the place of a large sized object and the remaining storage is used to store small sized objects. This increment of the number of small sized objects in the proxy cache helps reduce bandwidth usage and retrieval delay, because hits for small sized objects prevent expensive and long p2p lookups for them in the clients' storage. The clients' storage is generally used to lookup large sized objects, whose lookup overhead is negligible.

In a fully decentralized system where there is no central server, every client uses both the SOL and the LOL and every client acts performs the same role. Accordingly, this system distributes the object in a well balanced manner and stores more objects; however, user latency is not reduced.

3.3 Replacement Policy

All web caches have limited storage and thus require a replacement algorithm that chooses which objects are evicted when new objects are requested and new storage is needed. Generally, web caches evict large sized objects as soon as possible so as to store more small sized objects. However, in p2p storage, this approach harms the user-perceived latency due to the long latency and heavy control traffic involved in the lookup for an object. In order to prevent compulsory latency and traffic, we give large sized objects higher priority than small sized objects at the clients' storage. That is, the proxy cache is mainly responsible for storing small sized objects while the clients mainly maintain large sized objects. The implementation of well separated roles for each part of the p2p based web caching system helps increase both the hit ratio and the byte hit ratio.

For example, if a client that has a SOL and LOL, as shown in Fig. 3, requires space for new objects, it first evicts "D", which is a small sized object and the least recently used object. Then, if the client needs more space, it evicts other small sized objects until the SOL is empty. If a cache needs space and the SOL is empty, it is necessary to evict the blocks of large sized objects. In this case, we evict "B20", which is the last data block of the least recently used large sized object "B". If it is necessary to evict more objects, the sequence of the evicted object will be "B13", "B", "H5", "H", "E20", and so on. However, in the proxy, because it does not store the index and data blocks of large sized objects and there are numerous small sized objects, the replacement occurs at the SOL only. The object header blocks are only evicted when a lookup miss occurs and the client notices that a large sized object has been evicted.

Large sized objects are well distributed in the clients, but missing just one block can spoil the whole of the large sized object. The missing of a block occurs when a client leaves/fails or if a client evicts blocks to to store new blocks (replacements). It is possible to overcome client failures or departures by applying a simple replication strategy to the p2p protocol for the clients' storage [7], [11]. In order to solve the problem of a replacement, blocks to be evicted are transferred to other clients before they are evicted. When a client evicts a block, it first regenerates a different block key by hashing the block and an optional suffix having a random value. In order to move the block correctly, the client finds the object header block, index block and the block number of the evicted block through the URL, and updates the block pointer with the new block key. The possibility of moving blocks for one large sized object is permitted until the number of chances is larger than a threshold value, n. If a large sized object uses all n chances, all of the blocks of the object are removed from the clients' storage. This is known as the n-chance replacement policy. In our simulation, the n value is 5.

3.4 Analysis of the Response Time

In web caching systems, the hit ratio, the byte hit ratio and the perceived response time are all important parameters. If the response time is too long for the user to wait for a requested object, though a p2p-based web caching systems achieves a high hit ratio and a high byte hit ratio, these systems do not excel at web caching.

To determine how the proposed system affects the response time of web caching systems, simple models for the systems are utilized. Additionally, a number of simplifying assumptions regarding the p2p substrate are made. First, the data is assumed to be always available. Even if clients dynamically join or leave, the availability of the data is preserved by the fault-tolerant p2p substrate, and this fault handling does not affect response time. Second, the total number of clients is assumed to remain static, and the p2p lookup cost is limited and manages the average value by the ideal formula, O(logN).

For a set of N identical clients interconnected by high speed LANs whose bandwidth is B_i, the average RTT of the connection from a client to another client is R_i. There is a proxy cache which is placed at the front end of the intra network. The bandwidth of the outside link is B_o, and the average RTT of an internet connection is R_o. The hit ratio of the web caching system is H_r and the hit ratio of the proxy cache is H_{pr}, which does not include hits from the clients' storage. The object size is S. According to these parameters,

simple models of the expected latencies were made for the various p2p based web caching systems.

These following equations represent the response times for the only proxy cache (P), the fully decentralized system (C) and the central server based system (PC). Basically, when the miss occurs we need more transfer time to get the objects from outside links which have less bandwidth (B_o) than the bandwidth of inside links (B_i). Moreover, when we use the p2p method we need more lookup time such as $\mathrm{Log}(N)R_i$. Especially in the fully decentralized system, though the hit occurs, the lookup time still exists and this make the response time longer.

To overcome this lookup overhead, we should maximize both of the hit ratio and the byte hit ratio and reduce more transfer time from outside links. When the p2p based web caching systems use our policies, the both of the hit ratio and the byte hit ratio increase effectively, and this helps reducing more external traffic. For example, the fully decentralized system (C) has basically longer lookup overhead ($\mathrm{Log}(N)R_i$) than the only proxy cache (P), but if the hit ratio on C increases effectively by using our policies, the effect of high hit ratio compensates for the lookup overhead of the p2p based system. Moreover, on the central server based system (PC), we can reduce more lookup overhead than P and C. When we use our policies, the proxy cache can store more small sized object and the hit ratio of the proxy (H_{pr}) increases effectively. This increase of H_{pr} prevent to take the long lookup in the clients' storage ($\mathrm{Log}(N)R_i$) and the transfer time from outside links more effectively than P and C. Consequently, our new policies make the p2p based web caching systems reducing the response time and encourage the web caching systems to adopt the p2p method. On the Sect. 4, we can find out that the our policies help the p2p system reducing the latency.

$$
\begin{aligned}
P &= H_r(R_i + S/B_i) \\
&\quad + (1 - H_r)(R_i + R_o + S/B_i + S/B_o) \\
C &= H_r(\mathrm{Log}(N)R_i + S/B_i) \\
&\quad + (1 - H_r)(\mathrm{Log}(N)R_i + R_o + S/B_i + S/B_o) \\
PC &= H_r(H_{pr}(R_i + S/B_i) \\
&\quad + (1 - H_{pr})(\mathrm{Log}(N)R_i + S/B_i)) \\
&\quad + (1 - H_r)(R_i + R_o + S/B_i + S/B_o)
\end{aligned}
$$

4. Performance Evaluation

In this section, we present the results of extensive trace driven simulations that we have conducted to evaluate the performance of our caching policies. We design our p2p based web cache simulator to conduct the performance evaluation. We have assumed that we simulate the behavior of a proxy cache effectively. The proxy cache is error-free and does not store non-cacheable objects: dynamic data, larger size data than total cache storage, control data, and etc. We also assume that there are not any problems in the network, such as congestions and buffer overflows. The size of a proxy cache is 200 MBytes and each client has 10 MBytes storage. We assume the large sized object is bigger than 1 Mbytes and the size of blocks for them is 32 KByte.

In our trace-driven simulations we use traces from KAIST, which uses a class B ip address for the network. We show some of the characteristics of these traces in Table 1. Note that these characteristics are the results when the cache size is infinite.

We compare five systems such as only proxy (P), the central server based system (PC), the central server based system with our policies (PCL), the fully decentralized system (C) and the fully decentralized system with our policies (CL). P, PC and C use the LRU policies for the object management.

4.1 Preliminary Inspection

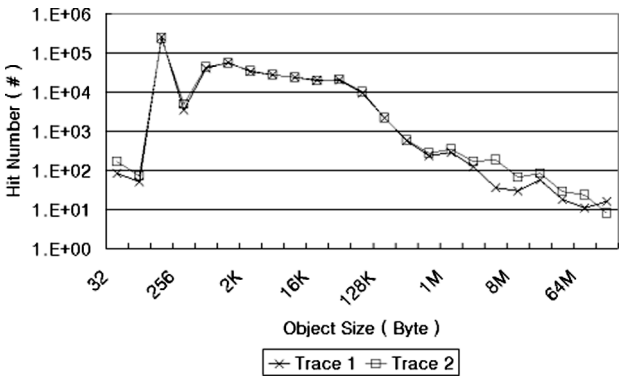
In order to estimate the performance of the proxy cache, observations were made that measured the number of objects handled by the proxy cache. If more hits occurred in a proxy cache, the proxy cache was said to achieve better performance. However, every hit does not have same weight. A good example of this is the byte hit. As the size of a requested object is variable, both the large number of hits for a small sized object and a small number of hits for a large sized object achieve similar byte hits. In Fig. 4, the distribution of the hits which occurs in a proxy cache is shown when the aforementioned traces are used, simulate a proxy cache whose storage is infinite. The hits are distributed by the size of a file. The maximum number of hits obtained takes place for files whose sizes are approximately 256 Bytes, and for a minimum value for 64 MB (approximate size) files. However, in terms of the byte hit, the byte hits on files close to 64 MB are larger than those for files of approximately 256 MB.

According to these results, if the general replacement algorithms evict large sized objects to achieve a high hit rate, they must sacrifice a high byte hit rate. To prevent this degradation of the byte hit rate, it is necessary to store large sized objects in an exclusive storage area having no relationship to the proxy cache. Large sized objects can be stored in the client-cluster, which is composed of clients and required no management cost for a proxy cache or other storage

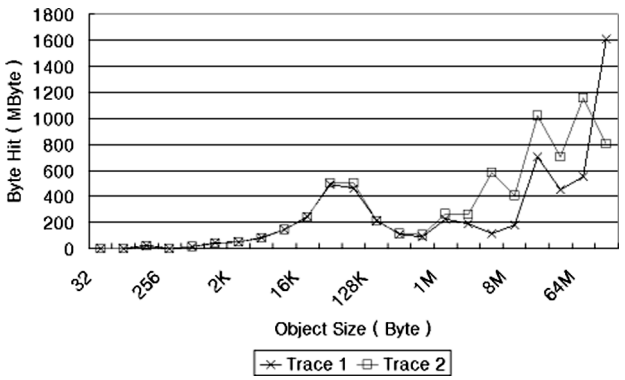
Additionally, in Fig. 4, hits decrease rapidly for files of nearly 1 MByte. This value, 1 MByte, is used as the threshold value for selecting large sized objects.

Table 1 Traces used in our simulation.

Traces	Trace 1	Trace 2
Measuring day	2001.10.08	2001.10.09
Requests Size	9.02 GB	11.66 GB
Object Size	3.48GB	4.92 GB
Request #	699280	698871
Object #	215427	224104
Hit Rate	69.19%	67.93%
Byte Hit Rate	63.60%	57.79%



(a) Hit number.



(b) Byte hit.

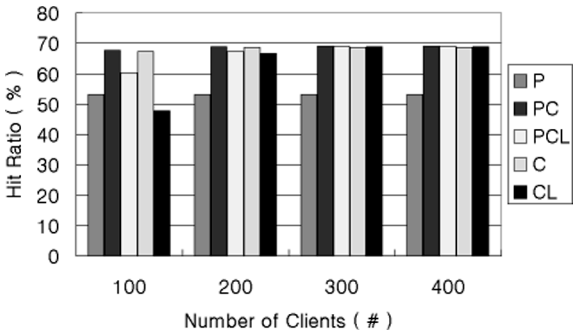
Fig. 4 Hit distribution in the cache system which has the infinite storage.

4.2 Hit Rate and Byte Hit Rate

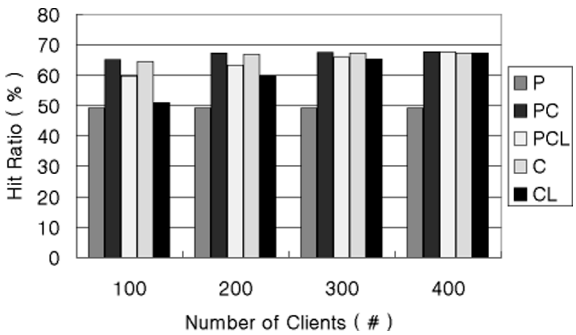
Figures 5 and 6 show comparisons of the hit rate and the byte hit rate. In Fig. 5, we find that the both of the p2p based web caching systems achieve about 20% higher hit rate than the normal web proxy caching system. However, in Fig. 6, even if the hit rate of the fully decentralized system is much higher than the only proxy, its byte hit rate is about half of the byte hit rate of the only proxy. In this system, the client which has limited cache storage can not store large sized objects and it can not get the effect of the hit for large sized objects. Oppositely, in the fully decentralized system with our policies, though the hit rate is slightly lower than the fully decentralized system, but the byte hit rate is much bigger than it. Moreover, in the central based system whose byte hit rate is already bigger than the only proxy, when we apply our policies to this system, the byte hit rate increases more.

Because large sized objects are given higher priority than small sized objects in order for the large sized objects to come to be many, the cache pollution can occur in the client's storage. In the Fig. 5, p2p web caching systems with our policy (PCL, CL) achieve lower hit rate than normal p2p web caching systems (PC, C). However, this degradation of hit rate dwindles away when the number of client increases and the storage of the system increase. Moreover, when a large sized object is evicted, the relatively large free space is obtained and small sized objects can be stored.

According to these, if we handle the large sized object efficiently in p2p based web caching systems, we achieve

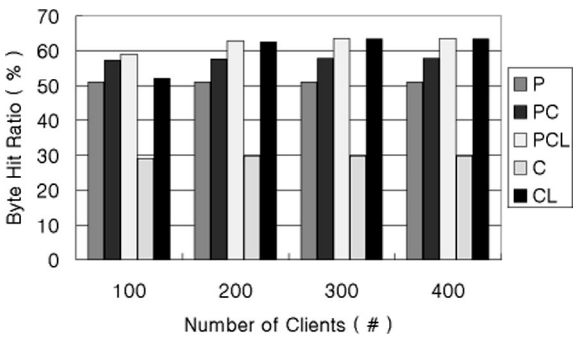


(a) Trace 1.

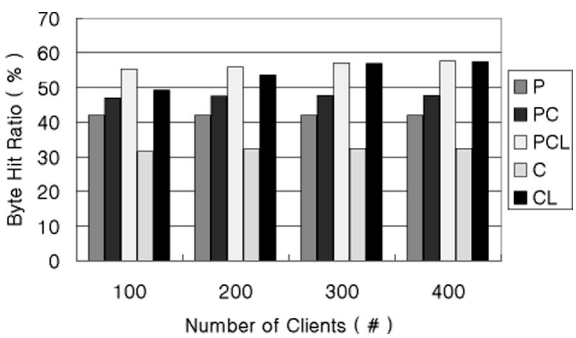


(b) Trace 2.

Fig. 5 Hit rate comparison.



(a) Trace 1.



(b) Trace 2.

Fig. 6 Byte hit rate comparison.

not only the high hit rate but also the high byte hit rate. In addition, when the number of clients increases, if we use our policies, both of the hit rate and the byte hit rate increase remarkably, otherwise, they increase little. This means the p2p based web cache systems with our policies are more scalable.

4.3 External Traffic

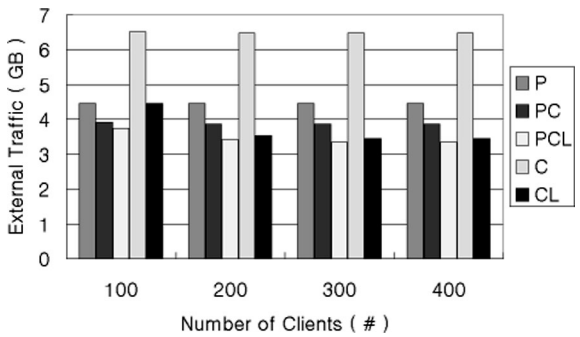
We define *external traffic* as the number of bytes transferred between the intra network and original servers including object data and control messages. This traffic, which uses the core-link of the inter network, should be of a small value for good performance of the Internet and the web caching system. Figure 7 show the external traffic. Like the results of hit rates and byte hit rates, if we use our policies, we save the external traffic remarkably. In the central server based system, our policies save about 1 GB traffic and in the fully decentralized system, our policies reduce about 2.5 GB traffic. The p2p based systems save more external traffic than only proxy except the fully decentralized system. Amazingly, the fully decentralized system wastes most of the external traffic among the systems. The reason is that though its hit rate is much higher than only proxy, this system achieves lowest byte hit rate. According to this, not only the hit rate but also the byte hit rate is important parameter for the p2p based web caching system to save outgoing bandwidth and our policies help them reducing the external traffic.

4.4 Control Traffic

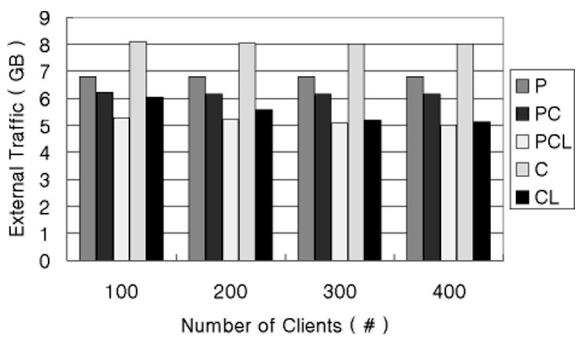
We divide large sized objects to many small blocks and we need more control traffic to gather the distributed blocks, such as lookup messages of p2p substrate, requests for index blocks and requests for data blocks. In our simulation, we set the block size to 32 KB and set the size of a control message to 32 B. Figure 8 shows the comparison of the control traffic. We find that the p2p based web caching systems use more control traffic than normal web caching system, because the lookup for the p2p substrate need the multiple routing hops and need additional messages for the routing process. Especially, the fully decentralized system use much more control traffic than the central server based system, because this system always use the p2p lookup. Moreover, when we use our policies we use more control traffic. However, in Fig. 7, though there is the additional control traffic, the p2p based web caching systems with our policies reduce much more external traffic, and both of the traffic for the object data and the traffic of control messages are included in the external traffic. Consequently, the increment of the performance of p2p based web caching systems with our policies covers up the additional control traffic for the large sized objects and the p2p lookups.

4.5 Response Time

Though the p2p based web caching systems support very big storage, the lookup cost is very heavy because the lookup of

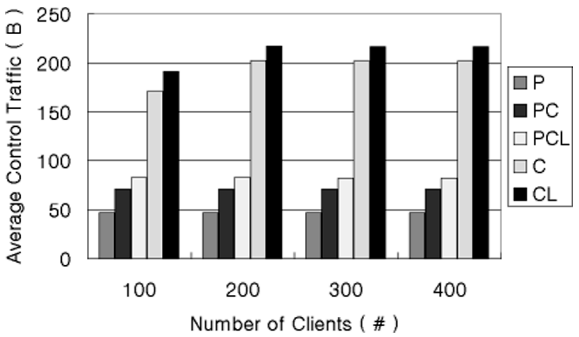


(a) Trace 1.

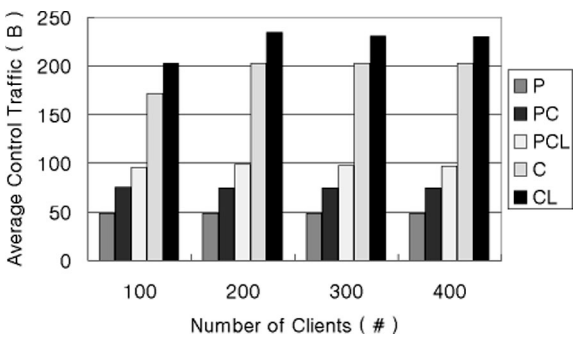


(b) Trace 2.

Fig. 7 External traffic comparison.



(a) Trace 1.



(b) Trace 2.

Fig. 8 Control traffic comparison.

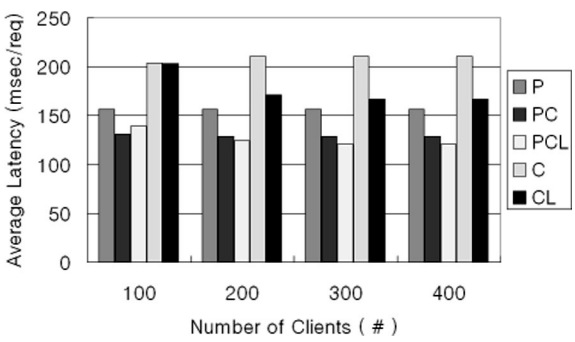
the p2p substrate needs the multiple routing hops. General DHT based p2p protocols need O(log N) lookups where N is the number of the live nodes in the system. In Fig. 9, we show the comparison of the average latency. Moreover, in Fig. 10, we show the comparison of the latency in cumulative density function style. Because of the lookup overhead, the fully decentralized systems need more latency than others. This lookup overhead is observed easily in the Fig. 10. Every request of C and CL takes about 50 msec lookup overhead. However, the central server based systems (PC, PCL) achieve less latency time than others even though they use DHT based p2p protocol. This means that they are more suitable to the web caching system than the fully decentralized systems. Moreover, when we use our policies with both of the central server based and fully decentralized systems, we reduce the average latency more in accordance with the increment of clients. Especially, in the fully decentralized systems, when the number of clients is 400, we reduce the latency by 45 msec.

In Fig. 10, when our policies are applied, the 90th percentile of the latency increases slightly. In clients' storage, large sized objects have higher priority than small sized objects and the evicted small sized objects are obtained from the outside link. This cache pollution effect decreases the hit rate and increases the latency slightly. However, this increase is only few msec and negligible. Moreover, when an user requests a large sized object and many small sized object, this slight increase of the latency is compensated with the profit of the hit for the large sized object.

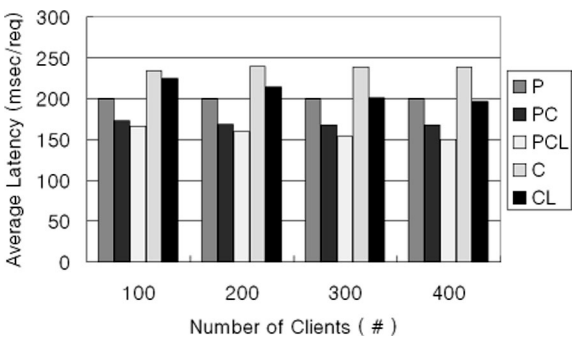
According to these facts, we find that our policies not only increases both of the hit rate and the byte hit rate, but also reduces the average latency.

4.6 Various Proxy Size

According to the previous results, our policies are more suited for the central server based p2p web caching than fully distributed p2p web caching. The Fig. 11 shows the performance comparison of a central server based p2p web caching with various sizes of a proxy cache. The number of clients is 100 and the size of a proxy cache varies from 100 MB to 500 MB. In PC system, the clients' storage stores many small sized objects and the system achieves very high hit rate without any relation to the size of a proxy cache. However, despite of the very high hit rate, the byte hit rate does not increase remarkably. When the proxy size increases and there is room for store large sized objects in the proxy cache, the byte hit rate of PC system increases. In PLC system which uses our policies, while the hit rate is lower than PC system, it achieves very high byte hit rate. The clients' storage tries to keep large sized objects and the proxy cache has the most frequently accessed small sized objects. When the size of a proxy cache increases, the number of stored small sized objects in the proxy cache increases and the hit rate increases linearly to the size of a proxy cache. Even if the trade-off of the performance exists between our policies and legacy policies, the average latency is similar to each other, like Fig. 11 (c).

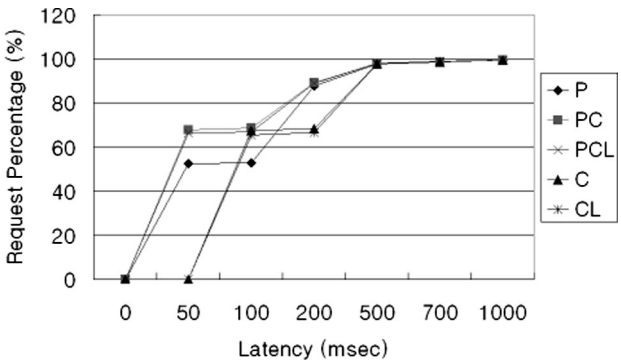


(a) Trace 1.

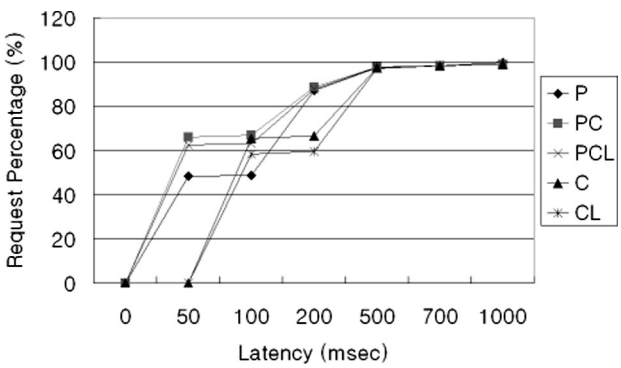


(b) Trace 2.

Fig. 9 Average latency comparison.



(a) Trace 1.

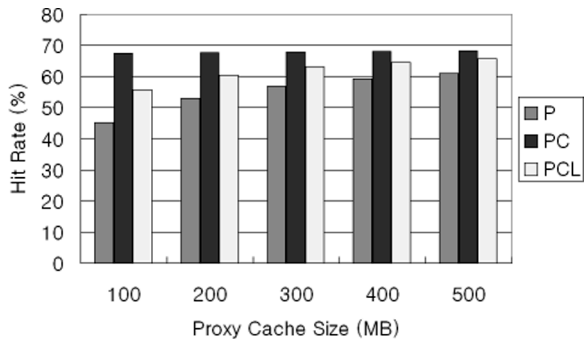


(b) Trace 2.

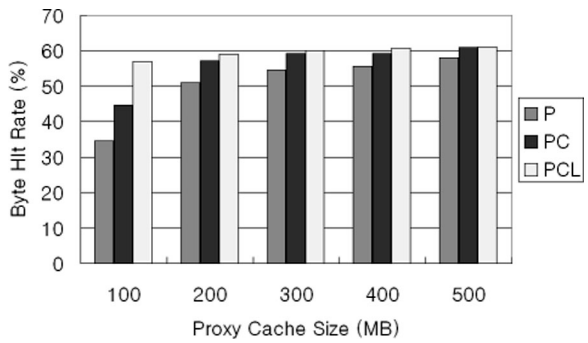
Fig. 10 Latency comparison in CDF style. (client number: 200)

Table 2 Summary of client loads for Trace 2.

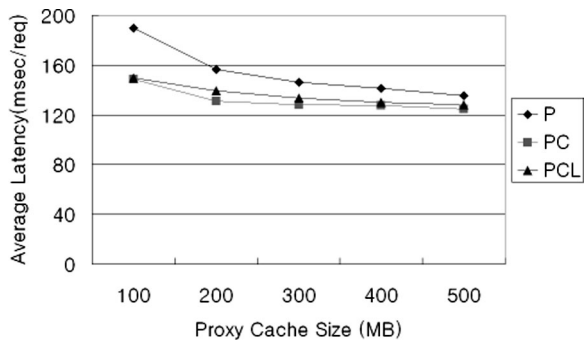
		Mean Size	Dev.	Mean Req.	Dev.	Mean Breq	Dev.
100	PC	9904 KB	9.91%	3519	2.82%	65772 KB	94.41%
	PCL	10360 KB	1.04%	3565	2.24%	71168361	60.13%
	C	9906 KB	9.84%	6918	12.54%	113489 KB	109.37%
	CL	10147 KB	1.68%	7568	11.48%	134783 KB	92.12%
200	PC	9406 KB	16.04%	1768	3.89%	33751 KB	127.08%
	PCL	10386 KB	0.71%	1793	3.22%	35961 KB	80.42%
	C	9494 KB	15.24%	3477	16.97%	58128 KB	152.58%
	CL	10074 KB	1.37%	3774	15.66%	67855 KB	130.79%



(a) Hit rate.



(b) Byte hit rate.



(c) Average latency.

Fig. 11 Performance comparison for Trace 1 with various proxy cache size.

4.7 Client Loads

The client loads are examined, including storage size, request number, byte request, and the deviation of each parameters, in order to verify that the clients balance storage and requested queries when our policies are applied. Table 2 shows a summary of client loads. When our policies are applied, while the deviation of request number decreases little, the deviations of storage size and byte request decrease remarkably. The previous p2p based web caching does not consider the web characteristics, especially the size and each client gets different loads even if the request queries are balanced. However, our policies divide the large sized object into many small blocks and not only the request queries but also the other loads are balanced. Moreover, because the client loads are balanced and the storage can be used efficiently, each client stores more object. It helps increasing the performance of p2p based web caching system, such as hit rate and byte hit rate.

5. Conclusions

In this paper, we suggest the efficient policies for the p2p based web caching systems to exploit the characteristics of web objects. Basically, we apply the different caching policies to the objects according to their size: *storage policy* and *replacement policy*. The small sized objects are stored by itself, but the large sized objects are stored by dividing into many small blocks because each client does not have enough storage for the whole of the object. The clients store both types of objects which are small sized objects and all blocks of large sized objects. The proxy stores small sized objects and the header objects for large sized objects to achieve high hit rate and reduce the proxy overhead. Unlike on the proxy cache, on the client we give large sized objects higher priority than small sized object and maximize the effect of hit for the large sized object which increase the byte hit rate. The trace based simulation confirms that our policies is efficient.

References

[1] A. Wolman, G.M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H.M. Levy, "On the scale and performance of cooperative Web proxy caching," Proc. SOSP 1999, pp.16–31, Dec. 1999.
[2] Z. Xu, Y. Hu, and L. Bhuyan, "Exploiting client cache: A scalable and efficient approach to build large Web," Proc. International Parallel and Distributed Processing Symposium, (IPDPS'04), pp.55–64, April 2004.
[3] L. Xiao, X. Zhang, and Z. Xu, "On reliable and scalable peer-to-peer Web document sharing," Proc. International Parallel and Distributed Processing Symposium, (IPDPS'02), pp.23–30, April 2002.
[4] K. Kim and D. Park, "Efficient and scalable client clustering for Web

proxy cache," IEICE Trans. Inf. & Syst., vol.E86-D, no.9, pp.1577–1585, Sept. 2003.
[5] S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: A decentralized peer-to-peer web cache," Proc. Principles of Distributed Computing'02, pp.213–222, 2002.
[6] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," Proc. ACM SIGCOMM 2001, pp.149–160, Aug. 2001.
[7] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," Proc. International Conference on Distributed Systems Platforms (Middleware), pp.329–350, Nov. 2001.
[8] B.Y. Zhao, J. Kubiatowicz, and A. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," UCB Technical Report, UCB/CSD-01-114, 2001.
[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," Proc. ACM SIGCOMM 2001, pp.161–172, 2001.
[10] P. Druschel and A. Rowstron, "PAST: A large-scale, persistent peer-to-peer storage utility," Proc. HotOS VIII, pp.75–80, May 2001.
[11] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS," Proc. SOSP 2001, pp.202–215, Oct. 2001.

Kyungbaek Kim received his B.S. degree and M.S. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 1999 and 2001, respectively. Current he is a Ph.D. candidate at KAIST in Korea. His research interests include operating system, distributed system, world wide web, peer-to-peer algorithm/network and overlay multicast.

Daeyeon Park received his B.S. and M.S. degrees in computer science from University of Oregon, USA, in 1989 and 1991, respectively and Ph.D. degree in computer science from University of Southern Califonia, USA, 1996. He worked at Hankuk University of Foreign Studies from 1996 to 1997. He joined the Department of Electrical Engineering at KAIST in 1998, where he is currently an Assistant Professor. His major interests include operating system, distributed system, parallel processing, and computer architecture. He is a member of KIEE, KISS, and IEEE.