DYNAMIC CACHE RECONFIGURATION STRATEGIES FOR A CLUSTER-BASED STREAMING PROXY *

Yang Guo, Zihui Ge, Bhuvan Urgaonkar, Prashant Shenoy, and Don Towsley
Department of Computer Science
University of Massachusetts at Amherst
Amherst, MA 01002
{yguo,gezihui,bhuvan,shenoy,towsley}@cs.umass.edu

Abstract The high bandwidth and the relatively long-lived characteristics of digital video are key limiting factors in the wide-spread usage of streaming content over the Internet. The problem is further complicated by the fact that video popularity changes over time. In this paper, we study caching issues for a cluster-based streaming proxy in the face of changing video popularity. We show that the cache placement problem for a given video popularity is NP-complete, and propose the *dynamic first fit* (DFF) algorithm that give the results close to the optimal cache placement. We then propose *minimum weight perfect matching*(MWPM) and swapping-based techniques that can dynamically reconfigure the cache placement to adapt to changing video popularity with minimum copying overhead. Our simulation results show that MWPM reconfiguration can reduce the copying overhead by a factor of more than two, and that swapping-based reconfiguration can further reduce the copying overhead compared to MWPM, and allow for the tradeoffs between the reconfiguration copying overhead and the proxy bandwidth utilization.

Keywords: Streaming proxy, Cache reconfiguration, Cluster-based

1. Introduction

The high bandwidth and the relatively long-lived characteristics of digital video are key limiting factors in the wide-spread usage of streaming content over the Internet. The problem is further complicated by the fact that video popularity changes over time. The use of a content distribution network (CDN) is one technique to alleviate these problems. CDNs cache partial or entire

*This research was supported in part by the National Science Foundation under NSF grants EIA-0080119, ANI-0085848, CCR-9984030, ANI-9973092, ANI9977635, ANI-9977555, and CDA-9502639. Any opinions, findings, and conclusions or recomendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

videos at proxies deployed close to clients, and thereby reduce network and server load and provide better quality of service to end-clients [1–3]. Due to the relatively large storage space and bandwidth needs of streaming media, a streaming CDN typically employs a cluster of proxies at each physical location. Each such cluster can collectively cache a larger number of objects and also serve clients with larger aggregate bandwidth needs. In this paper, we study caching issues for such a streaming proxy cluster in the face of changing video popularity.

Each proxy within the cluster contains two important resources: storage (cache) space and bandwidth. Each video file requires a certain amount of storage and bandwidth determined by its popularity. Assuming that video files are divided into objects, we study the *cache placement problem*, i.e., whether to cache an object, and if we do, which component proxy to place it on so that the aggregate bandwidth requirement posed on the servers and the network is minimized. Furthermore, since video popularities vary over time (e.g., many people wanted to watch the movie *The Matrix* again in preparation of the release of its sequel *The Matrix Reloaded* causing it to be very popular for a few weeks), the optimal cache placement also changes with time. *The proxy must be able to deal with dynamically varying popularities of videos and reconfigure the placement accordingly*.

In this paper, we first consider the offline version of the cache placement problem. We show that it is an NP-complete problem and draw parallels with a closely related packing problem, the 2-dimensional multiple knapsack problem (2-MKP) [4]. Taking inspiration from heuristics for 2-MKP, we propose two heuristics—static first-fit (SFF) and dynamic first-fit (DFF)—to map objects to proxies based on their storage and bandwidth needs. We then propose two techniques to dynamically adjust the placement to accommodate changing video popularities. Our techniques attempt to minimize the copying overheads incurred when adjusting the placement. The *minimum weight perfect match* (MWPM) reconfiguration method minimize the copying overhead associated with such a placement reconfiguration by solving a bipartite matching problem. In order to further reduce the copying overhead, we propose *swapping-based reconfiguration*, which mimics the hill climbing approach [5] used in solving optimization problems. The swapping-based reconfiguration also naturally allows us to trade off proxy bandwidth utilization against copying overhead.

We evaluate our techniques using simulation. We find that DFF gives a placement that is very close to the optimal cache placement, and that both DFF and SFF outperform a placement method that does not take bandwidth and storage requirements into account. We then examine the performance of MWPM reconfiguration, and show that it can reduce the copying overhead by a factor of more than two. Finally, we show that swapping-based reconfiguration can further reduce copying overhead relative to MWPM, and that further copying overhead reduction can be achieved by decreasing the proxy bandwidth utilization.

In summary, we study the dynamic cache reconfiguration problem for a cluster-based streaming proxy in the face of changing video popularities. Our contributions are twofold:

- We show that the cache placement problem is NP-complete, and propose DFF algorithm that is found to give the results close to the optimal cache placement.
- We propose MWPM and swapping-based techniques that can reconfigure the cache placement dynamically to adapt to changing video popularity with minimum copying overhead.

The remainder of the paper is organized as follows. In Section 1.2, we describe the architecture of a cluster-based streaming proxy. We formulate the optimal cache placement problem and present the baseline strategies in Section 1.3. The techniques for dynamic reconfiguration of cache placement are presented in Section 1.4. Section 1.5 is dedicated to performance evaluation. Section 1.6 includes the related work, and Section 1.7 concludes the paper.

2. Architecture of cluster-based streaming proxy

A cluster-based streaming proxy consists of a set of component proxies as shown in Fig. 1. These individual proxies are connected through a LAN or SAN, and are controlled by a coordinator residing on one of the machines.

The coordinator provides an interface for clients and servers so that the cluster-based streaming proxy acts as a single machine proxy from the perspective of clients and servers. In addition, the coordinator provides the following functionalities to the component proxies inside a cluster-based proxy.

- Coordinate component proxies to serve client requests. A client request may require multiple cached objects from different component proxies with a certain timing relationship. The coordinator needs to orchestrate the component proxies to serve these requests.
- Monitor and estimate the popularity (access frequency) of multimedia objects.
- Compute the optimal cache placement based on the current content popularity, and conduct dynamic cache reconfiguration if the degree of the popularity change demands cache reconfiguration.

In this paper we focus on the dynamic cache reconfiguration issues for such a cluster-based streaming proxy. We assume perfect knowledge of client access information. Accurately monitoring the popularity of streaming objects and

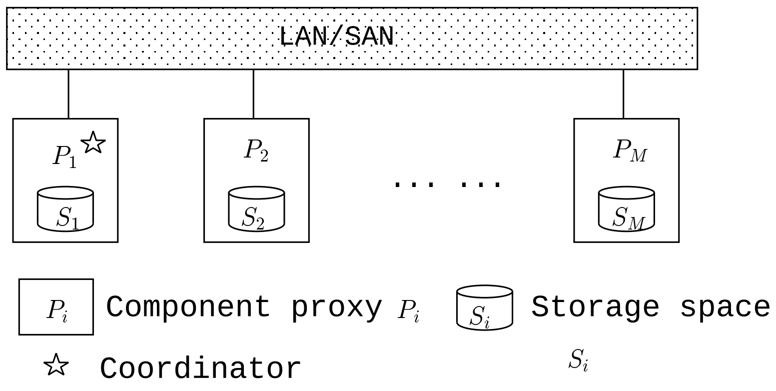


Figure 1. A cluster-based streaming proxy

efficiently coordinating component proxies to provide requested service are significant problems in their own right, and lie beyond the scope of this paper.

3. Optimal Cache Placement

In this section, we investigate the optimal cache placement problem for a cluster-based streaming proxy for a given video popularity. We formulate this problem as an integer linear programming problem, and show that the problem is NP-complete. We then present two baseline heuristics for it. In Section 1.4, we will show how these baseline heuristics can be enhanced to realize dynamic cache reconfiguration with minimum copying overhead.

Optimal Cache Placement: Problem Formulation

Consider a cluster-based proxy with M component proxies. Let S_j and Φ_j denote the available storage space and bandwidth at the j-th component proxy. Suppose that the cluster-based proxy services a set of videos that are divided into K distinct objects. Let x_i and b_i denote the storage space and bandwidth requirement of object i. The proxy caches a subset of the K objects to reduce the network bandwidth consumption on the path from remote servers to the proxy cluster. We focus on the problem of which objects to cache and where.

Let c_{ij} be a selection parameter that denotes whether object i is cached at proxy j—c_{ij} equals 1 if proxy j holds a copy of object i and is zero otherwise. Further, let ρ_{ij} denote the amount of bandwidth reserved for object i at proxy j (ρ_{ij} indicates how much of the aggregate demand for the object is handled by component proxy j). The objective of caching objects at the proxy is to *minimize* the bandwidth consumption on the network path from the remote servers to the proxy, or equivalently, to *maximize* the share of the aggregate client requests that can be serviced using cached videos. The resulting *optimal cache placement (O.C.P.)* problem can be formulated as follows:

$$\max \qquad B(\rho, c) = \sum_j \sum_i \rho_{ij} c_{ij} \tag{1}$$

$$\text{subject to:} \qquad \sum_i x_i c_{ij} \le S_j \tag{2}$$

$$\sum_i \rho_{ij} c_{ij} \le \Phi_j \tag{3}$$

$$\sum_j c_{ij} \rho_{ij} \le b_i \tag{4}$$

where $c_{ij} \in \{0, 1\}$, $i \in \{1, \ldots, M\}$, and $j \in \{1, \ldots, K\}$. $B(\cdot)$ denotes the allocated proxy bandwidth.

The solution to this problem yields values of c_{ij} and ρ_{ij} that completely describe the placement of objects and the bandwidth reserved for that object at a proxy. The solution may involve object replication across component proxies to meet bandwidth needs. Further, some objects may not be cached at any proxy, if it is not advantageous to do so.

PROPOSITION 3.1 *The optimal cache placement (O.C.P) problem is NP-complete. The problem is NP-complete even if all objects are of the same size.*

The proof is included in the Appendix.

Cache Placement Heuristics

We note that O.C.P. is similar to the 2-dimensional multiple knapsack problem (2-MKP) [4]. 2-MKP has one or more *knapsacks* that have capacities along two dimensions, and a number of *items* that have requirements along two dimensions. Each item has a profit associated with it. The goal is to pack items into the knapsacks so as to maximize the profit yielded by the packed items, while the capacity constraints along both dimensions are maintained. The component proxies and video objects in O.C.P. may be viewed as akin to the knapsacks and the items in 2-MKP respectively; the profits associated with the video objects are their bandwidth requirements. However there is an important difference between the two problems—the requirements of an item along both directions in 2-MKP are indivisible meaning the item may be packed in exactly one knapsack; the bandwidth requirement of a video object in O.C.P. is divisible and may be met by replicating the object on multiple component proxies.

Heuristics based on *per-unit weight* are frequently used for knapsack problems. Consequently, we define the *bandwidth-space ratio* of object i to be

b_i/x_i (the ratio of the required bandwidth and the object size), and the bandwidth-space ratio of proxy j to be Φ_j/S_j.

• **Static First-fit algorithm (SFF).** Static first-fit sorts proxies and objects in descending order of their bandwidth-space ratios. Each object (in descending order) is assigned to the first proxy (also in descending order) that has sufficient space to cache this object. If this proxy has sufficient bandwidth to service this object, the corresponding amount of bandwidth is reserved at the proxy, and the object is removed from the uncached object pool. On the other hand, if the proxy does not have sufficient bandwidth to service the object, the available bandwidth at the proxy is reserved for this object. The object is returned back into the un-cached object pool with the reserved bandwidth subtracted from its required bandwidth. The proxy is removed from the proxy pool since all of its bandwidth has been consumed. The algorithm is illustrated in Fig. 2.

• **Dynamic first-fit algorithm (DFF).** DFF is similar to SFF, except that the bandwidth-space ratio of a component proxy is recomputed after an object is placed onto that proxy and proxies are resorted by their new bandwidth-space ratios (in SFF, the ratio is computed only once, at the beginning). The intuition behind DFF is that the effective bandwidth-space ratio of a proxy changes after an object is cached, and recomputing this ratio may result in a better overall placement. In fact, as we will see in Section 1.5, DFF does perform better than SFF, and gives results close to the optimal cache placement.

So far we have focused on the optimal cache placement with fixed storage and bandwidth needs of a set of objects. In practice, bandwidth needs of objects vary over time due to changes in object popularities. The cache placement needs to be dynamically reconfigured in order to adapt to the changing popularities, e.g., newly popular objects may need to brought in from the servers, and cold objects may need to be ejected. We denote the number of objects that need to be transmitted among component proxies and from servers to proxy as the *copying overhead* of cache reconfiguration. In the following section, we study how to realize the cache reconfiguration with the minimum copying overhead.

4. Dynamic Cache Reconfiguration

A straightforward technique for dynamic cache reconfiguration is to recompute the entire placement from scratch using DFF or SFF based on the newly measured popularities, and bring in the objects from neighboring component proxies or remote servers. We denote such cache reconfiguration approaches as *simple DFF reconfiguration* or *simple SFF reconfiguration*. These approaches may yield close-to-optimal bandwidth utilization. However, they may cause many objects to be moved across proxies, resulting in excessive copying overhead as indicated by the simulation experiments in Section 1.5.

```
STATIC-FIRST-FIT (P, O)
sort P in descending order of bandwidth-space
  ratio
while ( O is not empty) {
  i = object with highest bandwidth-space ratio
  for (proxy j ∈ P in the sorted order) {
    if (x_i ≤ S_j) {
      c_ij = 1;      // cache object i
      S_j = S_j − x_i;
      if (b_i ≤ Φ_j)
        ρ_ij = b_i
        Φ_j = Φ_j − b_i
        remove object i from O
      else
        ρ_ij = Φ_j
        remove proxy j from P
        b_i = b_i − Φ_j;
        return modified obj. i into O
      break;
    } //end of if
  } //end of for loop
} //end of while loop
```

Figure 2. Static First-fit Placement Algorithm. Denote by P the collection of proxies that have bandwidth and space resources to provide caching service, and by O the collection of objects that have not been cached, or need additional bandwidth.

In the following, we propose two cache reconfiguration techniques that can reduce the copying overhead by exploring the existing cache placement. We first present the *minimum weight perfect matching reconfiguration method* (MWPM reconfiguration) by formulating the minimum copying overhead reconfiguration problem as a minimum weight perfect matching on a bipartite graph. We then describe the *swapping-based reconfiguration* method that mimics the hill-climbing approach [5] used in solving optimization problems. The swapping-based reconfiguration naturally allows us to trade the proxy bandwidth utilization for the copying overhead.

MWPM cache reconfiguration

Let $P_1, P_2, \ldots P_M$ denote the M proxies in the cluster. Let Ψ denote the current placement of objects onto the proxy cluster and let Ψ_{new} denote the new placement that is desired. There can be as many as $M!$ ways to reconfigure the placement. Ideally, we would like to reconfigure the placement such that the cost of moving (copying) objects from one proxy to another or from the server to a proxy is minimized.

The above problem is identical to the problem of computing the *minimum weight perfect matching* on a bipartite graph. To see why, we model the reconfiguration problem using a bipartite graph with two sets of M vertices. The first set of of M vertices represents the current placement Ψ. The second set of vertices represent the new placement Ψ_{new}. We add an edge between vertex P_u of the first set and vertex P_v of the second set if P_u has enough space and bandwidth to accommodate all the objects placed on P_v in Ψ_{new}. The weight of an edge represents the cost of transforming the current placement on the proxy represented by P_u to the new placement represented by P_v (the cost is governed by the number of new objects that must be fetched from another proxy or a remote server to obtain the new placement).

To illustrate this process, consider the example in Figure 3 with two identical proxies and five objects. In the current placement Ψ, the first three objects are placed on P_1 and the remaining two objects on P_2. The new placement Ψ_{new} involves placing $\{1, 2, 3\}$ on one proxy and $\{1, 5\}$ on the other proxy. Since the two proxies are identical, we add edges between both pairs of vertices in the bipartite graph. This is because either proxy can accommodate all the objects placed on the other proxy by the new placement. The weights on the edges indicate the cost of transforming each proxy's cache to one of these sets (e.g., transforming P_2's cache from $\{4, 5\}$ to $\{1, 5\}$ involves one copy whereas transforming it to $\{1, 2, 3\}$ involves three copies; deletions are assumed to incur zero overhead). It can be shown that finding a *minimum weight perfect match (MWPM)* for this bipartite graph yields a transformation with minimum copying cost. A perfect match is one where there is a one-to-one mapping from vertices in the former set to the latter; a minimum weight perfect match minimizes the weights on the edges for this one-one mapping.[1] Thus, in Fig. 3, leaving P_1's cache as is, and transforming P_2's cache from $\{4, 5\}$ to $\{1, 5\}$ is the minimum weight perfect matching with a copying cost of 1.

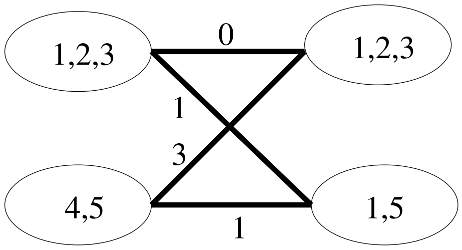


Figure 3. An example of using the minimum weight perfect matching to find the placement which minimizes object movement in a 2 node proxy.

Swapping-based cache reconfiguration

MWPM cache reconfiguration can significantly reduce the copying overhead, as indicated by the simulation experiments in Section 1.5. In this section, we describe a swapping-based reconfiguration technique that can further reduce the reconfiguration overhead. The swapping-based reconfiguration takes the current placement as the initial point, and use the well-known hill-climbing method [5] to solve the optimal cache placement (O.C.P.) problem. The reconfiguration cost incurred in MWPM is used as a control parameter to limit the overhead incurred by the swapping-based technique.

The hill-climbing method is characterized by an iterative algorithm that makes a small modification to the current solution at each step to come closer to the optimal solution. To apply the hill-climbing method to the cache reconfiguration problem, two issues need to be properly addressed: (1) how to modify the existing cache placement at each step to improve the proxy bandwidth utilization, and (2) when to stop. In the following, we address the above two issues.

Object swapping. We propose to swap the position of two objects at each step to modify the current placement. Our approach is to select a pair of objects such that the utilized proxy bandwidth increases after swapping. In fact, we select the pair that maximally increases the bandwidth utilization so as to minimize the copying overhead of reconfiguration.

Proxies are classified into two categories: overloaded proxies and underloaded proxies, as shown in Fig. 4. A proxy is said to be overloaded if the total bandwidth needs of objects currently stored at the proxy exceed capacity;

under-loaded proxies have spare bandwidth. All objects not currently stored on any proxy are assumed to be stored on a virtual proxy with bandwidth capacity zero (thus, the virtual proxy is also overloaded). The abstraction of a virtual proxy enables us to treat cached and uncached objects in a uniform manner. Intuitively, a "cold" object from an under-loaded proxy is selected and swapped with a "hot" object on an overloaded proxy, so that the total bandwidth utilization increases. We apply the following rules in selecting object:

- *Cold object selection*: Randomly select an underloaded component proxy with probability proportional to the amount of spare bandwidth, and then choose the least frequently accessed object in this proxy.
- *Hot object selection*: Select the object with the highest bandwidth requirement cached in the overloaded proxies or the virtual proxy.

These two objects are then swapped, and the corresponding proxies are relabeled as under-loaded or overloaded based on the new cache contents. [2] Randomization is introduced to overcome thrashing observed in experiments where two proxies repeatedly swap the same pair of objects, causing the algorithm to be stuck in a local minimum with no further improvement.

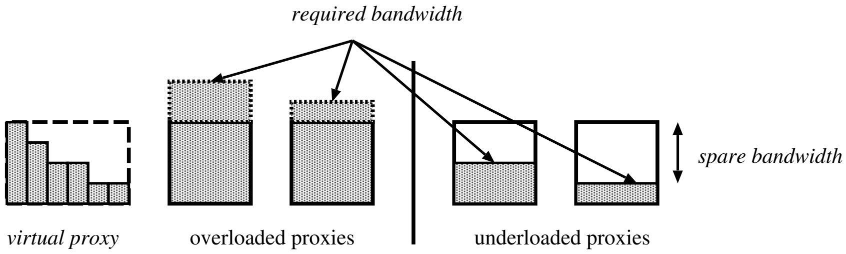


Figure 4. Selection of swapping objects in swapping-based reconfiguration

Termination condition. We now examine the termination condition of the swapping-based reconfiguration algorithm. Let $util_{DFF}$ denote the bandwidth utilization achieved by DFF and $cost_{DFF}$ denote the copying overhead of achieving this placement as computed by the MWPM reconfiguration. The swapping-based reconfiguration uses $(1 - \delta) \times util_{DFF}$ as its bandwidth utilization target, where δ ($0 \leq \delta < 1$) is a design parameter set by the proxy coordinator. Next, it runs the swapping-based heuristic to search for a placement that has a bandwidth utilization larger than $(1 - \delta) \times util_{DFF}$ but at a lower copying cost. The heuristic then chooses this placement, or reverts to the MWPM reconfiguration computed placement if the search yields no better placement. Thus, the swapping-based heuristic is run until one of the following occurs:

- The bandwidth utilization reaches the target $(1 - \delta) \times util_{DFF}$ at a cost lower than $cost_{DFF}$. Since a lower cost placement that closely approximates DFF is found, we pick this placement over MWPM DFF.

- The cost of the swapping-based heuristic reaches $cost_{DFF}$ but its bandwidth utilization is below $(1 - \delta) \times util_{DFF}$. No better placement is found, so we revert to the one computed using MWPM DFF.

By adjusting δ, we can trade the bandwidth utilization for the copying overhead. We will evaluate this in Section 1.5.0.0.

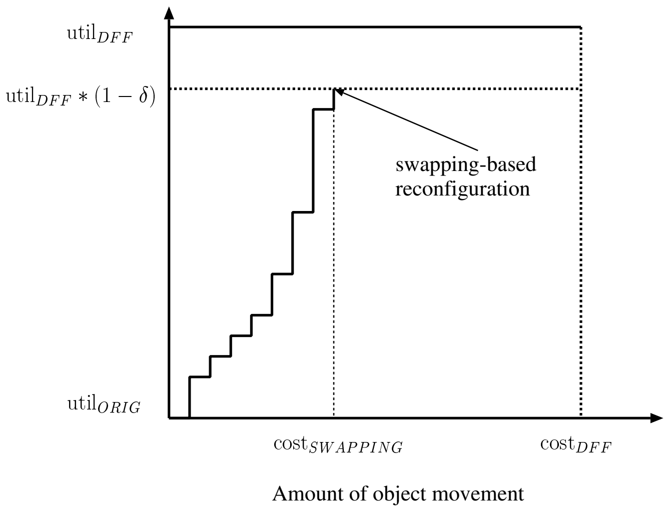


Figure 5. Swapping-based cache reconfiguration

5. Performance Evaluation

In this section, we conduct simulation experiments to evaluate the performance of MWPM and swapping-based reconfiguration algorithms. We start by evaluating the cache placement algorithms, DFF and SFF. We find that DFF yields a placement very close to the optimal cache placement, and both DFF and SFF outperform a placement method that does not take the bandwidth and storage requirement of objects into account. We then examine the performance of MWPM reconfiguration, and show that it reduces copying overhead by a factor of more than two. Finally, we show that the swapping-based reconfiguration can further reduce copying overhead in comparison to MWPM, and further copying overhead reduction can be achieved by decreasing the proxy bandwidth utilization.

Simulation setting

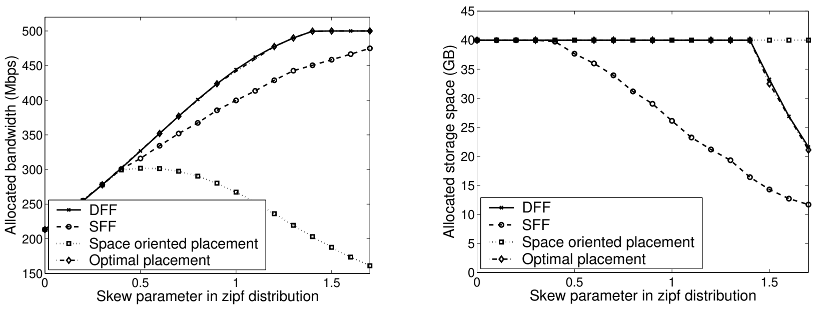
Assume that clients access a collection of 100 videos whose lengths are uniformly distributed from 60 minutes to 120 minutes. The playback rate of these videos is 1.5 Mbps (CBR), and their popularity obeys the Zipf distribution, i.e., the i-th most popular video attracts a fraction of requests that is proportional to $1/i^{\alpha}$, where α is the Zipf skew parameter. Each video is divided into equal sized segments and each segment is an independent cachable object. We consider a streaming proxy that consists of 5 component proxies. The bandwidth available at component proxy j, Φ_j, is 100 Mbps for $1 \le j \le 5$. We denote by S_j the storage space at the j-th proxy. We set S_i/S_{i+1} to be a constant for $1 \le i \le 4$, and denote it as *the space skew parameter*, p_s. The bandwidth-space ratios of component proxies can be tuned by adjusting p_s.

Evaluation of cache placement algorithms

We use the lp_solve [6] linear programming package to solve the optimal cache placement problem exactly. We also use *space oriented placement*, SOP, as the baseline algorithm. Space oriented placement uses storage space as the only constraint. It places the objects in descending order of access frequency, into proxies that are sorted in descending order of storage space.

We choose the aggregate storage space of 5 proxies to be 40 Gbytes. The proxy configuration is chosen in such a way that the bottleneck varies between proxy storage space and bandwidth based on the the proxy space skew parameter and clients' access behavior. In practice, the number of videos are much greater than 100 videos, so we assume that the storage space much larger than 40 Gbytes is required. The object size is set to be that of a one minute video segment. We will investigate the impact of object size later. The aggregate client request rate is 4 request/min.

DFF outperforms SFF and SOP consistently and achieves a proxy bandwidth utilization comparable to that of the optimal cache placement. Note that the computation time for obtaining an optimal cache placement using lp_solve is more than two hours on a 1GHz CPU, 1GB memory Linux box, while it takes a couple of seconds for DFF and SFF. Fig. 6 depicts the allocated bandwidth and used storage space as a function of Zipf skew parameter. Here we set the component proxy space skew parameter, p_s, to one, i.e., the storage space is evenly distributed among the proxies. We observe that DFF, SFF, and SOC all achieve the optimal bandwidth utilization when the Zipf skew parameter is less than 0.5. Intuitively, when the Zipf skew parameter is small, the client requests are evenly distributed among different videos. The number of client requests for different videos are comparable, and the required bandwidth is therefore nearly equal for all objects. The storage space at the proxy is the bottleneck

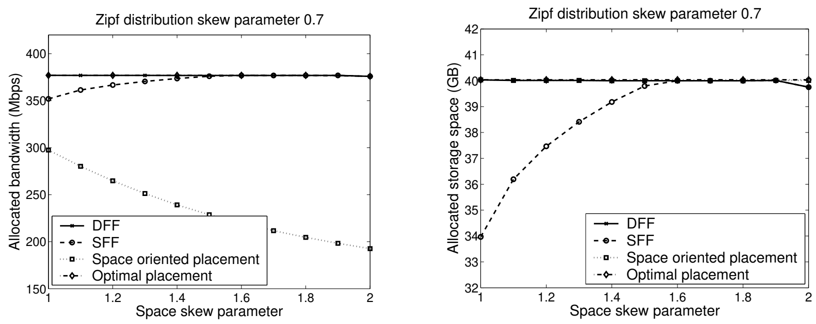


(a) Allocated bandwidth (b) Allocated storage space

Figure 6. Effect of Zipf distribution skew parameter

resource, and the maximum proxy bandwidth utilization is achievable as long as the proxy caches the “hot” objects and uses up the entire storage space.

DFF outperforms SFF and SOC as the Zipf skew parameter increases further. As the Zipf skew parameter increases, the discrepancy between the bandwidth required by “hot” and “cold” objects increases. As in the multi-knapsack problem, a right set of objects needs to be cached at each component proxy to fully utilize every component proxy’s bandwidth. SOC uses the storage space as the only resource constraint, and caches the hot objects on the component proxy with the largest storage space. Since the first component proxy consumes the hottest objects, the aggregate required bandwidth of these object surpasses this component proxy’s available bandwidth. Meanwhile, other component proxy’s bandwidth is wasted since only “lukewarm” or “cold” objects are available. The aggregate utilized bandwidth decreases as the Zipf skew parameter increases further. SFF does a better job since it considers the bandwidth as another resource constraint and stops caching the objects into a component proxy once its bandwidth has been fully utilized. However, SFF fails to balance the bandwidth and storage space utilization at component proxies. For instance, when the Zipf skew parameter is 0.7, two of the component proxies use up the bandwidth while having free storage space in SFF. On the other hand, the storage space is used up on the other three component proxies while bandwidth remains free. This is shown in Fig. 6(b), where SFF doesn’t fully utilize the storage space. In contrast, DFF distributes the hot objects among component proxies, and fully utilizes the storage space (which is the bottleneck resource compared to bandwidth for Zipf skew parameter equal to 0.7) and maximizes the utilization of proxy bandwidth.



(a) Allocated bandwidth (b) Allocated storage space

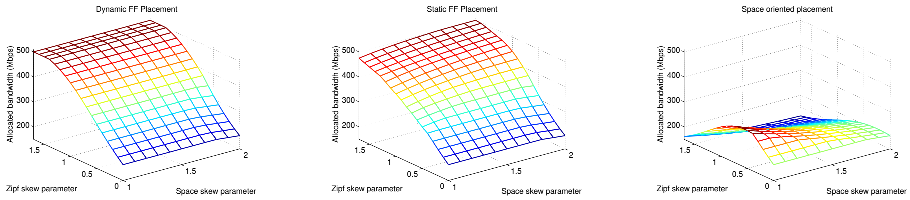
Figure 7. Effect of proxy space skew parameter (zipf skew parameter = 0.7)

Effect of space skew parameter. We further evaluate the performance of DFF and SFF in the case of a large space skew parameter. The space skew parameter, p_s, changes the storage space distribution among machines. For instance, when $p_s = 2$, the smallest storage space is 1.29 Gbytes. Hence the bandwidth-space ratios of proxies are widely skewed.

Again, DFF outperforms SFF and SOC, and achieves a proxy bandwidth utilization close to the optimal cache placement. Fig. 7 depicts the allocated bandwidth and allocated space as a function of the space skew parameters when the Zipf skew parameter is 0.7. We observe from Fig. 7(a) that the bandwidth is not fully utilized by SFF when $p_s < 1.6$. Notice that the storage space is also not fully utilized by SFF, as shown in Fig. 7(b). The failure of balancing the bandwidth and storage space utilization causes this behavior. However, as the space skew parameter increases, the performance of SFF improves. This is because the component proxy with large bandwidth-space ratio should cache more hot objects in order to achieve maximum proxy bandwidth utilization, and SFF happens to do that.

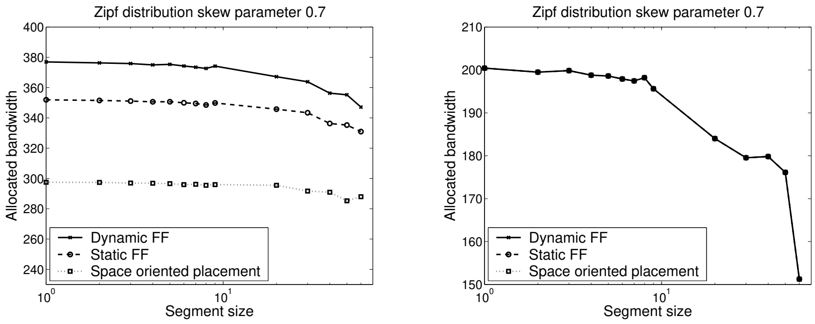
Fig. 8 depicts the performance of DFF, SFF, and SOC with respect to the proxy bandwidth utilization with different space and Zipf skew parameter. We observe that the utilized proxy bandwidth of DFF is consistently larger than that of SFF and SOC. We will use DFF in the following subsections.

Effect of object size. The object size may affect the utilization of proxy storage space; we investigate its impact in this section. In the results reported in the previous experiments, the object size is that of a one minute video segment. When we increase the object size while fixing the aggregate proxy storage space at 40Gbytes, the proxy bandwidth utilization gradually decreases as shown in Fig. 9(a) because the component proxy storage space is much larger



(a) Performance of DFF (b) Performance of SFF (c) Performance of SOC

Figure 8. Effect of zipf skew parameter and proxy space skew parameter on allocated bandwidth



(a) Storage space = 40 Gbytes (b) Storage space = 10 Gbytes

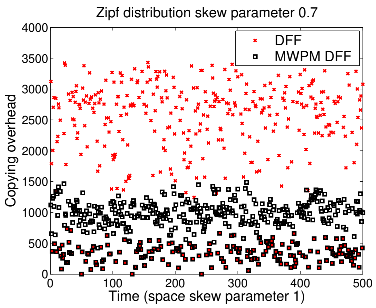
Figure 9. Effect of object size (zipf skew parameter = 0.7)

than the object size (the space skew parameter is selected to be one and Zipf skew parameter is 0.7).

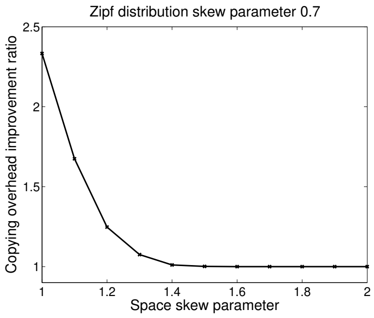
However, when the proxy storage space is relatively small, the impact of object size on performance becomes significant. Fig. 9(b) plots the proxy bandwidth utilization vs. the object size when the aggregate storage space is 10 Gbytes. The bandwidth utilization degrades dramatically as the object size increases beyond that of a 10 minute segment. The smaller object size helps better utilize the storage space and thus better utilize the bandwidth. We suggest that the object size be chosen sufficiently small compared to the component proxy storage space. In the following experiments we use an object size of one minute.

Evaluation of cache reconfiguration algorithms

The optimal cache placement changes over time as the video popularity varies. Hence the cache placement needs to be dynamically reconfigured. During the cache reconfiguration process, the objects have to be moved between the component proxies, or be brought in from remote servers in order to max-



(a) Copying overhead



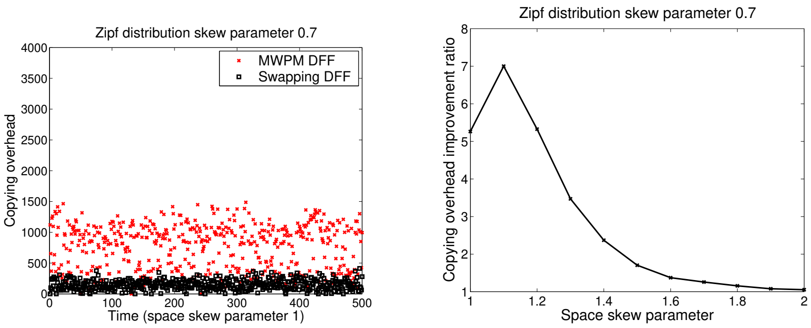
(b) Improvement ratio

Figure 10. Performance of MWPM cache reconfiguration

imize the proxy bandwidth utilization. The number of objects that need to be transmitted among the component proxies and from the servers is defined to be *copying overhead*. In the following, we evaluate the MWPM and swapping-based cache reconfiguration algorithm. We assume that the cache reconfiguration algorithm is executed periodically, and denote each period a round. We assume that 10% of the videos change their popularity ranks at the end of each round. DSS is used to generate the new cache placement according to the new video popularity.

Performance of MWPM cache reconfiguration. MWPM can significantly reduce the reconfiguration cost and is most effective when all component proxies have the same amount of storage space. Fig. 10(a) depicts the copying overhead with and without MWPM. The Zipf skew parameter is selected to be 0.7 and the space skew parameter is 1. MWPM can reduce the copying overhead by a factor of 2.3. On average, about 520 objects (about 7 to 8 videos) need to be transmitted to adapt to the new video popularity.

We further investigate how MWPM performs as the space skew parameter changes. Intuitively, as the space skew parameter increases, the storage space at component proxies become increasingly more skewed. Hence fewer edges exist between the old and new placements in the bipartite graph. This reduces the effectiveness of MWPM algorithm. Define *copying overhead improvement ratio* to be the ratio of average copying overhead without using MWPM to that using MWPM. Fig. 10(b) depicts the copying overhead improvement ratio vs. the space skew parameter. The effectiveness of MWPM decreases quickly as space skew parameter increases, and MWPM can not improve the copying overhead when the space skew parameter is larger than 1.4.



(a) Copying overhead (b) Improvement ratio

Figure 11. Performance of dynamic cache reconfiguration

Performance of Swapping-based reconfiguration. Swapping-based reconfiguration can further reduce the reconfiguration cost compared to MWPM. Fig. 11(a) depicts the copying overhead of swapping-based cache reconfiguration and MWPM with the Zipf skew parameter set to 0.7 and the space skew parameter set to 1. Here we choose δ to be 0.02, i.e., the swapping process stops once the allocated proxy bandwidth is within 98% of the allocated bandwidth computed by DFF. Swapping-based reconfiguration on average only needs to transfer 147 objects to adapt to the new video popularity. Fig. 11(b) depicts the copying overhead improvement ratio versus the space skew parameter. With the exception of space skew parameter of 1.1, the improvement ratio decreases as the space skew parameter increases, which suggests that we should configure the component proxy as homogeneous as possible.

• **Tradeoffs between reconfiguration cost and target proxy bandwidth utilization.** Fig. 12 depicts the reconfiguration copying overhead versus the target bandwidth utilization with 95% confidence interval. This curve is concave, where the copying overhead decreases quickly as the target bandwidth utilization decreases. This suggests that the swapping-based cache reconfiguration algorithm offers good tradeoffs between reconfiguration cost and target bandwidth utilization.

• **Sensitivity to changes in video popularity.** Swapping-based reconfiguration is more robust to a change in video popularity than MWPM. To investigate the sensitivity of a change in video popularity on the performance of swapping-based reconfiguration and MWPM, we vary the percentage of videos whose ranks change at each round. Fig. 13 depicts the average copying overhead vs. the video popularity change in terms of the percentage of videos (with 95% confidence interval). We first notice that for both MWPM and swapping-based reconfiguration, the copying overhead increases as video popularity change in-

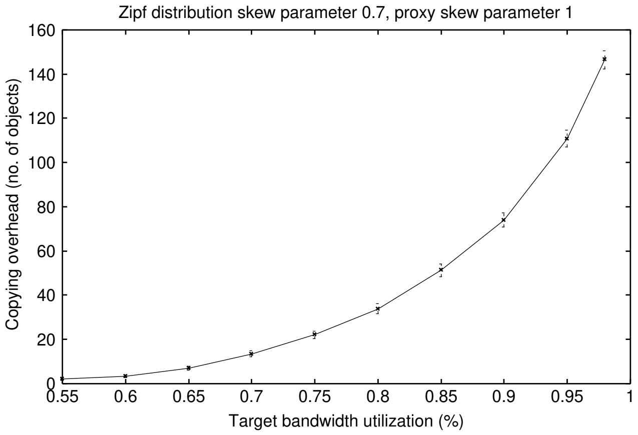


Figure 12. Tradeoffs of bandwidth utilization and copying overhead

tensifies. However, the copying overhead of MWPM is much larger than that of swapping-based reconfiguration, and the difference increases as the percentage of popularity change increases.

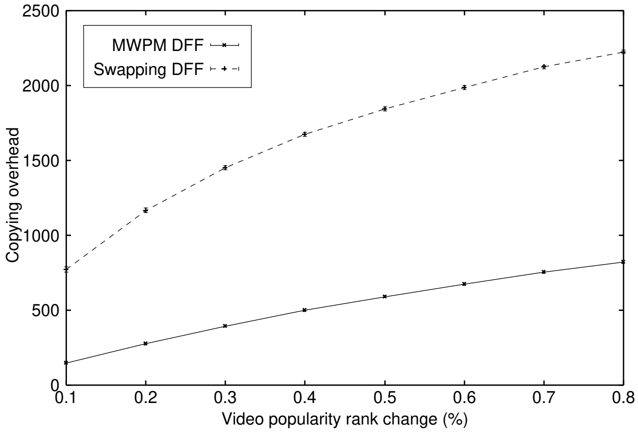


Figure 13. Sensitivity to the popularity change

• **Time-of-Day Effect.** The client request rate also varies over time. In the following, we examine the performance of MWPM and swapping-based reconfiguration under changing request rate. Suppose that the request rate is 10 requests/min during the peak hours (from 10am to 4pm), and 1 request/min during the off-peak hours. From 7am to 10am and 4pm to 7pm are two transition periods, during which the request rate linearly increases (decreases) from off-peak rate to peak rate. We assume that 10% of the videos change their ranks every 5 minutes. We use 5 minutes as the time period of a round. Fig. 14 depicts the copying overhead incurred by MWPM and the swapping-based reconfiguration, respectively, during a 24-hours time period.

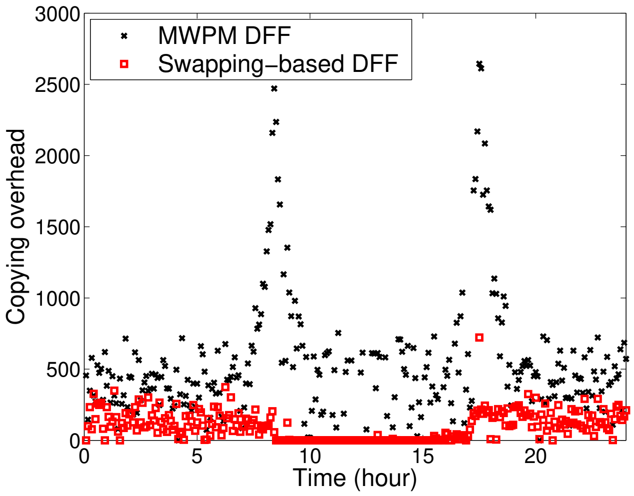


Figure 14. Impact of time-of-day effect and changing video popularity

The copying overhead of MWPM reconfiguration increases dramatically during the transition periods. Both the client request rate and the video popularity can affect the optimal cache placement. During the transition periods, the optimal cache placement changes more drastically since both video popularity and access rate are changing, leading to the high MWPM reconfiguration cost. In contrast, swapping-based reconfiguration continues to perform well even during in the transition periods. In off-peak hours, the swapping-based reconfiguration incurs less copying overhead than MWPM reconfiguration. During the peak hour, the copying overhead is even lower since now the proxy bandwidth is the bottleneck resource. There are more "hot" objects hence a lot of swapping is not necessary.

6. Related Work

In general, two bodies of work are related to streaming cache design, namely web caching for regular objects and distributed video server design.

The web caching systems such as CERN httpd [7], Harvest [8], and Squid [9] are designed for classical web objects and do not offer any support for streaming media. Our work is closest to [1, 3] where placement and replacement of streaming media were studied for proxy clusters. The Middleman proxy architecture described in [1] consists of collection of proxies coordinated by a centralized coordinator. Middleman caches only one copy of each segment in the proxy cluster, and uses demand-driven data fetch and a modified version of LRU-K local cache replacement policy called *HistLRUpick*. The work presented in [3] considers a loosely coupled caching system without a centralized coordinator. The system caches multiple copies of segments for the purpose of fault tolerance and load balancing. However, the above efforts focused on optimizing storage space at proxies; our results show that optimizing storage space alone is sub-optimal and that significant additional gains can accrue by considering both bandwidth and storage space constraints.

The effectiveness of using bandwidth and storage space-based metrics was demonstrated for single proxy environments in [10, 11] where a central non-cooperating architecture is assumed. Our techniques extend these works and other studies of single proxy cache placement [12, 2], and are specifically designed for proxy clusters that are typical in today's content distribution networks.

There are other related work in the area of multimedia servers [13–19]. In [20], a dynamic policy was proposed that creates and deletes replicas of videos, and mixes hot and cold versions so as to make the best use of bandwidth and space of a storage device. While the work in [20] focused on balancing the load on multiple storage devices, our focus is on maximizing the utilization of proxy bandwidth and the dynamic reconfiguration with minimum reconfiguration cost.

7. Conclusions and future work

In this paper, we considered the problem of caching popular videos at a proxy cluster so as to minimize the bandwidth consumption on the proxy-server path. We proposed heuristics for mapping objects onto proxies based on bandwidth and storage space constraints, and showed how these heuristics give the comparable results to the optimal cache placement. We further propose MWPM and the swapping-based reconfiguration schemes that handle dynamically changing popularities with minimum copying overhead. Our simulation results demonstrated the benefits of our approach, and showed that the swapping-based reconfiguration would allow the tradeoffs between the reconfiguration copying overhead and the proxy bandwidth utilization.

Future research can proceed along several avenues. We would like to further refine the cache reconfiguration techniques. For instance, we currently do not take into consideration the distance that an object is moved in the cache reconfiguration. How to compute the copying overhead reflecting the actual overhead consumed over the network remains an interesting problem. We would also like to reduce the switching overhead when servicing a client request that requires multiple objects residing in different component proxies. The cache placement should take this into account and try to place the objects of the same video in the same component proxy if possible. Finally, accurate estimate of the client request rate is an important and interesting research problem.

Appendix: *Proof of Proposition 3.1:*

O.C.P. problem $\in$ **NP**, since given an allocation, $\{\rho_{ij}\}$ and $\{c_{ij}\}$, validating them and compute objective function $B(\rho, c) = \sum_i \sum_j \rho_{ij} c_{ij}$ can be done in polynomial time.

To show that it is **NP**-hard, we prove that Equal-Size Partition(ESP) is polynomially reducible to BSCP problem, i.e., ESP $\leq_P$ BSCP problem. The Equal-Size Partition described below is an **NP**-complete problem [21]:

- INSTANCE: Finite set A, function $f : A \to \mathbf{Z}^+$.
- QUESTION: Can A be partitioned into 2 disjoint sets A_1, A_2, i.e., $A = A_1 \cup A_2$ and $A_1 \cap A_2 = \emptyset$, such that $|A_1| = |A_2|$ and $\sum_{a \in A_1} f(a) = \sum_{a \in A_2} f(a)$.

Given an instance of ESP with A and f, we can formulate an instance of the BSCP problem:

- Let $X = A$ and $\forall i \in X$, let $b(i) = f(i)$ and $x(i) = 1$.
- Let the number of proxies $M = 2$, and $S_1 = S_2 = \frac{|A|}{2}$, $\Phi_1 = \Phi_2 = \frac{\sum_{i \in A} f(i)}{2}$

By solving the BSCP problem, we can obtain an optimal allocation, $\{\rho_{ij}\}$ and $\{c_{ij}\}$. Now we establish $B(\rho, c) = \sum_{i \in X} f(i)$ if and only if an equal size partition exists for the ESP problem instance.

(i) If an equal size partition exists for $A = A_1 \cup A_2$, it is easy to verify that the following allocation $c = \{c_{ij}\}$ and ρ=$\{\rho_{ij}\}$ gives an optimal allocation: $B(\rho, c) = \sum_{i \in X} f(i)$.

$$c_{ij} = \begin{cases} 1 & i \in A_j \\ 0 & i \notin A_j \end{cases}$$

$$\rho_{ij} = \begin{cases} f(i) & c_{ij} = 1 \\ 0 & c_{ij} = 0 \end{cases}$$

where j =1 ,2.

(ii) If there exists ρ and c such that $B(\rho, c) = \sum_{i \in X} f(i)$, we construct $A_1 = \{i | c_{i1} = 1\}$ and $A_2 = \{i | c_{i2} = 1\}$. We need to establish that A_1 and A_2 equally partition A.

First, we show by contradiction that $A_1 \cup A_2 = A$.
Assume there exists $a \in A = X$, however $a \notin A_1$ and $a \notin A_2$, i.e., $c_{i1} = c_{i2} = 0$. Then,

$$B(\rho, c) = \sum_{i \in A} \rho_{i1} c_{i1} + \rho_{i2} c_{i2} \leq \sum_{i \in A / \{a\}} b(i) < \sum_{i \in X} f(i).$$

Therefore, a does not exist: $A_1 \cup A_2 = A$.
We now show that $|A_1| = |A_2| = \frac{|A|}{2}$ and $A_1 \cap A_2 = \emptyset$.
This is true since $|A_1| + |A_2| \geq |A|$ and by allocation constraint

$$\sum_{i \in X} x_i c_{ij} = \sum_{i \in X} c_{ij} = |A_j| \leq S_j = \frac{|A|}{2}, \qquad j = 1, 2.$$

Last, since, by allocation constraint,

$$\sum_{i\in X}\rho_{ij}c_{ij} \le \Phi_j = \frac{\sum_{i\in A} f(i)}{2}, \qquad j = 1,2$$

and

$$\sum_{j=1,2}\sum_{i\in X}\rho_{ij}c_{ij} = B(\rho, c) = \sum_{i\in A} f(i),$$

we have

$$\sum_{i\in X}\rho_{i1}c_{i1} = \sum_{i\in X}\rho_{i1}c_{i1} = \frac{\sum_{i\in A} f(i)}{2}$$

$$\Rightarrow \sum_{a\in A_1} f(a) = \sum_{c_{i1}=1}\rho_{i1}c_{i1} = \sum_{c_{i2}=1}\rho_{i2}c_{i2} = \sum_{a\in A_2} f(a).$$

Thus, A_1 and A_2 equally partition A.

This completes the proof. ∎

Notes

1. The bipartite graph can be constructed in $O(K+M)$ time and the minimum weight perfect matching is polynomial-time solvable with complexity $O(M^3)$.

2. No objects are physically moved at this time. Actual movement occurs after the algorithm is terminated.

References

[1] S. Acharya and B. Smith, "Middle man: A video caching proxy server," in *Proc. NOSSDAV 2000*, June 2000.

[2] R. Rejaie, H. Yu, M. Handley, and D. Estrin, "Multimedia proxy caching mechanism for quality adaptive streaming applications in the internet," in *Proc. IEEE INFOCOM*, April 2000.

[3] Y. Chae, K. Guo, M. Buddhikot, S. Suri, and E. Zegura, "Silo, rainbow, and caching token: Schemes for scalable fault tolerant stream caching," in *IEEE Journal on Selected Areas in Communications on Internet Proxy Services*, September 2002.

[4] P. Crescenzi and V. Kann(Eds.), *A compendium of NP optimization problems*. http://www.nada.kth.se/ viggo/problemlist/compendium.html.

[5] A. Bryson and Y. Ho, *Applied Optimal Control*. Taylor & Francis, 1975.

[6] "Lp_solve: Linear programming code." `ftp://ftp.es.ele.tue.nl/pub/lp_solve/`.

[7] T. Berners-Lee, A. Lutonen, and H. Nielsen, *Cern httpd*. http://www3.org/Daemon/Status.html, 1996.

[8] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Scwartz, and K. Worrell, "A hierarchical internet object cache," *Proceedings of the 1996 Usenix Technical Conference*, 1996.

[9] D. Wessels, *Icp and the squid cache*. National Laboratory for Applied Netowrk Research, 1999.

[10] R. Tewari, H. M. Vin, A. Dan, and D. Sitaram, "Resource-based caching for Web servers," in *Proc. SPIE/ACM Conference on Multimedia Computing and Networking*, January 1998.

[11] J. Almeida, D. Eager, and M. Vernon, "A hybrid caching strategy for streaming media files," in *Proc. SPIE/ACM Conference on Multimedia Computing and Networking*, January 2001.

[12] S. Sen, J. Rexford, and D. Towsley, "Proxy prefix caching for multimedia streams," in *Proc. IEEE INFOCOM*, April 1999.

[13] M. Buddhikot, G. Parulkar, and J. R. C. Jr., "Design of a large scale multimedia storage server," *Journal of Computer Netowrks and ISDN Systems*, 1994.

[14] W. Bolosky and et al., "The tiger video file-server," *Proceedings of NOSSDAV 96*, 1996.

[15] C. Bernhardt and E. Biersack, *The Server Array: A Scalable Video Server Architecture*. Kluwer Academic Press, 1996.

[16] P. Chen and et al., "Raid: High-performance, reliable secondary storage," *ACM Computing Surveys*, 1994.

[17] P. Shenoy, "Symphony: An integrated multimedia file system," *Doctoral Dissertation, Dept. of Computer Science, University of Texas at Austin*, 1994.

[18] R. Tewari, R. Mukherjee, D. Dias, and M. Vin, "Design and performance tradeoffs in clustered video servers," *proceedings of the IEEE ICMCS'96*, 1996.

[19] P. Lie, J. Lui, and L. Golubchik, "Threshold-based dynamic replication in large-scale video-on-demand systems," *Multimedia Tools and Applications*, 2000.

[20] A. Dan and D. Sitaram, "An online video placement policy based on bandwidth to space ratio (bsr)," *Proceedings of the 1995 SIGMOD*, 1995.

[21] M. R. Garey and D. S. Johnson, *Computers and Intractability A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.