Subway : Peer-To-Peer Clustering of Clients for Web Proxy

Kyungbaek Kim and Daeyeon Park

Department of Electrical Engineering & Computer Science,
Division of Electrical Engineering,
Korea Advanced Institute of Science and Technology (KAIST), Korea

Abstract

Many cooperated web cache systems and protocols have been proposed. But, these systems need the expensive resources, such as core-link bandwidth and proxy cpu or storage, and need administrative cost to achive enough performance when the client population increases. This paper suggests a self-organizable caching system, called Subway, which is scalable and fault tolerant, with a distributed peer-to-peer storage as a backup storage. Subway reduces consumption of expensive resources by using resources of participant clients, which use the service of a central proxy. We examine the performance of Subway via trace driven simulations.

key words : Peer-To-Peer, Clustering, Web Caching, Cooperated Caching

1. Introduction

The recent increase in popularity of the Web has led to a considerable increase in the amount of traffic over the Internet. As a result, the Web has now become one of the primary bottlenecks to network performance. Consequently, web caching has become an increasingly important topic. Web caching aims to reduce network traffic, server load, and user-perceived retrieval delay by replicating popular contents on caches that are strategically placed within the network. Browser caches reside in the clients' desktop, and proxy caches are deployed on dedicated machines at the boundary of corporate network and at Internet service providers.

By caching requests for a group of users, a proxy cache can quickly return documents previously accessed by other clients. Just using one proxy cache has the limitation of the enhancement of its performance, because the hit rate of the proxy is limited by the cache storage and the size of the client population. Multiple proxies should cooperate with each other in order to increase total client population, improve hit ratios, and reduce document- access latency. That is the cooperative caching.

Several cooperative-caching techniques has been proposed [11], [10], [1], [2]. However, these studies need high bandwidth, expensive infrastructure and high administrative cost. ICP-based cooperative caches communicate with other caches which are connected by busy core-links, to find requested objects in other caches. Even if requested objects are not in other caches, they spend bandwidth of core-links to obtain requested objects from others. Some cooperative caches use a proxy cluster, as a single big cache, to be overprovisioned in order to handle bursty peak loads. A growth in client population causes scalability issues, leading to a need for increasing the cluster size.

In this paper, we suggest a new web cache architecture which uses the resources of clients, called *Subway*. Subway is the self-organizing caching system with a distributed storage as a backup storage and it is scalable and fault tolerant. Subway is composed of two parts, a central proxy cache and a client-cluster. The central cache is a simple proxy cache and the client-cluster is a distributed storage. The client-cluster is managed by *Station* application which uses a DHT-based peer-to-peer protocol. This client-cluster stores evicted objects from the central proxy. To communicate between the central cache and the client-cluster Subway uses *Backward-ICP* which propagates form central cache to client cluster. Subway thus has the advantage of reducing consumption of core-bandwidth and administrative cost. Moreover, Subway enhances the overall performance of the proxy cache; hit rate, byte hit rate, and bandwidth.

This paper is organized as follow. In section 2, we describe cooperated web caching and peer-to-peer lookup algorithm brifely. Section 3 introduce the detail of Subway. The simulation environment and the performance evaluation are given in section 4. We mention other related works in section 5. Finally, we conclude in section 6.

2. Background

2.1. Cooperated Web Caching

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 the local web browser caches, web proxy caches, or original content servers - depending on which contain copies of requested objects. If the requested objects are in web browser caches, clients don't need to send GET requests. If not, clients send GET requests to web proxy caches, and so on. When closer cache to clients has the copy of the requested object, we can reduce more bandwidth consumption and more network traffic. The cache hit rate should be maximized and the cost of a cache miss should be minimized when designing a web caching system.

The performance of a web caching system depends on the size of its client comunity. When the user community increases, the probability of that a cached document will soon be requested again increases. Hence, caches sharing mutual trust may assist each other to increase the hit rate. A caching architecture should provide the paradigm for proxies to cooperate efficiently with each other. One approach to coordinate caches in the same system is to set up a caching hierarchy. With hierarchical caching, caches are placed at multiple levels of the network. The another approach is a distributed caching system, where there are only caches at the bottom level and there are no other intermediate cache levels. In a hybrid scheme, caches may cooperate with other caches at the same level or at a higher level using distributed caching.

Internet Cache Protocol(ICP) [1] is a typical cooperating protocol for a proxy to communicate with other proxies. The ICP operation is simple. If a requested object is not founded in a local proxy, it sends ICP queries to neighbor proxies; sibiling proxies and parent proxies. The neighbor proxies receive the queries and send ICP replies. The local proxy receives the ICP replies and decides where to forward the request. However, ICP wastes expensive resources; core bandwidth and cache storage. Even if the neighbor caches don't have the requested object, ICP uses the core-links, which is the inter-proxy links and very busy for many clients to use many different Internet applications. Another protocol for distributed caching is the Cache Array Routing Protocol (CARP) [2], which divides the URL-space among and array of loosely coupled caches and lets each cache store only the documents whose URL are hashed to it. Because of this feature, every request is hashed and forwarded to selected cache node. To do this, clients must know the cache array information and the hash function. According to these, the management of CARP is difficult and there are the other issues; such as load balancing, fault tolerance and etc.

Another problem of CARP, same of ICP, is scalability. Large corporate networks often employ a cluster of machines, which has to usually be overprovisioned to handle bursty peak loads. A growth in user population cause leading to a need for hardware upgrades. This scalability issue cannot be solved by ICP or CARP.

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 equibalent in functionality; redundant stoarage, selection of nearby servers, anonymity, search, and hierarchical naming. Despite this rich set of features, the core operation in most peer-to-peer systems is efficient location of data.

A number of peer-to-peer lookup protocols have been recently proposed, such as Pastry [8], Chord [9], CAN and Tapestry. In a self-organizing manner, these protocols provide a distributed hash-table (DHT) that reliably maps a given object key to a unique live node in the network. Because DHT is made by hash function, each live node has same responsibility of data storage and query load. If a node wants to find an item, it just sends a query with the object key to the selected node by the DHT. The length of routing is about

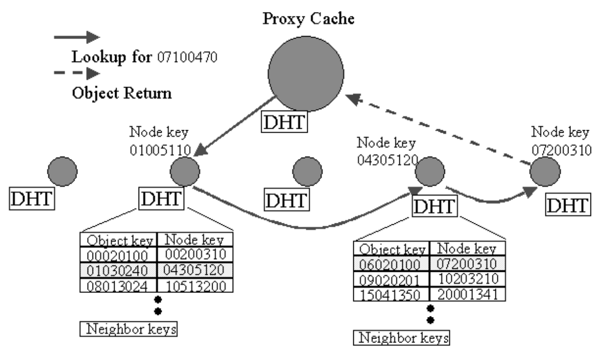


Figure 1. Overview of the Subway system

O(log n), where n is the number of nodes. According to these properties, a system which uses peer-to-peer lookup to locate data items, is scalable and does not worry about node join or failure. Moreover, the system balances data storage and query load, and provides efficient routing of queries.

3. Subway Architecture

3.1. Overview of Subway

Subway is a self-organizing caching system with a distributed storage as a backup storage and it is scalable and fault tolerant, caused from the properties of DHT based peer-to-peer system. It reduces core-bandwidth usage and administrative cost. Subway is composed of two parts, a central proxy cache and a client-cluster. The central cache is an ordinary proxy cache to store requested objects. The client-cluster whose participants are clients in the same AS, is a decentralized peer-to-peer storage to store evicted objects from the central cache, as a secondary storage for the central cache. Each participated client has a web browser and a Station program which is used for Subway to use storage of clients.

Figure 1 shows the lookup operation of Subway system. When a client sends a GET request to its web browser, it lookups it web browser cache first. If the requested object is not in the browser cache, the browser sends the request to a central cache and it checks wherether the object is in it. If the central cache does not have the object, it sends Backward-ICP Lookup request to the client-cluster. This Lookup message is routed by a peer-to -peer protocol; in this paper we use DHT based peer-to-peer lookup protocol. According to this behavior, Subway reduces usage of expensive infrastructures, such as core-link bandwidth, proxy-storage, and administrative cost.

3.2. Central Cache

The central cache acts as a simple proxy cache. It locates at boundaries of corporate networks or at Internet service providers and is managed by administrators of networks or ISPs. As described before, a cache has limited storage and evicts objects which will not be used by clients to achive high performance; hit rate, byte hit rate, and latency. In a normal proxy cache, evicted objects are dropped, but in Subway, if a central cache evicts objects, the central cache sends evicted objects to the client-cluster to store them.

The basic operation of the central cache is as follow; The central cache receives a request query from a client which is in the same network where it locates. If a cache hit occur, the requested object is returned to the client, otherwise, the central cache sends a Lookup message to the client-cluster. If the requested object is in it, the central cache obtains the object , saves the object and returns the object to the client. Otherwise, the central cache sends a request message to an original server. This behavior of central cache decreases the probability of sending requests to original server or cooperated caches, reduces core-bandwidth consumption, and increases performance of central cache.

3.3. Client-Cluster

The main function of the client-cluster is backup of the central cache by supporting resources of each client; storage, cpu power, bandwidth, etc. Each participant client needs another application whose name is Station, not a web browser, to support this function. A Station receives requests from the central cache and performs proper jobs to assist it; in this cache system, a Station checks whether there are requested objects in local cache or forwards requests to other Station. These Stations are managed by a DHT-based peer-to-peer protocol. Each Station has the unique node key and a DHT, that maps object keys to live node keys for routing request queries. Like Figure 1, if a Station receives a request, it gets the object key of requested object and selects a next Station by using the DHT and the object key. A seleted Station checks whether the requested object is in local cache. If a hit occurs, the Station returns the object to the central cache, otherwise, just returns null and the central cache sends requests to other servers which may have object copies.

This client-cluster is self-organizable, scalable and fault tolerant, caused from the properties of peer-to-peer protocols. When the number of clients increases, the central cache get more storage to store evicted objects and achieve high performance; hit rate and byte hit rate. When any client joins the network, this client sends a join message to any other client and is involved in the client-cluster by assigning the node key and a DHT. When any client leaves the client-cluster or fails, other clients detect the failure of the client and repair their DHT. On both of situations, all participant clients don't exchange the whole of the saved objects but just repair the DHT, because the objects in the client-cluster are just backup objects which has less popularity and locality than the objects which are in the central cache. According to these features, Subway reduces administrative cost and proposes the way to manage a web caching system efficiently and easily.

3.4. Backward-ICP

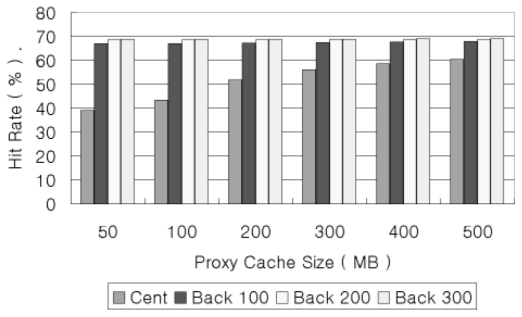
Because there are two part, we need a protocol to communicate with each others; that is the Backward-ICP. Communication between the central cache and the client-cluster is similar to that between the proxy caches by using ICP. However, the Backward-ICP uses local area networks rather than core-links and its direction of propagation is reverse to ICP. When a cache miss occurs in the central cache , it sends a Lookup message to the client-cluster to find and obtain objects without existance of the object.

This approach has a simple disadvantage; the big miss penalty at the client-cluster. For this, the central cache has the bloom filter which is used to check whether there are requested objects in the client-cluster or not.

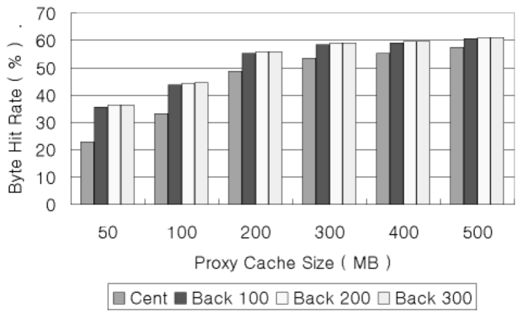
4. Performance Evaluation

4.1. Simulation Setup

We perform trace-driven simulation for performance evaluation of Subway. The trace from a proxy cache in KAIST contains over 3.4million requests in a single day. When the number of requests of the trace is 0.7million, the number of objects is 0.48 million, the total object size is 5.7GB, the infinite-hit rate is 68.78% and infinite-byte hit rate is 63.66%.



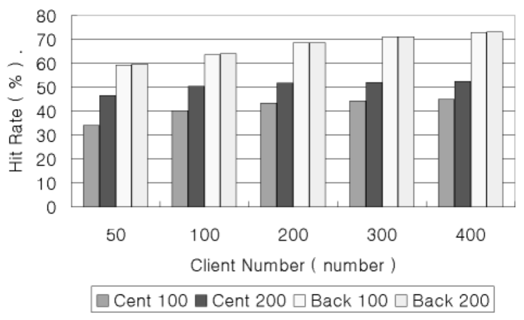
(a) Hit rate



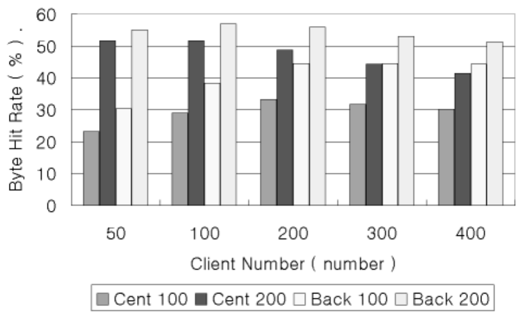
(b) Byte Hit Rate

Figure 2. performance comparison between only proxy cache(cent) and Subway(back n)

We assume the central cache is error-free and does not cache the non cachable objects; dynamic data, large size data, control data, etc. Every client has the same storage, 10MB, for Station not web browser cache and communicates each other by a peer-to-peer lookup algorithm; we use Chord. In this simulation, we don't consider the effects of browser caches. We use the bloom filter at the central cache to check whether the requested objects are in the client-cluster or not. By using this filter, we achieve near 100% hit rate of the client-cluster and we reduce the local-bandwidth consumption.



(a) Hit rate



(b) Byte Hit Rate

Figure 3. Hit rate comparison between only proxy cache(cent n) and Subway(back n) with variable client size

4.2. Hit Rate and Byte Hit Rate

Figure 2 shows the comparision of the hit rate and the byte hit rate. In this figure, cent means using only proxy cache and back-n means using Subway with n hundred Stations(clients). The hit rate of only central cache is greatly affected by the cache size, but the hit rate of Subway achieves nearly infinite-hit rate without any relationship to the central cache size. This means Subway reduces core-bandwidth consumption and administrative cost, because the cache performance is not affected by the configuration of central cache.

In the byte hit rate, we can find the similar result with the hit rate. However, in this case, Subway does not get infinite-byte hit rate at the point of small sized proxy cache. The reason of this result is that the large size object whose size is bigger than the whole size of proxy cache storage and the size of Station storage, 10MB. If the Station storage is large, the byte hit rate increases.

Client #	Mean Req.	Max Req.	Dev.
100	1024	1369	2.2
200	602	733	2.4
300	401	510	2.5
	Mean Byte Req.	Max. Byte Req.	Dev.
100	13422KB	316805KB	11.1
200	6711KB	315158KB	12.1
300	4474KB	314197KB	12.9

Table 1. Summary of Client Loads for Trace 1 with the 200MB proxy

4.3. Client Size Effect

In this section, we show that Subway is scalable. We assume every 100 client requests 0.35million requests and simulates with variable client number. The result is shown in Figure 3. In the result, the hit rate when only the central cache is used does not increase markedly. However, when we use Subway, the hit rate increases by 30-40% over that when only the central cache is used. Additionally, as the client number increases, the hit rate increases accordingly. For byte hit rate, Subway still has higher value than the central cache, but the effect of the large size object also exist. According to this result, using only central cache has the limitation with any administrative cost, but using Subway is scalable without other managements.

4.4. Client Load

We check the client loads, which are request number, stored size, stored objects, hit rate, etc, to verify the client-cluster balance the storage and the request queries. Table 1 shows the summary of request number and the size of the requested object. In the result, every client receives roughly same load, and when the client number increases the load of each client decreases. The reason of this result is the properties of DHT-base peer-to-peer protocols. In the byte request, we find the effect of the large size object again and we are convinced that is the only reason of the performance degradation in the byte hit rate.

5. Related Works

Cooperative web caching is found to be useful in improving hit rate in a group of small organizations. There are many forms such as hierarchical web caching [1], hash-based clustering [2] and directory-based scheme [5] and etc. These are efficient but need high resources and high administrative cost to improve the utility and the scalability of the caching system. On the other hand, in our scheme, the client-cluster is composed of residual resources of clients and the scalability is the natural characteristic of the client-cluster.

Many peer-to-peer applications such as Napster, Kazza and Morpheus, become popular. Additionally, large area filesystems using peer-to-peer are proposed such as PAST [4], CFS [3] and OceanStore [7]. The target of these systems is the wide area network and they have no concern about the characteristics of web objects such as size, popularity and update frequency.

A similar proposal for our approach appeared in Squirrel [6] is the decentralized web browser cache. Squirrel fully distributes the web caches storage among the browser caches of clients. According to this fact, when the availability of clients is asymmetric, some of bad clients decrease the total performance of the Squirrel network. Additionally, all contents are distributed and it is hard to manage the objects according to the characteristics of web objects. In our scheme, an web object is assigned to the proxy cache or the client-cluster according to the popularity of the object, which optimizes the overall performance of the proxy cache.

6. Conclusions

We propose the new web cache system, Subway, which is scalable , self-organizable, fault-tolerant and easily managable. The performance of Subway is not affected by the variation of the client population and the configuration of the central proxy cache. Moreover, Subway uses efficiently the expensive resources, such as core-bandwidth, proxy resources, etc.

In this paper, Subway is used for a web caching system. However, we can extend the usage of Subway to other proxy systems. When a proxy performs big jobs, such as encoding/decoding or complex calculations, for many clients, it uses the resources of the clients to do jobs for them. Moreover, we can improve the performance of the Subway. The first one is removing effect of large size objects and the second one is the efficient replication of objects in the client-cluster. These are our ongoing works.

References

[1] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. A hierarchical internet object cache. *In proceedings of the 1996 Usenix Technical Conference*, January 1996.

[2] J. Cohen, N. phadnis, V. valloppillil, and K. W. Ross. Cache array routing protocol v 1.0. *http://www.ietf.org/internet-drafts/draft-vinod-carp-v1-03.txt*, September 1997.

[3] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with cfs. *In Proceedings of ACM SOSP 2001*, 2001.

[4] P. Druschel and A. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. *In Proceedings of HotOS VIII*, 2001.

[5] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. *In Proceedings of ACM SIGCOMM 98*, 1998.

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

[7] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gumadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. *In Proceedings of ACM ASPLOS 2000*, November 2000.

[8] 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.

[9] 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*, August 2001.

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

[11] 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. *In Proceedings of the 17th ACM Symposium on Operating Systems Principles*, December 1999.