# Efficient Resource Management for the P2P Web Caching

Kyungbaek Kim and Daeyeon Park
Department of Electrical Engineering & Computer Science,
Division of Electrical Engineering,
Korea Advanced Institute of Science and Technology ( KAIST ),
373-1 Guseong-dong, Yuseong-gu, Daejeon 305-701, Republic of Korea

E-mail: kbkim@sslab.kaist.ac.kr and daeyeon@ee.kaist.ac.kr

## Abstract

*P2P based file sharing systems have become an extremely popular application and they are highly scalable, self-configurable and fault tolerant. Some researches exploit them for the web caching systems to solve the scalability issue. However, they treat the web objects as homogeneous objects and there is no concern about the various web characteristics.*

*This paper suggests the efficient web caching system which manages the objects by different policies to exploit the characteristics of web objects. Basically, we apply the different caching policies to the objects according to their size. The small sized objects are stored by itself, but the large sized objects are stored by dividing into many small blocks which are distributed in the clients and the object header block is stored rather than the large object itself. Moreover, we give large sized objects higher priority than small sized objects and maximize the effect of hit for the large sized object which increase the byte hit rate.*

*We examine the performance of the efficient policies via a trace driven simulation and demonstrate effective enhancement of the web cache performance.*

## 1 Introduction

In these days, peer-to-peer systems have become an extremely popular platform for large-scale content sharing. Unlike traditional server-based storage systems, which centralize the management of data in a few highly reliable servers, peer-to-peer storage systems distribute the burden of data storage and communications among thousands of individual clients. These systems provide large-scale, self-configuring, and highly scalable distributed storage services. To provide scalable storage services, the idle resources of client may be managed efficiently. Generally, these p2p systems [5], [6], [7], [8] manage the resources by the DHT ( Distributed Hash Table ) and they achieve the good load-balance and the efficient routing.

Some researches [4], [2], [1] exploit these p2p systems to support the web caching system. In the web caching system, the main limitation of the performance is the size of the cache storage. When the client community increases in size, the number of the requested objects also increases and the web caching system needs more storage to conserve the level of the performance. They address this limitation with the residual storage of clients which is managed by the peer-to-peer method. Every client which wants to use the proxy cache system provides the storage of itself and this storage is organized into the peer-to-peer based proxy cache system which is used to store requested objects. Each client uses DHT based p2p protocol to find the place where an object reside in the cache system and each client only manages its small DHT. When the number of clients increase, the storage for the cache system increases automatically and the cache system also preserves the level of the performance without any management cost.

Though these p2p based web caches have many advantages, there is common problems which limit the growth of the performance. First, these p2p based caches deal with web objects as homogeneous objects. Because of the nature of DHT based p2p protocol, they match an object to a client which has the numerically closest node ID to the object ID. However, the size of the web object is very various from 10byte to 10Mbyte or more and the storage of a client is limited. According to these facts, the loads of clients are unbalanced and some clients can not store the large sized object whose size is bigger than the maximum size of its storage. This behavior exhausts more outgoing bandwidth and reduce the performance of the web cache system. Second, they don't consider the characteristics of web objects.

Proceedings of the Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/E-Learning on Telecommunications Workshop
0-7695-2388-9/05 $20.00 © 2005 IEEE

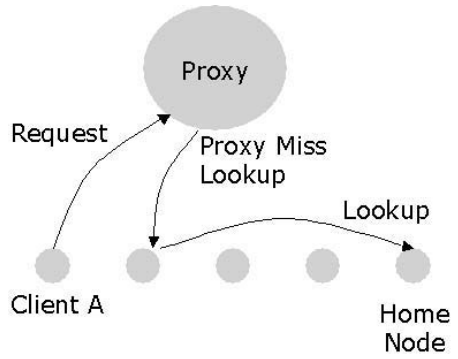
The small sized objects are requested frequently but the large sized objects are not. However, when the large sized objects hit, it reduces much more external traffic than the small sized objects. Though these diverse characteristics of the web objects exist, the previous p2p based caches can not store large sized objects with efficient way, and they try to store more small sized objects and evict large sized objects as soon as possible, to increase the hit ratio and to make a response for them as fast as possible. This management increases the hit ratio dramatically but the byte hit ratio does not, because there is few chance for large sized objects to hit. Consequently, even if we use previous p2p based web caches, we can not reduce outgoing traffic efficiently.

In this paper, for the web caching system to exploit the characteristics of web objects efficiently, we manage the distributed storage with the different caching policies which are dependent on the size of the web objects. Basically, we concentrate on storing large sized objects efficiently. If the number of clients which want to use the cache system increases, the total storage of the whole cache system increases too. This feature makes enough storage to store the requested objects, especially small sized objects and we can get the high hit rate. However, because of the limited storage of clients and low priority of large sized objects, we can not exploit the effect of the hit for the large sized objects which reduces the internet traffic enormously. If we store large sized objects efficiently, we can achieve not only the high hit rate but also the high byte hit rate. To store large sized object efficiently, we suggest two policies : *storage policy* and *replacement policy*.

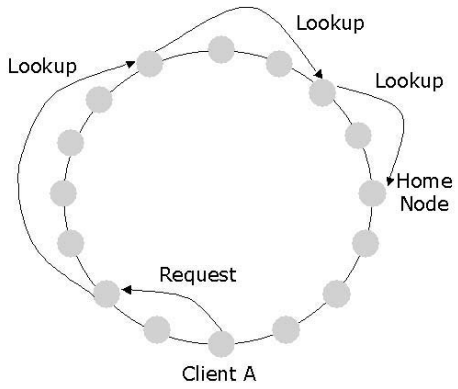
For the storage policy, the small sized objects are stored by itself, but the large sized objects are stored by dividing into many small blocks because each client does not has enough storage for the whole of the object. Each client which stores large sized objects just obtains the header objects for them and data blocks are distributed in other clients. According to this, the storage overhead for each client reduces and is balanced. The proxy cache which is used in the central server based p2p web caching systems [2], [1] stores only small sized objects and the header objects for the large sized objects to achieve the high hit rate.

When a cache needs space for new objects, it evicts less useful cached objects which are selected by our replacement policy. In this case, we evict small sized objects first. That is, we give the large sized objects higher priority than the small sized objects to increase the availability of the large sized objects. To prevent that the missing just one 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.

This paper is organized as follow. In section 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 simulation environment and the performance evaluation are given in section 4. Finally, we conclude in section 5.




(a) Central server based system




(b) Fully decentralized system

**Figure 1. Peer-to-peer based web cache systems**

## 2 Background

The web caching system stores the previous requested objects for the future requests. To increase the performance such as the hit ratio and the byte hit ratio the cache needs enough storage, and a growth in user population creates a need for new storages. That is the scalability issue which can not be solved by ordinary web caching system. In recent years, new solutions have been proposed to utilize the residual client storage which is managed by p2p method to

Proceedings of the Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/E-Learning on Telecommunications Workshop
0-7695-2388-9/05 $20.00 © 2005 **IEEE**

address the scalability issue. These solutions can be categorized into two types: the central server based systems [1], [2] and the fully decentralized systems [4]. The central server based systems use client storages as backup storages. As shown in figure 1(a), if a client request misses in local browser cache and the proxy cache, the proxy server will find the right object in another client's storage. In [1], the proxy server connecting to a group of networked clients maintains an index file of data objects of all clients' caches, but in [2], the proxy server only maintain small DHT which is used to construct client-cluster. In [4], a fully decentralized, p2p web cache, called *Squirrel*, is proposed. Web caching workloads are taken by all the clients and the dedicated proxy server is eliminated. Figure 1(b) shows the basic operation of Squirrel.

In these systems, they uses a self-organizing, p2p routing substrate , for its object location service, to identify and route to *the home node* that cache copies of a requested object [5], [6], [7], [8]. 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, a node simply sends a query with the object key corresponding to the object to the selected node determined by the DHT.

According to this behavior, in previous systems, an object is mapped to a live node. Because the web objects have very various size from 10byte to 10Mbyte or more, though the p2p protocols balance the number of objects for which each client is responsible, the storage usage and the requested traffic is unbalanced. Moreover, because the storage of a client is small and limited by the owner who is selfish maybe, this can not store some large sized objects whose size is bigger than the maximum client storage. Consequently, the load unbalance occurs and the byte hit rate is limited; that wastes more external bandwidth even if we solve the scalability problems with the p2p method.

## 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 *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 are 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 the block. According to this, all of blocks for a large sized object are distributed in the clients and the storage overhead for each client reduces and balances.

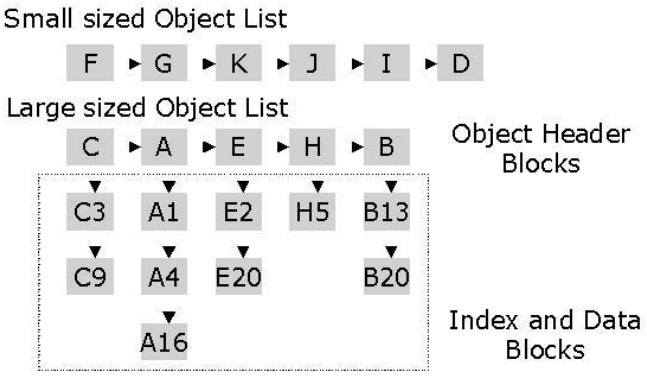



**Figure 2. SOL( Small sized Object List) and LOL( Large sized Object List )**

We use the index based allocation method to store large sized objects, because this method is simple ,cost-effective and easy to access randomly. First of all, we need the object header block which has the basic information about the large sized object, such as URL, size and modified time, and indirect pointers (IP), such as the single indirect IPs, the double indirect IP and the triple indirect IP. We do not use direct pointers in the object header block. In the general indexed method, the direct pointer is used to store small sized files to avoid making unnecessary index blocks. However, the size of the stored objects is enough large to neglect the overhead of index blocks. The home node for an large sized object stores this object header block instead of the object itself.

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

### 3.2 Storage policy

Figure 2 shows how the client or proxy stores the requested objects. We use two lists; the *Small sized Object List*(SOL) and the *Large sized Object List*(LOL) to store objects. The SOL is the simple linked lists of the small sized objects, because the small sized objects are stored by itself. The LOL is the linked lists of the object header blocks of large sized objects and each header has its block list which manages the index and data blocks of the large sized object

Proceedings of the Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/E-Learning on Telecommunications Workshop
0-7695-2388-9/05 $20.00 © 2005 **IEEE**

on this storage.

In the fully decentralized system where there is no central server, every client use both of the SOL and the LOL to store both type of objects. However, in the central server based systems, there is a powerful proxy server and clients just act as backup storages which store evicted objects from proxy server. Because the central server should handle the whole of requests and has very much connections with clients, it has to process a request as soon as possible to reduce the connection overhead. In this case, the role of the central server is different from the role of clients. The clients use both of the SOL and the LOL, otherwise the central server uses the SOL and the LOL without index and data blocks. According to this policy, the central server store more small sized objects and its hit rate increases more.

### 3.3 Replacement policy

All of web caches have the limited storage and they need the replacement algorithm that chooses which objects are evicted when the new objects is requested and new storage is needed. Generally, the web caches evict large sized objects as soon as possible to store more small sized objects. This make the hit rate higher, but the byte hit rate decreases. To prevent this degradation of the byte hit rate, we should give the large sized objects more chances for their hits.

In the clients, we give the large sized objects higher priority than the small sized objects to increase the availability of the large sized objects. In figure 2, if a cache needs space for new objects, it first evicts "D" which is a small sized object and if it needs more space, it evicts more small sized objects until the SOL is empty. If a cache needs space and the SOL is empty, we have to evict the blocks of large sized object. In this case, we evict "B20" which is the last data block of the least recently used large sized object. However in the proxy, because it does not store the index and data blocks of large sized objects and there are very many small sized objects, the replacement occurs at SOL only. This behavior increases the byte hit rate enormously.

Large sized objects are distributed in the clients very well, but missing just one block can spoil the whole of the large sized object. To prevent this block missing which spoils the large sized object, we use $n$ chance replacement policy. When a client evicts a block, it first regenerates the different block key by hashing the block and an optional suffix which is the random value. To move the block correctly, the client finds the object header block and the index block by URL and the block number of the evicted block and update the block pointer with the new block key. We permit this chance of moving blocks for one large sized object until the number of the chance is bigger than the threshold value, $n$. If an large sized object uses all of $n$ chance, whole of blocks of the object is removed. In our simulation, the $n$ value is 5.

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

**Table 1. Traces used in our simulation**

## 4 Evaluation

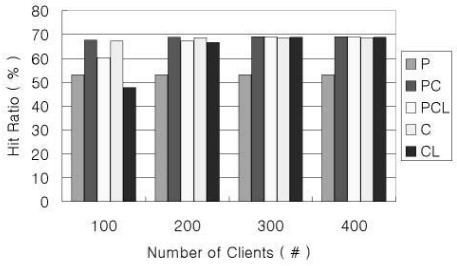
In this section, we present the results of extensive trace driven simulations that we have conducted to evaluate the performance of our caching policies. We design our p2p based web cache simulator to conduct the performance evaluation. We have assumed that we simulate the behavior of a proxy cache effectively. The proxy cache is error-free and does not store non-cachable 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 the 10 MBytes storage. We assume the large sized object is bigger than 1Mbytes and the size of blocks for them is 32KByte.

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.

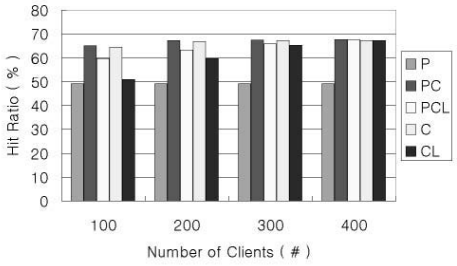
We compare five systems such as only proxy (P), the central server based system (PC), the central server based system with our policy (PCL), the fully distributed system (C) and the fully distributed system with our policy (CL). P, PC and C use the LRU policy for the object management.

### 4.1 Hit rate and Byte hit rate

Figure 3 and 4 show comparisons of the hit rate and the byte hit rate. In figure 3, we find that the p2p based web cache achieve higher hit rate than the normal web proxy cache. However, in figure 4, even if the hit rate of the fully distributed system is much higher than the only proxy, its byte hit rate is about half of 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. However, in fully distributed system with our policy, the hit rate is slightly lower than the

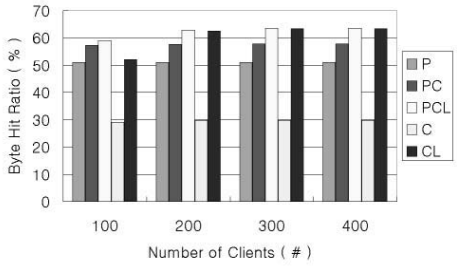



(a) Trace 1

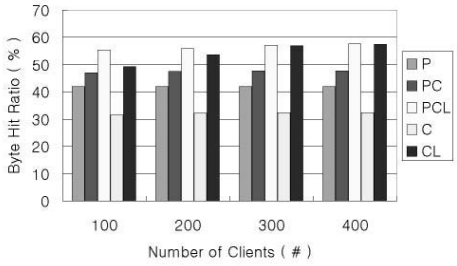



(b) Trace 2

**Figure 3. Hit rate comparison**




(a) Trace 1




(b) Trace 2

**Figure 4. Byte hit rate comparison**

fully distributed system, but the byte hit rate is much bigger. Additionally, if we use our policy, when the number of clients increases the hit rate and the byte hit rate increase remarkably, otherwise if we do not use the policy, there is little increment. According to this, if we handle the large sized object efficiently in p2p based web cache systems, we achieve not only the high hit rate but also the high byte hit rate.

### 4.2 Control Traffic

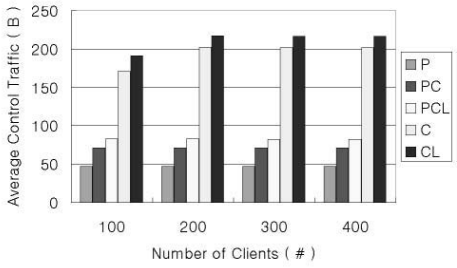
We divide large sized objects to many small blocks and we need more control traffic to gather the distributed blocks such as lookup messages of p2p substrate, requests for index blocks and requests for data blocks. In our simulation, we set the block size to 32 KB and set the size of a control message to 32 B. Figure 5 shows the comparison of the control traffic. According to our expectation, when we use our policy we use more control traffic.

We find that the p2p based web cache system uses more control traffic than normal proxy cache system. The fully decentralized system use much more control traffic than the central server based system, because the lookup for the p2p substrate need the multiple routing hops. However, though there is the additional control traffic, the p2p based web cache systems with our policy achieve the higher byte hit ratio. Consequently, the increment of the performance of p2p based web cache systems with our policy covers up the additional control traffic for the large sized objects or the lookups.

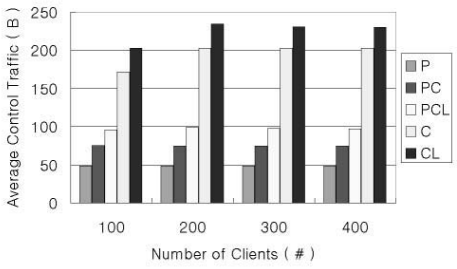
## 5 Conclusion

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 has 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. 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

Proceedings of the Advanced Industrial Conference on Telecommunications/Service Assurance with Partial and Intermittent Resources Conference/E-Learning on Telecommunications Workshop
0-7695-2388-9/05 $20.00 © 2005 IEEE




(a) Trace 1




(b) Trace 2

**Figure 5. Control traffic comparison**

policies is efficient and when we use any p2p based web cache systems with our policy, these systems achieve not only the high hit rate but also the high byte hit rate. Especially, the central server based system with our policy gets the best performance.

## References


[1] L. Xiao, X. Zhang, and Z. Xu. On Reliable and Scalable Peer-to-Peer Web Document Sharing. *In Proceedings of International Parallel and Distributed Processing Symposium,(IPDPS'02)*, April 2002.

[2] K.Kim and D.Park. Efficient and Scalable Client Clustering For Web Proxy Cache. *IEICE Transaction on Information and Systems*, E86-D(9), September 2003.

[3] J.Wang. A Survey of Web Caching Schemes for the Internet. *ACM Computer Communication Review*, October, 1999.

[4] S.Iyer, A.Rowstron, and P.Druschel. Squirrel: A decentralized peer-to-peer web cache. *In Proceedings of Principles of Distributed Computing'02*, 2002.

[5] I.Stoica, R.Morris, D.Karger, M.F.Kaashoek, and H.Balakrishnan. Chord: a scalable peer-to-peer lookup service for internet applications. *In Proceedings of ACM SIGCOMM 2001*, August 2001.

[6] A.Rowstron and P.Druschel. Pastry: scalable, decentralized object location and routing for large-scale peer-to-peer systems. *In Proceedings of the International Conference on Distributed Systems Platforms(Middleware)*, November 2001.

[7] 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.

[8] S.Ratnasamy, P.Francis, M.Handley, R.Karp, and S.Shenker. A scalable content-addressable network. *In Proceedings of ACM SIGCOMM 2001*, 2001.