OPTIMIZING VIDEO-ON-DEMAND WITH SOURCE CODING

S.-H. Gary Chan   *Zhuolin Xu*

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
Kowloon, Hong Kong
Email: {gchan, fanniexu}@cse.ust.hk

Ning Liu

School of Software
Sun Yat-sen University
Guangzhou, China, 510006
Email: liuning2@mail.sysu.edu.cn

ABSTRACT

In order to cost-effectively serve a large number of users, video-on-demand (VoD) content providers often place distributed servers close to user pools. These servers have heterogeneous streaming and storage capacities, and collaboratively share contents with each other. A critical challenge is how to optimize movie storage and retrieval so as to minimize system deployment cost due to server streaming, server storage, and network transmission between servers.

Using a general and comprehensive cost model, we propose a novel VoD architecture using linear source coding. All the movies are source-encoded once at the repository, by coding k source symbols of movie m to $n^{(m)}$ source-coded symbols. These coded symbols are then distributed to the servers. We optimize $n^{(m)}$ and the number of symbols to retrieve from each server for a request. Our solution approaches asymptotically to global optimum as k increases. We show that even when k is low (say, 30), near optimality can be achieved. Furthermore, the solutions on $n^{(m)}$, symbol distribution and retrieval can be efficiently computed with a linear program (LP). Through extensive simulation, our algorithm is shown to achieve substantially the lowest cost, outperforming traditional and state-of-the-art heuristics by a significantly wide margin (by multiple times in many cases).

***Index Terms*—** Video-on-demand; Source coding; LP optimization

1. INTRODUCTION

Distributed video-on-demand (VoD) has emerged as an important and lucrative cloud service. In order to provide such service in a cost-effective manner scalable to large number of users, a content provider often deploys distributed servers close to user pools. These servers cooperatively store and retrieve movies depending on their popularity. In this work, we aim to minimize the total deployment cost by optimizing movie storage and retrieval in the servers.

We consider a distributed and cooperative VoD network which consists of a central server (repository) storing all the movies and proxy servers placed close to user pools.[1] While the central server stores all the movies, the proxy servers are of limited (and possibly heterogeneous) storage which can only locally store a fraction of the movies. Each user has a home (or local) server to serve his movie request. If the request is a hit, the home server directly streams to the users from its local storage. On the other hand, if it is a miss, the home server searches for the content in the server network and requests the content from a remote server (which is either a proxy server or the central server) to serve the request. In other words, the missed content is streamed "via" the home server to the request. Therefore, the bandwidth of the central and proxy servers is used to stream not only to its own home users (if any), but also to remote servers requesting their contents. In this paper, we use the term "servers" to collectively refer to the central and proxy servers.

A critical challenge is to minimize the total system *deployment cost* given by the sum of server and network costs by optimizing movie storage and retrieval at the servers. We consider *server cost* as a general function of its storage and bandwidth (maximum or utilized). In addition, we consider *network cost* as the bandwidth cost to stream symbols from a server to another (a function of the data traffic). If two servers cannot have direct connection, we may set the network cost between them as infinite.

For efficient movie storage and retrieval, we propose in this paper a novel VoD network making use of linear source coding (SC) which achieves *asymptotically optimal* solution. Even under the general and realistic case far from the asymptotic condition, the system performs closely optimal, significantly better than the other state-of-the-arts heuristics (usually by many times).

In the network, movie m is source-encoded only once at the repository by taking k source symbols to generate

This work was supported, in part, by Microsoft Research Asia Academic Program (MRA12EG01), HKUST (FSGRF12EG05 and FSGRF13EG15), and the Fundamental Research Funds for the Central Universities (under Grant no. 2010620003161035). Corresponding author is Ning Liu.

[1]In this paper, we use "client" and "user" interchangeably. We also use "movie," "video" and "content" interchangeably.

$n^{(m)} \geq k$ coded symbols (using a general linear source coding technique such as Reed-Solomon code, Maximum-Distance-Separable (MDS) systematic erasure code, etc.), where $n^{(m)}$ is the optimizing parameter depending on the movie popularity, and k is a network-wide parameter depending on how much coding complexity and decoding delay one is willing to accept. So long as k out of the $n^{(m)}$ symbols are collected, the original source symbols can be recovered.

The repository stores k of these $n^{(m)}$ coded symbols, so that contents are always available in the network. The remaining symbols are stored at the other servers without duplication. A proxy server hence stores any number (obviously no more than k) of the coded symbols.

To serve a local request for a movie, one hence may imagine that the request carries a "bucket" of size k symbols to be filled by *any* of the servers in the network. Once k symbols are collected, the source symbols can then be recovered and played back. The bucket is first filled by the coded symbols at its home server. If this does not fully fill up the bucket, the home server collects by pulling the remainder of the symbols from the other servers (including the repository).

The major issues are hence, given k, what the optimal $n^{(m)}$ is for movie m, how many symbols should be stored at a server and how much to retrieve from other servers to serve a request in order to minimize system cost.

Our contributions are three-folds:

- *General and comprehensive consideration of bandwidth and storage for video-on-demand:* Previous work in VoD seldom considers the inter-dependency among server bandwidth, server storage and network traffic in cost optimization. We present the optimization of a VoD network capturing all these parameters. Our cost model is hence more general and comprehensive.

- *Bucket-filling: A novel movie distribution and retrieval algorithm with source coding:* We propose a novel video-on-demand network using linear source coding. A request for a movie can be satisfied by filling a bucket of size k symbols. Our scheme, termed bucket-filling, is remarkably simple and effective for movie distribution and retrieval.

- *Asymptotically optimal performance for distributed video-on-demand:* By optimizing $n^{(m)}$ for movie m, bucket-filling is able to make the best use of limited server storage, efficiently utilize server bandwidth, and greatly reduce network access cost. With Bucket-filling, we can use an efficient linear program (LP) to optimize symbol distribution and retrieval. The performance can be arbitrarily close to the exact optimum as k increases (i.e., asymptotically optimal in terms of k). We show that even under the most general and realistic condition of low values of k, the system performs closely optimal.

We conduct extensive simulation and comparison study of bucket-filling with other traditional and state-of-the-art schemes. Our results show that bucket-filling outperforms them by a significantly wide margin (by multiple times in most cases). Our results show that the performance of many previously proposed heuristics are still far from optimal, and bucket-filling can achieve performance arbitrarily close to the optimum.

We briefly discuss previous work as follows. Many heuristics have been proposed for movie replication and retrieval [1–6]. These algorithms are generally sub-optimal and their performance bounds are not easy to analyze or derive. In contrast, bucket-filling achieves *asymptotically optimal* performance by increasing the parameter k. For the work studying the cost issue of VoD [1, 4, 7–9], they often have not sufficiently considered the more general case with network access cost, storage constraint and streaming cost of the servers. Our model captures all these elements, leading to a more complete, realistic and practical formulation.

This paper is organized as follows. We describe bucket-filling algorithm in Section 2, in terms of its operation with source coding, problem formulation as a linear program, and its solutions for movie storage and retrieval. In Section 3, we present illustrative simulation results. We conclude in Section 4.

2. BUCKET-FILLING ALGORITHM

2.1. System description

A movie m is source-coded only once at the repository by taking k source symbols to generate $n^{(m)} \geq k$ equal-sized coded ones. As a user has to collect k symbols in order to decode the video, k is a tunable system parameter depending on the level of coding delay and complexity the provider is willing to tolerate. Out of the $n^{(m)}$ coded symbols, the repository stores any of the k coded symbols, and distributes the remainder without replication to the proxy servers.

In the network, movies are distributed and retrieved according to the following:

- *Coded Symbol Distribution:* The repository encodes the movies once and then distributes the coded symbols of the movies to each server. The symbol distribution needs to be done only upon major system changes, e.g., upon the introduction and removal of movies or change in movie popularity which affect movie storage in a major way.

- *Coded Symbol Retrieval:* A movie request carries a "bucket" of size k symbols. If its home server has not stored, and hence cannot supply, enough k symbols to serve the request, it "pulls" the missing ones from the other proxies or central servers. Through this *bucket-filling* mechanism, the servers cooperatively store and supply symbols on-demand with each other to fulfill requests.

2.2. Cost optimization as a linear program

In this section, we present the cost-optimization problem of our VoD network as an LP, which can be efficiently solved at the central server. The overlay network is modeled as a directed graph $G = (V, E)$, where V is the set of central and proxy servers and $E = V \times V$ is the set of overlay edges connecting nodes in V (may not be complete). Let M be the set of movies and $L^{(m)}$ be the movie length in seconds (i.e., movie length before network coding). Let $p^{(m)}$ be the popularity of movie m, which is the probability that a user requests movie m, where $\sum_{m \in M} p^{(m)} = 1$.

Each movie is network-coded to different length (obviously no less than $L^{(m)}$). Let $I_v^{(m)}$ (seconds) be the amount of coded movie m that server v stores. Obviously, we require

$$0 \leq I_v^{(m)} \leq L^{(m)}, \quad \forall v \in V, m \in M. \tag{1}$$

Note that for the repository (i.e., central server), we require $I_v^{(m)} = L^{(m)}, \forall m \in M$.

Server v has a certain storage capacity B_v (seconds). To meet storage requirement, we require

$$\sum_{m \in M} I_v^{(m)} \leq B_v, \quad \forall v \in V. \tag{2}$$

Let λ_v be the total movie request rate at server v (requests per second); the request rate for movie m at the server is hence $p^{(m)}\lambda_v$. Further let $\alpha^{(m)}L^{(m)}$ be the average holding (or viewing) time for movie m, where $\alpha^{(m)} \geq 0$.

Each user retrieves data from the servers (including his home server) proportional to his holding time. Let $r_{uv}^{(m)}$ (seconds) be the amount of movie m supplied from server u to server v for a user holding time of $L^{(m)}$. In order to playback the movie, the supply of the coded symbols must satisfy

$$\sum_{u \in V} r_{uv}^{(m)} \geq L^{(m)}, \quad \forall v \in V, m \in M. \tag{3}$$

The actual amount of streamed data is given by $\alpha^{(m)} r_{uv}^{(m)}$. As the server cannot supply more than that it stores, we need

$$0 \leq r_{uv}^{(m)} \leq I_u^{(m)}, \quad \forall u, v \in V, m \in M, \tag{4}$$

and, by definition,

$$r_{vv}^{(m)} = I_v^{(m)}. \tag{5}$$

Let Γ_{uv} (bits/s) be the total network bandwidth used for symbol transmission from server u to v, which can be obtained as $\Gamma_{uv} = \sum_{m \in M} p^{(m)} \lambda_v \alpha^{(m)} r_{uv}^{(m)} s, \forall u, v \in V$. By definition, $\Gamma_{uu} = 0$, for $u \neq v$.

Let C_{uv}^N be the network cost due to the traffic from server u to v. It is a monotonically non-decreasing piece-wise linear function in Γ_{uv}. Γ_{uv}, i.e., $C_{uv}^N = \mathbb{C}_{uv}^{\mathbb{N}}(\Gamma_{uv}), \forall u, v \in V$, with $C_{uu}^N = 0$. The total network cost C^N is hence $C^N = \sum_{u,v \in V} C_{uv}^N$.

The servers help each other using "cache and stream" mechanism, i.e., a remote server streams to a user *through* his home server. In other words, the home server is an intermediate node between the remote server and users. For any remote server $v \in V$, the data rate the server u "pulls" from server v for movie m is $p^{(m)}\lambda_u(\alpha^{(m)} r_{vu}^{(m)} s)$. The total rate (bits/s) that server v serves other servers is hence $R_v = \sum_{m \in M} \sum_{u \in V, u \neq v} \Gamma_{vu}, \forall v \in V$.

Let C_v^S be the cost of operating server v, which is a monotonically non-decreasing piece-wise linear function in B_v and R_v, i.e., $C_v^S = \mathbb{C}_v^{\mathbb{S}}(B_v, R_v), \forall v \in V$. In other words, the server cost consists of storage cost and streaming cost. The aggregated server cost C^S, therefore, is $C^S = \sum_{v \in V} C_v^S$.

Finally, the total system deployment cost C is

$$C = C^S + C^N. \tag{6}$$

We state our cost-optimization problem as follows:

Optimal Movie Distribution and Retrieval Problem to Minimize Deployment Cost: Given topology G, user demand $\{\lambda_v\}$, storage capacity $\{B_v\}$, movie popularity $\{p^{(m)}\}$ and cost functions $\mathbb{C}_{uv}^{\mathbb{N}}$ and $\mathbb{C}_v^{\mathbb{S}}$, we seek to minimize the total cost given by Equation (6), subject to Equations (1) to (5). The output is the optimal solution of the amount of the movie stored in each server (i.e., $\{I_v^{(m)}\}$) and the retrieval amount between servers (i.e., $\{r_{uv}^{(m)}\}$).

Note that, for any arbitrary piece-wise linear functions of C_v^S and C_{uv}^N, the above problem becomes a linear programming (LP) problem which can be solved efficiently.

2.3. Parameter discretization

The LP yields optimal solution for system parameters $\{I_v^{(m)}\}$ and $\{r_{uv}^{(m)}\}$ for movie m. Given these parameters, the movie can then be encoded, distributed and retrieved according to the following:

- *Movie encoding:* To obtain the encoding parameter $n^{(m)}$, observe that the network-coded and raw movie lengths must satisfy the following equation:

$$\frac{\sum_{v \in V} I_v^{(m)}}{L^{(m)}} = \frac{n^{(m)}}{k}, \quad \forall m \in M,$$

  i.e.,

$$n^{(m)} = \frac{\sum_{v \in V} I_v^{(m)}}{L^{(m)}} k, \quad \forall m \in M. \tag{7}$$

- *Symbol distribution (storage):* The number of symbols that server v stores is given by

$$n_v^{(m)} = \frac{I_v^{(m)}}{L^{(m)}} k, \quad \forall m \in M. \tag{8}$$

- *Symbol retrieval (collection):* The number of symbols for server u to stream to server v is

$$n_{uv}^{(m)} = \frac{r_{uv}^{(m)}}{L^{(m)}}k, \quad \forall m \in M. \tag{9}$$

Note that $\{n^{(m)}\}$, $\{n_v^{(m)}\}$ and $\{n_{uv}^{(m)}\}$ in Equations (7)–(9) are the asymptotic optimal solutions when k is large. For finite k, they should be discretized to integral values. We present below a simple discretization approach which converges to the asymptotic optimum as k increases. The basic idea is that each proxy tries to match the optimal LP solution as much as possible through integer rounding. In symbol retrieval by filling a "bucket," the shortfall in symbols due to rounding can be obtained from the repository:

- *Discretize* $\{n_v^{(m)}\}$: We first round down the result $\{n_v^{(m)}\}$ as obtained in Equation (8) to its closest integers. For each server v, it first stores according to these integers for all the movies. This clearly does not violate its storage constraint (given in Equation (2)). For the residual storage it then stores a symbol of each movie in decreasing popularity until its total storage is fully used up.

  After this, the new $\{n_v^{(m)}\}$ are of integral values. The coding information $n^{(m)}$ for movie m is then given by

$$n^{(m)} = \sum_{v\in V} n_v^{(m)}, \quad \forall m \in M. \tag{10}$$

- *Discretize* $\{n_{uv}^{(m)}\}$: This is similar to the discretization of $\{n_v^{(m)}\}$. First we write $\{n_{uv}^{(m)}\}$ in Equation (9) as the sum of an integral part and a positive fractional part. Clearly, the integral part does not violate the supply constraint as given in Equation (4).

  To recover the source packet (Equation (3)), we first rank the movies according to decreasing popularity. We then conditionally round up the fractional parts to 1 of the movies until Equation (3) is satisfied, and the remaining fraction is rounded down to 0. If Equation (3) is still violated after all the rounding, the remaining symbols are assigned to the repository.

It is clear from above that the system performance can be arbitrarily close to the exact optimum as k increases (i.e., asymptotically optimal). The discretization steps guarantee that all the movies can be recovered at each server. In Section 3, we can see from the experiments that the performance penalty due to rounding is very low.

3. ILLUSTRATIVE SIMULATION RESULTS

3.1. Simulation environment and performance metrics

In this section, we present our simulation environment and performance metrics to study the performance of bucket-filling.

The VoD network consists of a number of distributed proxy servers. All our results are obtained at steady state. Unless otherwise stated, we use the default values as follows for our system parameters (the baseline case): $k = 30$, 20 proxy servers, 200 movies, skewness of movie popularity is 0.6, movie length is 90 minutes, movie streaming rate is 1 Mbits/s, total request rate in the network is 0.6 requests/s (equally distributed to the proxies), average movie holding time is movie length (i.e., $\alpha^{(m)} = 1$), c_{uv} between central and proxy server is 0.01 unit/second, c_{uv} between proxies is Zipf with skewness 0.6 and mean 0.005 unit/s, average server storage is 20 movies, skewness of server storage is 0.4, average proxy server bandwidth capacity is 160 Mbits/s, skewness of server bandwidth is 0 (i.e., same bandwidth).

Movie popularity follows the Zipf distribution with skewness parameter s, i.e., the request probability of the ith movie, denoted as $f(i)$, is given by $f(i) \propto 1/i^s$. Requests arrive at each proxy server according to a Poisson process with total rate λ (req./second). The central server has no home users. The proxy servers have heterogeneous storage space and bandwidth following a Zipf distribution (independent of each other). The repository stores all the movies with a streaming capacity twice of the average streaming capacity of the proxy servers.

We consider the network cost function from server u to server v proportional to the bandwidth between them, i.e., $C_{uv}^N(\Gamma_{uv}) = c_{uv}\Gamma_{uv}, \forall u, v \in V$, where c_{uv} is some constant (by definition, $c_{vv} = 0$).

The server cost is a function of its storage and its total bandwidth used to serve the remote servers, modeled as $C_v^S = \sigma_B B_v + C_v(R_v), \forall v \in V$, where σ_B is a constant ($\sigma_B = 0.02$ in our simulation), and $C_v(R_v)$ is a piece-wise linear function monotonically increasing in R_v.

We show in Figure 1 $C_v(R_v)$ versus R_v/U_v in our simulation, where U_v is the streaming capacity of the server and hence R_v/U_v is the bandwidth utilization of the server. There are three linear segments formed by points $(0, 0)$, $(0.8, 0.125)$, $(0.93, 0.4375)$ and $(0.99, 1.925)$ (these coordinates are obtained from the queuing model $\sigma_S/U_v - R_v$, where σ_S is some constant). The cost increases with the bandwidth utilization. As the consumed bandwidth R_v approaches the bandwidth capacity U_v, the server cost increases more sharply.

We compare bucket-filling with the following traditional and recent movie replication schemes:

- *Random*, where each server randomly stores movies without considering their popularity. This is a simple storage strategy.
- *MPF (Most Popular First)*, where each server stores the most popular movies. This is a greedy strategy, but does not take advantage of cooperative replication.
- *Local Greedy* [1], which divides the movies into three categories, those popular ones which all servers store (full

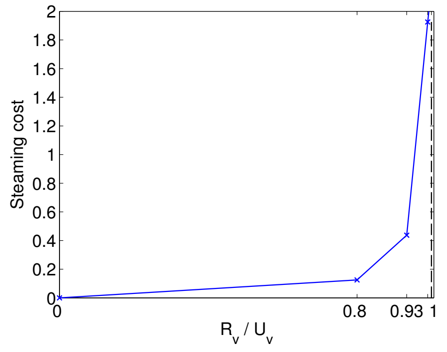

Fig. 1. Streaming cost model at proxy server.

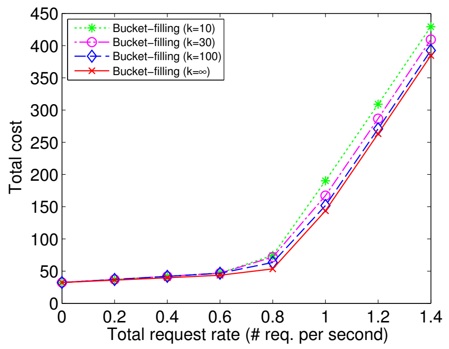

Fig. 2. Total cost versus request rate given k.

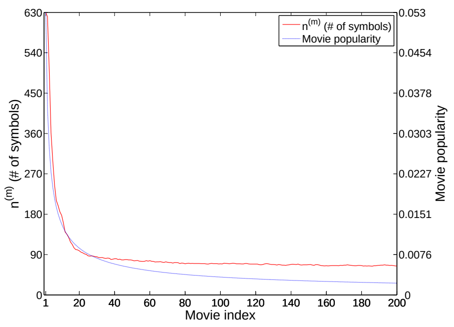

Fig. 3. Optimal $n^{(m)}$ versus movie index.

replication), those medium popular ones which only one proxy server store (single copy), and those unpopular ones which only the repository stores (no copy). By formulating an LP problem, it seeks to minimize network cost. As Local Greedy assumes homogeneous access cost, we set its access cost to be equal to the average access cost between servers in our network.

For all the comparison schemes, upon a miss request, the home server v chooses an available server u which has the requested content with probability proportional to $1/c_{uv}$. It is a reasonable, simple and effective strategy because the server with lower access cost has higher chance to be chosen. With this probabilistic approach, a server with low access cost is not always selected so as to avoid congestion, and hence high network cost, at the server.

3.2. Illustrative results

We plot in Figure 2 the total cost versus request rate given k. The total cost increases with the request rate mainly because of the increase in network load. As k increases, the network approaches the exact optimum (corresponding to the case $k = \infty$). However, for humble value of k (say 30), the performance is already very close to the optimum (less than 6% deviation in this). This shows that our network is highly efficient, with closely optimal performance even for all the practical (finite) value of k.

We show in Figure 3 the optimal $n^{(m)}$ versus movie index. Also shown is the corresponding movie popularity (default setting). We see that the movie popularity exhibits some skewness with a tail (with $s = 0.6$ and $M = 200$, the top 30% of the movies account for close to 60% of the total traffic). The optimal $n^{(m)}$ decreases with movie popularity. This is reasonable because the servers tend to locally store more of those popular movies to reduce transmission cost in the network. For the unpopular ones, fewer symbols are generated and stored in the whole network. We see that no matter how unpopular the movie is, the number of symbols is higher than k, meaning that some symbols are stored in the network besides those at the repository.

We compare in Figure 4 the total cost versus the request rate for different schemes. Total cost increases with request rate mainly due to the increase in network traffic. Bucket-filling clearly achieves much lower total cost among all the schemes, beating them by multiple times. In other words, given the same deployment budget, bucket-filling can support much higher request rate (i.e., more concurrent users in the system). MPF does not perform well because it mainly relies on the central server to serve the requests for the unpopular movies. Random, due to its popularity-blind nature, stores insufficient copy of the popular movies, leading to considerable cost. Local Greedy has lower cost due to its cost optimization. Bucket-filling achieves by far the best performance because it achieves near optimality by taking into account of not only the network transmission cost but also the server storage and streaming cost.

We plot in Figure 5 individual server cost for different schemes. We sort the proxy servers according to their storage in ascending order (as their streaming capacity is the same in the default setting), and the last one refers to the repository. As the storage of a proxy increases, its cost increases because it needs to server more remote requests. Bucket-filling utilizes very well the finite storage and bandwidth resources of proxy servers, leading to significantly lower repository streaming cost. It has strong server cooperation to achieve nearly optimal system performance. As MPF only stores the most popular movies at the proxy servers, it has lower proxy cost at the steep sacrifice of repository cost. The proxies barely contribute their bandwidth and storage to cooperatively help each others. Local Greedy, with network cost optimization, outperforms Random in both proxy server cost and repository cost.

We show in Figure 6 the cost of each movie for different schemes. The movies are sorted according to their popularity in descending order. The popularity-based schemes (i.e., bucket-filling, Local Greedy and MPF) tend to locally store the popular movies, and hence those popular ones enjoy lower cost at much sacrifice of those not-so-popular movies.

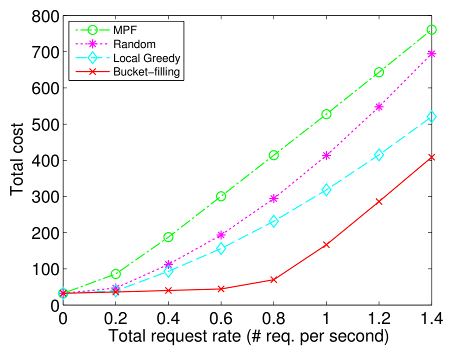

Fig. 4. Total cost versus request rate given different schemes.

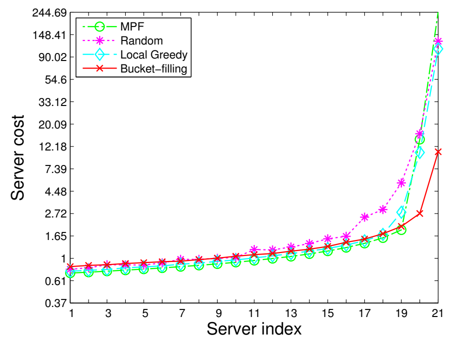

Fig. 5. Server cost distribution given different schemes.

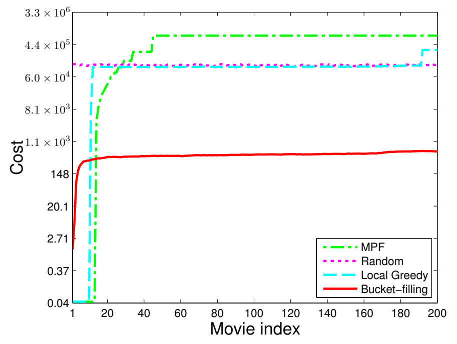

Fig. 6. Cost of each movie given different schemes.

Bucket-filling makes much better movie storage and retrieval decisions by cooperatively storing the movies. While having slightly higher cost for popular movies, most of the movies in bucket-filling have quite uniform and low access cost. Bucket-filling accomplishes much better optimality with the cost of not-so-popular movies strikingly much lower by orders of magnitude than the other schemes. This is the main factor of its success. For MPF, its high cost mainly comes from the less popular movies. Random treats each movie equally and thus has the most uniform cost distribution.

4. CONCLUSION

In this work, we have studied optimal movie distribution and retrieval to minimize deployment cost for video-on-demand (VoD) with distributed servers. The deployment cost captures the costs of server streaming, server storage and network transmission cost. We have studied a VoD network using source coding which asymptotically achieves exactly optimum depending on a coding parameter. Movies are distributed and retrieved efficiently using our proposed "bucket-filling" algorithm.

We have formulated the optimization problem as a linear program which can be solved efficiently. We have conducted extensive simulation to compare the performance of bucket-filling with other traditional and state-of-the-art schemes. Our results show that bucket-filling achieves its close optimality with substantially much lower cost, and outperforms the other schemes by a wide margin (multiple times in many cases, and more than 100% in most cases).

5. REFERENCES

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

[2] S. Zaman and D. Grosu, "A distributed algorithm for the replica placement problem," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 9, pp. 1455–1468, Sept. 2011.

[3] Le Chang and Jianping Pan, "Reducing the overhead of view-upload decoupling in peer-to-peer video on-demand systems," in *IEEE International Conference on Communications (ICC '11)*, June 2011, pp. 1–5.

[4] A. Nimkar, C. Mandal, and C. Reade, "Video placement and disk load balancing algorithm for VoD proxy server," in *IEEE International Conference on Internet Multimedia Services Architecture and Applications (IMSAA '09)*, Dec. 2009, pp. 1–6.

[5] M. Hefeeda and B. Noorizadeh, "On the benefits of cooperative proxy caching for peer-to-peer traffic," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 7, pp. 998–1010, July 2010.

[6] Wai-Pun Ken Yiu, Xing Jin, and S.-H. Gary Chan, "VMesh: Distributed segment storage for peer-to-peer interactive video streaming," *IEEE Journal on Selected Areas in Communications Special Issue on Advances in Peer-to-Peer Streaming Systems*, vol. 25, no. 9, pp. 1717–31, Dec. 2007.

[7] Weijie Wu and J. C. S. Lui, "Exploring the optimal replication strategy in P2P-VoD systems: Characterization and evaluation," in *Proceedings of IEEE INFOCOM 2011*, Apr. 2011, pp. 1206–1214.

[8] S.-H. Gary Chan, "Operation and cost optimization of a distributed servers architecture for on-demand video services," *IEEE Communications Letters*, vol. 5, no. 9, pp. 384–386, Sept. 2001.

[9] Yung R. Choe, Derek L. Schuff, Jagadeesh M. Dyaberi, and Vijay S. Pai, "Improving VoD server efficiency with bittorrent," in *Proceedings of conference on Multimedia (MULTIMEDIA '07)*, New York, NY, USA, 2007, pp. 117–126, ACM.