Caching large files by using p2p based client-cluster for web proxy cache

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
email : kbkim@sslab.kaist.ac.kr and daeyeon@ee.kaist.ac.kr

ABSTRACT
Many web cache systems and policies have been proposed. These studies, however, consider large sized objects less useful than small sized objects for the performance and evict them as soon as possible. Even if this approach increases the hit rate, the byte hit rate decreases and more bandwidth is used to obtain large sized objects.

This paper suggests the web cache system which uses the client-cluster which is composed of the residual resources of clients as an exclusive storage for large sized objects. To manage the client-cluster, we use DHT based peer-to-peer lookup protocol which makes the system self-organizing, fault-tolerant, well-balanced and scalable. We manage the large sized object by the index based allocation method and balance the loads of all clients well. Consequently, this proposed system achieves not only the high hit rate but also the high byte hit rate, and reduces the outgoing traffics.

We examine the performance of the cache system via a trace driven simulation and demonstrate effective enhancement of the proxy cache performance.

KEY WORDS
peer-to-peer, clustering, web caching, replacement algorithm

1 Introduction

The recent increases in popularity of the Web has led to a considerable increase in the amount of Internet traffic. Especially, requests for large sized objects such as music and video files increase exponentially. As a result, web caching has become an increasingly important issue. Web caching aims to reduce network traffic, server load, and user-perceived retrieval delay by replication popular content on caches that are strategically placed within the network. Web caches are often deployed by institutions(corporations, universities, and ISPs) to reduce the traffic on the access links between the institution and its upstream ISP.

By caching requests for a group of users, a web proxy cache can quickly return objects previously accessed by other clients and reduce bandwidth consumption and network traffic. To maximize the cache performance, a proxy cache tries to handle as many requests as possible. However, the storage of a proxy cache is limited and it can not store all requested object in its storage. If a cache is full and needs space for new objects, it evicts the other objects which is not useful for cache performance; this is the replacement policy of a web proxy cache [9], [2], [1], [3], [8]. Generally, if a cache needs space, it evicts large sized objects first in compliance with the replacement policy. According to this behavior, a cache has more small sized objects and achieves higher hit rate and reduces more access links between the institution and its upstream ISP.

Using these replacement policies, large sized objects are not cached for long time and there is few chance for large sized objects to hit in a cache. According to this, though these replacement policies increase the hit rate, they reduce the byte hit rate which is the number of bytes that hit in the proxy cache as a percentage of the total number of bytes requested. This degradation of byte hit rate makes a proxy cache use more outgoing traffic even if few access links exist. To prevent this degradation, we should store large sized objects and maximize the chance to hit these objects. As a naive approach, a proxy cache increases its local storage. However this approach is only a temporary solution and is still affected by the general replacement policies. Therefore we need an exclusive storage for large sized objects. We can use CDN services to do this, but these services need expensive cost for hardwares and managements. Moreover, these approaches need too much administrative cost for the frequent variation of clients. For example, a growth in client population necessitates increasing the storages and updating the system information.

In this paper, we suggest a new web cache system which uses the residual resources of clients which are interconnected by high speed LANs. Basically, a web proxy cache stores small sized objects and resources of clients are used to store large sized objects. This separation of storages make a proxy cache storing more small sized objects, because it does not need to store any large sized objects, and large sized objects are stored in an exclusive storage which is supplied by clients. According to this behavior, a proxy cache keeps or improves its performance such as the hit rate

,the byte hit rate and the usage of outgoing bandwidth. Furthermore, the size of an exclusive storage increases as more clients use a proxy, and this reduces the administrative cost and makes the proxy cache more scalable.

The client-cluster is composed of the client's residual resources. Since clients join and leave dynamically, in order to use its storage efficiently, the client-cluster must be self-organizing and fault tolerant and the load of each client should be balanced. To cope with these requirements, we manage the client-cluster by using Distributed Hash Table (DHT) based peer-to-peer protocol. By using this protocol, all clients receive roughly the same load because the hash function balances load with high probability. Moreover, the proxy cache does not need to manage information about these clients and we save administrative cost.

This protocol matches an object with a client. However, we try to store large sized objects in client-cluster and it is hard and unfair for a client to store a large sized object. Therefore, we break up a large sized object into many small sized blocks and store these blocks to many clients by using the index based allocation method. All of blocks are distributed in the client-cluster and the storage overhead for each client reduces balances. When a proxy cache sends requests to a client-cluster and the requested objects are not stored in it, the proxy cache takes on additional latency. To prevent this latency, we use a cache summary with a Bloom filter, which determines whether the requested objects are in the client-cluster.

This paper is organized as follow. In section 2, we describe web caching and peer-to-peer lookup algorithm briefly. Section 3 introduces the detail of the client-cluster storing for large sized objects. The simulation environment and the performance evaluation are given in section 4. Finally, we conclude in section 5.

2 Background

2.1 Web caching and replacement policy

The basic operation of the web caching is simple. Web browsers generate HTTP GET requests for Internet objects such as HTML pages, images, mp3 files, etc. These are serviced from a local web browser cache, web proxy caches, or an original content server - depending on which cache contains a copy of the object. If a cache closer to the client has a copy of the requested object, we reduce more bandwidth consumption and decrease more network traffic. Hence, the cache hit rate and byte hit rate should be maximized and the miss penalty, which is the cost when a miss occurs, should be minimized when designing a web cache system.

If a web cache has the infinite storage, there is no problem for caching objects and a web cache achieves maximum of hit rate and byte hit rate. A web cache, however, has the size-limited storage and if a cache needs space for new objects, it evicts the other cached objects which is not useful for cache performance. In this case, the policy of selecting object is the replacement policy.

Many replacement policies are proposed and generally evict large sized object first for new objects and some policies even give up storing large sized objects [9], [2], [1], [3], [8]. Because of these policies, the web cache stores more objects ,increases the hit rate and decreases the number of access links between the institution and its upstream ISP. However, because these policies evict large sized objects early, the large sized objects are not cached for long time and there is few chance for the large sized objects to hit in the cache. Although the web cache can achieve high hit rate with these policies, it can not achieve high byte hit rate and wastes more upstream bandwidth for retrieving large sized objects. If the requests of large sized objects increase, this degradation appears remarkably.

2.2 Peer-to-Peer Lookup

Peer-to-peer systems are distributed systems without any centralized control or hierarchical organization, where the software running at each node is equivalent in functionality; this includes redundant storage, selection of nearby servers, anonymity, search, and hierarchical naming. Among these features, lookup for a data is an essential functionality for peer-to-peer systems.

A number of peer-to-peer lookup protocols have been recently proposed, including Pastry, Chord, CAN and Tapestry [7], [6], [10], [5]. In a self-organizing and decentralized manner, these protocols provide a DHT (distributed hash-table) that reliably maps a given object key to a unique live node in the network. Because DHT is made by a hash function that balances load with high probability, each live node has the same responsibility for data storage and query load. 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. Typically, the length of routing is about O(log n), where n is the number of nodes. According to these properties, peer-to-peer systems balance storage and query load, transparently tolerate node failures and provide efficient routing of queries.

3 Proposed Idea

3.1 Overview

As we described in the previous section, a web proxy cache evicts large sized objects first to get free space which is used to store a new cached object and this feature reduces the cache performance, especially the byte hit rate. According to this, the large sized object is the main obstacle of the cache performance. To solve this problem, we exploit the residual resources of clients for a proxy cache. That is, any client that wants to use the cache provides small resources

to the cache and the proxy cache uses these additional resources to maintain the proxy cache system. This feature makes the proxy cache resourceful and scalable.

We use these resources as an exclusive storage for large sized objects. Generally a web proxy cache stores all of requested objects in the local storage, but in our system, a cache only stores small sized objects and large sized objects are stored in the exclusive storage which is distributed among the client cluster. The behavior of the cache depends on the size of the requested object. If the size is small, the cache deals with the object in the local cache, otherwise, it turns over the object to the exclusive storage. According to this behavior, the proxy cache stores more small sized objects and the hit rate increases more. Additionally, we can achieve the high byte hit rate and save more outgoing bandwidth.

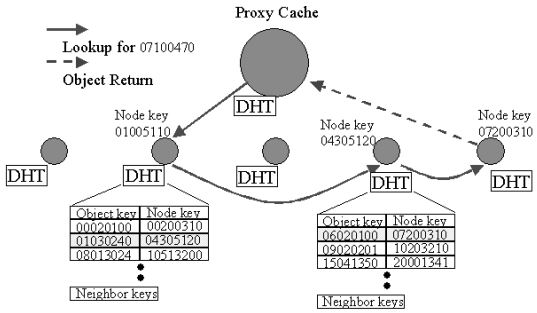


Figure 1. Basic lookup operation in the client-cluster. In this figure, total hop count is 3 for an object.

3.2 Client-Cluster Management

In our scheme, a proxy cache uses the resources of clients that are in the same network. Generally, if a peer wants to use other peers, it should have information about those. This approach is available when the other peers are reliable and available. However, the client membership is very large and changes dynamically. If the proxy cache manages the states of all clients, too much overhead is created to manage the client information and complex problems such as fault-tolerance, consistency and scalability arise. In consideration of these issues, we establish the proxy cache such that it has no information for the clients and the client-cluster manages itself.

We design the client-cluster by using DHT(distributed hash table) based peer-to-peer protocol [6], [7]. To use this protocol, each client needs an application whose name is *Station*. A Station is not a browser or a browser cache, but a management program to provide clients' resources for a proxy cache. A client can not use resources of a Station directly, while a proxy cache sends requests issued from clients to Stations in order to use resources of a client-cluster. When a Station receives requests from a proxy cache, it forwards requests to another Station or checks whether it has the requested objects. Each Station has a unique node key and a DHT. The unique node key is generated by computing the SHA-1 hash of the client identifier, such as an ip address or an ethernet address, and the object key is obtained by computing the SHA-1 of the corresponding URL. The DHT describes the mapping of the object keys to responsible live node keys for efficient routing of request queries. It is similar to a routing table in a network router. A Station uses this table with the key of the requested object to forward the request to the next Station. Additionally, the DHT of a Station has the keys of *neighbor Stations* which are numerically close to the Station, like the leaf nodes in PASTY or the successor list in CHORD.

The basic operation of the lookup in a client-cluster is shown in figure 1. When a proxy cache sends a request query to one Station of a client-cluster, the Station gets the object key of the requested object and selects the next Station according to the DHT and the object key. Finally, the *home Station*, which is a Station having the numerically closest node key to the requested object key among all currently live nodes, receives the request and checks whether it has the object in local cache. If a hit occurs, the home Station returns the object to the proxy cache; otherwise, it only returns a null object. In figure 1, the node whose key is 07200310 is the home Station for the object whose key is 07100470. The cost of this operation is typically O(log n), where n is the total number of Stations. If 1000 Stations exist, the cost of lookup is about 3, and if 100000 Stations, the cost is about 5. Since the RTT for any server in the Internet from one client is 10 or 100 times bigger than that for another client in the same network, we reduce the latency for an object by 2 or 20 times when we obtain the object in the client-cluster.

The client-cluster can cope with frequent variations in client membership by using this protocol. Though the clients dynamically join and leave, the lazy update for managing the small information of the membership changes does not spoil the lookup operation of this protocol. When a Station joins the client-cluster, it sends a join message to any one Station in the client-cluster and gets new DHT and other Stations to update their DHT for the new Station lazily. On the other hand, when a Station leaves or fails, other Stations which have a DHT mapping with the departing Station detect the failure of it lazily and repair their DHT. According to this feature, the client-cluster is self-organizing and fault-tolerant.

All Stations have roughly the same amount of objects, because the DHT used for the lookup operation provides a degree of natural load balance. Moreover, the object range, which is managed by one Station, is determined by the number of live nodes. That is, if there are few live nodes, the object range is large; otherwise, it is small. According to this, when the client membership changes, the object range is resized automatically and the home Stations for every object are changed implicitly.

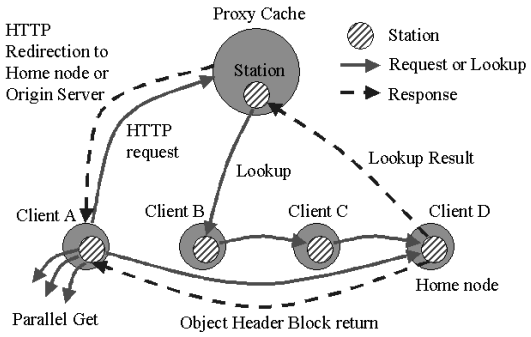


Figure 2. Operation for the Large Sized File

Traces	Trace 1	Trace 2
Measuring day	2001.10.08	2001.10.09
Requests size	9.02 GB	11.66 GB
Objects size	3.48 GB	1.38 GB
Request number	699280	698871
Object number	215427	224104
Hit Rate	69.19%	67.93%
Byte Hit Rate	63.60%	57.79%

Table 1. Traces used in our simulation

3.3 Large File Backup

In our system, large sized objects are stored in the client-cluster. Basically, the client-cluster stores the object in the corresponding node which has numerically closest node key to the object key. However, each node in the client-cluster supports the residual resource which are not used by a node 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 nodes. Each block has the block key which is obtained by hashing the block itself and the home node that has numerically closest node key to the block key stores the block. According to this, all of blocks for a large sized object are distributed in the client-cluster and the storage overhead for each client reduces and balances.

We use the index based allocation method to store large sized objects. That is, we use the index block to access the data block. Each pointers to the blocks includes the block key which is used to find the home node. First of all, we make the object header block which has the basic information about the large sized object, such as the object key, URL, size and IMS information, and indirect pointers, such as the single indirect, the double indirect and the triple indirect. We do not use direct pointers in the object header block, because the size of the object is larger than 1Mbyte. This header block is stored at the home node for the large sized object, that is, we store the header block instead of the object itself at the home node.

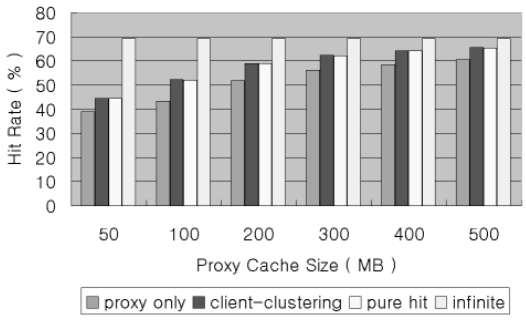
The basic operation for requesting the large sized object is shown in figure 2. Client A wants to get a large sized object and sends a request to the proxy cache. The proxy first checks its local storage, because it has no idea about the size of the requested object. However, the requested object is a large sized object, and the proxy can not find it in its storage and sends a lookup message to the client-cluster. In this figure, Client B gets this lookup message first and forwards it to Client C, and finally this message arrives at Client D, the home node for the large sized object. This home node returns a lookup result which indicates whether the node has the object or not. According to this result, the proxy chooses the destination of the HTTP redirection and returns it to the client. That is, if the home node has the object header, Client A sends request to the Client D and gets it, otherwise, Client A gets the large sized object from the original server. When a client gets a large sized object from the original server, it takes a charge of storing this object. The client makes the object header block, index blocks and real data block and distributes these blocks into the client-cluster.

4 Evaluation

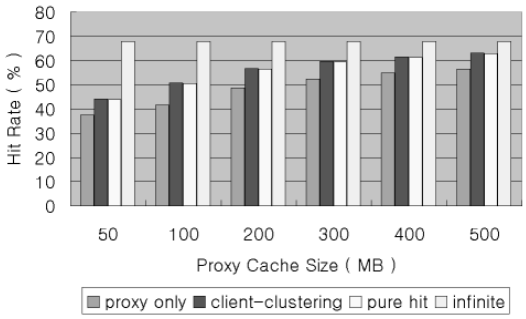
In this section, we present the results of extensive trace driven simulations that we have conducted to evaluate the performance of our system. We design our proxy cache simulator to conduct the performance evaluation. This simulator illustrates the behavior of a proxy cache and client-cluster. We have 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, control data and etc. We also assume that there are not any problems in the network, such as congestions and overflow buffers. The size of a proxy cache is in the range from 0.5MByte to 500MBytes. Each client uses one Stations which has the storage, about 40MBytes. The client-cluster stores large sized objects whose size is bigger than 1MBytes.

4.1 Traces used

In our trace-driven simulations we use traces from KAIST, which uses a class B ip address for the network. The trace from the proxy cache in KAIST contains over 3.4 million requests in a single day. We have run our simulations with traces from this proxy cache since October, 2001. 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. However, our simulations assume limited cache storage and ratios including hit rate and byte hit rate can not be higher than *infinite-hit rate* and *infinite-byte hit rate*, which are the hit rate and the byte hit rate when the infinite storage is used.

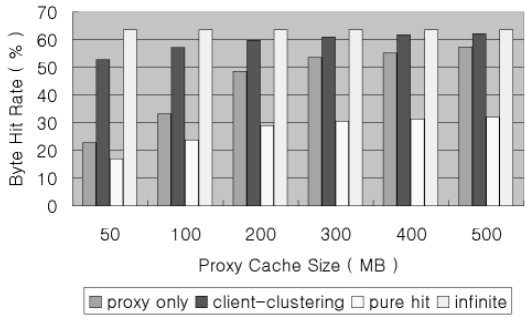


(a) Trace 1

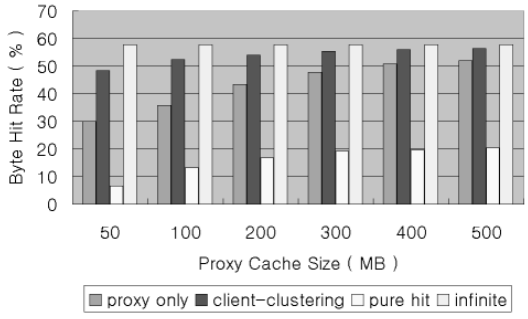


(b) Trace 2

Figure 3. Hit rate comparison between only proxy cache and client-clustering



(a) Trace 1



(b) Trace 2

Figure 4. Byte Hit rate comparison between only proxy cache and client-clustering

4.2 Hit Rate and Byte Hit Rate

Figure 3 and 4 show comparisons of the hit rate and the byte hit rate. By the hit rate, we mean the number of requests that hit in the proxy cache as a percentage of total requests. A higher the hit rate means the proxy cache can handle more requests and the original server must deal with proportionally lighter load of requests. The byte hit rate is the number of bytes that hit in the proxy cache as a percentage of total number of bytes requested. A higher byte hit rate results in a greater decrease in network traffic on the server side.

In the figures, *proxy only* means using only a proxy cache and *client-clustering* means using the client-cluster to store large sized objects. When we use the client-cluster, a hit does not only occurs at the local storage of a proxy cache but also at the client-cluster. To separate the two types of hits, we use *pure hit* that indicates the hit or byte hit rate which is obtained at the local storage of a proxy cache. *infinite* is the rate when a proxy cache has infinite storage.

The figure 3 shows the effect of using a client-cluster to the hit rate. When we use a client-cluster to store large sized objects, the hit rate increases by about 10% without any relation to the proxy cache size. Moreover, in every case, the hit rate of *pure hit* is very similar to the hit rate of *client-clustering*. Numerically the difference of these two value is only about 0.2%. Most of hits occur in a local storage of a proxy cache and few hits (about 0.2%) occur in a client-cluster. When using a client-cluster, the increasing effect of the hit rate is due to the increase of the hit rate in the local storage of a proxy cache. That is, a proxy cache does not handle large sized objects any more and it can store more small sized objects, and the hit rate of the whole system increases.

From the result of the hit rate, when we use a client-cluster few hits occur in the client-cluster for large sized objects. However, these hits bring very big byte hits because of the large size of requested objects. In figure 4, using a client-cluster, the byte hit rate increases remarkably and it achieves similar value to the infinite-byte hit rate without any relation to a proxy cache size. Differently from the hit rate, the byte hit rate of *pure hit* is smaller than the rate of *client-clustering* or the rate of *proxy only*. Especially, in the result of trace 2, the byte hit rate which is obtained from the local storage of a proxy cache is smaller than a

Client	100	200	300
Mean Size	20967KB	10586KB	7081KB
Max Size	21659KB	11141KB	7372KB
Dev	1.65	2.48	1.98
Mean Req.	3662	1842	1230
Max Req.	3899	2001	1375
Dev	2.12	2.99	4.1
Mean Byte Req.	120025KB	60374KB	40327 KB
Max Byte Req.	127762KB	65568KB	45056KB
Dev	2.1	2.98	4.09

Table 2. Summary of client loads for Trace 1 with the 200MB proxy

third of the byte hit rate from a client-cluster. According to these, though a proxy cache achieves high hit rate, small sized objects in the proxy cause the low byte hit rate. We can cope with this weak point to store large sized objects in a client-cluster. Consequently, to use a client-cluster which is a exclusive storage for large sized objects, we achieve not only the high hit rate, but also the high byte hit rate. We can preserve and improve the performance of a proxy cache without the expensive management cost.

4.3 Client Load

We examine the client loads, which include the request number, storage size, stored object and etc, to verify that the client-cluster balances the storage and requested queries. Table 2 shows a summary of the supported storage size, the requested number and the requested byte of clients. According to this table, in order to store the whole of large sized objects in the client-cluster, each client should supply about 10MB – 20MB storages to the proxy cache and handle about 1000 – 4000 requests for a day. These loads are enough for any client to handle in these days and if a number of client is 300 or more, these loads are negligible. The deviation value of each metric is less than 4% and each client receives roughly same load. Furthermore, when the client number increases the load of each client decreases. We are sure of the scalability of our system according to these results.

5 Conclusion

In this paper, we propose and evaluate the peer-to-peer client-cluster which is used as an exclusive storage for a web proxy cache. The proxy cache with this client-cluster achieves not only high hit rate but also high byte hit rate. This behavior reduces the outgoing traffic which occurs over the congested links and improves the performance of the connections to the outside world. Moreover, the client-cluster supported by the clients which use the proxy cache is highly scalable and the proxy just needs low administrative cost. Even if the clients take the load, this load has been verified on a range of real workloads to be low and well balanced. Additionally, if we use this client-cluster not only as an exclusive storage of a proxy cache but also as a backup storage of a proxy cache, we achieve the high value for both of the hit rate and the byte hit rate, which is similar to the value when we use the infinite cache. This is our ongoing work.

References

[1] C. Aggarwal, H. L. Wolf, and P. S. Yu. Caching on the world wide web. *IEEE Transaction on Knowledge and data Engineering*, 11(1), January 1999.

[2] P. Cao and S. Irani. Cost-aware www proxy caching algorithms. *In Proceedings of Usenix symposium on Internet Technology and Systems*, December 1997.

[3] K. Kim and D. Park. Least popularity per byte replacement algorithm for a proxy cache. *In Proceedings of IPADS 2001*, June 2001.

[4] 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.

[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. *In Proceedings of ACM SIGCOMM 2001*, 2000.

[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] 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.

[8] J. Wang. A survey of web caching schemes for the internet. *ACM Computer Communication Review*, 29(5):36–46, October 1999.

[9] S. Williams, M. Abrams, 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.

[10] 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.