An Effective Data Replacement Scheme within Video Proxy for VOD System

Yu-Jin Wang, Yi-Shou Lin, S.W.Huang and Cheng Chen
Department of Computer Science and Information Engineering
1001 Ta Hsueh Road, Hsinchu, Taiwan, 30050, Republic of China
Tel:(886)35712121 EXT:54734, Fax:(886)35724176
Email:{ugwang, linys, huangsw, cchen }@csie.nctu.edu.tw

Abstract

Due to the advances in computer and multimedia techniques, VOD is more popular than before. However, the bandwidth is still a bottleneck in the VOD environment. In order to resolve the problem, we proposed a video proxy architecture in this paper. A good video proxy should have higher hit rate. Thus, we design an efficient data replacement scheme by using Knapsack to increase total system performance. Besides, we also provided a prediction method to predict the distribution of access patterns. By the method, we can decide which data could be put into the video proxy. According to our simulation and evaluation, our method can improve hit rate much as well as save a lot of bandwidth. The detail will be described in the literature.

Key words: *Multimedia, Algorithm, Proxy server, Bandwidth*

1. Introduction

Recent advancement in multimedia has brought about new applications which allow us to exchange information efficiently [1-3]. Among these applications, video-on-demand (VOD) is an interesting application [4]. Our main purpose is to improve VOD architecture and then reduce the bandwidth usage. Caching popular video in local proxy is one of the most attractive ways to resolve this problem [1-3,5,7]. In general, there are three architectures in accordance with the location of the proxy [2]. Among them, the independent proxy structure is more interesting because it can share the cached data to every client and reduce the response time for a client. In this paper, we propose a cooperative video proxies architecture to save the bandwidth more. No doubt, how to increase hit rate in our architecture is also important. Many factors can increase hit rate, such as job scheduling [8-10], data placement [4,7], data replacement [11-14], and so on. Several replacement policies have been presented [11-14]. However, they aren't suitable in VOD because of the characteristics of video data. Thus, we proposed a two-phase replacement scheme in our cooperative video proxies. In the first phase, we will design a popularity comparison method to decide when we will trigger the replacement operation. Here, we propose a prediction method to predict the popularity of requests. And then in the second phase, we designed an effective replacement by using the Knapsack algorithm method to save the bandwidth dramatically [18-19].

In order to evaluate the proposed architecture and two-phase replacement scheme, we design and implement a simulation environment. By our evaluation results, we only take a little time and some disk capacity to increase the total performance. We can save about 1624GB bandwidth one day than that of without cooperative proxies. And, if we adopt our two-phase replacement, we can save more 277GB than that of several other policies.

The remainder of this paper is organized as follows. In Section 2, some background knowledge will be surveyed briefly. In Section 3, we will explain the design concept and principle of our VOD system. In Section 4, we will introduce the basic concept and principle of two-phase replacement scheme in our video proxy. In Section 5, related performance gains will be evaluated and analyzed. Finally, some concluding remarks will be given in Section 6.

2. Fundamental Background

Basically, the VOD systems suffer from the problem of communication bandwidth [6]. Hence, how to resolve the problems become more important. One of the popular ways is to cache popular video in local proxy [1-3,5,7]. In general, there are three architectures of the proxy,

named Proxy-at-Client, Proxy-at-Server and Independent Proxy [2]. Independent Proxy can share the cached data to every client and reduce the response time for a client [2]. Thus, we will focus on this kind of architectures. Because the single proxy inherently limits in scalability and robustness, research trend turns to cooperative caching servers [2,5,15]. One of the most popular Squid caching servers is to use the Internet Cache Protocol (ICP) to maintain the consistency of the objects in the caches [16]. However, ICP is not a scalable protocol because the local caching server sends many ICP queries to search the requested object in other siblings when a local cache miss [15,16]. All siblings have to receive and process the ICP query. As the number of the cooperative proxies increases, the overhead of ICP will becomes serious quickly. To solve the scalability problem, Caching and Replication for Internet Service Performance (CRISP) was proposed [15]. Although the CRISP may reduce the number of messages in the sibling-lookup process, it doesn't reduce the access latency for an object missing in the local server. The sibling-lookup time is still needed. Another drawback is that it needs a failure handing process if the mapping server is over loaded or down. Thus, a distributed protocol was proposed [15]. This protocol can eliminates the sibling-lookup time. Every caching server has the directory recording the caching entry of other caching servers. Thus, the local proxy can immediately determine which peer server contains the requested object if a local miss occurs. Besides, it can also increase the robustness. When any proxy is down, the other proxies still know the location of object cached in the other proxies. However, this protocol can't still reduce the number of local proxy miss. Thus, in the following sections, we will introduce our video proxy architecture with an efficient replacement technique to increase the total performance in some degree.

3. Design of the System Architecture

As the growth of client number, the requirement of bandwidth usage is much more. We can cache popular video in local proxy to save a lot of bandwidth. According to scalability and robustness, our video proxy is based on cooperative proxies. It stands as a cache between the home server and the clients. We use Chang's protocol to let all of the cooperated video proxies communicate with each other efficiently [15]. But, our protocol still has a difference to it. In our protocol, when the requested object is not cached in local proxy, the local proxy will get the object from peer proxy or home server, and supplies the object to the client, instead of saving the object in its cache each time. It is because the size of video is huge and we don't know if the new video is popular than any of the cached video. According to the location of queried video, there are five situations will occur, named Directly Hit, Peer Hit, Download Peer, Home Hit, and Download Home.

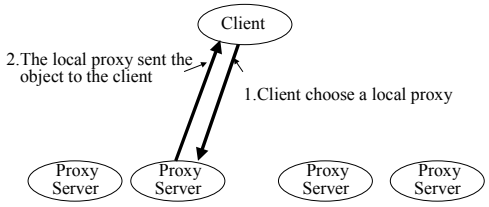

Figure 1. Directly Hit

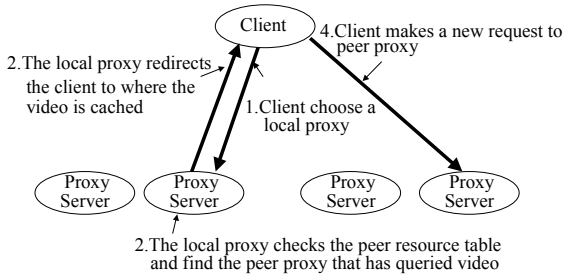

Figure 2. Peer Hit

In Directly Hit, as shown in Figure 1, the queried video is cached in the local video proxy. The video proxy sends the object to the client directly. In Peer Hit, as shown in Figure 2, if the queried video is not cached in the local video proxy, then the video proxy server will check his peer resource table. The table contains the total videos that peer proxy cached. If the table has the video, the video proxy will redirect the client to this peer video proxy. Then, the client will send a new request to the peer video proxy.

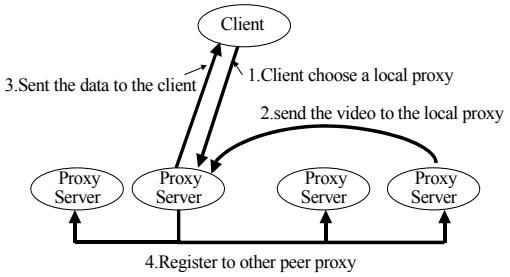

Figure 3. Download Peer

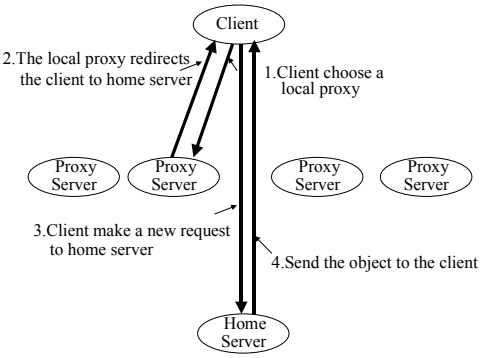

Figure 4. Home Hit

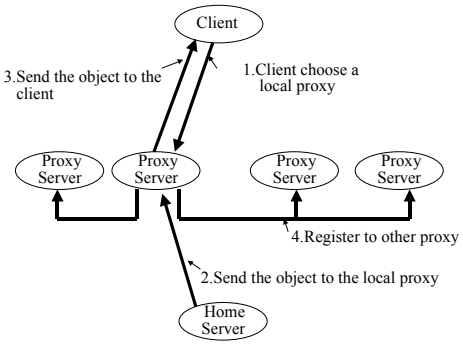

Figure 5. Download Home

In Download Peer, as shown in Figure 3, if a video is often requested in local video proxy and this video is cached in peer video proxy, it may be downloaded to the local video proxy from the peer video proxy. Data placement module will calculate the frequency of the requested video and the total videos that are cached in video proxy. If the value of requested video is larger than any of videos cached in video proxy, the requested video will be downloaded into the local video proxy from the peer video proxy. And then, the local video proxy will tell other peer video proxies that it has a new video and ask them to update their peer resource table. By this way, the response time is well as the bandwidth between the client and the peer video proxy can be reduced.

In Home Hit, as shown in Figure 4, if the video is not cached in the local video proxy or the peer video proxies, the local video proxy will redirected the client to home video server. The client will send a new request to the home video server. Then, the home video server will return the queried video to the client. Basically, Download Home shown in Figure 5 is similar to the situation of Download Peer. If a video is often requested and it is cached in home video server, it may be downloaded to the local video proxy from the home video server. Data placement module will calculate the frequency of requested video and the total videos cached in video proxy. If the value of requested video is larger than that of any of the videos cached in video proxy, the requested video will be downloaded into the local video proxy from the home video server. And, the local video proxy will tell other peer video proxies that it has a new video and ask them to update their peer resource table. Figure 6 shows the flow chart of all kinds of operational scenario. In the next section, we will introduce an effective replacement scheme used in our video proxy architecture.

4.Two Phase Replacement Scheme

There are two main design issues in our replacement scheme. The first is when we will trigger the replacement scheme. The second is how to choose a victim to be replaced. In the prior policy, the proxy will choose a least use video as a victim [11-14]. But, it may not save the most bandwidth. For example, the size of the video A is 100MB and the number of access is 20. The other video B has the size with 50 MB and its access number is 30. Based on the traditional policies, the video B will be cached in local proxy because its access number is larger than that of the video A. However, the bandwidth usage of the video A is 2000MB which is more than that of the video B. Thus, caching video A in local proxy is more useful to save the network bandwidth. For this reason, the video size is an important factor in designing our replacement scheme.

By taking these two main design issues into considerations, we propose a two-phase replacement scheme. In the first phase, we will design a popularity comparison method to

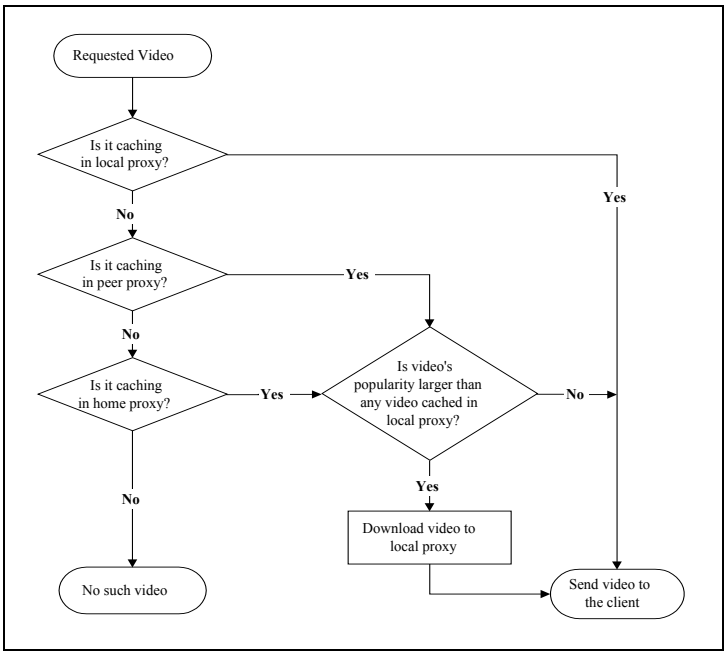

Figure 6. System Flow Chat

decide when we will trigger the replacement operation. And then in the second phase, we will design an effective replacement method to choose a victim. Below, we will introduce the popularity comparison method and two-phase replacement scheme in some detail.

4.1 Popularity Comparison Method

The main function of the popularity comparison method is to decide when we will trigger the replacement method. Basically, when a new video is coming, we will compare its access frequency with the total cached videos. If the access frequency of the new video is larger than that of any other cached video, the local proxy will go to the second phase to trigger the replacement method. In order to obtain the optimal performance/cost, we develop two different modes of the popularity comparison method. One is the *non-predict mode* and the other is the *predict mode*.

4.1.1 Non-Predict Mode

In this mode, if the access frequency of the new video is larger than that of any other cached video cached, then the process goes to the second phase. But, we may not have the highest performance in this mode. Given the same example as before, because the access frequency of the video A is not larger than that of video B, the process won't go to the second phase. If we let the process go to the second phase until the access frequency reaches 30, it will waste 100*10 bandwidth. In order to avoid this kind of drawback, the *predict mode* is proposed.

4.1.2 Predict Mode

In this mode, if the access frequency of the new video is larger than that of any other cached video, we let the process go to the second phase like the non-predict mode. If not, we define P as the probability of the progress towards the second phase for each process. In our system design, a random number, C, will be generated automatically to compare with P when a newly request is coming. If C is smaller than P, second phase will be triggered and vice versa. We can determine the P depending on the access frequency of the new video. If the access frequency of newly requested video accumulates rapidly, P will be larger. However, we do not know the distribution of the video accesses today. Fortunately, access patterns may not change dramatically every day [17]. We can predict the growth speed of video access frequency newly requested according to its access pattern occurred within the last time period before (e.g. yesterday). If we record the whole access pattern within this period, the overhead of computing is too large. Without loss of generality, we may take one day as a period and then partition one day into four intervals. We record the least access frequency in four intervals to predict the distribution of the video access for the next period (e.g. tomorrow). Now, we introduce a method to predict the distribution of the video access. We define the last period as T_p, current period as T_c. Four last intervals are I_{p1}, I_{p2}, I_{p3}, I_{p4} and four current intervals are I_{c1}, I_{c2}, I_{c3}, I_{c4}. And, we also define $f_a(I_{pi})$ as the least accumulation access frequency during each last interval I_{pi} ,

i=1,2,3,4. Assume a video V is coming and its access frequency is F_v. We want to know whether the access frequency accumulation of the video V grows rapidly. We use $f_a(I_{pi})$ to be a criterion to predict the distribution of the video access in current period. With the time increasing, $f_a(I_{pi})$ will change in each interval. We define $F_{threshold}(t_c)$as the criterion in each interval given in Equation 4-1. And the detailed algorithm is given in Appendix A.

$$F_{threshold}(t_c) = \begin{cases} f_a(I_{p1}), & t_c \in I_{c1} \\ f_a(I_{p2}), & t_c \in I_{c2} \\ f_a(I_{p3}), & t_c \in I_{c3} \\ f_a(I_{p4}), & t_c \in I_{c4} \end{cases} \quad (4\text{-}1)$$

where t_c is defined as the current time

When the accumulation of the video V is increasing, F_v will approach to $F_{threshold}(t_c)$ It means that the video V has more probability to download into proxy. Thus, a higher probability P is assigned. Formally, the P can be defined by the Equation 4-2.

$$P = \begin{cases} 100\%, & F_v \geq F_{threshold}(t_c) \\ \dfrac{F_v}{F_{threshold}(t_c)}, & \text{otherwise} \end{cases} \quad (4\text{-}2)$$

4.2 The Replacement Method

The main function of the second phase is to decide which video as a victim will be replaced, so that we may cache the most popular video in the local proxy. Thus, how to store different size of videos in the limited capacity of the local proxy to raise the performance becomes very important. This problem is similar to the 0-1 knapsack problem [18-19] because the capacity of the local proxy is limited, the size of the videos and the popularity of the videos are all variable. Since the total size is smaller than proxy capacity, we can use the dynamic programming technique to obtain the maximum profit [18-19]. The main goal of the replacement scheme is to reduce the provision of the home server. Thus, we hope the throughput getting from the local proxy is as much as possible. And, the size of a video multiplied by the frequency of the video is the network throughput. We can map and model our scheme by using knapsack algorithm as follows. Assume that there are n videos, item[1] through item[n], cached in the local proxy, and a new video item[n+1] is coming. Let

f_j = *access frequency* of video j, during period T

w_j=*size* of video j,

$p_j = f_j * w_j$,

c = *capacity* of the local proxy,

The f_{n+1} is the access frequency of the new video and the w_{n+1} is the size of the new video. The value of the p_{n+1} is $f_{n+1} * w_{n+1}$. By applying the Knapsack algorithm, we can obtain the maximum profit. We give an example here. Assume there are 3 videos cached in the local proxy and the capacity of the proxy is 8. There is a new video coming. The characteristics of these videos are shown in Table 4-1.

Table 4-1. The Characteristics of All Videos

Video (i)	1	2	3	4(new)
w_I	2	1	5	7
f_I	4	4	2	2
p_i	8	4	10	14

By Knapsack algorithm, we can obtain the optimal value, 22, derived from video 1, video 2, and video 3. Because the new video 4 is not in the result, we will not download the new video from the home server. The detailed algorithm is illustrated in Figure A-1 to Figure A-3.

Algorithm 1: Popularity Comparison

Input: a new request: A

number of cached videos: N

prediction mode: M

total videos cached in the proxy:

Video[1],…..,Video[N]

$F_{threshold}(t_c)$: F

Output: enter /* if enter is true, go to the second and vice versa */

Program:

```
switch(M){
non-predict:
  for(int i=1;i<=N;i++){
    if(A.frequency>Video[i].frequency)
        enter=true;
    else{
      enter=false;
      break;
```

```
        }
    }   /* check whether A.frequencylarger
                    than all cached videos */
    if(enter==true)
        probability=100%;break;
predict:
    if(A.frequency>F)
        probability=100%;
    else
        propbbility=A.frequency/F;
        break;
}
```

Figure A-1. The Algorithm of Popularity Comparison Method

Algorithm 2: Knapsack

Input: number of cached videos: N
capacity of knapsack: C
profit of total videos cached in the proxy: p[1],…,p[N]
weight of total videos cached in the proxy: w[1],….,w[N]

Output: knapsack table: K_{C*N}
profit table: P_{C*N}
the coordinates of largest profit: (z_x,z_y)

Program:

```
K[0][0]=1;
K[0][1...n]=0;
for(i=1;i<n+1;i++){
  for(j=0;j<c+1;j++){
    if(K[i-1][j]>0){
      K[i][j]=1;              //not select
      P[i][j]=P[i-1][j];
    }
    if (j-w[i]>=0){
      if(K[i-1][j-w[i]]>0){
        if(K[i-1][j]==0){
          K[i][j]=2;      //select
          P[i][j]=P[i-1][j-w[i]]+p[i];
        }else{      //select or not
          K[i][j]=3;
          if(P[i-1][j]>=P[i-1][j-w[i-1]]+p[i-1])
            P[i][j]=P[i-1][j];
          else
            P[i][j]=P[i-1][j-w[i-1]]+p[i-1];
        }
        if(P[i][j]>z){
          z=P[i][j];
          z_x=i;
          z_y=j;
        } }
    }                    //end of knapsack loop
}       //end of stone loop
```

Figure A-2. Algorithm 2: Knapsack Algorithm

Algorithm 3: Backward

Input: knapsack table: K_{C*N}
profit table: P_{C*N}
number of cached videos: N
the coordinates of largest profit: (z_x,z_y)
profit of total videos cached in the proxy: p[1],…..,p[N]
weight of total videos cached in the proxy: w[1],….,w[N]

Output: the desired items: item[1],…item[N]
/* if item[i] is true, the item[i] will be cached in the proxy. i∈ $\{1,....,n\}$*/

Program:

```
i=z_x;
n=j=z_y;
k=m=i-1;
while(k>0){
  if(K[k][n-w[m]]==3){
    if(P[k-1][n-w[m]]>=
      P[k-1][n-w[m]-w[i-1]]+p[k-1])
      K[k][n-w[m]]=1;   //not select
    else
      K[i][j]=2;                  //select
  }
  else if(K[k][n-w[m]]==1){  k--;
    item[k]=false;                  //not select
  }
  else if(K[k][n-w[m]]==2){
```

```
        item[k-1]=true;                    //select
        n=n-w[m];
        m=k-1;
        k--;
      }
  }
```

Figure A-3. Algorithm 3: Backward Algorithm

When we use the knapsack algorithm as our second phase replacement method, we can save the most server provision. We will aznalyze it in the following. Obviously, the total network usage D_{total} is calculated by the Equation 4-3.

$$D_{total} = D_{home} + D_{local} \qquad (4\text{-}3)$$

where

$$D_{total} = \sum_{i} video[i]_{size} * video[i]_{frequency}$$

$i \in$ all request (4-4)

$$D_{home} = \sum_{i} video[i]_{size} * video[i]_{frequency}$$

$i \in$ request home server (4-5)

$$D_{local} = \sum_{i} video[i]_{size} * video[i]_{frequency}$$

$i \in$ request local server (4-6)

The $video[i]_{size}$ is the size of the video requested by the client *i*. The $video[i]_{frequency}$ is the frequency of the video requested by the client *i*. If the clients are fixed, the total network usage D_{total} is fixed. Our purpose is to minimize the provision of the home server D_{home}. Therefore, we have to maximize the provision of the local server D_{local}. In our replacement scheme, it fits the concept. Thus, the profit in our algorithm is the $video[i]_{size} * video[i]_{frequency}$. It is the provision of the local server D_{local} shown in Equation 4-6. In knapsack algorithm, we always choose the largest profit of all kinds of the combinations, i.e. the largest D_{local}. Because the total bandwidth usage is fixed, we can obtain the minimum of D_{home} by the Equation 4-3. Below, we give a brief deduction for our result.

Theorem 1: Given a set of total requests, R, and a set of requests which get data from the local proxy, S, and $|S| \leqq |R|$. By the knapsack algorithm, the provision from home server

$$D_{home} = \sum_{i}^{S} video[i]_{size} * video[i]_{frequency},$$

is minimum. In other words, we can reduce the most bandwidth from the proxy to the remote home.

Proof: Assume there is a $D_{home'}$ smaller than $D_{home.}$ Thus, there is a $D_{local'}$ larger than $D_{local.}$ But, by knapsack algorithm, we always choose the largest D_{local} from the profit table. It is a contradiction. Therefore, by the knapsack algorithm, we can reduce the most bandwidth from the proxy to the remote home.

In conclusion, we have described the popularity comparison method as our first phase. Its complexity is O(*n*). The replacement method is our second phase. Its complexity is O(*kn*). Thus, the complexity of the two-phase replacement scheme is O(*kn*). Here, the *k* means the capacity of the proxy and the *n* means the number of the total videos cached in local proxy. Because *k* is constant and the number of the total cached videos *n* is small, the complexity is not high and it is acceptable. Applying the two-phase replacement scheme, we can save the most bandwidth.

5. Simulation Environment and Performance Evaluations

In this section, we will describe the structure of our simulation environment first. Then, the input data model employed in the simulation environment will be explained. Lastly, several performance evaluations are given to illustrate the main advantages of our two-phase replacement scheme used in the video proxy architecture.

5.1 Overview of Our Simulation Environment

In our simulation, Request Generator simulates user behavior to request a video. In general, we can configure the request average arrival time following Poisson distribution [10,20]. Moreover, we apply Zipf distribution for video selection [10]. Admission Control controls the incoming requests. Every user's requests will be accepted until admission criteria are satisfied. Generally, it examines if requested video is in the server and system capability is enough for new requests. Job Scheduling is used to control proxy to retrieve video data from disks and buffer them in a cycle fashion. Disk Scheduling is used to optimize the performance of reading data from disks by rearranging the order of retrieval commands from job schedule module. Data Placement acts as the file system manger of video server. It handles the mapping from logical address to physical address of the video blocks. By peer resource table, we can know that peer servers cache what videos so as to search the peer servers quickly. Statistical Table records the

characteristics of every video such as access frequency etc. When the knapsack algorithm is triggered, we can obtain the characteristics by the table. All video proxies can communicate with each other by the communicative stub module. When a video proxy downloaded a new video from the remote home server, and then it will tell other video proxies that it has a new video by this module. And, the other video proxies will update their peer resource table. The whole structure is shown in Figure 7. In the following, we will describe the video data layout, request model and simulation parameters before introducing our evaluation vectors.

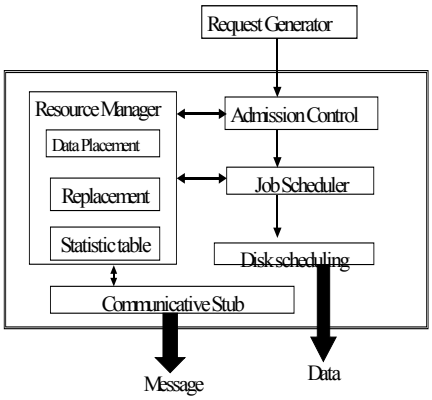

Figure 7. Simulation Environment

5.2 Video Data and Input Model

Block size B is default to 94KB and server round τ is about 250ms [10]. Given the rotation latency (t_{rot}), the average seek time (t_{seek}), and the transfer rate (r_{disk}) of disks, we can employ SCAN policy of disk scheduling and the disk capability M be deduced from Equation 5-1, where n denotes the number of requests [10,21-22].

$$M = \max\left\{n \middle| n * (t_{rot} + B / r_{disk}) + 2 * t_{seek} \le \tau\right\} \quad (5\text{-}1)$$

The number of videos is 1000. In order to model different size of videos, the content size is from 600MB to 1800MB. To model different scales of our proxy, the capacity of proxy is from 40GB to 640GB. Similarly, to model different scales of cooperative proxies, the proxy number is from 1 to 5 [15]. The network delay in LAN is 1ms and in WAN is 200ms [15]. Request generation is modeled as Poisson arrival process with Zipf distribution of video selection [10,20].

5.3 Preliminary Performance Evaluations

In the following subsections, we will assess the merits of our two-phase replacement scheme by the measurement of hit rate, bandwidth reduction, average network delay and average CPU time. We will evaluate our two modes and compare it with LRU, Size-max (always replace the victim which has maximum size), Size-min (always replace the victim which has minimum size) techniques. At last, we will evaluate our proposed cooperative video proxies architecture under various numbers of proxies.

5.3.1 Evaluation of Two Modes

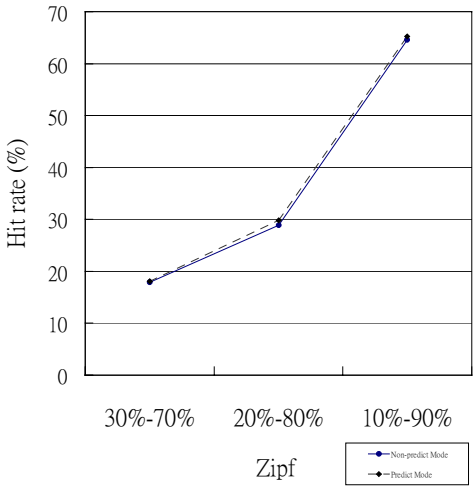

Figure 8. Hit rate v.s. Zipf

Figure 8 shows the change of hit rate under different Zipf distribution, while C=80GB. The more serious access skew is, the higher hit rate has. We can find that the access patterns are more skew, the number of hot videos is smaller so that the more clients can get videos from local proxy. Thus, it will have higher hit rate. Besides, the predict mode has higher hit rate than that of non-predict mode. Because even the access frequency of every video cached in local proxy is larger than that of the new video, the new video still has a chance to run Knapsack algorithm. However, we find that the hit rates of predict mode and non-predict mode are very closed

5.3.2 Comparisons with Other Policies.

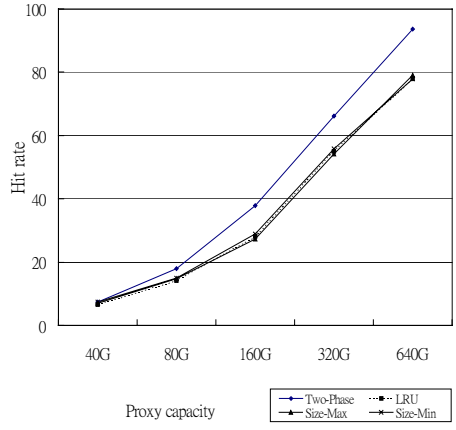

Figure 9. Hit rate v.s. Proxy capacity

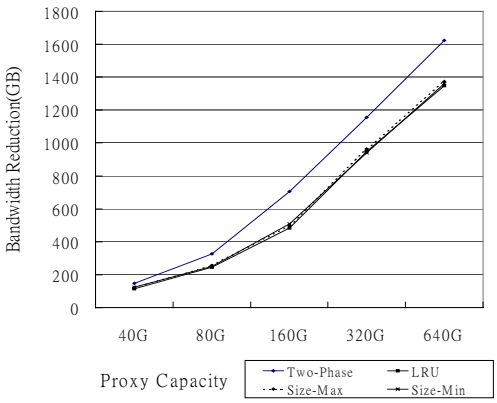

Figure10.Bandwidth Reduction v.s. Proxy Capacity

Figure 9 illustrates the proxy hit rate as function of the proxy capacity, for Two-Phase, LRU, Size-max, Size-min policies. The hit rate of our two-phase replacement is higher than that of other policies. And, we notice that as the proxy capacity grows, the performance difference between our policy and other policies becomes larger as well. Because when the number of cached videos grows, the probability to find unused videos in the proxy increases. It may increase the probability of overwriting a hot video in other policies. This situation hardly occurs in our two-phase replacement because we will check whether the popularity of the new video is larger than that of any cached video before replacing it.

As we discussed previously, a good replacement policy is not only increasing hit rate but also reducing network bandwidth. Figure 10 shows the proxy bandwidth reduction in one day as function of the proxy capacity, for Two-Phase, LRU, Size-min, Size-max policies. When the proxy capacity reaches 640GB, we can save about 277GB network bandwidth. Comparing our policy with other policies, we find that although hit rate doesn't increase obviously but bandwidth is saved so much. This is because the bandwidth reduction D_{local} is calculated by $Video_{frequency}$ * $Video_{size}$. We have the higher bandwidth reduction due to considering the size of video in our scheme. Thus, the size of video is an important factor to design replacement policy.

Average Network Delay Time =
Hit rate*D_{LAN} + Miss rate*D_{WAN} (5-2)

Figure 11 shows that the average network delay calculated by Equation 5-2. We can find that the two-phase scheme has the least average network delay because the two-phase scheme has higher hit rate so that the more number of clients can get data in the local proxy. Thus, the average network delay will decrease.

Although our method has higher performance, it may cause the penalty of the CPU time shown in Figure 12. When proxy capacity increases, the knapsack table grows so that the computation overhead will grow. However, as the capability of CPU is more powerful, the CPU time is less. It will not be a problem. Contrarily, to solve the problem of bandwidth or network delay time will be more significant.

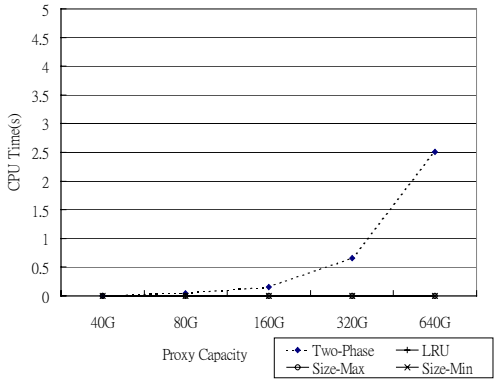

Figure 11. Average Network Delay Time

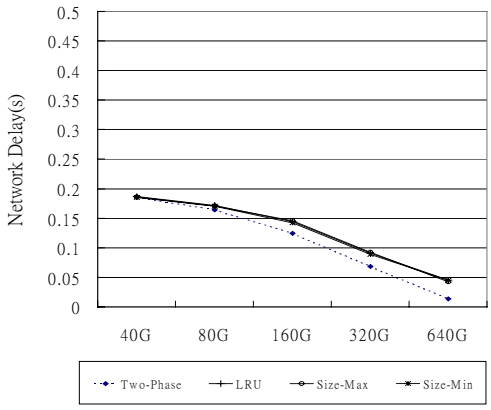

Figure 12. CPU Time

5.3.3 Number of Cooperative Video Proxies

Figure 13 shows that the hit rate increases as the number of proxy grows. We can find that the total hit rate has more improvement as the number of proxy is from one to four. While the number of cooperative proxies is from four to five, the improvement is limited. It is because that as the number of cooperative proxies grows, the variation of videos cached in cooperative proxies is limited. Thus, we suggest that the number of cooperative proxies within 5 is enough to design efficient cooperative video proxies.

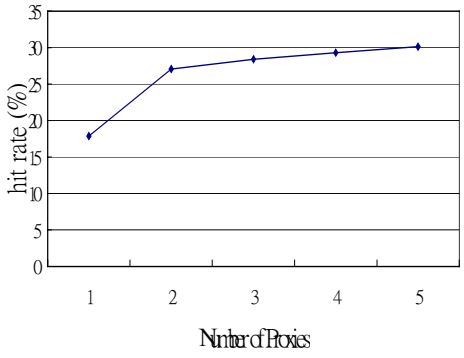

Figure 13. Hit Rate v.s. Number of Proxies

6.Concluding Remarks

With rapidly progress in information communication, the need of bandwidth is more and more. How to reduce the network bandwidth becomes very important. In this paper, we have proposed cooperative video proxies with a two-phase replacement scheme to save a lot of bandwidth so as to lighten the network load. We also build a simulation environment to evaluate the performance of our method. From our evaluation results, we will save more 147~1624GB bandwidth one day than without cooperative proxies under the number of proxies with one to four. If adopt our two-phase replacement scheme, we may save more 23~277GB than that of other policies under proxy capacity with 40GB to 640GB. Besides, the performance of predict mode is similar to non-predict mode. But, it needs more overhead of computing than non-predict mode. Thus, we suggest that the non-predict mode is more cost-efficiency than predict mode.

In the future, some significant issues need further exploration. In our video proxy, we cache a whole video so as to cache limited number of videos. In order to increase the number of cached videos, we can cache partial videos in local proxy. Thus, how to partition a video into several video segments and how to place these video segments into video proxies are interesting topics. We might partition a video into different number of segments according to its popularity. If the video is hotter, we place more number of video segments into the local proxy so as to increase the hit rate of the proxy. In the other aspect, in our replacement scheme, if the number of cached videos increases, the size of the knapsack table will become larger. Thus, how to reduce the knapsack table still is an important issue. We suggest that the videos can be classified according to their size. When a new video is coming, we will find its classification of videos in the proxy. From the members of this classification, we will choose one as a victim to be replaced by the new video. Because the number of the members is small, the size of the table can be reduced.

References

[1] Meira, W., Jr., Fonseca, E., Murta, C., and Almeida, V., "Analyzing Performance of Cache Server Hierarchies", *Proc. of XVIII International Conference of the Chilean Society on Computer Science*, pp. 113 –121, 1998.

[2] Y.B. Lee, "Parallel Video Servers : A Tutorial", *IEEE Multimedia* No. 5, Issue 2, pp. 20 –28, April-June 1998.

[3] A.; Das, Sun-Euy Kim and C.R., "Analyzing Cache Performance for Video Servers", *Proceedings of the 1998 ICPP Workshops on Sivasubramaniam, Architectural and OS Support for Multimedia Applications/Flexible Communication Systems/Wireless Networks and Mobile Computing*, pp.38 –47, 1998.

[4] Chiu, Y.M. and Yeung, K.H., "Partial video sequence caching scheme for VOD systems with heterogeneous clients", *Proc. Of 13th International Conference on Data Engineering*, pp.323 –332, 1997.

[5] Radhika Malpani, Jacob Lorch, and David A. Berger, "Making World Wide Web Caching Servers Cooperate", available at http://www.bmrc.berkeley.edu/research /publication/1995/138/paper-59.html.

[6] Hua, K.A.; Sheu, S.; Wang, J.Z, "Earthworm: A Network Memory Management Technique for Large-Scale", *Proceedings of IEEE* Volume: 3, pp.990 –997, 1997.

[7] Sen, S. and Rexford, J.; Towsley, D., "Proxy Prefix Caching for Multimedia Streams", *Proceedings. IEEE* Volume: 3, pp.1310 -1319 vol.3, 1999.

[8] Park Kyeongho, Yanghee Choi and Chong Sang Kim, "Scheduling of storage and cache servers for replicated multimedia data", *High Performance Computing on the Information Superhighway*, pp.484 –487, 1997.

[9] Michele Colajanni , P.S.Yu , and Daniel M.Dias, "Scheduling Algorithms for Distributed Web Servers", *Proc. of International Conference on Distributed Computing Systems*, pp.169-176, 1997.

[10] Y.Z. Hou, *An Effective Scheduling Policy in Distributed Video Server Environment*, Master Thesis, CSIE, NCTU, June, 1999.

[11] Tatarinov, I., Rousskov, A.and Soloviev, V., "Static Caching in Web Servers, Computer Communication and Networks", *Proc. of the Sixth International Conference on Computer Communications and Networks*, pp.410 –417, 1997.

[12] Sonah, B. and Ito, M.B. Modeling, "New Adaptive Object Replacement Policy for Video-On-Demand Systems", *Proc. of the Sixth International Symposium on Analysis and Simulation Computer and Telecommunication Systems*, pp.13-18, 1998.

[13] Charu C. Aggarwal, Joel L. Wolf and Philip S. Yu, **On Caching Policies for Web Objects**, *IBM Research Report*, RC20619, November, 1996.

[14] Charu C. Aggarwal and philip S.Yu, **On Disk Caching of web objects in proxy servers**, *IBM Research Report*, RC20636, November, 1996.

[15] Y. Chang, K.M. Yu, C.C. Wang, and C.N. Po, "An Efficient Cache Coherence Protocol for Cooperative WWW Caching", available at http://www.mi.chu.edu.tw/~ykchang/www-research.html.

[16] Wessels, D. and Claffy, K., **Internet Cache Protocol (ICP)**, *Technical Report Internet-Draft, IETE Network Working Group,* Version 2,April 1997.

[17] W.S. Huang, *An Effective Data Placement Scheme For Supporting Fault-Tolerance in Distributed Video Server Environment*,Master Thesis, CSIE, NCTU, June, 1999.

[18] David K. Smith, *Dynamic Programming*, pp.37-54, 1991.

[19] Silvano Martello and Paolo Toth, *Knapsack Problem*, pp.36-45, 1990.

[20] A. Dan, D. Sitaram and P. Shahabuddin, "Dynamic Batching Policies for An On-Demand Video Server", *Multimedia Systems*, Vol. 4, No.3, pp.112-121, 1996.

[21] B. Ozden, R. Rastogi and A. Silberschatz, "Disk String in video Server Environments", *IEEE international Conf. On Multimedia Computer and Systems*, pp.147-154, June 1995.

[22] J. Gafsi and E. Biersack, "Data String and Reliability Aspects in Distributed Video Servers", *Cluster Computing: Networks, Software Tools and Application*, Feb, pp.35-48, 1999.

[23] http://www.maxtor.com/techdocs/highcapacity.html