Optimal Content Placement for a Large-Scale VoD System

David Applegate, Aaron Archer, Vijay Gopalakrishnan, *Member, IEEE*, Seungjoon Lee, and K. K. Ramakrishnan, *Fellow, IEEE*

***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., 100 Ks 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 linear programming (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 least recently used (LRU) or least frequently used (LFU) cache replacement policies. We also investigate the tradeoff between disk space and network bandwidth.**

***Index Terms*—Content placement, integer programming, video on demand.**

I. 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 interconnected 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.

Manuscript received September 08, 2014; revised April 12, 2015; accepted June 04, 2015; approved by IEEE/ACM Transactions on Networking Editor N. A. L. Reddy. Date of publication October 26, 2015; date of current version August 16, 2016.

D. Applegate and V. Gopalakrishnan are with AT&T Labs—Research, Bedminster, NJ 07921 USA (e-mail: david@research.att.com; gvijay@research.att.com).

A. Archer is with Google, New York, NY 10011 USA (e-mail: aarcher@google.com).

S. Lee is with Two Sigma Investments, New York, NY 10013 USA (e-mail: slee@twosigma.com).

K. K. Ramakrishnan is with the University of California, Riverside, Riverside, CA 92521 USA (e-mail: kk@cs.ucr.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2015.2461599

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 [9]. The equation is one of growth—production of content and making them available, versus the economic viability of adding more and more storage at each node over time, as the amount of content grows—and which factor will outpace the other. At the current moment, 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 appear to be outpacing the ability of providers to economically add storage and replicate content. For example, a 1-h 4 K video takes up about 20 GB of disk [2], and today's VoD providers are estimated to have a library larger than 100 K videos [1]. Assuming the average video length is 1 h, the total disk size for a single copy of such high-quality video library would be 2 PB (or even larger if the storage back-end system uses replication internally [24]). With the advent of adaptive bit rate (ABR) video where multiple (5–10 for certain VoD providers) representations of the video are stored to deliver the most appropriate quality of the video to match the bandwidth currently available to the user, this problem is only exacerbated. Replicating an entire library at each VHO is economically nontrivial in this setup and even more challenging in case of a larger-scale library (e.g., millions of videos). Our work studies how to use intelligent content prepopulation and request load-balancing to serve VoD requests when disk storage is small relative to the entire library.

Multiple studies [10], [15] 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 [13], [19]. 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 [18]. 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 on the system in the context of VoD as this results in a high-bandwidth stream transferred for an extended period of time. Furthermore, 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 IV, 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 [7], [9], [23]. 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 [7]. Adding them introduces the aspect of multicommodity flow [14], 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 tradeoff 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 5 K videos and 55 VHOs. We overcome this by employing a decomposition technique based on the exponential potential function method and Lagrangian relaxation [4]. We thus obtain a near-optimal LP solution orders of magnitude faster (Section V-C), 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 subexponential bound on the worst-case running time of our rounding heuristic (Section V-D). In practice, bulk of the running time is spent in solving the LP, not in the rounding phase. When 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 a worst-case performance guarantee for our rounding heuristic; there may be 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 VI). 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 least recently used (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 [10]. Our results show that our approach outperforms existing caching schemes (Section VII). For the same amount of disk space, our approach only requires a little more than half the peak bandwidth as LRU or least frequently used (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.

We highlight the differences between this paper and our previous conference paper [5]. First, we have made significant improvements on our optimization strategy and rounding scheme, which we describe in more detail. Second, we present new experiment results from more diverse settings, including different real-world network topologies, different deployments scenarios, larger video library sizes, and varied disk sizes. Last, we compare our work to more recent works published since our conference paper.

II. 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 [12] and the references therein. Zhou and Xu [26] 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.* [23] propose an LP-based heuristic to calculate the number of video copies placed at customer home gateways. Borst *et al.* [9] 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

[1]Predicting popularity of new videos is an active area of research [6], [17].

diverse disk and link bandwidth constraints, where different locations show different video request patterns. We also consider popularity change over both short and long term, which poses a significant challenge in maintaining link bandwidth constraints.

Baev *et al.* [7] consider the *data placement problem* where the objective is to minimize cost without taking bandwidth into account. They prove their problem is NP-hard via a reduction of the uncapacitated facility location problem, and present a 10-approximation algorithm for uniform-length videos. They show that for the nonuniform case, even deciding feasibility is NP-hard, so no approximation algorithm is possible unless $\mathrm{P} = \mathrm{NP}$. Our problem is strictly more complex since we also consider link constraints. Also, 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 that applies to all instances would imply $\mathrm{P} = \mathrm{NP}$.

Content placement and traffic engineering in content distribution networks (CDNs) has been a focus of considerable recent research. Recent work by Sharma *et al.* [20] investigates the interplay between distribution strategies and traffic engineering, using traffic from a large CDN and realistic ISP topologies. While their conclusion is that optimization-based approaches do not offer significant benefits compared to simple approaches (e.g., LRU caching), we believe it is significantly dependent on the size of the cache deployed. Their storage ratio is quite large (of the order of 4 or more) for the deployed caches. Moreover, the approximation approach used eliminates over 50% of the videos for the primary solutions. We believe these significantly influence the results, in that there is a long tail in the access patterns as observed in our traces, and the bandwidth consumed in the network is indeed influenced by these transfers. We do observe in our work that as the amount of storage is increased and gets close to the working set size, the differences in performance across different methods do diminish. In addition, [20] does show that if demand can be predicated, like we do in our paper, optimization approaches do outperform simple caching. Qiu *et al.* [19] 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*) [3], [8]. 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. Thouin and Coates [22] consider the problem of server provisioning for VoD service. They consider a predefined set of server models and come up with a number of heuristics to find a low-cost deployment for given aggregate video demands. Their work is orthogonal to our work, in that our work focuses on placing and routing VoD requests on already provisioned server and networks.

Peer-to-peer (P2P) schemes that reduce the backbone bandwidth usage have been proposed for VoD delivery [16]. Such P2P systems, however, still need to use VoD servers when video delivery from peers is not possible. Our work is complementary to these proposals. Zhao *et al.* [25] 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 accounting for the skew in content popularity and replicating videos to minimize backbone network bandwidth consumption.

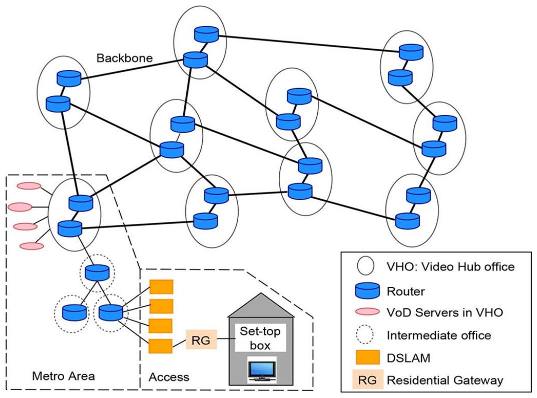


Fig. 1. Typical architecture for IPTV service.

III. 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 served by a single office. Each office has a certain amount of storage; this 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 predetermined path between the VHOs (e.g., based on shortest path routing), which is more realistic than arbitrary routing [14]. Fig. 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 that balances the tradeoff 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.

IV. Challenges

In this section, we use traces from a nationally deployed VoD service to show why content placement is challenging and why simple caching-based approaches alone will not suffice.

A. Large Working Set Sizes

In Fig. 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

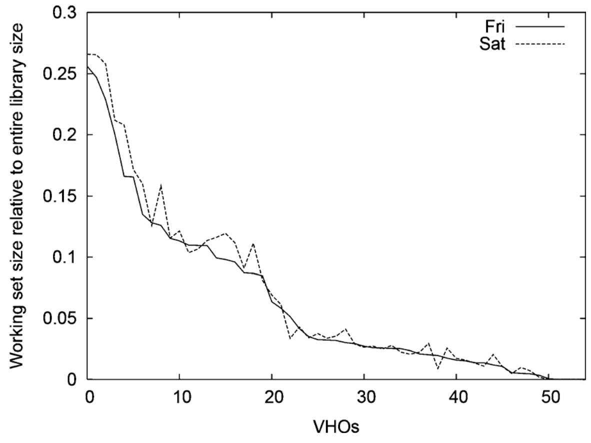


Fig. 2. Working set size during peak hours.

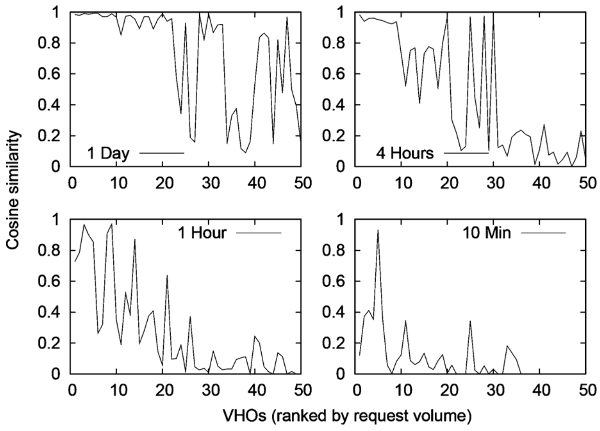


Fig. 3. Similarity in videos requested during different time windows.

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 VII.

B. *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 Fig. 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.

TABLE I
INPUT PARAMETERS AND DECISION VARIABLES USED IN THE MIP

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 at VHO j
$f_j^m(t)$	# of requests for video m at VHO j active at time t
c_{ij}	cost of transferring one GB from VHO i to VHO j
decision variable	**meaning**
y_i^m	binary variable indicating if video m is stored at VHO i
x_{ij}^m	fraction of requests for video m at VHO j served by VHO i

V. MIP FORMULATION

In this section, we formulate the 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.

A. *Input Parameters*

Table I 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 Mb/s). 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 gigabyte 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

[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.

the video, we show in Section V-B-1 that the actual values do not affect which solution is optimal.

B. 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. $x_{jj}^m = 1$ implies that j serves the requests itself (because it has the video 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:

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

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

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

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

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

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

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

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 $\langle \text{video}, \text{VHO}\rangle$ 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 $\sum_{i\in V} y_i^m \ge 1$, $\forall m \in M$.)

1) Cost Coefficients Matter Little: In (1), we introduce two coefficients α and β for the cost of serving content. Since all feasible solutions satisfy constraint (3), we can expand and rearrange the objective (2) as

$$\alpha \sum_{m\in M}\sum_{i,j\in V} s^m a_j^m |P_{ij}| x_{ij}^m \tag{9}$$

$$+ \beta \sum_{m\in M}\sum_{j\in V} s^m a_j^m. \tag{10}$$

The first term (9) scales uniformly with α, while the second term (10) is independent of the decision variables x_{ij}^m and y_i^m. Thus, we have the following proposition.

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

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

$$\sum_{m\in M}\sum_{i,j\in V} s^m a_j^m c_{ij} x_{ij}^m + w \sum_{m\in M}\sum_{i\in V} s^m c_{oi} y_i^m \tag{11}$$

where w is a parameter that allows a different weight given to the additional transfer cost. For example, with $w = 0$, (11) 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 (11) when we evaluate the impact in Section VII-H.

C. Finding a Fractional Solution

As mentioned in Section II, finding an optimal solution to the above MIP is NP-hard [7]. Instead, we first solve the LP obtained by relaxing the integral constraint (8) to be just $y_i^m \ge 0$, 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 V-D.

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, which we describe in more detail in the Appendix. This approach computes a solution to an LP instance that provably achieves approximate optimality and feasibility—it terminates with a solution that violates disk and link bandwidth constraints by at most 1% and its objective value is within 1% of the optimal.

The basic idea is to decompose a given large LP instance into mostly independent subproblems. In our case, we decompose the overall MIP into a subproblem for each video, and each subproblem then becomes a (fractional) uncapacitated facility location problem, which is more tractable. Specifically, in the LP formulation, we observe that constraints (3) and (4) are independent for each video, while the disk space and link bandwidth constraints (5) and (6) require aggregating resource usage across all videos. We then replace the nonindependent constraints (5) and (6) with a penalty function that is exponential in their normalized violations. In this way, we convert a large linear optimization problem into a convex optimization problem over a large collection of independent subproblems. We then use gradient descent to minimize the potential function contributed by an individual subproblem (i.e., individual video) and go through an entire set of subproblems to complete one *pass*. We then repeat multiple passes of the gradient descent process, until all the constraint violations are within the desired 1% threshold. Note that this process provably converges [4].

We also use Lagrangian relaxation, where we relax the disk space and link bandwidth constraints and use the exponential coefficients as Lagrangian multipliers. After finishing a gradient descent pass, we calculate a lower bound using the current Lagrangian multipliers, which allows us to check how close to optimal our current solution is. Our scheme terminates if the current objective is within 1% of a lower bound and if violates disk space and link bandwidth constraints by less than 1%.

In addition to this basic idea, our decomposition-based approach employs various optimizations (e.g., faster heuristics, parallelism) to achieve orders of magnitude improvement in running time and scalability over CPLEX (see Section VII).

D. Rounding

Based on the fractional solution, we round fractional y_i^m to integer values by sequentially solving a MIP for each video m. If all of the y_i^m for video m are already integer, the y_i^m and x_{ij}^m are unchanged. Otherwise, we solve a subproblem for video m based on the current potential function. Since we want y_i^m to be an integer in this case, the problem becomes an integer facility location problem, which is NP-hard. As solving this problem can be computationally expensive, we use an approximation algorithm by Charikar and Guha [11], which finds a provably good solution in polynomial time. When this final rounding pass is complete, the resulting solution will have integer values for all y_i^m variables; we use this for placement.

Rounding Performance: We briefly report the performance of the rounding step. In terms of computation time, the rounding pass is comparable to a single gradient descent pass. Another aspect is the quality of final integer solution. While the rounding step can cause the objective and constraint violation to increase beyond the 1% threshold that we guarantee with a fractional solution, in our experiments (detailed setup is described in Section VII-A), we find that the increase due to the rounding step is marginal. Specifically, with libraries of 5 K videos, the optimality gap on average is 4.1% (or 3.1% increase beyond our target 1%), and the constraint violation is less than 4.4%. However, with 200 K video libraries, the optimality gap is 1.0% and the constraint violation is less than 0.8% (smaller than the target 1%), which suggests that our rounding step performs better with larger libraries.

VI. Practical Considerations

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

A. 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. Furthermore, from our analysis,

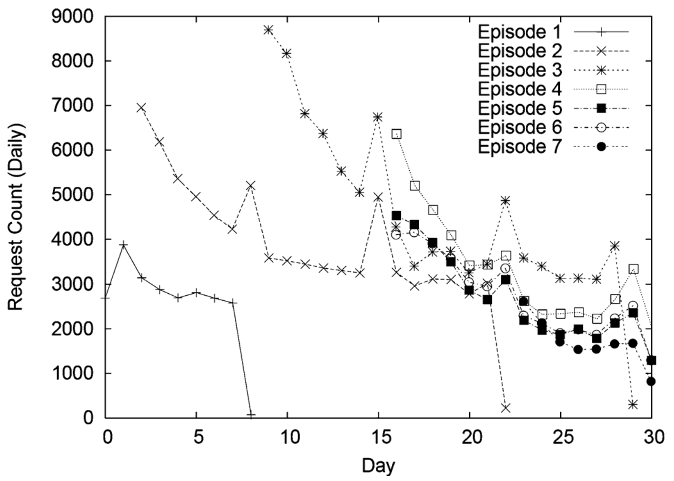


Fig. 4. Daily request count for different episodes of a series show.

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 [6], [17] 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. Fig. 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 VII-H.

We use another simple estimation strategy for blockbuster movies. From exogenous information [6], 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).

B. 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 reevaluated.

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 $|T|$. Therefore, we identify a very small number of *peak* demand periods (typically, we use $|T|=2$) over which to enforce the link constraints. By satisfying the link bandwidth constraints for these peak demand periods, we found that the demand during nonpeak periods did not overload any links in our instance. In the general case, we would iteratively identify these additional time periods that overload some links and add them to the set of peak demand periods, such that a solution to the new problem instance would satisfy the link constraints during these additional time periods.

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 over-provision capacity for our MIP to become feasible. We examine this consideration by experimenting with several window sizes in Section VII-G.

C. 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 tradeoff between more and less frequent updates in Section VII-H.

VII. 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 an LRU or an LFU cache replacement strategy. With respect to user request traces for our evaluation, we have two choices: either project library sizes to the future and "completely invent" the user request traces, or use the data we have of real user request traces and reduce disk sizes to something that yields an interesting point on the disk space/link bandwidth tradeoff curve. In our study, we have used the latter approach because we believe that using real data at smaller scale is more useful than making up data to illustrate the exact same point at a larger scale. As a result, we use relatively small disk size when evaluating the performance benefit of our proposed scheme, while we employ additional synthetic input traces to study the scalability of our MIP solution scheme.

A. 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+ bidirectional links connecting these locations. We assume that all these links have equal capacity. However, we vary the actual value to understand the tradeoff 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 min, 30 min, 1 h, and 2 h, 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 Mb/s. We start with a baseline scenario of a backbone network with each link being 1 Gb/s and the aggregate disk capacity across *all* VHOs being 2 times the entire library size. We then vary many of the parameters to understand the various tradeoffs 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 h 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 [23].

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 needs to identify the best peer VHO to fetch the video from. In our experiments, we assume the existence of an *Oracle* that informs us of the nearest location with a copy of the video. This gives the best case for these alternates in terms of minimizing the total bandwidth needed for the transfer.

We also conducted experiments with publicly available network topologies: Tiscali, Sprint, and Ebone, taken from Rocketfuel [21]. To understand how our approach performs with different library sizes, we generated synthetic video request traces for video libraries of different sizes (ranging from 5 K videos to 1 M videos) for those networks based on the YouTube popularity distribution published by Cha *et al.* [10]. We designed our trace generation process, such that it mimics the salient real-world properties (e.g., city-level population, popularity distribution of the videos). More details about the network topologies and synthetic trace generation are available at http://www2.research.att.com/~vodopt/.

B. 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 Gb/s 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 to 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 3 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 3-week period in Fig. 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 Mb/s, while the maximum value for Random + LRU is 2400 Mb/s, 2366 Mb/s for Random + LFU, and 2938 Mb/s for Top-100 + LRU. Note that the maximum value for our scheme is slightly larger than 1 Gb/s, 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 Gb/s (see Table V).

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 Fig. 6. We calculate the aggregate transfers across all links and calculate the average over 5-min intervals. We see similar trends 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 breakup of disk utilization in each VHO based on one solution to our MIP formulation in Fig. 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 (Fig. 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 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

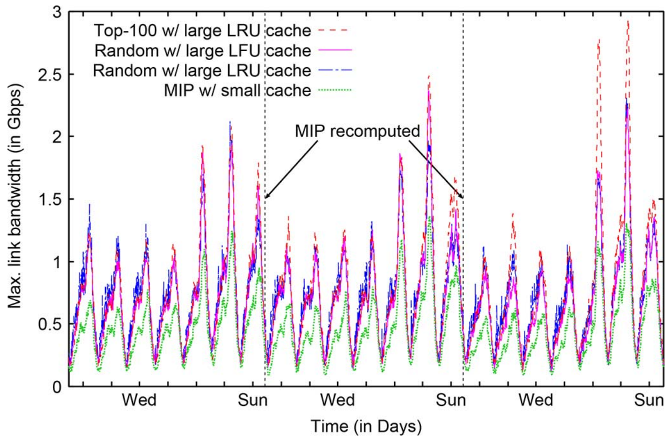


Fig. 5. Peak link bandwidth utilized, measured every 5 min.

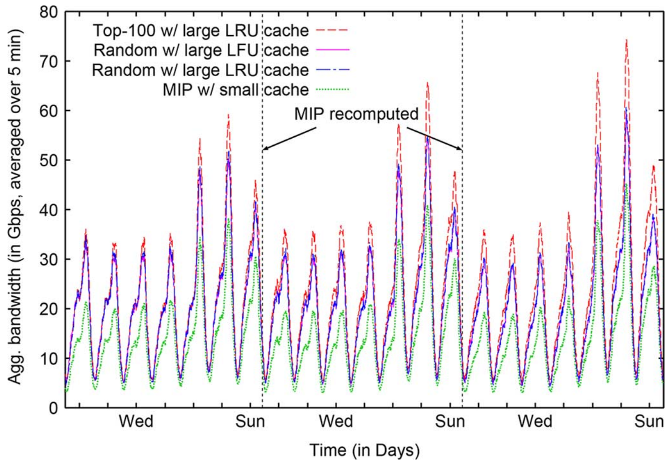


Fig. 6. Aggregate bandwidth across all links, averaged over 5 min.

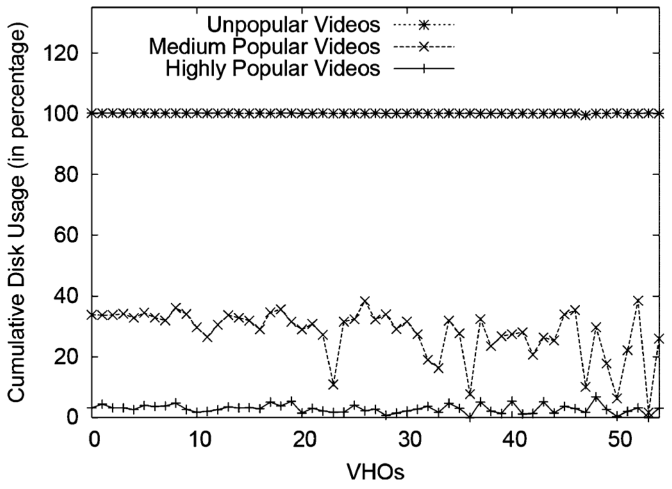


Fig. 7. Disk usage based on video popularity.

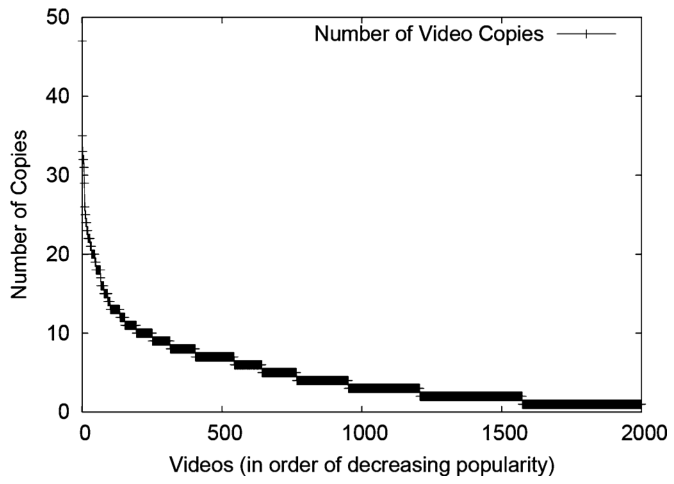


Fig. 8. Number of copies of each video, ranked by demand.

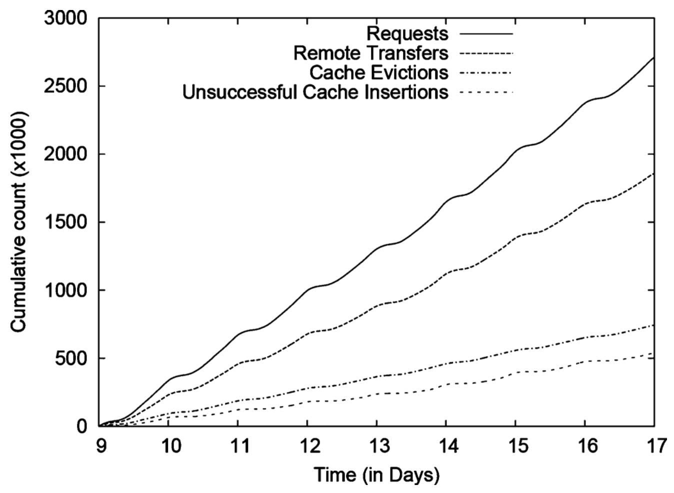


Fig. 9. Performance of LRU caches.

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 Fig. 9. As is clear from the figure, not only does the cache cycle, but 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.

Comparison to Origin Server and LRU-Cache: To compare our approach to recent proposals [20], we ran experiments that assume the use of simple LRU caching in conjunction with origin servers. Since identifying the optimal location of an origin server is not trivial, we partitioned our network into four regions, each served by a separate origin server that is connected to one of the VHOs in that region. Thus, the underlying connectivity does not change, and traffic from an origin server to VHOs likely traverses inter-VHO links in the underlying network. We then allocate the same amount of disk space to caches as we use in our approach (e.g., $2\times, 6\times$). The origin servers have additional capacity to store the entire library.

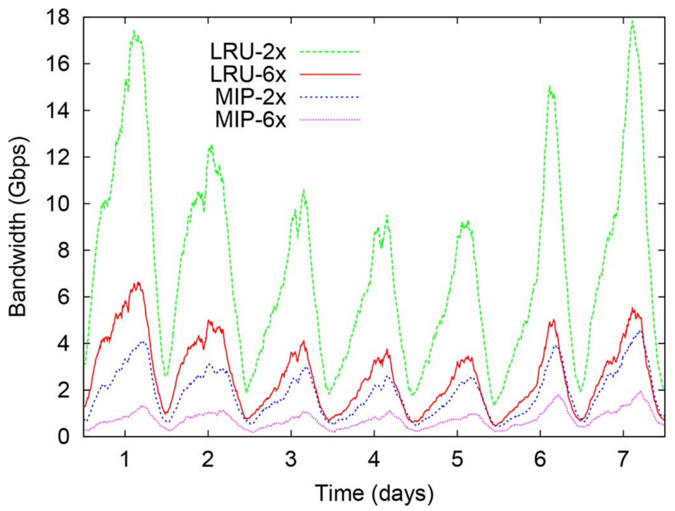


Fig. 10. Maximum link bandwidth used by MIP versus LRU caching with origin servers.

TABLE II
COMPARISON OF MIP VERSUS LRU CACHING WITH ORIGIN SERVERS

	2x-Disk		6x-Disk	
	MIP	LRU	MIP	LRU
Peak link B/W (Gbps)	4.5	17.8	1.9	6.6
Max. aggregate bandwidth (Gbps)	155.9	121.3	41.6	47.2
Cache Hit Rate	68%	62%	95%	86%

Thus, we give benefit to the caching solutions to have an extra $4\times$ storage. In fact, we did not account for this extra storage in the origin servers, to emphasize this benefit for the competing approach. We once again ran the experiments over the 2-week period. With caching, we use the first week as the warm-up phase, while with our approach, we run the MIP before the first and second weeks. We compare performance during the second week in Fig. 10 and Table II.

Our results show that our approach outperforms simple caching with origin servers for both $2\times$ and $6\times$ disks sizes. The maximum link bandwidth at any time during the second week is lower by $\sim 3.5\times$ in both cases. In fact, our results show that the maximum link utilization with our approach is lower even when our approach uses $2\times$ disk space while caching uses $6\times$ disk space. We also computed the sum bandwidth across all the links at various points in time and note the peak of these values in Table II. We see that our approach transfers more data than caching when disk space is limited $(2\times)$, but performs better when there is more disk $(6\times)$. Finally, we also note that the cache hit rate is consistently better with our approach than with caching.

C. *Tradeoff Between Storage and Bandwidth*

To understand the tradeoff 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 Fig. 11, 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 Fig. 11). When each link has a capacity of 0.5 Gb/s, and all VHOs have the same

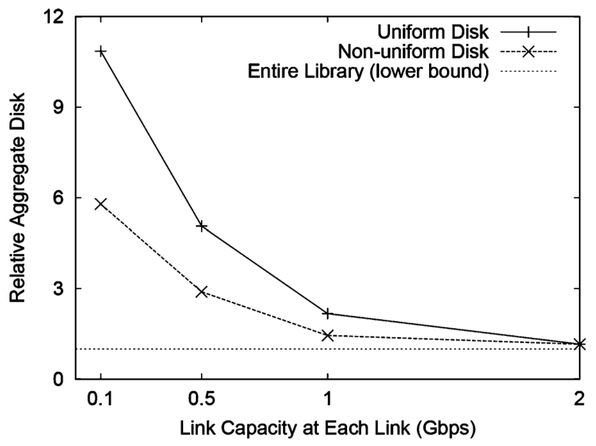


Fig. 11. Feasibility region.

amount of disk (denoted by "Uniform Disk"), we need at least 5 times more disk 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 Fig. 11 corresponds to the case of nonuniform 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-Gb/s links, the total disk we need is less than 3 times the entire library size (versus 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 nonuniform cases decreases and converges to the library size.

D. 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 Fig. 12. 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.

E. Scalability

In the next set of experiments, we experiment with growth in the VoD library, growing from 5 K videos to 1 M videos. We use the synthetically generated traces as described in Section VII-A. Given the library size, we consider six different experiment scenarios: two different disk sizes (small, large) and three different networks (Tiscali, Sprint, Ebone). For the small disk size, the aggregate disk is twice the library size; for the large disk size, an average VHO can store 20% of the entire video library. For each scenario, we experimented with 100 different runs using different random seeds and obtained the arithmetic means of each metric. Then, we use the geometric mean of the six arithmetic means for a given library size. One of the metrics considered is the running time. To obtain this, we ran all these experiments on a single machine (with two 6-core 2.67 GHz Intel Xeon X5650 CPUs and 48 GB of memory).

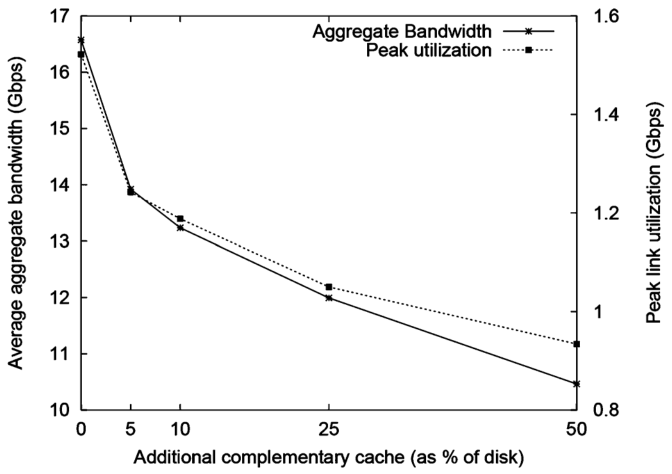


Fig. 12. Importance of complementary caches.

TABLE III
RUNNING TIME AND MEMORY USAGE. EACH ROW AGGREGATES SIX SCENARIOS (I.E., 3 NETWORKS AND 2 DISK TYPES)

library	CPLEX		Proposed Approach		
size	time (s)	mem (GB)	time (s)	mem (GB)	speedup
5,000	894.47	10.15	1.39	0.11	644x
10,000	2062.10	19.36	1.77	0.18	1168x
20,000	5419.57	37.63	2.62	0.33	2071x
50,000			5.44	0.77	
100,000			10.45	1.52	
200,000			20.03	3.02	
1,000,000			98.61	15.00	

In Table III, we first report the running time and memory usage of our proposed approach and compare it to CPLEX. We observe that the running time and memory footprint of our proposed scheme both scale almost linearly as a function of library size. The memory usage of CPLEX also scales linearly, but its running time scales superlinearly. In absolute terms, CPLEX needs more than 1.5 h and 37 GB of memory to solve the instance of a 20 K video library. Because of its large memory footprint, CPLEX cannot handle larger instances due to the memory limitation. In comparison, our proposed scheme takes less than 3 s (or 2000$\times$ faster) and uses only 0.3 GB of memory (or 100$\times$ smaller footprint). Furthermore, our proposed scheme can scale to much larger instances. For examples, it takes about 1.5 min and 15 GB of memory to solve a 1-M video library instance.

We next report how the link capacity requirement changes as we increase the number of videos in the library. Note that in our synthetic traces, the number of requests is proportional to the number of videos. In Fig. 13, we show the link capacity needed for each scenario, normalized by the number of videos in the library. We use the small disk scenarios, with the aggregate disk space being the same for all the three networks (2$\times$ library

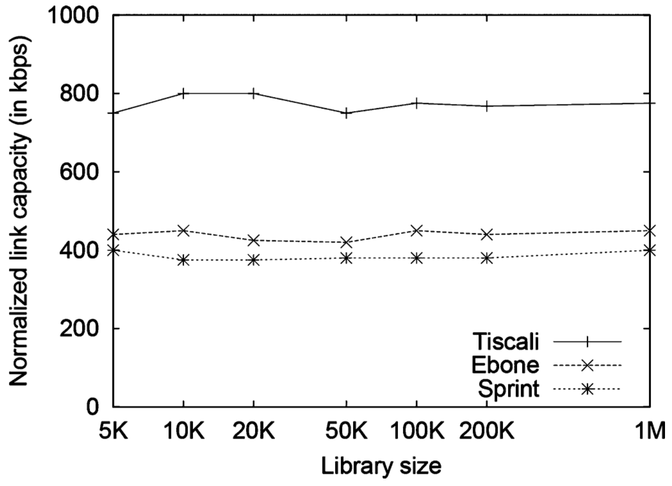


Fig. 13. Link capacity versus library size growth.

TABLE IV
TOPOLOGY VERSUS LINK CAPACITY

Topology		# Nodes	# Links	Feasibility Constraint (Gbps)
VoD Backbone	Original	55	76	0.8
	Tree	55	54	2.3
	Full Mesh	55	1485	0.05
RocketFuel Topologies	Tiscali	49	86	2.5
	Sprint	33	69	0.6
	Ebone	23	38	0.6

TABLE V
PEAK WINDOW SIZE VERSUS BANDWIDTH

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

size). We first observe that the required link capacity stays similar when we vary the number of videos. We also observe that the link capacity for Tiscali is the highest. Since Tiscali has more locations (49) than the other networks (see Table IV), each Tiscali location on average has smaller disk space, which results more remote transfers over the network. While Ebone has fewer locations than Sprint (23 versus 33), Ebone also has fewer network links than Sprint (38 versus 69), which results in a similar link capacity requirement for Ebone and Sprint.[3]

F. 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 report experimental results using the three network topologies from RocketFuel. For this set of experiments, we use the same video library and real VoD service traces for ease of comparison. Since the number of VHOs in the VoD backbone is larger than the number of nodes in any of the RocketFuel networks, we sort the VHOs starting with the largest request count and use the top n VHOs, where n is the number of nodes in each network (e.g., 33 top VHOs for Sprint). In all these experiments, we use the same amount of aggregate disk ($3\times$ the library size) across all VHOs.

In Table IV, we present our experimental results. As expected, given a set of nodes (i.e., the VoD backbone), we observe that with more links, we can serve all requests with lower link capacities. For instance, 1 Gb/s capacity for each link is more than sufficient in the case of original set of links, while we need more than 2 Gb/s for the tree topology. This result also clearly illustrates that the different network topologies require different link capacities for video delivery. Specifically, the link capacity requirement for Tiscali is much larger than that for Sprint. Note that another factor that influences these result is that we chose VHOs with larger request counts for the RocketFuel experiments, which would result in larger capacity requirements. A thorough understanding of the impact of topology and demand pattern on resource requirements is an interesting topic of future work.

G. Time-Varying Request Pattern

As discussed in Section VI-B, 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 tradeoffs on choosing the time window by varying it from 1 s to 1 day. Results are shown in Table III. Using the peak request demand for 1-s time windows, we find that a feasible solution exists for the MIP when each link is 0.5 Gb/s. We also observe that the maximum link utilization during the corresponding time window is 0.5 Gb/s. However, outside the peak window, some of the links are loaded up to 0.85 Gb/s. This is because the MIP solution considers the link constraints only during the 1-s 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-min windows. On the other hand, with 1-day time windows, a feasible solution requires 2-Gb/s links. However, we observe that all links always carry less than 1 Gb/s 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-h windows, the feasible solution requires 1-Gb/s links. Correspondingly, the maximum link bandwidth over the entire 7-day period is also less than 1 Gb/s. Thus, 1-h windows seem to give the best tradeoff between accurate estimation of the peak window and actual link utilization.

H. 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 VI, we show the maximum link bandwidth usage, total data transfer, and the fraction of requests served locally for the last 2 weeks. We consider both video size and number of hops to calculate total data transfer, as in (2). We do

[3]The results in Fig. 13 and Table IV are slightly different because the request count in Fig. 13 is proportional to the population of each location, while we use the locations with largest request volumes in Table IV.

TABLE VI
IMPACT OF UPDATE FREQUENCY ON THE PLACEMENT PERFORMANCE

	Max BW (in Gbps)	Total Transfer (TB × 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

not use the complementary LRU-cache here. We observe that if we update the placement once in 2 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 VI, 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 Gb/s), 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, 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 (11), such that we consider the cost of migration based on the previous mapping (refer Section V-B-2). In our experiments with this modified objective term, we find that around 2.5 K videos need to be transferred between two placements. We argue that this is a small cost compared to the number of requests (e.g., 100 K's 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.

VIII. CONCLUSION

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 1-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 tradeoffs 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 speedup due to our solution approach makes it eminently suitable for practical deployment.

APPENDIX
EXPONENTIAL POTENTIAL FUNCTION (EPF) METHOD AND ITS APPLICATION

While our previous work provides a more detailed description [4], we briefly describe the general principles of the EPF framework and highlight how we apply it to our VoD application.

A. General Description

The EPF framework is a Dantzig–Wolfe decomposition method that uses exponential penalty functions to define: 1) a potential function that encodes relaxed feasibility problems, and 2) Lagrange multipliers for computing lower bounds. Consider the following LP:

$$\min cz \text{ s.t. } Az \le b, z \in F^1 \times \cdots \times F^K \subseteq \mathbb{R}^n \tag{12}$$

where each F^k is a polytope, $A = (a_{ij})$ is an $m \times n$ matrix, $c \in \mathbb{R}_n$ and $b \in \mathbb{R}^m$. Let OPT denote its optimum.[4] Solution $z \in F$ is ϵ-*feasible* if $Az \le (1+\epsilon)b$ (i.e., it violates each constraint by at most $1+\epsilon$), and it is ϵ-*optimal* if $cz \le (1+\epsilon)$OPT. Given constant ϵ, we aim for an ϵ-feasible, ϵ-optimal solution. Equivalently, we will guess a value B for OPT and consider the problem FEAS(B), wherein we replace the objective in (12) with the constraint $cz \le B$. Note that a solution is ϵ-feasible for FEAS(OPT) iff it is ϵ-feasible, ϵ-optimal for (12).

Dantzig–Wolfe decomposition takes advantage of fast algorithms for optimizing linear objective functions over each F^k. We consider scenarios where the constraints constitute a *packing* problem, i.e., $a_{ij} \ge 0$ and $b_i > 0$. Let the set R index the rows of A, let $a_i \in \mathbb{R}_n$ denote row i of A, let index 0 refer to the objective, and define $R^* = R \cup \{0\}$. Given a row vector of Lagrange multipliers $\lambda \in \mathbb{R}_{R^*}$ with $\lambda \ge 0$ and $\lambda_0 > 0$, define $c(\lambda) = c + \frac{1}{\lambda_0}\lambda A$. Whenever $k \in \mathcal{B} := \{1, \ldots, K\}$, a superscript k denotes the portion of an object corresponding to block k, e.g., z^k, A^k, F^k, or $c^k(\cdot)$. Define

$$LR^k(\lambda) = \min_{z^k \in F^k} c^k(\lambda) z^k \tag{13}$$

and $LR(\lambda) = \sum_{k \in \mathcal{B}} LR^k(\lambda) - \frac{1}{\lambda_0}\lambda_R b$, where the notation λ_R means to restrict vector $\lambda \in \mathbb{R}_{R^*}$ to its R components (i.e., exclude λ_0). Standard duality arguments show that $LR(\lambda) \le$ OPT. All of our lower bounds derive from this fact.

[4]The variables in this Appendix are consistent with the EPF literature and our accompanying paper and not to be confused with those used in Section V.

Define $\alpha(\delta) = \frac{\gamma \log(m+1)}{\delta}$, where $\gamma \geq 1$ and δ is a scale factor that evolves over the course of the algorithm. Let $r_i(z) = \frac{1}{b_i} a_i z - 1$ be the relative infeasibility of constraint i, and define aliases $a_0 := c$ and $b_0 := B$ so that $r_0(z) = \frac{1}{B} cz - 1$. Define $\delta_c(z) = \max_{i \in R} r_i(z)$ as the max relative infeasibility over the coupling constraints, and $\delta(z) = \max(\delta_c(z), r_0(z))$. Let $\Phi_i^\delta(z) = \exp(\alpha(\delta) r_i(z))$ define the potential due to constraint i, and $\Phi^\delta(z) = \sum_{i \in R^*} \Phi_i^\delta(z)$ define the overall potential function we aim to minimize (for fixed δ). If z is feasible for (12), then each $r_i(z) \leq 0$ so $\Phi^\delta(z) \leq m+1$, whereas if even one constraint i has $r_i(z) \geq \delta$, then $\Phi^\delta(z) > \Phi_i^\delta(z) \geq (m+1)^\gamma \geq m+1$. Thus, minimizing $\Phi^\delta(z)$ either finds a δ-feasible z, or proves that none exists.

The plan is to minimize $\Phi^\delta(z)$ via gradient descent. Let $\pi_i^\delta(z) = \Phi_i^\delta(z)/b_i$ for $i \in R^*$, and $g(z) = \pi_0^\delta(z) c + \pi_R^\delta(z) A$. The gradient of the potential function is

$$\nabla \Phi^\delta(z) = \alpha(\delta) g(z) = \alpha(\delta) \pi_0^\delta(z) c\left(\pi^\delta(z)\right) \qquad (14)$$

which is a positive scalar times $c\left(\pi^\delta(z)\right)$. By gradient descent, we mean to move z along some segment so as to decrease $\Phi^\delta(z)$ at maximum initial rate. More precisely, defining $z(\tau) = (1-\tau)z + \tau\hat{z}$, we choose $\hat{z} \in F$ to minimize the directional derivative $\frac{d}{d\tau}\Phi^\delta(z(\tau))|_{\tau=0} = \nabla\Phi^\delta(z)(\hat{z} - z) = \alpha(\delta)\pi_0^\delta(z) c\left(\pi^\delta(z)\right)(\hat{z} - z)$. This is equivalent to solving the optimization problem (13) with $\lambda = \pi^\delta(z)$, once for each block $k \in \mathcal{B}$. Thus, solving the Lagrangian relaxation of (12) with this choice of multipliers serves twin purposes: giving not only a lower bound on OPT, but also a primal search direction. That is, we can optimize just a single block k and step in that block to reduce $\Phi^\delta(z)$ without changing other blocks.

B. Application to Video Placement

We now describe how our VoD placement problem maps to (12). Decision variables z's are mapped to where to replicate videos and how to fetch videos from remote locations, and coefficients of z's are determined by video size, popularity, and transfer cost between VHOs as described in (2). As mentioned in Section V-C, constraints (3), (4), (7), and (8)[5] are independent between different videos and define a polytope for each individual video (i.e., F^k for video k). In contrast, disk and bandwidth constraints (5) and (6) require usage aggregation across all videos and are mapped to the main constraints of (12). Specifically, A is determined by video size (for disk) and the streaming rate and number of concurrent flows (for bandwidth), and b by the capacity values.

Given a set of current values for decision variables z's, we can determine how much disk (or network bandwidth) i is being used relative to the capacity, and thus obtain $r_i(z)$. Along with similarly obtained $r_0(z)$, they determine all the parameters (e.g., $\delta, \pi^\delta(z)$) for individual block optimizations.

Algorithm 1 presents a high-level pseudocode. Our full algorithm is extremely intricate, and we refer readers to our accompanying paper [4] for more detail. Here, we highlight some of the key improvements that our algorithm makes over existing work. As mentioned in Section V, fast block heuristics (e.g., [11]) dramatically speed up the algorithm (steps 7 and 15), and we can partially parallelize the chunk iteration (steps 6–10), both with minimal impact on the number of iterations. The simple idea of shuffling the round-robin order for each pass (step 4) has a surprisingly dramatic impact on convergence. Specifically, compared to processing all video blocks in any fixed order, processing them in a different random order for each pass results in fewer iteration passes by a factor of 40+. We also improve the algorithm running time and convergence by employing new update mechanisms (steps 11, 14, and 15) for $\delta, \pi^\delta(z)$, and B, which are different from the theory and previous experimental work.

[5]Our focus here is on the LP relaxation case where $y_i^m \geq 0$.

Algorithm 1 EPF-based Algorithm Pseudocode

```
 1: Parameters: approximation tolerance $\epsilon > 0$, exponent
    factor $\gamma \approx 1$, smoothing parameter $\rho \in [0,1)$, chunk
    size $s \in \mathbb{N}$
 2: Initialize: solution $z \in F$, LB = valid lower bound on
    OPT, UB $\leftarrow \infty$, objective target $B \leftarrow$ LB, smoothed
    duals $\bar{\pi} = \pi^\delta(z)$, scale parameter $\delta = \delta(z)$, number
    of chunks $N_{\text{ch}} = \lceil K/s \rceil$
 3: for Pass = $1, 2, \ldots$ do
 4:    Select a permutation $\sigma$ of the blocks $\mathcal{B}$ uniformly at
       random, and partition $\mathcal{B}$ into chunks $C_1, \ldots, C_{N_{\text{ch}}}$,
       each of size $s$, according to $\sigma$.
 5:    for chunk $C = C_1, \ldots, C_{N_{\text{ch}}}$ do
 6:       for each block $k \in C$ do
 7:          optimize block: $\hat{z}^k \leftarrow \arg\min_{z^k \in F^k} c^k(\pi^\delta(z))$
 8:          compute step size: $\tau^k \leftarrow$
             $\arg\min_{\tau \in [0,1]} \Phi^\delta(z + \tau(\hat{z}^k - z^k))$
 9:          take step in this block: $z^k \leftarrow z^k + \tau^k(\hat{z}^k - z^k)$
10:          Save $\hat{z}^k$ for possible use in shortcutting step 15
             later.
11:       shrink scale if appropriate: $\delta = \min(\delta, \delta(z))$
12:       if $\delta_c(z) \leq \epsilon$ (i.e., $z$ is $\epsilon$-feasible) and $cz <$ UB
          then UB $\leftarrow cz, z^* \leftarrow z$
13:       if UB $\leq (1+\epsilon)$LB then return $z^*$
14:       $\bar{\pi} \leftarrow \rho\bar{\pi} + (1-\rho)\pi^\delta(z)$
15:    lower bound pass: LB $\leftarrow \max(\text{LB}, LR(\bar{\pi}))$,
       $B \leftarrow$ LB
16:    if UB $\leq (1+\epsilon)$LB then return $z^*$
```

Acknowledgment

The authors thank R. Jana for providing the trace data. They also thank R. Jana, D. Phillips, and the anonymous referees of the conference papers for their comments and feedback.

References

[1] "Amazon Prime vs. Netflix—Difference and comparison," [Online]. Available: http://www.diffen.com/difference/Amazon_Prime_Instant_Video_vs_Netflix

[2] C. Denison, "Sony feeds starving 4K early adopters with over 70 titles of 4K movies and TV shows," Sep. 2013 [Online]. Available: http://www.digitaltrends.com/home-theater/sony-launches-4k-video-unlimited-download-service-with-70-titles

[3] H. A. Alzoubi, S. Lee, M. Rabinovich, O. Spatscheck, and J. V. der-Merwe, "Anycast CDNs revisited," in *Proc. WWW*, 2008, pp. 277–286.

[4] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. Ramakrishnan, "Content placement via the exponential potential function method," in *Integer Programming and Combinatorial Optimization*, M. Goemans and J. Correa, Eds. Berlin, Germany: Springer, 2013, vol. 7801, Lecture Notes in Computer Science, pp. 49–61.

[5] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan, “Optimal content placement for a large-scale vod system,” in *Proc. 6th ACM Co-NEXT*, New York, NY, USA, 2010, pp. 4:1–4:12.

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

[7] I. D. Baev, R. Rajaraman, and C. Swamy, “Approximation algorithms for data placement problems,” *SIAM J. Comput.*, vol. 38, no. 4, pp. 1411–1429, 2008.

[8] A. Barbir *et al.*, “Known content network (CN) request-routing mechanisms,” RFC 3568, Jul. 2003.

[9] S. Borst, V. Gupta, and A. Walid, “Distributed caching algorithms for content distribution networks,” in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.

[10] 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 *Proc. ACM IMC*, 2007, pp. 1–14.

[11] M. Charikar and S. Guha, “Improved combinatorial algorithms for facility location problems,” *SIAM J. Comput.*, vol. 34, no. 4, pp. 803–824, 2005.

[12] L. W. Dowdy and D. V. Foster, “Comparative models of the file assignment problem,” *Comput. Surv.*, vol. 14, no. 2, pp. 287–313, 1982.

[13] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, “Summary cache: A scalable wide-area web cache sharing protocol,” *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, Jun. 2000.

[14] N. Garg and J. Könemann, “Faster and simpler algorithms for multicommodity flow and other fractional packing problems,” *SIAM J. Comput.*, vol. 37, no. 2, pp. 630–652, 2007.

[15] L. Guo, E. Tan, S. Chen, Z. Xiao, and X. Zhang, “Does internet media traffic really follow Zipf-like distribution?,” in *Proc. ACM SIGMETRICS*, New York, NY, USA, 2007, pp. 359–360.

[16] C. Huang, J. Li, and K. W. Ross, “Can internet video-on-demand be profitable?,” in *Proc. ACM SIGCOMM*, 2007, pp. 133–144.

[17] Y. Koren, “Factorization meets the neighborhood: A multifaceted collaborative filtering model,” in *Proc. ACM SIGKDD*, 2008, pp. 426–434.

[18] D. Lee *et al.*, “LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies,” *IEEE Trans. Comput.*, vol. 50, no. 12, pp. 1352–1361, Dec. 2001.

[19] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, “On the placement of web server replicas,” in *Proc. IEEE INFOCOM*, 2001, pp. 1587–1596.

[20] A. Sharma, A. Venkataramani, and R. K. Sitaraman, “Distributing content simplifies isp traffic engineering,” in *Proc. ACM SIGMETRICS*, 2013, pp. 229–242.

[21] N. Spring, R. Mahajan, and D. Wetherall, “Measuring ISP topologies with RocketFuel,” in *Proc. ACM SIGCOMM*, 2002, pp. 133–145.

[22] F. Thouin and M. Coates, “Equipment allocation in video-on-demand network deployments,” *Trans. Multimedia Comput. Commun. Appl.*, vol. 5, no. 1, pp. 5:1–5:22, Oct. 2008.

[23] V. Valancius, N. Laoutaris, L. Massoulié, C. Diot, and P. Rodriguez, “Greening the internet with nano data centers,” in *Proc. ACM CoNEXT*, 2009, pp. 37–48.

[24] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, “Ceph: A scalable, high-performance distributed file system,” in *Proc. 7th USENIX Symp. Oper. Syst. Design Implement.*, 2006, pp. 307–320.

[25] Y. Zhao, D. L. Eager, and M. K. Vernon, “Network bandwidth requirements for scalable on-demand streaming,” *IEEE/ACM Trans. Netw.*, vol. 15, no. 4, pp. 878–891, Aug. 2007.

[26] X. Zhou and C.-Z. Xu, “Optimal video replication and placement on a cluster of video-on-demand servers,” in *Proc. IEEE ICPP*, 2002, pp. 547–555.

David Applegate received the B.Sc. degree from the University of Dayton, Dayton, OH, USA, and the Ph.D. degree in computer science from Carnegie Mellon University, Pittsburgh, PA, USA, in 1991.

He then spent 4 years as a Member of Technical Staff with AT&T Bell Laboratories, Murray Hill, NJ, USA, and 4 years as an Associate Professor with Rice University, Houston, TX, USA, before joining AT&T Labs. He is currently a Lead Inventive Scientist with AT&T Labs—Research. His research interests include combinatorial optimization, data structures and algorithms, and network design and management.

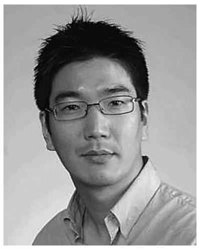
Dr. Applegate has received the 2000 Beal-Orchard-Hays Prize, the 2007 Frederick W. Lanchester Prize, the 2007 William R. Bennett Prize, the 2013 George Polya Award, and the 2013 AT&T Fellows honor.

Aaron Archer received the B.S. degree in mathematics from Harvey Mudd College, Claremont, CA, USA, and the Ph.D. degree in operations research from Cornell University, Ithaca, NY, USA, where he was supported by a Hertz Fellowship.

He then spent nearly a decade with the AT&T Shannon Research Laboratory, Florham Park, NJ, USA, before joining Google, New York, NY, USA, in 2013, where he is a Research Scientist with the Market Algorithms and Optimization group. His research interests span combinatorial optimization, approximation algorithms, algorithmic game theory, and large-scale graph algorithms, plus industrial applications of each, such as wired and wireless network design and optimization, scheduling, and streamlining of data center operations.

Vijay Gopalakrishnan (M'09) received the M.S. and Ph.D. degrees in computer science from the University of Maryland, College Park, MD, USA, in 2003 and 2006, respectively.

He has been with AT&T since 2006 and has worked on innovative solutions in the space of network management, content delivery, and the mobile Web. He is currently a Director with the Network and Service Quality Management Center, AT&T Labs—Research, Bedminster, NJ, USA, leading a team of researchers focused on systems challenges in the architecture, protocols and management of networks.



Seungjoon Lee received the Bachelor's and Master's degrees from Seoul National University, Seoul, Korea, in 1996 and 2000, respectively, and the Ph.D. degree from the University of Maryland, College Park, MD, USA, in 2006, all in computer science.

He is currently with Two Sigma Investments, LLC, New York, NY, USA. Before joining Two Sigma in 2014, he was with AT&T Research, Bedminster, NJ, USA, for more than 8 years. His research interests include large-scale systems, network management, content distribution, cloud computing, and mobile computing.

K. K. Ramakrishnan (S'76–A'83–M'03–SM'04–F'05) received the M.S. degree in automation from the Indian Institute of Science, Bangalore, India, in 1978, and the M.S. and Ph.D. degrees in computer science from the University of Maryland, College Park, MD, USA, in 1981 and 1983, respectively.

He is a Professor with the Computer Science and Engineering Department of the University of California, Riverside, CA, USA. From 1994 until 2013, he was with AT&T, most recently a Distinguished Member of Technical Staff with AT&T Labs—Research, Florham Park, NJ, USA. Prior to 1994, he was a Technical Director and Consulting Engineer in Networking with Digital Equipment Corporation, Littleton, MA, USA. Between 2000 and 2002, he was with TeraOptic Networks, Inc., Sunnyvale, CA, USA, as Founder and Vice President. He has published more than 200 papers and has 145 patents issued in his name.

Dr. Ramakrishnan is also an AT&T Fellow, recognized in 2006 for his work on congestion control, traffic management and VPN services, and for fundamental contributions on communication networks with a lasting impact on AT&T and the industry. He has been on the editorial board of several journals and has served as the TPC Chair and General Chair for several networking conferences. He received an AT&T Technology Medal in 2013 for his work on mobile video delivery strategy and optimization. His work on the “DECbit” congestion avoidance protocol received the ACM SIGCOMM Test of Time Paper Award in 2006.