The Content-Aware Caching for Cooperative Transcoding Proxies

Byoung-Jip Kim, 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 Kusong-dong Yusong-gu, Taejon, 305-701, Korea
{bjkim, kbkim}@sslab.kaist.ac.kr
daeyeon@ee.kaist.ac.kr

Abstract. The Web is rapidly increasing its reach beyond the desktop to various devices and the transcoding proxy is appeared to support web services efficiently. Recently, the cooperative transcoding proxy architecture is proposed to improve the system performance to cope with the scalability problem of a stand-alone transcoding proxy. However, because of the multiple versions, the communication protocol of the cooperative caches is very complex and causes additional delay to find best version for a requested object.
In this paper, we propose efficient cooperative transcoding proxy architecture which uses the content-aware caching. The main purpose of the proposed system is simplifying the communication protocol of cooperative caches. We associates a home proxy for each URL and the home proxy is responsible for transcoding and maintaining multiple version of an URL. This mechanism reduces the amount of messages exchanged and communication latency involved. To prevent the hot-spot problem, each proxy cache has the private cache which stores the recently requested objects. We examine the performance of the proposed system by using trace based simulation with Simjava and show the effective enhancement of the cooerative transcoding proxy system.

1 Introduction

In recent years, the technologies of the network and the computer have developed enormously and the diverse devices such as PDAs, mobile phones, TVs and etc which are connected to the network with various ways such as wired or wireless interfaces. These diverse devices have been able to use the web contents, but some clients can not use the web contents directly because their capabilities differ from those of the web content provider's expectation. For these clients, the content adaptation, called the *transcoding*, is needed. This transcoding transforms the size, quality, presentation style, and etc of the web resources to meet the capabilities of the clients. The main features of the transcoding can be summarized with two. First is that multiple versions exist for the same web content due to the diverse client demand. Second is that the transcoding is a very computational task. These two features of the transcoding bring many issues to design a transcoding system.

The existing approaches of the transcoding system can be classified into three categories broadly, depending on the entity that performs the transcoding process: *client-based*, *server-based*, and *intermediary-based* approaches. In the client-based approaches, the transcoding is performed in client devices and the transcoder has direct access to the capabilities of the various devices. However, these approaches are extremely expensive due to the limited connection bandwidth and computing power of clients. Conversely, in the server-based approaches, the content server transforms objects into multiple versions on online or offline. These approaches preserve the original semantic of the content and reduce the transcoding latency during the time between the client request and the server response. However, keeping the multiple versions of an object wastes too much storage and the content providers actually can not provide all kind of versions of contents for the diverse clients. In the intermediary-based approaches, edge servers or proxy servers can transform the requested object into a proper version for the capability of the client before it sends the object to the client. These approaches need additional infrastructures in the network and the additional information (e.g., client capability information, semantic information of contents). Although these additional needs exist, this intermediary-based approaches address the problems of the client-based and server-based approaches and many researches have been emerged.

Although the intermediary-based approaches are considered most appropriate due to their flexibility and customizability, they have some system issues to be addressed. Because the costly transcoding has to be performed on demand in proxy servers, the scalability problem arises. To address the scalability problem and improve the system performance, researchers have proposed caching the transcoding results. Because of the cached results, we can reduce repeated transcoding tasks and the system performance can be improved. Recently, the cooperative transcoding proxy architecture is proposed to improve the system performance[3,4]. However, applying the traditional cooperative caching directly to the transcoding proxy architecture is not efficient due to inherent problems of the content transcoding such as multiple versions of contents[1,5]. Because of the multiple versions, the communication protocol of cooperative caches is more complex than existing protocols, such as ICP and causes additional delay which is incurred by finding more similar version of an object during the time for discover the object in cooperative caches. Additionally, each cooperative caches consumes too much storage to store redundant multiple versions for the same object. These hurdles decrease the system performance and utilization.

In this paper, we propose the efficient cooperative transcoding proxy architecture which uses the content-aware caching. The main purpose of the proposed system is simplifying the communication protocol of cooperative caches. To cope with the problem which is caused by the multiple version, we propose that every version for an object is stored at one designated proxy together. Each transcoding proxy is mapped with the hashed value of the URL and a proxy stores its transcoded result at the designated proxy which is mapped with the URL of the requested object. By using this concept, we gather the whole of the version for an object in its designated proxy, so called a *home proxy*, and find the best version for an object deterministically. According to this behavior, every version of an object resides at one designated proxy and the discovery process becomes simple.

While the proxies store objects deterministically, they should fear for the hot spot problem; a small fraction of objects will be hot which could lead to excessive load at nodes which are their homes. To prevent this overload, we divide a cache storage into two; *public storage* and *private storage*. The general content-aware caching mechanism uses the public storage which is used to find the best version of the requested object. The private storage contains the hot objects of the local clients. If the requested object is found in the private storage of a local proxy, there is no need to check the public storage of the home proxy. That is, we reduce the load of the home proxies of hot objects.

Moreover, we refine the cooperation mechanism for the proposed system to perform more efficiently. There are three main processes: the discovery process, the transcoding process and the delivery process. We exploit the characteristics of the content-aware caching to make the discovery process simpler than the previous process and design the transcoding process to increase the performance of the proxies. By using the redirection in the delivery process, we reduce the network traffic which is needed to manage the system.

We evaluate the performance of the proposed system by using trace based simulation. We use the Simjava to simulate the cooperative transcoding proxy system. We compare the system response time, the cache hit ratio and the communication cost between the previous cooperative caching and the proposed content-aware caching and show that the performance increases when the content-aware caching is used.

The rest of this paper is organized as follow. Section 2 briefly represent the related works and the problem of them. Section 3 presents our proposed architecture for cooperative transcoding proxy. The performance evaluation is on section 4. Finally, we concludes this paper on section 5.

2 Background

In recent years, some proposals have exploited both of transcoding and caching to reduce the resource usage at the proxy server, especially for transcoding time. The main idea of these approaches is that caching the transcoding results improves the system performance by reducing the repeated transcoding operation. Moreover, some studies extend a stand-alone transcoding proxy to cooperate each other to increase the size of the community of clients. This cooperative caching increases the hit ratio of the cache system by cooperating with each other caches for discovery, transcoding and delivery. As result of the increased hit ratio of system, it reduces not only the repeated transcoding operations, but also the user perceived latency for an object.

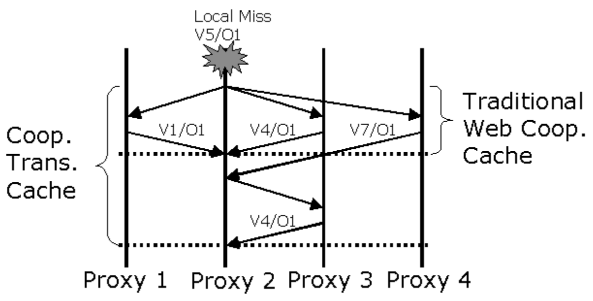


Fig. 1. The discovery process of the cooperative transcoding proxies

The transcoding proxy should manage the multiple version of an object, because the transcoding results depend on the various capability of clients. According to the multiple versions, there are two types of hit; *exact hit* and *useful hit*. The exact hit means the proxy cache contains the exact version required by the client, and the useful hit means the proxy cache does not have the exact version of the requested object but contains a more detailed and transcodable version of the requested object that can be transformed to obtain a less detailed version that meets the client request.

These two types of hits make the communication protocol of cooperative caches, especially the discovery protocol, more complex than existing protocols, such as ICP. Figure 1 shows the difference of the discovery process between the cooperative transcoding proxies and the traditional web proxies. The proxy 2 gets a request for an object, O1, whose version is the version 5, V5, and misses the object, then the proxy 2 sends queries to other proxies to find the object. If we use the traditional web proxies, the discovery process is over after getting any object from any proxy. In this figure, the proxy 1 or 3 returns the object O1 to the proxy 2 and the process is over. However, if we use the transcoding proxies, we should consider not only the object but also the version, then we have to wait for the best version that minimize the transcoding operation. In this figure, though the proxy 1 and 3 return the objects with version V1 and V4, the proxy 2 does not know that the proxy 4 has the exact version and has to wait for the responses from proxy 4. After the proxy 2 gets all responses form all proxies, it chooses the proxy which has the best version, in this figure the proxy 3, and sends a query to get the object itself. This behavior takes for long time to determine the best version that minimize the transcoding operation because a local transcoding proxy have to wait for potentially better version. Also, it generates enormous query messages to discover an object. That is, each transcoding proxy has to process redundant query messages for every discovery request.

3 Main Idea

3.1 Content-Aware Caching

We mentioned the problems of the query-based discovery protocol of the cooperative transcoding proxy in the section 2. A cause that a proxy waits for a potentially better version is that each version of the same content resides irregularly at different proxies. In this situation, a proxy should send queries to every proxy to discover the best version because it does not know which proxy has the best version of the content. However, if the different versions of a content are cached together at the designated proxy, a proxy can determine the best version of a content with only one query message.

Each proxy has to store transcoded versions of an object at designated proxy being aware of the object. We would refer this caching scheme as *content-aware caching*. In content-aware caching, a transcoding proxy stores its transcoded results at a designated proxy according to the URL of objects. Then, every version of the same URL is stored at a designated proxy. We would refer this designated proxy as a *home proxy* of an objects. Each URL is mapped into its home proxy by using URL hashing. The 128bit ID space is generated by the hash function which balances the ID with high probability such as SHA-1 and each proxy which is the participant of the system manages the partial ID space which is determined by the proxy node ID that is computed by hashing the unique value of node such as an ip address. Each object has the object ID which is obtained by hashing the URL of the object and is stored at the home proxy that manages the object ID.

This content-aware caching has several advantages. First, a proxy can discover the best version of an object deterministically. A proxy can find the best version at a home proxy with only one query and does not wait for potentially better version after it receives an exact or useful hit message. Second, the redundant query processing is reduced significantly. In the previous system, a proxy sends a query to every peer proxy, and each proxy which receives a query performs the query processing to find a proper version of an object. However, in the content-aware caching system, only home proxy performs query processing. Third, the network traffic is reduced because the number of query messages is reduced significantly.

3.2 Prevention of Hot Spot

When we use the content-aware caching, the request for an object is always forwarded to the home proxy. If the hot spot for an object occurs, the home proxy which has the responsibility for the object has to deal with every request and the home proxy is overloaded and out of order. To prevent this case, we divide a cache storage into two : *public storage* and *private storage*. The public storage is used to store the every version of an object for the content-aware caching and the private storage is used to store the hot objects of the local clients. That is, a proxy stores the clusters of version for objects whose object IDs are managed

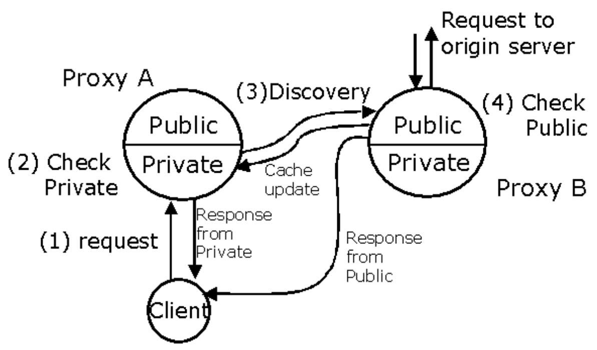


Fig. 2. Overall of the Content-aware caching system

by itself in the public storage and caches the frequently requested objects from the local clients in the private storage.

Figure 2 shows the overall of the cache architecture and the cooperation mechanism. When a proxy receives a request (1), it first checks its private storage (2). If a exact hit occurs, the proxy returns the object to the client. However, if either a local useful hit or a local miss occurs, the proxy tries to discover a matched transcoded object at the home proxy of the requested object (3). Then, the home proxy checks its public storage for the object (4). If there is the object which is exact or useful, the home proxy transforms the object into the exact version and returns the object to the client, and updates the private storage of the local proxy if the home proxy decides that the object is frequently requested. Otherwise, if a miss occurs, this proxy gets the new object from the origin server. According to this behavior, the hot objects reside at the local private storage with high probability and we can reduce the excessive load of the home proxies of the hot objects.

3.3 Cooperation Mechanism

There are mainly three cooperation processes: the discovery process, the transcoding process, and the delivery process. The first, the discovery process takes advantages of the content-aware caching. In our proposed system, different versions of the same content are cached together at the public storage of the home proxy. If a proxy gets a request and a miss occurs at the private storage, it need not send queries to every peer proxies but send only one query to a home proxy of the requested URL. Therefore, in the discovery process, we can reduce not only the query messages but also the waiting time for finding a potentially better version. Moreover, because the home proxy could have almost versions of an object, the exact hit ratio increases and the system performance would increase.

When we find the useful object in the public storage of the home proxy, we should decide the location of the transcoding. If the local proxy which gets the request from the client performs the transcoding, it has to update the public storage of the home proxy because the home proxy manages the whole version of an object. That is, we preserve the advantage of the discover process by using this redundant traffic. According to this, we performs transcoding task for the requested object at the home proxy to eliminate the redundant transmission.

After the transcoding task, the new version of the object is stored at the public storage of the home proxy. If the home proxy returns the new object to the local proxy and the local proxy returns it to the client, this indirect transmission causes the redundant object transmission that generally makes the response time long. To prevent this redundant traffic and reduce the response time, the home proxy redirects the response to the client which request the object. When the local proxy forward the request to the home proxy, the forwarding message includes the redirection information. However, this redirection mechanism can cause the hot spot problem at the home proxy. To cope with this problem, the home proxy has to update the private storage of the local proxy. If the exact hit occurs at the public storage, the home proxy checks how frequently the object is requested. If the object is decided as a hot object, the home proxy sends this object to the local proxy which requests it and the local proxy stores it at the private storage. This cache update policy compensates the effect of the hot objects with the local private storage.

4 Evaluation

4.1 Simulation Setup

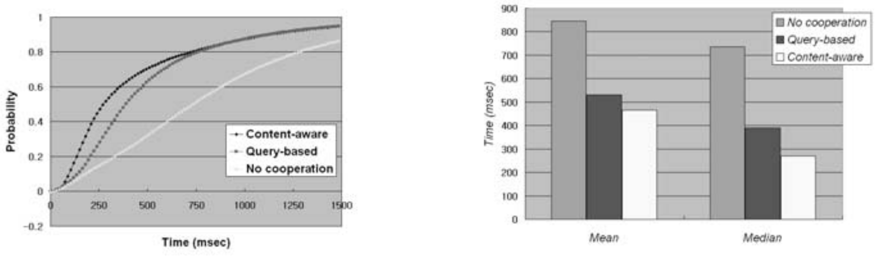
We simulate the cooperative transcoding proxy architecture to evaluate its performance. We use Simjava to simulate the architecture. Simjava is a toolkit for building working models of complex systems. It is based around a discrete event simulation kernel [6].

We try to reflect the real environment in our simulation as accurate as possible. We examine the previous papers on the cooperative caching to extract the simulation parameters [7]. The size of a cache storage is 300MB and 30% of the total storage is assigned to the public storage. We use 4 caches which are cooperated with each other and use the LRU replacement policy. The establishing HTTP connection takes 3 msec and the cache lookup needs 1.5 msec. The processing the ICP query need 0.3 msec and the hashing calculation for the content-aware caching takes 0.6 msec. The transmission time to content server takes 300 msec as average and the transmission time in local backbone network takes 40 msec as average. The simulation parameters about the transcoding operation are extracted from the previous paper [5]. The transcoding takes 150 msec as the mean value and 330 msec as the 90th percentile.

We use a trace file of IRCache [2]. The trace date is October 22, 2003 and the total duration is 1 day. The total number of requests is 416,015 and the mean request rate is 4.8 requests per second. We assume that the 100 clients use one proxy cache and consider a classification of the client devices on the basis of their capabilities of displaying different objects and connecting to the assigned proxy

Device type	**PC**	**Laptop**	**TV Browser**	**PDA**	**Mobile phone**
Percentage	40%	15%	15%	15%	15%

Table 1. Client device types and population



(a) Cumulative distribution

(b) Mean and Median

Fig. 3. The comparison on the system response time

server. The classes of devices range from high-end workstations/PCs which can consume every object in its original form, to mobile phones with very limited bandwidth and display capabilities. We introduce five classes of clients. Table 1 shows the client types and their population.

4.2 System Response Time

The system response time is the time between sending requests of clients and receiving of responses of clients and it is generally used as a criterion of system performance. Figure 3 shows the comparison on the system response time. It shows clearly that the cooperative architecture provides better performance than the stand-alone architecture. Also, it shows that the content-aware cooperation architecture provides better performance than the query-based cooperation architecture. The 90th percentile of the response time is similar between the content-aware architecture and the query-based architecture. However, the median of the response time is much better in the content-aware architecture.

The main reason of the different response times is the cooperative discovery protocol. The query-based discovery protocol has a problem of a long decision time due to the two-phase lookup. To address this problem, we proposed content-aware caching mechanism and this significantly reduces the decision time in the multiple-version lookup. Therefore, the performance of the system increases.

However, the 90th percentile is similar because the global cache miss causes the long round-trip time to the original server. This long round-trip time is the system bottleneck for both architectures. Although the content-aware coopera-

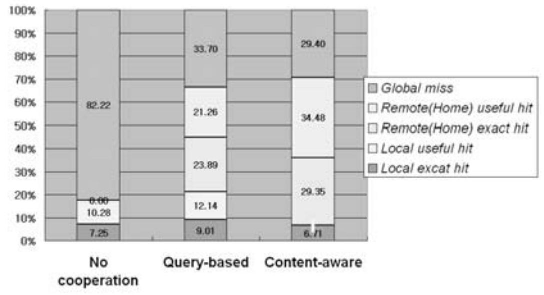


Fig. 4. The comparison on the cache hit ratio

tive architecture provides fast decision in multiple-version lookup, the dominant factor of the long transmission time from content server to a proxy server causes long user response time. Therefore, high hit ratio of a proxy cache is important.

4.3 Cache Hit Ratio

Cache hit ratio is the important factor that affects the system performance of the transcoding proxy. A cache in the multiple-version environment has three different event: an exact hit, a useful hit, and a miss. The high exact hit ratio improves the system performance by eliminating the transcoding overhead that generally involves a long processing time. The high useful hit ratio improves the system performance by reducing the redundant transcoding process. The high cache miss ratio degrades the system performance since this case needs the content transmission from a content sever to a transcoding proxy and the complete transcoding task.

Figure 4 shows the cache hit ratio of each scheme. The cooperation schemes provide much higher ratio of both an exact hit and a useful hit. The hit ratio of the content-aware scheme is slightly higher than the query-based scheme. In the query-based scheme, the exact hit ratio of the local cache is 9.01% and the exact hit ratio of the remote cache is 23.89%. In the content-aware scheme, the exact hit ratio in the private storage is only 6.71% which is smaller than the query-based but the exact hit ratio of the public storage is 29.35% which is much bigger than the query-based. Even if the local exact hit ratio of the content-aware scheme is smaller, the main factor of high exact hit ratio is the remote exact hit ratio for both schemes. In this case, the global lookup process of the query-based scheme causes the long decision time due to two-phase lookup in the multiple-version environment mentioned in the section 2. However, the content-aware scheme finds the exact object with the simple discovery process which takes only one query to the home proxy. Therefore, the content-aware scheme can provide better performance than the query-based scheme.

We can see that the useful hit ratio is increased in case of the content-aware cooperation architecture. The reason is that each useful version is clustered to be discovered directly in the content-aware cooperation architecture and they use the cache storage more efficiently without the redundant copies. Additionally, in the content-aware scheme, the local useful hit ratio is 0 because the private storage is used to find the exact objects only.

5 Conclusions

In this paper, we propose the efficient cooperative transcoding proxy architecture which uses the content-aware caching. We cope with the problem which is caused by the multiple version environment by using the content-aware caching, which means that every version for an object is stored at one designated proxy together. The proposed architecture makes the communication protocol between each proxies simpler, especially the discovery process and reduces the number of messages which are used to maintain the cache system such as ICP queries and object responses. Moreover, because of gathering all versions of an object at one proxy, the exact hit ratio increases and the performance of the system increases too. This architecture has an improvement of 20 percentage points of response time, an increase of 10 percentage points of cache hit ratio, and improved scalability on bandwidth consumption. Though the many advantages exist, the hot spot problem can be appeared. We prevent this problem by using the private cache and the cache update policy. The detail of the cache update policy is our ongoing work.

References

1. V. Cardellini, M. Colajanni, R. Lancellotti, and P. S. Yu. *A distributed architecture of edge proxy servers for cooperative transcoding* In Proc. of 3rd IEEE Workshop on Internet Applications, June 2003.
2. IRCache project, 2003. http://www.irchache.net
3. A. Maheshwari, A. Sharma, K. Ramamritham, and P. Shenoy. *TransSquid: Transcoding and caching proxy for heterogeneous e-commerce environments* In Proc. of 12th IEEE Int'l Workshop on Research Issues in Data Engineering, pages 50-59, Feb. 2002.
4. A. Singh, A. Trivedi, K. Ramamritham, and P. Shenoy. *PTC: Proxies that transcode and cache in heterogeneous Web client environments* In Proc. of 3rd Int'l Conf. on Web Information Systems Engineering, Dec. 2002.
5. C. Canali, V. Cardellini, M. Colajanni, R. Lancellotti, P. S. Yu. *Cooperative Architectures and Algorithms for Discovery and Transcoding of Multi-version Content* In Proc. of 8th Int'l Workshop on Web Content Caching and Distribution, Sep. 2003.
6. F. Howell, R. McNab. *SimJava: a discrete event simulation package for Java with applications in computer systems modeling* In Proc. 1st Int'l Conference on Web-based Modelling and Simulation, San Diego CA, Society for Computer Simulation, Jan. 1998.
7. C. Lindemann, O. P. Waldhorst. *Evaluating cooperative web caching protocols for Emerging Network Technologies* In Proc. of Int'l Workshop on Caching, Coherence and Consistency, 2001.