Optimal Content Placement for a Large-Scale VoD System

David Applegate, Aaron Archer, Vijay Gopalakrishnan
Seungjoon Lee, K.K. Ramakrishnan
AT&T Labs Research, 180 Park Ave, Florham Park, NJ, 07932 USA

ABSTRACT

IPTV service providers offering Video-on-Demand currently use servers at each metropolitan office to store all the videos in their library. With the rapid increase in library sizes, it will soon become infeasible to replicate the entire library at each office. We present an approach for intelligent content placement that scales to large library sizes (e.g., 100Ks of videos). We formulate the problem as a mixed integer program (MIP) that takes into account constraints such as disk space, link bandwidth, and content popularity. To overcome the challenges of scale, we employ a Lagrangian relaxation-based decomposition technique combined with integer rounding. Our technique finds a near-optimal solution (e.g., within 1-2%) with orders of magnitude speedup relative to solving even the LP relaxation via standard software. We also present simple strategies to address practical issues such as popularity estimation, content updates, short-term popularity fluctuation, and frequency of placement updates. Using traces from an operational system, we show that our approach significantly outperforms simpler placement strategies. For instance, our MIP-based solution can serve all requests using only half the link bandwidth used by LRU or LFU cache replacement policies. We also investigate the trade-off between disk space and network bandwidth.

1. INTRODUCTION

Content and network service providers are facing an explosive growth in the demand for Video-on-Demand (VoD) content. To counter this and to scale the distribution, they are building out Video Hub Offices (VHOs) in each metropolitan area to serve subscribers in that area. Each of these offices has a large number of servers to store and serve videos. These offices are inter-connected using a high-bandwidth backbone. To deal with the high demand, providers currently replicate the entire library in all the locations. This allows them to circumvent problems such as content popularity prediction and overload due to flash crowds.

Despite disk space being plentiful and affordable for today's libraries, we believe that this approach is not only wasteful, but also economically infeasible. The cost of a gigabyte of storage is going down; however, the video libraries are also growing [6]. The rate of creation of content, the ever-increasing demand for high-quality content (e.g., high-definition, 3-D video), and the space needed to store them appears to be outpacing the ability of providers to economically add storage and replicate content everywhere.

Multiple studies [8,12] have observed "long tail" properties in the popularity of videos; this means that a large number of videos are requested infrequently. Hence storing copies of these videos in all locations is overkill. Inspired by this, we consider the problem of placing on-demand videos at the VHOs of a large-scale IPTV system. Our goal is to treat these VHOs as part of a large distributed store and distribute videos among them such that we can serve all users' requests, and do so efficiently. The problem of placing video content in large scale is quite challenging. It requires predicting what content is going to be popular, when it is going to be popular and where it is going to be requested most.

The traditional approach to address this problem is to take advantage of the skew in request popularity and use caches, as in applications such as DNS and Web [10,16]. When using caches, each video is replicated at a few locations. When a requested video is locally unavailable (*cache miss*), the video is fetched from a remote location that has a copy, and is then cached locally. When the cache at a location is full, videos are replaced using a replacement policy [15]. Thus, caches allow the system to dynamically adjust the number of copies of videos according to their current popularity. Caching, however, is extremely dependent on the size and stability of the *working set* (i.e., items being actively accessed at a given time). A cache miss imposes significant burden

ACM CoNEXT 2010, November 30 – December 3 2010, Philadelphia, USA.
Copyright 2010 ACM 1-4503-0448-1/10/11 ...$10.00.

on the system in the context of VoD as this results in a high-bandwidth stream transferred for an extended period of time. Further, a video being viewed needs to be retained in the cache for the length of the video, thereby occupying the cache for a long period. As we show in Section 4, the working set size can be quite large in typical VoD deployments and changes dramatically over time. This means that the caches have to be fairly large for them to be useful over long periods.

Another approach to video content placement is to employ optimization-based techniques [3, 6, 18]. However, most existing schemes are based on unrealistic simplifying assumptions (e.g., same demand pattern from all locations), a particular class of networks (e.g., tree topology), or heuristics to make the problem tractable. Also, many of them do not consider link capacity constraints, which is a crucial aspect in delivering high-quality videos for an extended period of time. Without link bandwidth constraints, the problem is a variety of facility location [3]. Adding them introduces the aspect of multicommodity flow [11], which makes the models computationally much more difficult. In this paper, we seek a general framework that can scalably find a placement solution *optimized for all the videos in the library across all locations with an arbitrary network topology and video demand pattern.*

We propose to pre-position videos so as to minimize the system resource usage while serving all requests and satisfying all disk and link capacity constraints. Our approach also enables us to explore the trade-off between the disk capacity requirement at each VHO and bandwidth. When a requested video is not available locally, we take advantage of the high-speed backbone to satisfy the request from a remote VHO.

We use a mixed integer programming (MIP) formulation to determine the placement of videos. The solution of the MIP tells us how many copies of each video are needed and where each copy should be placed so as to minimize the total network bandwidth consumption. However, due to the scale of the problem, we find that even the linear programming (LP) relaxation of our MIP is too large for off-the-shelf software (e.g., CPLEX) to find an optimal solution within a reasonable time. For instance, it took CPLEX more than 10 days to optimally solve an instance with 5K videos and 55 VHOs. We overcome this by employing a decomposition technique based on Lagrangian relaxation [5]. We thus obtain a near-optimal LP solution orders of magnitude faster (Section 5.3), and then use a rounding heuristic to convert it into a solution for our MIP.

Our algorithm is practical to run on realistic instances, even though there is no sub-exponential bound on the worst-case running time of our rounding heuristic (Section 5.4). In practice, the bulk of our running time is spent in solving the LP, not in the rounding phase. While producing the LP solution, our algorithm simultaneously proves a lower bound on the objective value of the optimal solution, via Lagrangian relaxation. Therefore, by comparing the objective value of our final integer solution with this lower bound, we can bound the gap between our solution and the optimal one. For the instances we have studied, arising from real-world large-scale systems, we have observed that the solutions are near-optimal (e.g., typically within 1-2%). However, we have not proven any worst-case performance guarantee for our rounding heuristic, so there may be some instances that exhibit worse optimality gaps.

For real-world applicability of the MIP-based strategy, we present a number of simple strategies to address practical issues (Section 6). Specifically, the MIP requires the demand for each video as an input. We make use of request history to predict the demand for videos. This, however, only lets us predict demand for videos already in the library.[1] In this paper, we use a simple strategy where we identify a similar video in the past (e.g., same TV series show) and use its request history to predict the demand for a new video. Our experimental results show that our simple strategy is quite effective. To overcome errors in our prediction, including unexpected flash crowds, we make use of a small LRU-cache at each location.

We have performed extensive simulations using real trace data from a nationally deployed VoD service and synthetic trace data generated from YouTube crawls [8]. Our results show that our approach outperforms existing caching schemes (Section 7). For the same amount of disk space, our approach only requires a little more than half the peak bandwidth as LRU or LFU schemes. The total number of bytes transferred is also significantly lower. Our results confirm that the approach scales well and can be used in practice to place libraries with hundreds of thousands of videos.

2. RELATED WORK

Content replication has been a topic of extensive research. There exists a large body of work related to file placement, which typically focuses on a relatively small system connected through a local area network. We refer readers to the survey by Dowdy and Foster [9] and the references therein. Zhou and Xu [20] consider the problem of minimizing the load imbalance among servers subject to disk space and network bandwidth constraints. However, they only consider egress link capacity from servers. Our focus is different in that we consider the link constraints inside a backbone network.

There have also been several efforts to address the problem of content placement. Valancius et al. [18] propose an LP-based heuristic to calculate the number of

[1]Predicting the popularity of new videos is an active area of research [2, 14].

video copies placed at customer home gateways. Borst et al. [6] focus on minimizing link bandwidth utilization assuming a tree structure with limited depth. They formulate an LP and observe that assuming symmetric link bandwidth, demand, and cache size, they can design a simple local greedy algorithm that finds a solution close to optimal. Both proposals focus on a particular network structure (e.g., tree) that is applicable for the distribution network to consumers. In contrast, our work considers arbitrary networks with diverse disk and link bandwidth constraints, where different locations show different video request patterns. Our work is therefore applicable for content placement in the backbone as well. We also consider popularity change over both short and long term, which poses a significant challenge in maintaining link bandwidth constraints.

Baev et al. [3] consider the *data placement problem* where the objective is to minimize the cost without taking bandwidth into account. They prove their problem is NP-hard via a reduction to the uncapacitated facility location problem, and present a 10-approximation algorithm for the case of uniform-length videos. They also show that for the non-uniform case, even deciding feasibility is NP-hard, so no approximation algorithm is possible unless P=NP. Our problem is strictly more complex, since we also consider link constraints. Since the data placement problem is a special case, our problem is also NP-hard, and suffers from the same inapproximability result. This motivates our decision to design an algorithm with per-instance performance guarantees, since proving an approximation guarantee applying to all instances would imply that P=NP.

Content Distribution Networks have also been the focus of recent research. Qiu et al. [16] consider the problem of positioning web server replicas to minimize the overall cost. Others have focused on ways to direct user requests to replicated servers (also known as *request routing*) [1, 4]. Most existing work, however, focuses on minimizing latency given constraints (e.g., server capacity), but do not consider replicating individual content while taking into account backbone network bandwidth.

Peer-to-peer (P2P) based schemes that reduce the backbone bandwidth usage have been proposed for VoD delivery [13]. However, such peer-to-peer systems still need to use VoD servers when content delivery from peers is not possible. Thus, our work is complementary to these since we consider how to place content on the VoD servers. Zhao et al. [19] focus on evaluating network bandwidth requirements for a single file when they vary tree construction and streaming delivery strategies in different network settings. By contrast, we focus on how to consider the skew in content popularity and replicate content to minimize the backbone network bandwidth consumption.

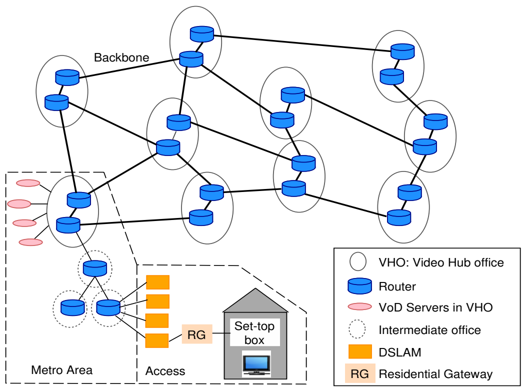

Figure 1: Typical architecture for IPTV service.

3. SYSTEM ENVIRONMENT

We assume a typical IPTV network with an associated VoD service in which the provider has offices (i.e., VHOs) in multiple metropolitan areas that are connected via a high-bandwidth backbone. Each office may cover one or more cities; we use the term "metro area" to describe all the cities serviced by a single office. Each of these offices has a certain amount of storage, which may be the same across all offices or may vary based on the size of the metro area (we experimentally study the effect of this heterogeneity.) We consider a scenario where each office has storage space to hold only a subset of the videos offered through the catalog. A VHO receives and satisfies all the requests from users in its metro area. In case a video is not available at the local office, we assume that the system has the ability to deliver the video from a remote office to satisfy the user's request transparently. In this case, we assume a pre-determined path between the VHOs (e.g., based on shortest path routing), which is more realistic than arbitrary routing [11]. Figure 1 shows a typical setup.

It is important to note that the links between offices need not be dedicated for the VoD service and may be shared with other services. Similarly, only a portion of storage space may be available for the VoD service. Hence, the goal of our work is to find an optimal operating point which balances the trade-off between the amount of storage space used and the amount of network bandwidth needed, while satisfying all requests. Also note that while we have considered a typical IPTV environment in this paper, our solution is applicable to a CDN setting where a user's request can be dynamically directed to any CDN location.

4. CHALLENGES

In this section, we show why content placement is challenging and provide evidence that using simple cache replacement policies alone will not suffice. Our analysis uses traces from a nationally deployed VoD service.

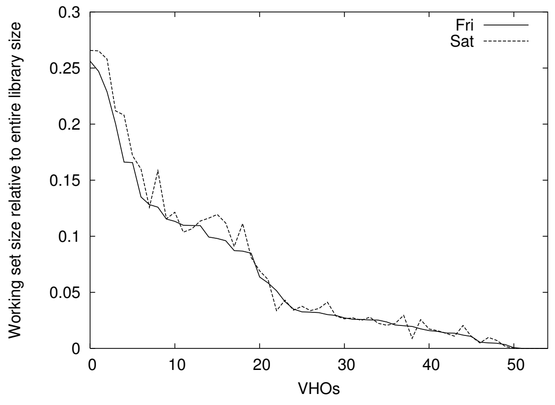

Figure 2: Working set size during peak hours.

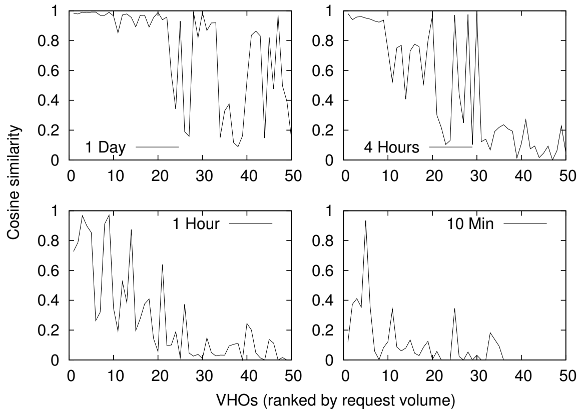

Figure 3: Similarity in videos requested during different time windows.

4.1 Large working set sizes

In Figure 2, we count the number of distinct videos requested from each VHO (i.e., the "working set") during the peak hour of a Friday and a Saturday (the two busiest days of the week). We observe that the working set size (disk space) for certain VHOs is large compared to the entire library size. The maximum is around 25% of the entire library, and about 10 of the VHOs see requests for almost an eighth of the library size. Thus, a proportionally large cache is necessary to store these videos. We also show the impact of a simple LRU caching policy on the miss ratio in Section 7.

4.2 Time-varying demand pattern

We observe from VoD request traces that the demand pattern for videos at each VHO significantly changes even over short periods of time. To quantify this (relatively rapid) change in request pattern, we use the cosine similarity metric that is commonly used in Information Retrieval.[2] For a given time window size, we partition the entire time duration into multiple intervals of the same size. Then, for each interval, we model the request pattern at a VHO as a vector composed of the request count for each video during the interval. In Figure 3, for each VHO, we compute the cosine similarity between the vector for the interval containing the peak demand instant and the vector for the previous interval. We also vary the time window size to change the aggregation time granularity. We observe that while the request mix is similar across days, there are significant differences in the request mix as the time window size decreases. This indicates that caches employing simple replacement policies have to be provisioned carefully as it can result in significant 'cycling' of the cache.

[2]Given two vectors v_1 and v_2, the cosine similarity is $\frac{v_1 \cdot v_2}{|v_1||v_2|}$. The value is between $[0, 1]$, with 1 indicating that the two vectors are alike.

5. MIP FORMULATION

In this section we formulate the mixed integer program (MIP) we use to determine the optimal content placement. Given a request pattern for each video at each VHO over a time period, our goal is to find a video placement that minimizes the total network consumption while *satisfying all user requests* within the link bandwidth and disk capacity constraints.

5.1 Input parameters

Table 1 summarizes the symbols used and their meaning. The top section lists the input parameters, which our MIP treats as fixed. Let V denote the set of VHO locations, L the set of directed links between these locations, and M the set of videos in our catalog. The set of time slices at which we enforce the link bandwidth constraints is T. Each VHO i has disk capacity D_i, and the size of video m is s^m. For each pair of VHOs $i, j \in V$, we assume that there is a fixed directed path P_{ij} from i to j. For the purposes of the MIP, only the set of links used in the path matters (not their order), so we take $P_{ij} \subseteq L$. Serving a request locally requires no links, so $P_{ii} = \emptyset$. The capacity of link $l \in L$ is B_l, while the bit rate of video $m \in M$ is r^m (both in Mbps). For each video $m \in M$, VHO $j \in V$ receives a_j^m requests during the entire modeling period, but at any given time slice $t \in T$, the number of concurrent streams is $f_j^m(t)$. This includes not only the requests initiated at time t, but also those that start before t and are still being streamed.

We denote the cost of serving one GB of video from i to j by c_{ij}. We focus on the scenario where the cost of remote service is proportional to the number of hops between i and j. Specifically, we use:

$$c_{ij} = \alpha|P_{ij}| + \beta, \tag{1}$$

where $|P_{ij}|$ denotes the hopcount from VHO i to VHO j, α the cost of transferring video over any link in L, and β a fixed cost for serving a request locally at any VHO (e.g., lookup cost). While we expect that the values of α and β would, in practice, take into account the monetary cost of storing and transferring the video, we show in Section 5.2.1 that the actual values do not affect which solution is optimal.

5.2 MIP model

Our MIP model has just two types of decision variables. For each VHO $i \in V$ and each video $m \in M$, y_i^m is a binary variable indicating whether we should store m at i (i.e., yes, if $y_i^m = 1$; no, if $y_i^m = 0$). When a request for video m arrives at VHO j, it is served locally if the video is stored at j; otherwise, it must be fetched from some other VHO storing m. If there are multiple such VHOs, then j chooses which one to use. The variable x_{ij}^m tells what fraction of requests should be served from VHO i. In the event that the $\{x_{ij}^m\}_{i\in V}$ are strictly fractional for some $m \in M$, $j \in V$, there are multiple approaches to implement it in practice (e.g., weighted round-robin, interleaving, etc.). In our experiments, for simplicity we select a server i (that has video m) at random with probability x_{ij}^m and fetch the entire video m from i. If $x_{jj}^m = 1$, this means that j serves the requests itself (because it has the content locally).

Our objective for the content placement problem is to minimize the cost of the total byte transfer, subject to disk space and link bandwidth limits. This can be formulated as the following MIP:

$$\min \sum_{m\in M} \sum_{i,j\in V} s^m a_j^m c_{ij} x_{ij}^m \tag{2}$$

$$\text{s.t.} \sum_{i\in V} x_{ij}^m = 1, \ \forall m \in M, j \in V \tag{3}$$

$$x_{ij}^m \le y_i^m, \ \forall i,j \in V, m \in M \tag{4}$$

$$\sum_{m\in M} s^m y_i^m \le D_i, \ \forall i \in V \tag{5}$$

$$\sum_{m\in M} \sum_{\substack{i,j\in V:\\ l\in P_{ij}}} r^m f_j^m(t) x_{ij}^m \le B_l, \ \forall l \in L, t \in T \tag{6}$$

$$x_{ij}^m \ge 0, \ \forall i,j \in V, m \in M \tag{7}$$

$$y_i^m \in \{0,1\}, \ \forall i \in V, m \in M \tag{8}$$

The objective expressed by (2) is to minimize the overall cost of serving all the requests, in terms of network consumption, for the entire period. Constraint (3) ensures that the total fraction of requests served locally and remotely is 1, for each <video,VHO> pair. Constraint (4) captures the fact that location i can serve video m only when it has chosen to store a copy locally. Constraint (5) reflects the limited disk space at each VHO, while constraint (6) captures the bandwidth limit for each link at each time slice. We include constraint (8) because we always store either the entire video or none of it at a VHO. If we wanted to break up videos into chunks and store their pieces in separate locations (as typically considered in some other studies), we could accomplish that by treating each chunk as a distinct element of M. Constraints (3) and (4) combine to ensure that every video is stored in at least one VHO. (That is, they imply $\sum_{i\in V} y_i^m \ge 1$, $\forall m \in M$.)

parameter	meaning
V	set of VHOs (vertices)
L	set of directed links
M	set of videos (mnemonic: movies)
T	set of time slices
D_i	disk capacity at $i \in V$ (in GB) reserved for fixed storage
s^m	size of video $m \in M$ (in GB)
P_{ij}	set of links on path used by $i \in V$ to serve requests at $j \in V$
B_l	capacity of link $l \in L$ (in Mbps)
r^m	bitrate of video $m \in M$ (in Mbps)
a_j^m	aggregate # of requests for video $m \in M$ at VHO $j \in V$
$f_j^m(t)$	# of requests for video $m \in M$ at VHO $j \in V$ that are active at time $t \in T$
c_{ij}	cost of transferring one GB from $i \in V$ to $j \in V$
decision variable	**meaning**
y_i^m	binary variable indicating whether to store video $m \in M$ at VHO $i \in V$
x_{ij}^m	fraction of requests for video $m \in M$ at $j \in V$ served by $i \in V$

Table 1: Input parameters and decision variables used in the MIP

5.2.1 *Cost coefficients matter little*

In (1), we introduce two coefficients α and β for the cost of serving content. In fact, since all feasible solutions satisfy Constraint (3), we can prove the following. (We omit the proof for the lack of space.)

PROPOSITION 5.1. *The set of optimal solutions is independent of the values α and β in (1), provided $\alpha > 0$.*

5.2.2 *Transfer cost due to video placement*

The above objective function (2) only considers transfer cost due to video requests. However, realizing a placement solution requires transferring videos to the location. Suppose we have a placement solution where i stores a copy of video m. Let us assume there is an origin o for all videos and the transfer cost between o and i is c_{oi}. We can easily account for this cost by including a second term in our objective function:

$$\sum_{m\in M} \sum_{i,j\in V} s^m a_j^m c_{ij} x_{ij}^m + w \sum_{m\in M} \sum_{i\in V} s^m c_{oi} y_i^m, \tag{9}$$

where w is a parameter that allows a different weight given to the additional transfer cost. For example, with w=0, (9) reduces to (2), and with w=1, we treat the transfer due to placement the same as any transfer due to a user request. Note that the initial placement of videos into the library is done before being made available to users. This can be achieved in multiple ways (e.g., using DVDs or using spare capacity, without regard to real-time deadlines). However, incremental updates to implement a new solution can potentially incur considerable cost. We use the equation (9) when we evaluate the impact in Section 7.8.

5.3 Finding a Fractional Solution

As mentioned in Section 2, finding an optimal solution to the above MIP is NP-hard [3]. Instead, we first solve the LP obtained by relaxing the integral constraint (8) to be just $y_i^m \geq 0$, allowing fractional values for y_i^m. Then, based on the fractional solution, we apply a rounding heuristic to obtain an integer solution, as described in Section 5.4.

Although LPs can be solved in polynomial time, our instances are too large to solve efficiently even with a state-of-the-art commercial LP solver. To overcome this challenge, we use an approach based on the potential function method described by Bienstock [5]. This approach computes a solution to the LP that strictly satisfies all of the constraints and achieves an objective value that is provably within 1% of optimal.

The basic idea is to use Lagrangian relaxation to relax the disk space and link bandwidth constraints ((5) and (6)) and incorporate these terms into the objective function using Lagrangian multipliers. This causes the LP to decompose into a bunch of independent (fractional) uncapacitated facility location problems [7], one for each video. The Lagrangian multipliers are exponential in the violations of the relaxed constraints, and are also used to define a potential function that we minimize using gradient descent. We iteratively find solutions to the 1-video sub-problems and update the overall solution and Lagrangian multipliers, using a line search to improve the potential function.

In addition, after every fixed number of ordinary iterations, we identify a subset of variables with non-negligible values in the overall solution. Then, we find a feasible fractional solution from a restricted version of the original LP consisting of only those variables, which is typically much smaller than the full LP. Also, by using the optimal dual variables from the restricted LP, we can obtain very good lower bounds, which allow us to check how close our current solution is to optimal and terminate if the gap is within the desired 1%.

On the instance mentioned earlier, while CPLEX used over 100 GB and yet could not solve in 10 days, our code uses 1 GB and takes about 1 hour to find a feasible fractional solution whose objective is within 1% of optimal. Both experiments were run on a machine with a 64-bit 1.5 GHz Itanium processor and 256 GB of memory.

5.4 Rounding

Based on the fractional solution, we round the fractional y_i^m to integer values by sequentially solving a MIP

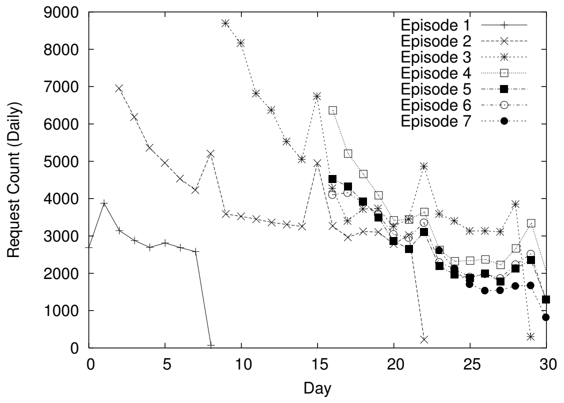

Figure 4: Daily request count for different episodes of a series show

for each video m, processed in order of decreasing total demand. If all of the y_i^m for video m are already integer, the y_i^m and x_{ij}^m are unchanged. Otherwise they are replaced with the solution to a MIP which minimizes the cumulative increase (over the videos processed so far) in the disk (5), link bandwidth (6), and objective (2), while enforcing the service constraints (3) and (4) and variable bounds (7) and (8). (Since our goal is to minimize the exceedence over the disk and link capacity limits as well as the fractional solution objective, a decrease is not rewarded, but an increase is penalized.) Since the impact of rounding y_i^m on the disk constraint (5) depends significantly on the video size s^m, we maintain separate cumulative increases for each value of s^m.

This MIP for rounding video m is an (integer) uncapacitated facility location problem with piecewise-linear costs. Even though these MIPs are NP-hard, they tend to be very easy to solve in practice. In our experiments, the entire rounding process, solving a MIP for every video m with a fractional y_i^m, only took about 5 minutes. Also, although there is no guarantee for the performance of this rounding heuristic, it only leads to around 1-2% overage for disk, link bandwidth, and the objective when compared to the fractional solution for the instances we studied.

6. PRACTICAL CONSIDERATIONS

We address a number of practical considerations for the algorithm design and parameter selection in real-world deployments in this section.

6.1 Demand estimation

Our MIP formulation needs the demand for videos to compute placement. This, however, is not known *a priori*. Our approach is to use the recent history (e.g., the past 7 days) as a guide to future demand for the videos. We use this history as an input to our formulation.

While history is available for existing videos, new videos are added to the library continually. Further, from our analysis, we find that many such newly added videos receive a significant number of requests. Hence we also need to address the problem of placement of new content into the system. While demand estimation for such new videos is an active area of research [2, 14] and beyond the scope of this paper, we use a simple estimation strategy. It is based on the observation that a significant number of the newly added videos belong to TV series, and that videos from a TV series exhibit similar demand patterns. Figure 4 presents the daily request count for different episodes of a particular series show during one month. Although there is some variation, we observe considerable similarity in the request volume for each episode of the series. For instance, on the day of release, episode 2 was requested around 7000 times, and episode 3 around 8700 times. In our system, we base our demand estimate for a new episode of a TV series on the requests for the previous week's episode of the same series (e.g., request pattern of episode 2 is used as demand estimate for episode 3). We show the effectiveness of our approach in Section 7.8.

We use another simple estimation strategy for blockbuster movies. From exogenous information [2], we assume that we are informed of a list of blockbuster movies (e.g., 1–3 movies each week). Then, we take the demand history of the most popular movie in the previous week and use it as the predicted demand for the blockbuster movies that are released this week.

Complementary caching: While TV shows and blockbuster movies account for the majority of requests for new videos — series episodes account for more than half of the requests for new releases — we still do not have a demand estimate for the remaining new videos (music videos, unpopular movies, etc.). Our current system uses a small LRU cache to handle load due to new releases for which we do not have estimates. This cache also handles unpredictable popularity surges to some videos (which is often why LRU caches are used).

6.2 Time-varying demand

We observe that the request pattern changes quite significantly over time, both in aggregate intensity and its distribution over the individual items in the library. For instance, users typically make significantly more requests on Fridays and Saturdays, while the traffic mix during the peak intervals on those two weekend days are quite different. The bandwidth required to serve these requests will correspondingly vary when they are served from remote VHOs. The placement should be able to handle such change and still satisfy the link constraints throughout the entire time period that the placement would remain before it is re-evaluated.

While accounting for link utilization at all times (e.g., each second during a 7-day interval) might guarantee that we never exceed the link constraint, it makes the problem computationally infeasible as the number of link bandwidth constraints (6) is proportional to the number of time slices in $|T|$. We find that the demand during non-peak periods does not overload any links. Therefore, we identify a very small number of *peak* demand periods (typically, we use $|T| = 2$) for which to enforce the link constraints. Picking the size of time window to compute load is also critical. If we pick a small time window, we may not capture the representative load and hence will not place videos appropriately. If we use a large window, we may considerably overprovision capacity for our MIP to become feasible. We examine this consideration by experimenting with several window sizes in Section 7.7.

6.3 Placement update frequency

Another consideration is the frequency of implementing a new placement using our algorithm. While updating our placement more frequently allows the system to adapt to changing demands and correct for estimation errors more gracefully, each update incurs overhead, both in computing the new MIP solution and migrating the content. We evaluate the trade-off between more and less frequent updates in Section 7.8.

7. EXPERIMENTAL RESULTS

We evaluate the performance of our scheme and study various "what-if" scenarios through detailed simulation experiments. We compare our scheme against existing alternatives of using a least recently used (LRU) or a least frequently used (LFU) cache replacement strategy.

7.1 Experiment Setup

We perform our experiments using a custom built simulator. By default, we use a 55-node network modeled from a backbone network of a deployed IPTV service. The network has 70+ bi-directional links connecting these locations. We assume that all these links have equal capacity; however, we vary the actual value to understand the trade-off between disk capacity and link bandwidth. Similarly, we focus on the scenario where all VHOs have equal disk space, but also present results where VHOs have heterogeneous disk capacities. To simulate user requests, we use one month's worth of VoD request traces from a nationally deployed VoD service. This trace contains requests to various types of videos, including music videos and trailers, TV shows, and full-length movies. For simplicity, we map these videos to four different video lengths: 5 minutes, 30 minutes, 1 hour, and 2 hours and assume that we need 100 MB, 500 MB, 1 GB, and 2 GB respectively for storing them on disk. We assume that all videos are of standard definition and stream at 2 Mbps. We start with a baseline scenario of a backbone network with each link being 1 Gbps and the aggregate disk capacity across *all*

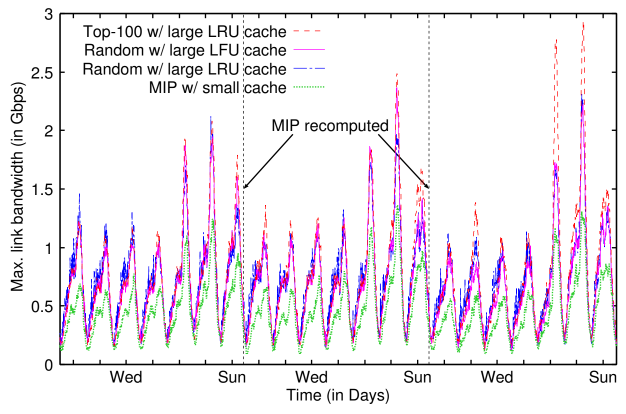

Figure 5: Peak link bandwidth utilized, measured every five minutes.

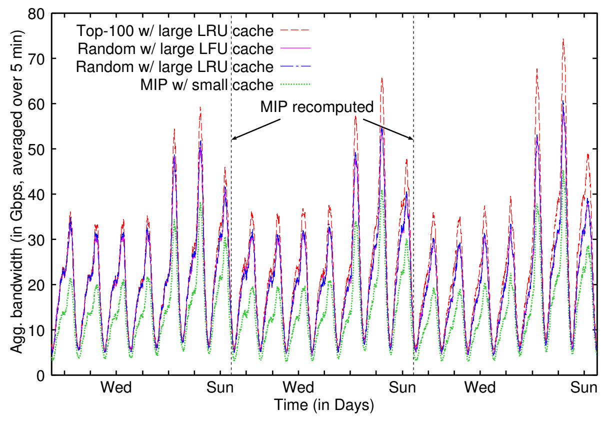

Figure 6: Aggregate bandwidth across all links, averaged over five minutes.

VHOs being 2 times the entire library size. We then vary many of the parameters to understand the various trade-offs and sensitivity of the system.

In our experiments, we use our MIP formulation to place the videos in the VHOs. Unless stated otherwise, we update our MIP-based placement every week using the video requests in the previous 7 days as history. We assume time windows of 1 hour each across two time slices to capture the link constraints. For comparison, we simulate three alternative approaches:

- Random + LRU: For each video, we place one copy at a randomly chosen VHO. The rest of disk space in each VHO is used for LRU cache.
- Random + LFU: This is similar to Random + LRU, but uses LFU instead of LRU.
- Top-K + LRU: We replicate top K videos at every VHO. The remaining videos are assigned randomly to one location. The remaining disk space at each location is used for LRU cache. This is a highly simplified version of [18].

Due to the local cache replacement in all the alternative approaches, if a VHO does not have a local copy of a requested video, it is difficult in practice for the VHO to find which VHO is best to fetch the video from. In our experiments, we assume the existence of an *Oracle* that can tell us the nearest location with a copy of the video, which is the best case for caches in terms of minimizing the total bandwidth consumed for the transfer.

7.2 Performance of our MIP scheme

In the first experiment, we solve an MIP instance and place the videos according to that solution. Then, we play out the request log based on the solution. For each week, we construct a new parameter set based on previous week's demand history and recompute a new MIP instance. We use a link capacity of 1 Gbps for MIP constraints. The aggregate disk space is around 2 times the entire library size. Of this, around 5% of the disk space at each VHO is used as an LRU cache. We compare our scheme with the three alternatives using the same disk space. For the top-K, we experimented with both K=10, and K=100. We present the results only for K=100, as K=10 was highly similar to Random + LRU. We use the first nine days' requests to warm up the caches and run the tests using the remaining three weeks of requests.

Maximum Link Bandwidth. We identify the maximum link usage across all links at each time instant and show how it varies over the three-week period in Figure 5. We observe that, for the same amount of disk space, our proposed scheme can satisfy all requests using significantly lower peak bandwidth. Specifically, the maximum bandwidth needed for our case is 1364 Mbps, while the maximum value for Random+LRU is 2400 Mbps, 2366 Mbps for Random+LFU, and 2938 Mbps for Top-100+LRU. Note that the maximum value for our scheme is slightly larger than 1 Gbps, which is the link capacity provided for the MIP instance. This is because each week introduces new videos, some of which we do not have a good estimate. While the small LRU cache helps absorb some of the errors in estimation, we believe a more sophisticated estimation strategy will help even further. We confirmed this through experiments assuming perfect knowledge of traffic pattern: the maximum bandwidth in that case always stayed within the constraint of 1 Gbps (See Table 3).

Total Bytes Transferred. We calculate the total amount of network transfer where each video transfer is weighted by the video size and hop count. A good placement scheme will result in a small value because most of the requests would be satisfied locally or by nearby neighbors. We present the results in Figure 6. We calculate the aggregate transfers across all links and calculate the average over five minute intervals. We see similar trends

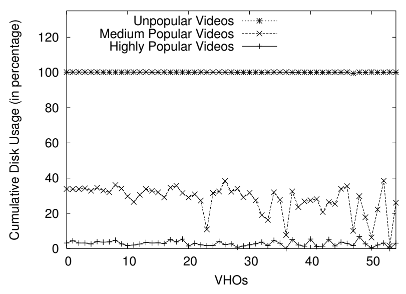

Figure 7: Disk usage based on video popularity.

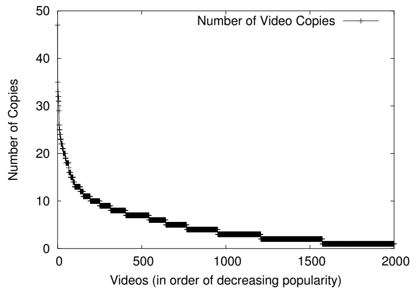

Figure 8: Number of copies of each video, ranked by demand.

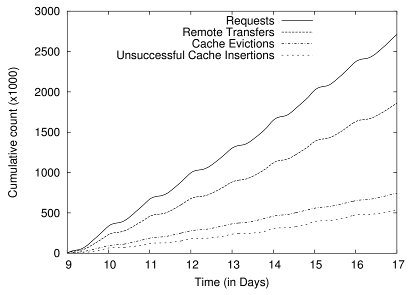

Figure 9: Performance of LRU caches.

to what was observed for the peak bandwidth. Our scheme consistently transfers fewer bytes compared to the other caching based schemes. LRU and LFU perform almost identically. Surprisingly, Top-100 + LRU results in a higher peak utilization and total bytes transferred. We attribute this to the fact that video popularity does not have a very high skew; even the less popular videos incur significant load. With the Top-100 videos occupying significant storage, there is less space for the LRU cache, and hence the performance becomes worse.

To analyze this further, we present the break-up of disk utilization in each VHO based on one solution to our MIP formulation in Figure 7. We characterize the top 100 videos as highly popular, the next 20% of videos as medium popular, and the remaining as unpopular. The highly popular videos occupy a relatively small portion of the total disk space, while the medium popular videos occupy a significant proportion of the total disk space in the system (e.g., more than 30%). We also present the number of copies for each of the top 2000 videos in one of our MIP solutions (Figure 8). We observe that our solution intelligently places more copies for popular videos. This is to avoid remotely fetching frequently requested videos, which not only increases the overall cost (i.e., byte transfer), but also leads to link capacity violations. However, in our solution, even highly popular videos are not replicated everywhere (e.g., less than 30 VHOs have a copy of the 10th most popular video). On the other hand, we observe that more than 1500 videos have multiple copies in the entire system. These two figures indicate that medium popular videos result in significant load and have to be dealt with intelligently. A given movie needs only a few copies — anywhere from 2 copies to 10 copies — but together these videos consume significant space. As a result, our solution carefully distributes copies of these videos across the VHOs. Unfortunately, caching schemes will have difficulty dealing with medium popular videos, unless the cache size is sufficiently large.

Comparative LRU Cache Performance. We performed a simple experiment to understand the performance of a dynamic LRU cache replacement strategy. The aggregate disk space across all locations is around twice the entire library size while each location has the same disk space (and equal to the disk used in the MIP experiments). More than half of the space in each VHO was reserved for the LRU cache. We present the results in Figure 9. As is clear from the figure, not only does the cache cycle, a large number of videos are not cachable because all the space in the cache is currently being used. Almost 20% of the requests could not be cached locally due to this. All this results in around 60% of requests being served by remote offices.

Other results. We ran other experiments, which we only summarize here for lack of space. We repeated the experiments with the aggregate disk space being *5 times* the library size. We find that our approach still results in lower aggregate and peak bandwidth, although the difference between our scheme and the other approaches is smaller. We also experimented with the case of infinite link bandwidth to compare with the unconstrained link case [3]. Then, the maximum link bandwidth used by such a solution sometimes grows to more than twice the link bandwidth our scheme needs. In general, since a solution to the unconstrained problem does not have a limit on link usage, the maximum link usage can grow arbitrarily large, while our scheme finds the best tradeoff, given the link and disk constraints.

7.3 Trade-off between Storage and Bandwidth

To understand the trade-off between storage and bandwidth, we identify how much disk space is needed to find a feasible solution to the MIP, given the link capacity. In Figure 10, we show the feasibility region (where we can serve all the requests without violating disk and link constraints) when we vary the link capacity. Note that the minimum aggregate disk space in the system must be as large as the entire library size, to store at least one copy of each movie (the bottom line in Figure 10). When each link has a capacity of 0.5 Gbps, and all VHOs have the same amount of disk (denoted by “Uniform Disk”), we need at least 5 times more disk

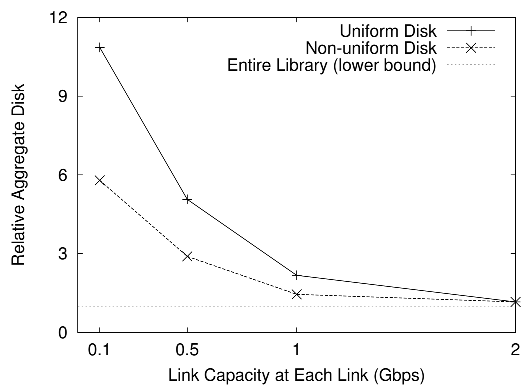

Figure 10: Feasibility region.

than what is needed to store one copy of each movie. We also observe that if we increase link capacity, then we can satisfy all the requests with much smaller disk.

We also consider the case where there are three different types of VHOs. Based on the number of subscribers at individual VHOs, we identify 12 *large* VHOs, 19 *medium* VHOs, and 24 *small* VHOs. In our experiments, a large VHO has twice the disk of a medium VHO, which in turn has twice the disk of a small VHO. The middle line in Figure 10 corresponds to the case of non-uniform VHOs. We observe that compared to the uniform VHO case, we need significantly smaller aggregate disk space to satisfy all the requests. Specifically, with 0.5 Gbps links, the total disk we need is less than 3 times the entire library size (vs. 5 for the uniform VHO case). This is because the majority of requests originate from those large VHOs and some of the medium VHOs. With a larger disk, these VHOs can serve more videos locally. Not surprisingly, as we increase the link capacity, the gap between uniform and non-uniform cases decreases and converges to the library size.

7.4 Scalability

In the next set of experiments, we vary the library size and request load and investigate how the system resource requirement varies accordingly. In the first experiment, we fix the library size and increase the number of requests, while maintaining the same popularity distribution as the trace, which can be approximated by a combination of zipf with an exponential drop-off [8]. We also maintain the diurnal patterns as we scale the request intensity. In Figure 11, we plot the minimum capacity for each link to serve all requests without violating constraints. We show three cases with different disk capacities at each VHO. Obviously, we need larger link capacity with smaller disk or more requests (e.g., larger user base). However, we observe that the growth rate for link capacity is slightly smaller than the growth rate of traffic intensity. This is because a local copy of a highly popular video is able to handle many requests without using the network.

We next experiment with growth in the VoD library. We analyzed the logs by Cha et al. [8] and found that the skew parameter stays similar even for the larger libraries (e.g., 250K videos). We simplified the process of trace generation by only sampling from a zipf distribution rather than a combination of zipf and exponential drop-off. We set the disk size at each VHO such that the aggregate disk space is around 3 times the library size. For ease of comparison, we use the same total number of requests regardless of the library size. We present the results in Figure 12 where we see two interesting trends. First the link bandwidth needed for a feasible solution (with 3x disk) drops as the library size increases. This is due to the combination of increased disk space and distribution of requests to videos. Second, we see that the number of active flows in the peak (i.e., total number of requests for videos) goes up with library size. This is because as the number of videos increases, for the same total number of requests, the distribution of requests is more dispersed.

7.5 Complementary caching

In this experiment we examine the effect of complementary caching on the performance of the MIP solution. We vary the amount of cache as a percentage of the disk space at each VHO and add that space to each VHO. We run the experiment for one week's worth of requests and measure the peak link utilization and the average aggregate bytes transferred. The results are shown in Figure 13. As expected, both the peak and the aggregate decrease with increasing cache size. The reduction is significant as we go from no cache to 5% cache. The reduction, however, is not as significant as we increase the amount of cache further. This result shows that while a cache is important to handle errors in estimation or sudden changes in popularity, it is more important to get the placement correct.

7.6 Topology

We investigate how different topologies affect the capacity required to meet all requests. In addition to the *backbone* network used in the previous sections, we consider two hypothetical networks: *tree* and *full mesh* (where each pair of nodes has a direct link). We also consider an artificial topology generated by combining Sprint and Abovenet topologies from the RocketFuel traces [17]. In each experiment, we use the same amount of aggregate disk across all VHOs, set at 3x the library size. In Table 2, we present our experimental results. As expected, we observe that with more links, we can serve all requests with lower link capacity. For instance, 1 Gbps capacity for each link is more than sufficient in the backbone case, while we need more than 2 Gbps for the tree topology. A thorough understanding of the impact of topology is an interesting topic of future work.

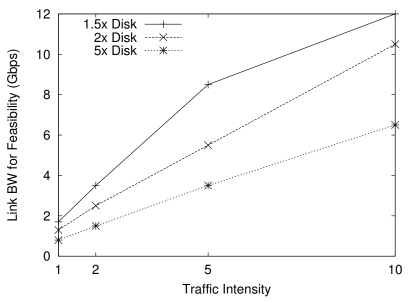

Figure 11: Effect of request intensity.

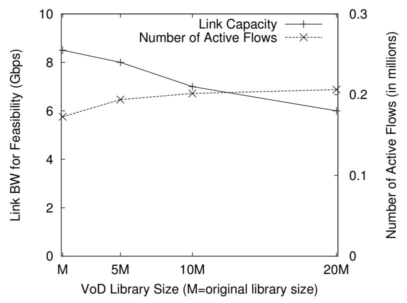

Figure 12: Effect of library growth.

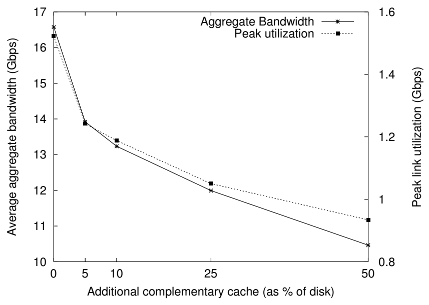

Figure 13: Importance of complementary caches.

Topology	# Nodes	# Links	Feasibility Constraint (Gbps)
Backbone	55	76	0.8
Tree	55	54	2.3
Full Mesh	55	1485	0.05
RocketFuel	40	74	0.8

Table 2: Topology vs. link capacity.

Window Size	Link Bandwidth (in Gbps) Feasibility Constraint	Max during LP window	Max during entire period
1 second	0.5	0.5	0.85
1 minute	0.5	0.49	0.88
1 hour	1.0	0.68	0.80
1 day	2.0	0.94	0.96

Table 3: Peak window size vs. bandwidth.

7.7 Time-varying request pattern

As discussed in Section 6.2, we use only a small number of time windows over which we evaluate the peak demand of videos requested and examine if the placement ensures we remain within the link capacity constraints throughout the week. We performed experiments to understand the trade-offs on choosing the time window by varying it from 1 second to 1 day. Results are shown in Table 3. Using the peak request demand for 1-second time windows, we find that a feasible solution exists for the MIP when each link is 0.5 Gbps. We also observe that the maximum link utilization during the corresponding time window is 0.5 Gbps. However, outside the peak window, some of the links are loaded up to 0.85 Gbps. This is because the MIP solution considers the link constraints only during the 1-second window, and due to highly varying request mix, the request pattern during the window is not representative. As a result, the placement solution is not able to satisfy the link constraint outside of the window. Similar conclusions apply for 1-minute windows. On the other hand, with 1-day time windows, a feasible solution requires 2 Gbps links. But we observe that all links always carry less than 1 Gbps of traffic during the entire 7-day period. Thus, 1-day windows lead to a significant over-estimation of the required link bandwidth. With 1-hour windows, the feasible solution requires 1 Gbps links. Correspondingly, the maximum link bandwidth over the entire 7-day period is also less than 1 Gbps. Thus, 1-hour windows seem to give the best trade-off between accurate estimation of the peak window and actual link utilization.

7.8 Frequency of placement update and estimation accuracy

Running the placement often allows us to handle demand changes or estimation errors gracefully. However, each iteration incurs computational and transfer overheads. We experimented with different placement frequencies to see how they affect performance. In Table 4, we show the maximum link bandwidth usage, total data transfer, and the fraction of requests served locally for the last two weeks. We consider both video size and number of hops to calculate total data transfer, as in (2). We do not use the complementary LRU-cache here. We observe that if we update the placement once in two weeks, then the maximum bandwidth grows significantly. This is because with less frequent updates, the error in demand estimation accumulates over time, as the current placement does not adapt to changes in the demand pattern. We observe that compared to weekly updates, daily updates only modestly improves the maximum bandwidth usage or miss ratio. However, by utilizing the most recent request history information, we can achieve around 10% improvement in terms of total data transfer. Using a 14-day history with weekly placement updates, we did not find any meaningful differences compared to a 7-day history.

In Table 4, we quantify the error in our estimation of demand for new videos by presenting the performance when we have perfect knowledge. When we have perfect knowledge, our MIP-based solution always maintains the link utilization below capacity (< 1 Gbps), serves all the requests while using less total network bandwidth, and serves a majority of requests locally. On the other hand, without any estimation for new videos,

	Max BW (in Gbps)	Total Transfer (TB $\times$ hop)	Locally served
once in 2 weeks	2.23	3284	0.545
weekly	1.32	2776	0.575
daily	1.30	2448	0.585
perfect estimate	0.97	2052	0.606
no estimate	8.62	7042	0.144

Table 4: Impact of update frequency on the placement performance

we observe that the maximum bandwidth grows to over 8+ times the link capacity, and the resulting placement results in lots of remote transfers. Our simple estimation strategy, while not perfect, allows for performance comparable to when we have perfect knowledge.

Cost of placement updates: One aspect to consider when determining the frequency of updates is the network transfer cost due to video migration for a new placement. We can slightly modify equation (9), such that we consider the cost of migration based on the previous mapping (refer Section 5.2.2). In our experiments with this modified objective term, we find that around 2.5K videos need to be transferred between two placements. We argue that this is a small cost compared to the number of requests (e.g., 100Ks per day) and hence is quite manageable. In practice, we can even lower the update costs by piggybacking on requests. That is, when a new placement requires a particular VHO i to store video m, i can wait until a user requests m, fetch it and store a copy in the pinned portion of disk. We plan to investigate this aspect further in the future.

8. CONCLUSIONS

To meet the growing need for on-demand content, we considered the problem of placing video content across distributed offices of a VoD provider connected via a high-bandwidth backbone. Our mixed integer program formulation considers the constraints of disk space and link bandwidth to scalably find a near-optimal placement solution optimized for all the videos in the library, across all locations in an arbitrary network topology, and video demand pattern. For real-world applicability, we presented a number of simple strategies to address practical issues. For instance, we made use of the request history over a one-week period to place content at the VHOs to serve requests over the next week.

We performed extensive experiments using trace data from an operational nationwide VoD service. Our proposed scheme significantly outperforms existing schemes, including LRU and LFU-based caching schemes. We also investigated the performance trade-offs and sensitivity when we vary disk space, link bandwidth, request volume, and library size. Our replication strategy scales with the growth of VoD service and the orders of magnitude speed-up due to our solution approach makes it eminently suitable for practical deployment.

9. ACKNOWLEDGMENTS

We thank Rittwik Jana for providing the trace data. We also thank Rittwik Jana, Debra Phillips and the anonymous referees for their comments and feedback.

10. REFERENCES

[1] H. A. Alzoubi, S. Lee, M. Rabinovich, O. Spatscheck, and J. V. der Merwe. Anycast CDNs revisited. In *Proceeding of WWW*, 2008.

[2] S. Asur and B. A. Huberman. Predicting the future with social media. *arXiv:1003.5699v1 [cs.CY]*, 2010.

[3] I. D. Baev, R. Rajaraman, and C. Swamy. Approximation algorithms for data placement problems. *SIAM J. Computing*, 38(4):1411–1429, 2008.

[4] A. Barbir, B. Cain, F. Douglis, M. Green, M. Hofmann, R. Nair, D. Potter, and O. Spatscheck. Known Content Network (CN) Request-Routing Mechanisms. RFC 3568, July 2003.

[5] D. Bienstock. *Potential Function Methods for Approximately Solving Linear Programming Problems: Theory and Practice*. Kluwer, 2002.

[6] S. Borst, V. Gupta, and A. Walid. Distributed caching algorithms for content distribution networks. In *Proceeding of IEEE Infocom*, 2010.

[7] J. Byrka and K. Aardal. An optimal bifactor approximation algorithm for the metric uncapacitated facility location problem. *SIAM Journal on Computing*, 39(6):2212–2231, 2010.

[8] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *Proceedings of ACM IMC*, 2007.

[9] L. W. Dowdy and D. V. Foster. Comparative models of the file assignment problem. *ACM Comput. Surv.*, 14(2):287–313, 1982.

[10] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, 2000.

[11] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.*, 37(2):630–652, 2007.

[12] L. Guo, E. Tan, S. Chen, Z. Xiao, and X. Zhang. Does internet media traffic really follow zipf-like distribution? In *Proceedings of ACM SIGMETRICS*, New York, NY, USA, 2007.

[13] C. Huang, J. Li, and K. W. Ross. Can internet video-on-demand be profitable? In *Proceedings of ACM Sigcomm*, 2007.

[14] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceeding of ACM SIGKDD*, 2008.

[15] D. Lee, J. Choi, J. H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim. LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Trans. Comput.*, 50(12):1352–1361, 2001.

[16] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the placement of web server replicas. In *Proceeding of IEEE Infocom*, 2001.

[17] N. Spring, R. Mahajan, and D. Wetherall. Measuring isp topologies with rocketfuel. In *Proceedings of ACM SIGCOMM*, 2002.

[18] V. Valancius, N. Laoutaris, L. Massoulié, C. Diot, and P. Rodriguez. Greening the internet with nano data centers. In *Proceedings of ACM CoNEXT*, 2009.

[19] Y. Zhao, D. L. Eager, and M. K. Vernon. Network bandwidth requirements for scalable on-demand streaming. *IEEE/ACM Trans. Netw.*, 15(4):878–891, 2007.

[20] X. Zhou and C.-Z. Xu. Optimal video replication and placement on a cluster of video-on-demand servers. In *Proc. IEEE ICPP*, 2002.