Journal of Universal Computer Science, vol. 14, no. 14 (2008), 2329-2342
submitted: 30/9/07, accepted: 30/4/08, appeared: 28/7/08 © J.UCS

An Optimization of CDN Using Efficient Load Distribution and RADS Caching Algorithm

Yun Ji Na
(Advanced Institute of Convergence Information Technology, Gyeoungju, S. Korea
yjna@dongguk.ac.kr)

Sarvar Abdullaev
(Division of Computer Science, Dongguk University, Gyeoungju, S. Korea
sarrtv@yahoo.com)

Franz I. S. Ko
(Division of Computer Science, Dongguk University, Gyeoungju, S. Korea
isko@dongguk.edu)

Abstract: Nowadays, while large-sized multimedia objects are becoming very popular throughout the Internet, one of the important issues appears to be the acceleration of content delivery network (CDN) performance. CDN is a web system that delivers the web cached objects to the client and accelerates the web performance. Therefore the performance factor for any CDN is vital factor in determining the quality of services. The performance improvement can be achieved through load balancing technique, so the server load could be distributed to several clustered groups of machines and processed in parallel. Also the performance of CDN heavily depends on caching algorithm which is used to cache the web objects. This study investigates a method that improves the performance of delivering multimedia content through CDN while using RADS algorithm for caching large-sized objects separately from small-sized ones. We will also consider the efficient distribution of requests outgoing from local servers in order to balance the CDN load. This method uses various types of factors such as CPU processing time, I/O access time and Task Queue between nearby servers. At the end of the paper, we will see the experimental results derived from implementing the proposed optimization technique and observe how it could contribute to the effectiveness of CDN.

Keywords: load balancing, caching algorithm, divided cache scope, content delivery network
Categories: C.4, H.3

1 Introduction

This study attempts to optimize the content delivery network through deploying efficient load distribution mechanism along with new caching algorithm with divided cache scope. It is necessary to consider certain physical factors in order to achieve this optimization. The factors that should be considered while deploying CDN can be summarized as follows:

1. The physical capacity of CDN affects the service speed. An improvement of physical capacity should be considered in respect with technical modernization issues and costs at the same time.

2. However there is a limitation to improvement of CDN performance by deploying additional capacity. A CDN is periodically or non-periodically updated as occasion demands. But it is difficult to introduce new technologies to existing CDN's physical structure because of high integration costs.
3. The service response time is dependent on the geographical distance between clients and servers. Although the performance of a CDN mentioned above largely affects the service response speed, the geographical distance between clients and servers has more significant impact on the service response speed.

In order to solve these issues, the enhancements from both local cache servers' and server group manager's perspectives should be undertaken. A clustered web system [Ko1, 07] [Cardelli, 99] [Kangsharju, 00] [Kangsharju2, 99] that uses a load balancing technique on web server side can improve the service response speed. Also, it can be accelerated through efficient web caching [Ko1, 07] [Ko2, 06] [Ko3, 05] [Williams, 96] algorithm on local cache server side. The main problem is how to cache the large multimedia content in most efficient way. We have several ways of enhancing the performance of overall system through implementing load distribution techniques along with web caching algorithms.

This study sets group of neighboring servers that is geographically close to clients and dynamically balances the load. We propose a load balancing method that adaptively distributes client requests to grouped servers. The proposed technique reflects a solution for certain geographical problems and various costs related with load distribution. In addition, this technique can reduce costs by sharing a servers' capacity and can improve the processing power of the system. Consequently, it significantly reduces the response time and eases the overload on servers. Along with proper load distribution system, we will introduce a new algorithm for effective caching of large data and examine the results of the whole system against traditional cases.

In order to evaluate the traffic load involved in CDN, it is useful to characterize the web objects requested by users. The data requested by user can have the following reference characteristics [Bolot, 95]. Firstly, web object size is greatly volatile, so the objects with various size range should be efficiently handled by web cache. Second reference characteristic is the variety of user preferences while requesting web objects. Third reference characteristic is the existence of time and space localities. It is important to consider the size heterogeneity of requested web objects in some a replacement algorithm. However the traditional caching algorithms do not consider the size heterogeneity of the web objects. In this study, we proposed a new replacement algorithm called RADS (Replacement Algorithm with Divided cache Scope) which deals with size heterogeneity of a web objects. Usually a number of small-sized objects are removed in case if some large sized object has to be cached to the web cache. RADS is designed with a divided scope which takes into account the size reference characteristic and treats each object according to its own size classification. Consequently it can improve the object-hit ratio and the response time for any classic caching algorithm. At the end of this page we will review some experimental results and make conclusion upon the performance of proposed optimization technique.

2 Related Works

Until now web servers have been servicing mainly the text and the image objects. As it was predicted that web will deliver mostly multimedia data in next few years, it is taking up 50% or more of web traffic since 2003 [Sahu, 99].

In this section we will discuss the web caching algorithms previously researched. If client sends request to web cache, the caching algorithm determines whether a required object exists in the storage scope of the web cache server. If the object that a client requested exists in the storage scope of web cache, a cache hit is achieved. On the other hand, cache miss occurs if it does not exist, which becomes a factor to decrease the performance of a CDN. In the case of a cache hit, a corresponding object is transmitted from the storage scope of a cache to a client. If the corresponding object from requested URL does not exist as in the case of cache miss, it is first requested from web server and then transmitted to a client. In this case we must provide storage of a cache for a new object to be used. A replacement algorithm is required in order to replace previously cached useless objects with new objects. Web caching improves the computing performance of the Internet as an object used often is saved in a cache, and provides a fast response to a client by decreasing the load of the web server and the traffic of the total networks finally.

Web caching is classified into several types: the client caching, the server caching, the proxy caching, the hierarchical caching, and the cooperation server caching. The common goal of these web caching methods is an efficient management of the limited storage scope [Barish, 00] [Aggarwal, 99] [Abdullaev, 07].

We can increase the performance of web caching by saving the frequently used object in the storage scope of cache. Therefore, an efficient object replacement algorithm is an important factor for improving the performance of caching [Niclausse, 98] [Aggarwal, 99].

Web-caching algorithms to relieve network congestion and access latency have proliferated in recent years. The following descriptions indicate how each algorithm selects a victim to purge from the cache.

1. **LRU**(least recently used): Removes the least recently referenced object first. LRU is an algorithm to replace an unused object in storage scope so that a new object gets a storage space.
2. **LFU** (least frequently used): Removes the least frequently referenced object first [Kangsharju, 00]. Each tuple in LFU cache list includes special counter which counts the number of requests to certain object, so the object with least references are deleted next when space is required for caching new objects.
3. **Size:** Removes the largest object first. SIZE is an algorithm to replace the largest object among the objects saved in storage scope so that a new object gets a storage space. The web cache is different from traditional cache such as a hardware cache and a file system cache. In web cache the unit of an exchange is a web object, and the size of an object is very variable. Therefore, in a case where the size of many objects is small, they are

removed from storage scope by one object whose size is large [Cardelli, 99]. The algorithm of SIZE improved this issue by replacing the greatest object among objects of cache storage scope for new object.

4. **LRU-min:** Removes the least recently referenced object whose size is larger than desired free space of size s [Kangsharju, 00]. If enough space is not freed, LRU-min sets s to s/2 and repeats the procedure until enough space has been freed. LRU-min is a transformation of LRU.
5. **SLRU** (size-adjusted LRU): $Value(i) = \frac{1}{t_{ik}} \times \frac{1}{s_i}$, where s_i is the size of object i, and t_{ik} is the number of references till the last time i was referenced [Bahn, 02].

Thus, we observed different types of caching algorithms and we identified that it affects the performance of web caching if high size heterogeneity exits. Therefore we have to consider this characteristic properly in replacement algorithm, so that less frequently used large-sized object is not saved in cache and but rather a greater number of frequently accessed small-sized objects should be saved instead. Consequently, we can improve the performance of cache server. Therefore, a study on the web-caching algorithm that includes various characteristics of web object is required.

3 User's View: Caching

A system of web caching is able to process the request of the user in an electronic commerce system and improve the performance of Internet. The performance of a web caching is dependent on effective management of limited caching space. Thus, studies on replacement techniques have been largely conducted to maintain frequently used web objects in the storage scope of web caches [Ko2, 06] [Ko3, 05] [Williams, 96]. The replacement technique for web caches reflects the characteristics of web objects. The characteristics of web system users can be summarized as follows [Ko1, 07] [Ko2, 06].

- The deviation of web objects, which are requested by users, presents a large value. Thus, web caches must effectively support the variable objects required by the users.
- Referenced web objects have a certain reference locality according to the time and region. These reference characteristics vary as the time passes and become major factors to decrease the performance of the existing caching technique.
- The characteristics of users, such as age, Internet literacy, and education level, affect the reference characteristics.
- The type and characteristics of a web service affect the reference characteristics of users.
- The variety of the reference characteristics increases the deviation of object-hit ratios.
- The reference characteristics are not always predictable.

Changes in the variety web object reference characteristics of users decrease the performance of a web caching. However, the characteristics of users in a web system have not been reflected in any of the existing web caching techniques. This was due to the fact that studies on the web caching techniques have been largely focused on the improvement of object-hit ratios and reduction of caching costs [Ko2, 06] [Ko3, 05] [Williams, 96]. Thus, studies on the web caching techniques that reflect the characteristics of users are required.

As previously mentioned, studies from both web server's and client's point of view are required to improve the performance of a web system.

Moreover, studies on web caching are required from a web service using system's point of view. This study proposes a web caching technique that adaptively reflects the heterogeneity of web objects. In order to increase the efficiency of web caches, the proposed technique investigates the issue of a web caching technique as follows [Ko2, 06] [Ko3, 05].

- The heterogeneity of web objects is closely related to the size deviation of objects. Thus, the identification of objects according to the heterogeneity makes it possible to identify objects according to their size.
- A web object size based algorithm, such as SIZE and LRUMIN methods, can reduce the number of cases where a large amount of small sized objects are removed by large sized objects in a storage scope. However, in the case of the reference characteristics that have a large size deviation, the performance will be decreased.
- If the object is managed according to its size, the object deviation in each region becomes more homogeneous in size. In this case, the number of small sized objects that are produced to replace a single large sized object can be reduced due to the decrease in the size deviation.
- The reference characteristics of objects are very variable. Thus, the deviation of the heterogeneity is also variable. An adaptive caching technique is required to reflect variable reference characteristics.

The proposed web caching technique can improve the performance of a web caching technique due to the fact that it is able to adaptively reflect the reference characteristics of web objects.

4 Load Distribution

A web server group of WSGi that consists of several web servers can be presented in a form of tree shape as illustrated in Figure 1. Means and functions of the node presented in the figure are as follows:

- WSG_i : Web Server Group i
- $LSGM_i$: Local Server Group Manager i
    - managing load information for each LS_i in a group
    - managing transmission costs according to the geographical condition between LS_i s in a group

   - task switching between LS_i s
- LSG_j : Local Server Group j, 1-j-m, m is the number of local server groups in $LSGM_i$
- LS_i : Local Server i, 1-i-k, k is the number of local servers in WSG
- C_{xy} : terminal nodes (clients) of each WSG,
   - 1-x-k, 1-y-n, k: is the number of local servers in WSG
   - n : number of clients in local server

As presented in Figure 1, lines that connect each node present a network connection between servers or between servers and clients. The operation of WSG_i can be expressed as follows.

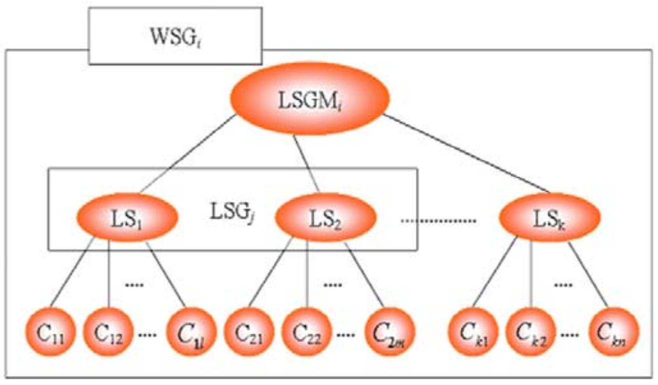

Figure 1: Tree structure of WSG_i

- Step 1) Requesting the task of C_j for LS_i
- Step 2) Processing tasks if LS_i can process the task of C_{xy}
   - Step 1) Transmitting requests to $LSGM_i$ if the load of LS_i can't process tasks
   - Step 2) $LSGM_i$ transmits a HTTP request by finding LS_k , which has a small load
   - Step 3) LS_k processes the delegated task
   - Step 4) Transmitting the results to LS_i
   - Step 5) LS_i transmits the results to C_j

4.1 Factors for the load state of each server

Load states should be checked in each server to distribute loads. In order to check C_{load} that presents a load state of server, L_S and Δ that present states and performance coefficients of each server, respectively, are required. Severs that are actually used in a certain system present differences in their performances. Thus, the coefficient Δ is used to present these differences. Although a load state of L_S in a server presents a little high level, the load of the server, $\approx$, may present a low level if the server has a very high performance.

Load verifications in each server can be performed using the CPU past history that extremely affects loads of a server, I/O past history, and wait loads of the task that is on standby in Task Queues. Thus, the load state of L_S of a server consists of three metrics namely.

$$L_S = (\alpha, \beta, \lambda)$$

L_S : load state for each server

α : average CPU past history in a time period of (t_h - t_c)

β : wait loads of the task that is on standby in Task Queues of a CPU in a time period of (t_c - t_r)

λ : I/O past history in a time period of (t_h - t_c)

where α and λ present the past history of CPU and I/O for a specific time period at the present time, and β presents the wait load of tasks, which will delegate a CPU, after finishing the present task.

4.2 Load distributions of a server according to the prior

If a task switching is required in $LSGM_i$ based on the load state of a server, loads can be distributed by verifying the load of a local server itself using a load state adapter.

Changes in the load state of a server should be adaptively reflected in order to dynamically operate a system according to the change in the state factor of the load of a server. The states for each server can be classified as six different states, such as minimal loaded state, lightly loaded state, mean value loaded state, lightly overloaded state, overloaded state, and maximal overloaded state according to the value of α, β, and λ that affects the load of a server.

Using a load state adapter, certain load states can be dynamically linked to other factors that affect the load of a server and presents the merit that the load of a local server can be managed in real-time.

In the case of the LS_i that is unable to process its own tasks, $LSGM_i$ is able to select a proper LS_i using the information of a load state adapter which is classified to following levels: Minimal loaded state server, lightly loaded state server, mean value loaded state server, lightly overloaded state server and overloaded state server.

In addition, if all servers in a group are in an overloaded state, a task switching will not be generated and will be on standby. Also, it waits for certain loads of a server, which requested task switching, and changes in load state of other servers in a group.

If the load state becomes the same value for each other, a server that has a low transmission cost according to a certain geographical location for a server, which requested a task switching, is to be selected.

5 Replacement Algorithm with Divide Cache Scope

In order to reach the maximum performance, it is necessary to apply efficient caching algorithm on local cache servers. Through handling most of the client requests locally, we will save the processing time of global server. Global server also stands for server group manager which does not let local traffic congest the backbone network connecting the local servers into one group. Replacement algorithm with divided cache scope could be the ideal one for handling multimedia content in large networks with a number of local cache servers. According to the size of web objects frequently requested in certain geographic region we can set the caching memory into several partitions according to the size group of web objects. For example, if in some area we have high demand on large-sized multimedia content, we can allocate larger partition for large-sized data at the expense of small-sized data which is not frequently used. This section will deeply discuss the essence of this algorithm.

This algorithm proposes to classify objects based on size characteristics, and to manage the divided storage of cache separately. Reference characteristics of object are variable. Therefore, efficiency based on the divided size of cache storage is variable, too.

The number of partitions, the volume of partitions and the determination of size to classify the objects are the crucial factors that impact the performance of web caching in this algorithm. Storage scope of the object that has an influence on web caching must be assigned in order to increase the hit ratio of cache.

In this algorithm we use some classification algorithm which labels the objects according to their size. The threshold for classifying the objects to LARGE and SMALL differs upon the context where the replacement algorithm is deployed. We assume that the threshold is 10 K. The objects with size of 10 k or above are labelled as LARGE, and the others which are less than 10 k are denoted as SMALL. Through denoting web objects with such labels we can allocate them to corresponding partition and manage them separately. Figure 2 shows the replacement algorithm with partitioned cache memory.

When object requests by client arrive, the cache server confirms the existence of an object in the corresponding partition according to the size of an object. The service of the object that a client required would be provided if a cache-hit occurs. Then, the cache server updates the time record on when it was used so that high ranking is assigned to this object in an LRU-MIN replacement.

If a cache miss occurs, cache server requests the corresponding URL for the service of the object, and the object is transmitted to the cache server. Then, a transmitted object is classified into a corresponding grade according to size, and a cache administrator confirms whether there is a space for this object in corresponding scope of cache memory.

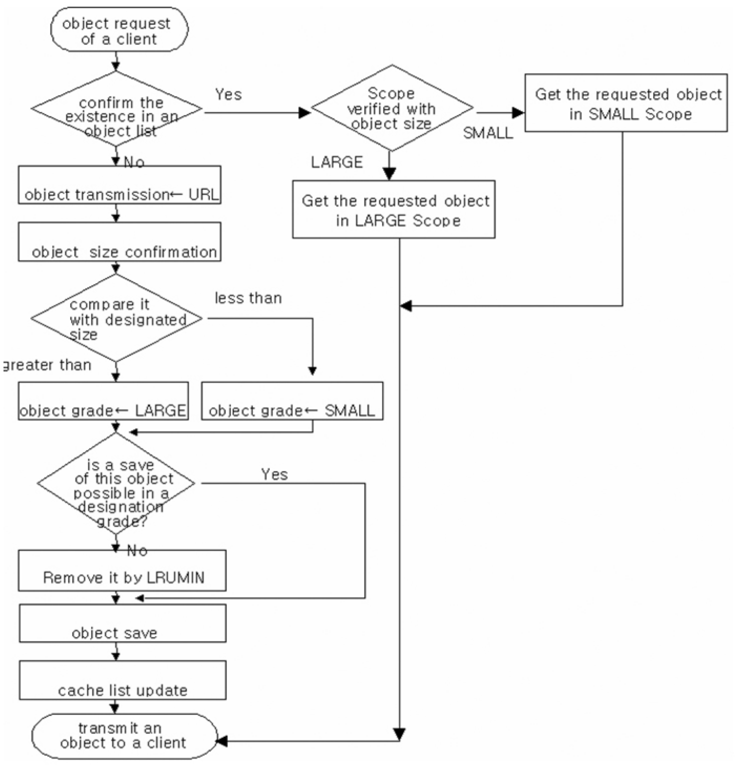
object request of a client
confirm the existence in an object list
Yes
No
Scope verified with object size
SMALL
LARGE
Get the requested object in SMALL Scope
Get the requested object in LARGE Scope
object transmission← URL
object size confirmation
compare it with designated size
less than
greater than
object grade← LARGE
object grade← SMALL
is a save of this object possible in a designation grade?
Yes
No
Remove it by LRUMIN
object save
cache list update
transmit an object to a client

Figure 2: Replacement Algorithm with Divided Cache Scope

If there is a space to save in cache scope, this object is saved, and this object is saved by LRU-MIN replacement algorithm if it is not there. Then, web object saved in a corresponding scope by substituting object of the same rank which is least recently used. Also, a time record of the newly arrived object is saved, and high ranking is assigned to this object in an LRU-MIN replacement process.

This study experimented on the performance of a cache scope divided by 6:4 and 7:3. But the efficiency of web caching may be increased more if division scope of variable size is used according to reference characteristics of an object than by division scope of size that has been fixed.

6 Experimental Results

Generally, for the performance evaluation measure of the replacement algorithm, cache-hit ratio, byte-hit ratio, delay-decrease ratio, a cost-saving ratio and response time are used. In this experiment, the performance of the proposed algorithm is evaluated by comparing an average object-hit ratio and response time.

The cache-hit ratio can indicate an object-hit ratio in web caching. The average object-hit ratio can calculate an average value of an object-hit ratio on a requested object following Formula (1).

$$Average\_object\_hit\_ratio = \frac{\sum_{i=1}^{n} s\_o_i \cdot n\_hit_i}{\sum_{i=1}^{n} s\_o_i \cdot n\_req_i} \times 100 \quad (1)$$

s_oi : size of object i

n_hiti : number of hits for object i

n_reqi : number of requests for object i

An average object-hit ratio is calculated by a ratio of hit object size in total object size that a client requested. It means that a high object-hit ratio provides a faster response regarding a request of a client in web caching, and is capable of efficiently reducing the load of a system. Also, it can decrease the traffic of the main system and the local server system.

And response time RT have the several response time notations, RT_{ch}(Response Time of cache-hit) on the cache-hit and response time RT_{cm}(Response Time of cache miss) on the cache miss. Response time for an object request of a client has the following delay factors.

①$TDT_{client_to_cache}$: Transmission delay time that occurs when a client requests an object to cache server

②DDT_{cache}: Delay time required for a determination of cache-hit or cache miss of cache server

③SDT_{cache}: The delay time required for a search of an object saved in Large or Small scope of cache

④$TDT_{cache_to_client}$: Delay time required when an object is transmitted from cache server to a client

⑤$TDT_{cache_to_server}$: Transmission delay time required when cache server requests an object from web server

⑥$TDT_{server_to_cache}$: Delay time needed when an object is transmitted from web server to cache server

⑦RDT_{cache}: Delay time that occurs in cache server when an object is replaced

1) A case of cache-hit

$RT_{ch}=TDT_{client\_to\_cache}+DDT_{cache}+SDT_{cache}+TDT_{cache\_to\_client}$ - Formula (2)

2) A case of cache-miss

$RT_{cm}=TDT_{client\_to\_cache}+DDT_{cache}+TDT_{cache\_to\_URL}+TDT_{URL\_to\_cache}+RDT_{cache}+TDT_{cache\_to\_client}$ - Formula (3)

The response speed has a close relationship with the object-hit ratio. Four delay factors occur with response time of a hit object in cache as in Formula (2), but six delay factors occur following the pattern of Formula (3) in the response time of a missed object. Among these delay factors, $TDT_{client\_to\_cache}$, $TDT_{cache\_to\_client}$, $TDT_{cache\_to\_URL}$, and $TDT_{URL\_to\_cache}$ are affected by the physical environment of networks. Therefore, delay time gets longer than cache-hit for the cache miss because of the many influences of the physical environment. Also, the RDT_{cache} is the delay time that occurs in cache server when objects replaced have a lot of influences on the performance of web caching in Formula (3). Therefore, if we increase the object-hit ratio in experiment one, we can improve the response time in experiment two.

Firstly we measured object-hit ratio. The experiment method is as follows. 70% on the cache scope was assigned first to a LARGE grade, and 30% was assigned to a SMALL grade. And the experiments were conducted on object-hit performance of this algorithm and LRU-MIN and SIZE. Second, 60% on the cache scope was assigned to a LARGE grade, and 40\% was assigned to a SMALL grade. Also, we experimented on the performance of these algorithms. In the figure, which showed the examination result, DIV6:4 shows 6:4 division, and DIV7:3 shows 7:3 division. The experiment results of object-hit ratio are shown in Figure 3 - Figure 4.

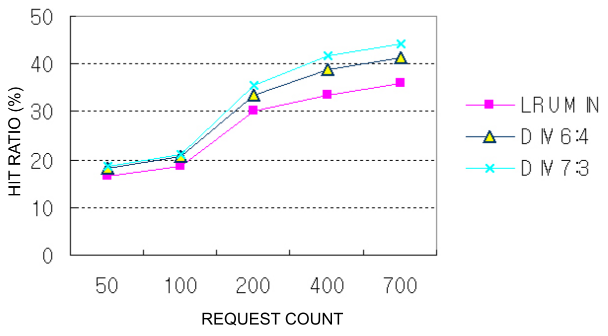

Figure 3: Object-hit ratio(%): compared with LRU-MIN

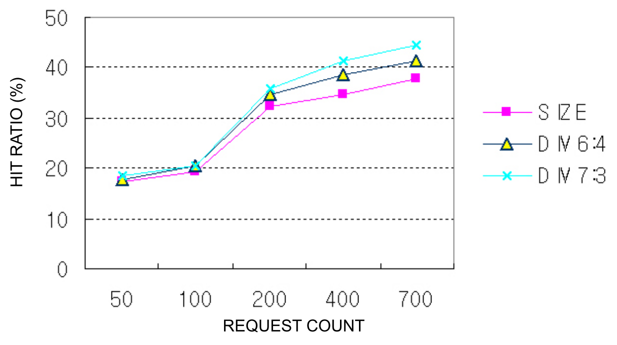

Figure 4: Object-hit ratio(%): compared with SIZE

The figures above show how the RADS method could improve the hit ratio of some classic caching algorithms. For instance, we can see how the SIZE caching algorithm gets improved as the number of requests grows. Definitely, the improvement in hit ration of caching algorithm can consequently result in the optimization of CDN.

In these experiments of a hit ratio, we can get the following conclusion:

- In the cache with small-sized capacity, the hit ratio of LRU-MIN, SIZE and the proposed algorithm were almost similar.
- As the cache capacity grew larger, the performance of LRU-MIN and SIZE is improved. In experiment result, we can see the performance of SIZE and LRU-MIN almost the same.
- And as the capacity of cache grew larger, the hit ratio performance of the proposed algorithm is more efficient than traditional replacement algorithms, and we can get 10%-15% performance enhancement.

Response time is the time required to provide the requested web object to a client. Figure 5- Figure 6 show the results of the experiment on response time.

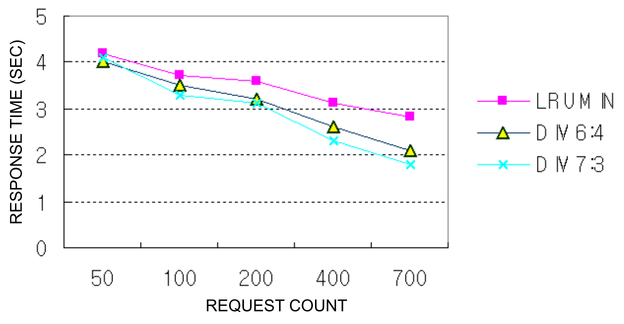

Figure 5: Response time(sec.): compare with LRU-MIN

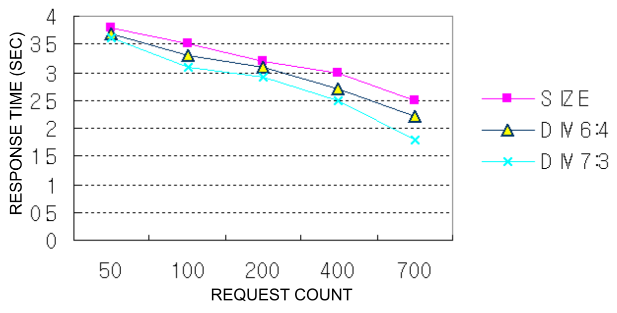

Figure 6: Response time(sec.): compare with SIZE

In figures above, we can also see how the RADS approach can decrease the response time of some classic caching algorithm like LRU-MIN or SIZE. For example, in Figure 6 it is notable that the response time for RADS approach with 7:3 divided scope ration is gradually decreasing as the number of requests grows. After conducting this experiment on response time performance of RADS, LRU-MIN and SIZE, we reached the following conclusions:

- As the cache scope grows larger, the proposed algorithm and LRU-MIN, SIZE all improved their response time.
- The performance of traditional algorithms and the proposed algorithm were almost similar with small-sized cache.
- As the capacity of cache grew larger, the response time performance of the proposed algorithm is more efficient than traditional replacement algorithms.

7 Conclusion and Future Perspectives

CDN is based on a web-based system where the performance of the system is the major factor determining the quality of services. An improvement in the system performance can be achieved using a type of load uniformity by producing a clustered group of servers. This study investigated a method that accelerates the performance of CDN through deploying new RADS caching algorithm in local cache servers. We tested the performance of new algorithm against traditional caching algorithms like LRU, LRU-MIN and SIZE and concluded that it shows better object-hit ratio which results in better performance of local cache servers and will keep the local client traffic in-bound to local network. Also the outgoing traffic from local cache servers will be properly managed by server group manager and will be directed to the least loaded machine which in turn easily returns the requested data using proposed caching algorithm. In this way we can enhance the response time and balance the load within backbone network. There are still many issues that have to be studies to improve the proposed algorithm. For example, the static presumed threshold for classifying the web objects to LARGE and SMALL grades can be dynamically adapted to the network. But it involves a special Adaptor unit analyzing the statistics of cached objects and making proper decision while calculating the threshold.

References

[Abdullaev, 07] Abdullaev, S., Ko, I.S.: A Study on Successful Business Intelligence Systems in Practice, *Journal of Convergence Information Technology* (JCIT), 2(2): 89-97, June 2007.

[Abrams, 95] Abrams, M., Standridge, C., Abdulla, G., Williams, S., Fox, E.: Caching Proxies: Limitations and Potentials, In Proc. of 4th Int. World Wide Web Conf., December 1995, 119-133

[Aggarwal, 99] Aggarwal, C., Wolf, J., Yu, P.: Caching on the World Wide Web, *IEEE Trans. Knowledge and Data Engineering*, 11(1): 94-107, January 1999.

[Bahn, 02] Bahn, H., Noh, S., Min, S. L., Koh, K.: Efficient Replacement of Nonuniform Objects in Web Caches, *IEEE Computer*, 35(6):65-73, June 2002.

[Barish, 00] Barish, G., Obraczka, K.: World Wide Web Caching: Trends and Algorithms, *IEEE Communications*, Internet Technology Series, May 2000, 444-455.

[Bolot, 95] Bolot, J. C., Hoschka, P.: Performance Engineering of the World-Wide Web: Application To Dimensioning And Cache Design, In Proc. of the 5th Int. World Wide Web Conf., May 1996, 110-128.

[Cardelli, 99] Cardellini, V., Colajanni, M., Yu, P. S.: Dynamic Load Balancing on Web-Server Systems, In Proc. Of Int. Conf. IEEE Internet Computing, May 1999, 28-39.

[Kangsharju2, 99] Kangasharju, J., Ross, K. W.: A Clustering Structure for Reliable Multi-casting, Computer Communications and Networks, In Proc. of Int. Conf. INFOCOM 1999, USA, March 1999, 378-383.

[Kangsharju, 00] Kangasharju, J., Ross, K. W.: A Replicated Architecture for the Domain Name System, In Proc. of Int. Conf. INFOCOM2000, Israel, March 2000, 2: 660-669.

[Ko1, 07] Ko, I. S.: Design and Performance Analysis of the Web Caching Algorithm for Performance Improvement of the Response Speed, In Proc. of Int. Conf. on MMM2007, Singapore, LNCS 4352, Springer-Verlag, January 2007, pp. 686-693.

[Ko2, 06] Ko, I. S.: ACASH: An Adaptive Web Caching method based on the Heterogeneity of Web Object and Reference Characteristics, *Journal of Information Sciences*, 176 (12): 1695-1711, June 2006.

[Ko3, 05] Ko, I. S.: An Adaptive Web Caching Method based on the Heterogeneity of Web Object, ISPA2005, Japan, LNCS 3758, Springer-Verlag, November 2005, pp. 853-858.

[Niclausse, 98] Niclausse, N., Liu, Z., Nain, P.: A New and Efficient Caching Policy for the World Wide Web, In Proc. Workshop on Internet Server Performance (WISP 98), June 1998, 94-107.

[Rizzo, 00] Rizzo, L., Vicisano, L.: Replacement Polices for a Proxy Cache, *IEEE/ACM Trans. Networking*, 8(2):158-170, April 2000.

[Sahu, 99] Sahu, S., Shenoy, P., Towsley, D.: Design Considerations for Integrated Proxy Servers, In Proc. of IEEE NOSSDAV'99, 247-250, June 1999.

[Williams, 96] Williams, S., Abrams, M., Standridge, C. R., Abhulla, G., Fox, E. A.: Removal Policies in Network Caches for World Wide Web Objects, In Proc. of Int. Conf. ACM SIGCOMM 1996, USA, October 1996, 293-304.