Joint Optimization of Content Replication and Server Selection for Video-On-Demand

Huan Huang Pengye Xia S.-H. Gary Chan
Department of Compute Science and Engineering
The Hong Kong University of Science and Technology
Hong Kong, China
{huangzunhuan, xiapengye, gchan}@cse.ust.hk

Guangyu Shi Hongbo Zhang
Huawei Technologies Co., Ltd
Shenzhen, 518129, China
{shiguangyu, zhanghongbo888}@huawei.com

***Abstract*—We study providing large-scale video-on-demand (VoD) service to distributed users. In order to achieve scalability in user capacity and reduce the load of the core network, local servers with heterogeneous storage are deployed. Each server replicates the movie segments depending on their access probabilities. Considering the realistic scenario that underlay delay is a function of the total traffic in the link (including cross-traffic), we address two important problems to achieve low user interactive delay: 1) Which segments should each server replicate under the constraints of their capacities to achieve network-wide good locality effect? This is the so-called content replication (CR) problem; and 2) Given a number of remote servers with the requested segment, which one should serve the user? This is the so-called server selection (SS) problem.**

CR and SS problems couple with each other. In this paper, we propose a simple and distributed algorithm which seeks to jointly optimize CR and SS. The algorithm, termed CR-SS, achieves good caching locality by adaptively replacing segments and selecting servers with a simple lookup. Simulation results on Internet-like topologies show that CR-SS outperforms existing and state-of-the-art approaches by a wide margin, achieving substantially lower user delay.

***Index Terms*—Joint optimization; distributed algorithm; content replication; server selection; video-on-demand**

I. Introduction

With the penetration of broadband Internet to the home, video-on-demand (VoD) service has attracted much attention recently [1], [2]. This is evident from numerous deployed on-demand Internet movie applications, such as hulu.com, pplive.com and netflix.com. The traditional client-server approach is obviously not scalable to serve a large number of users. In this paper, we consider a scalable distributed server architecture for large-scale VoD, and study its joint distributed optimization of content replication (CR) and server selection (SS).

In the VoD network, there is a repository storing all the video contents. To scale up the streaming capacity and reduce network load, a number of distributed servers are placed close to user pools in the network. The capacity of each server is heterogeneous, and may not replicate all the contents.[1] As streaming servers, they stream their contents to their local users.[2] Each movie is divided into fixed-sized *segments* (say, 5-20 minutes). Each server can replicate any of the segments depending on their capacities.

In the network, the clients may be some set top boxes or PCs and may at any time perform random seeks to any segment of the movies they are viewing. Each user has a home (or local) server, which is his contact point to the VoD service. The user "pulls" the segment of interest from his home server, which serves the user directly if it has replicated locally the segment of interest (we consider the general case that the bandwidth between the users and their home servers is not a bottleneck). Otherwise, it re-directs the request to a remote server (including repository) with the segment stored. The remote server then streams directly to the user once network bandwidth is available. We call such remote server the *streaming parent* of the home server. The users may access the video segments with different probabilities which may slowly vary over time.

We consider the realistic scenario that the underlay link has a certain bandwidth, and its delay is a function of its total traffic including all the cross traffic. There are two important issues which need to be addressed. First, given a server's limited storage capacity, which segments should it replicate to achieve network-wide good locality effect? This is the so-called *content replication (CR)* problem. Second, in case of a miss in the home server and given a number of available remote servers storing the requested segment, which of them should the home server choose to achieve good overall user delay experience? This is the so-called *server-selection (SS)* problem.

The objective of CR is to cooperatively optimize the segments stored at each server. A home server running a good CR strategy should therefore satisfy most of its segment requests by servers within its proximity (content replication), so as to achieve low interactive delay in accessing the segments

This work was supported, in part, by the General Research Fund from the Research Grant Council of the Hong Kong Special Administrative Region, China (611209), Proof-of-Concept Fund at the HKUST (PCF.005.09/10 & PCFX05-15C00610/11ON) and Huawai research award (HUAW14-15C00609/10PN).

[1]For example, the size of a 100-minute movie of bitrate 5 Mbps is around 4GB. To store 10,000 movies, a server requires a capacity of 40TB. Replicating all the movies at a local server is considered expensive, especially when most of them are not very popular.

[2]In this paper, we use "client" and "user" interchangeably.

and to conserve network bandwidth. On the other hand, the objective of SS is to optimize the choice of servers with the same goal of achieving low user delay and avoiding network congestions. Clearly, CR and SS couple with each other, i.e., given a different CR strategy, SS may have a different optimal solution, and vice versa. A joint optimization of CR and SS is hence necessary to achieve the overall best performance.

In this work, we address this joint optimization problem. Optimal replication in VoD is regarded as NP-hard (see, for example, [3]). We propose a distributed and cooperative heuristic called CR-SS (**C**ontent **R**eplication and **S**erver **S**election) to address the joint optimization problem. With CR-SS, each server uses a simple routing table to store segment location. By exchanging the routing table with its neighbors, each server is able to find the best set of server(s) for each segment and makes efficient segment replacement decision. Using a simple penalizing function, CR-SS is fully distributed and efficient, and guaranteed to converge to steady state. It has low server and computational loads.

This paper is organized as follows. In Section II we review related work. We present the distributed algorithm CR-SS in Section III. In Section IV we present illustrative simulation results on the performance and comparison of CR-SS. We conclude in Section V.

II. Related Work

There has been much work on VoD based on file downloading (e.g., [4]). In contrast, our work considers interactive *streaming* with problems in content replication and segment replacement. Other work on VoD focuses on its infrastructure and framework [5]–[7]. Although this body of work contains elements of CR and SS, it has not addressed the critical *distributed joint optimization* of CR and SS which we consider here. The SS problem in the context of overlay routing has been well studied in the literature [8], [9]. There has been work to optimize CR [4]. While these works treat CR and SS independently, we consider the joint optimization of CR and SS here.

Joint optimization of CR and SS has been studied in [10], [11]. We differ by considering the influence of cross traffic in each link and the more realistic scenario where the delay of the underlay links as a non-linear function of the link traffic. This calls for a totally different approach and solution. Borst et al. [10] assumes a specific tree topology with homogeneous link bandwidth and user demand in studying CR and SS problems. However, it has not taken into account the influence of cross traffic in each underlay link, and cannot be extended to a general mesh network topology with heterogeneous link bandwidth and user demand. The work in [11] considers joint optimization of CR and SS for *multi-view* video, which assumes a special coding structure and considers the optimization in the view dimension only.

III. Distributed and Cooperative CR-SS Algorithm

In the VoD network, there is a certain segment access delay between any pair of servers, including repository (the delay may be traffic dependent). We present in this section CR-SS (**C**ontent **R**eplication and **S**erver **S**election), which decides the segments to replicate in each server to minimize the average access delay in the system.

In CR-SS, each server independently minimizes the average segment delay in its local region through segment replacement. It jointly optimizes CR and SS by iteratively minimizing a delay function in a fully distributed manner. Such iteration is guaranteed to converge.

We present first how CR-SS uses a probabilistic framework to select servers (parents), then the segment replacement of CR-SS, followed by the complexity analysis of CR-SS.

A. Probabilistic Server Selection

Every server running CR-SS has a simple routing table, each entry of which corresponds to a video segment. For each video segment, the row contains a certain maximum number n_s of servers with the segment and hence can be chosen as its streaming parent. Each server periodically exchanges its routing table with its neighbors, and pings the path delay to them.

Let E_j^i be the set of servers stored in the entry for segment j at server i. Let d_j^{ik} be the access delay of segment j from server k for users homed at server i. For each server $k \in E_j^i$, the entry also stores a utility value indicating the importance of server k as U_j^{ik}. Obviously, U_j^{ik} should be a monotonically decreasing function in d_j^{ik} (note that our algorithms are not restricted to any particular form of the function). In order to choose close servers as streaming parents and not to direct all the traffic to a particular server, we adopt a probabilistic framework to achieve load spreading. In the framework, the probability that server k is chosen as the streaming parent for segment j is given by

$$P_j^{ik} = \frac{U_j^{ik}}{\sum_{t \in E_j^i} U_j^{it}}. \quad (1)$$

From the equation, if the home server has stored the segment (and hence E_j^i is the server itself), it has probability 1 to become the streaming parent (and hence surely serves its local users). On the other hand, if the segment is not found in the home server, the larger the path delay, the lower is its probability to become the streaming parents.

Consider a user request sent to its home server i for segment j. Server i then inspects the parent list in the corresponding entry of its routing table. The server chooses one of them as streaming parent according to Equation (1).

B. Segment Replacement of CR-SS

From Equation (1), the expected segment delay for users at home server i to get segment j is given by

$$\bar{d}_j^i = \sum_{k \in E_j^i} P_j^{ik} d_j^{ik}. \quad (2)$$

Let λ_j be the popularity of segment j, which is defined as the probability that a user accesses segment j. The average segment access delay at server i is hence given by

$$\bar{d}_i = \sum_j \lambda_j \bar{d}_j^i. \quad (3)$$

Each server has a certain number n neighbors, which are the closest servers according to the delay. Denote this set of neighbors for server i as N^i. The local region $L^i = N^i \cup \{i\}$ is defined as the set including the server i and its neighbors in N^i. The average access delay in L^i is hence given by

$$\bar{d}_{L^i} = \sum_{k \in L^i} \bar{d}^k. \quad (4)$$

In CR-SS, each server i independently minimizes the average segment access delay in its local region (i.e., $\bar{d}_{L^i}$) through segment replacement presented below.

The server initially replicate some segments. The initial assignment is not likely to be optimal, and hence the server goes through segment replacement to improve the performance. A server decides which segment to replace using the routing tables exchanged from its neighbors. Given all its neighbors' routing tables, a server examines the gain (in terms of reduction in access delay) of a certain segment replacement.

When a server starts to replace the segments. It examines the routing table obtained from its neighbor k. Recall that E_j^k is the entry which stores the parent list for segment j in server k. If $i \in E_j^k$, server i is possibly chosen by k as the streaming parent for segment j. However, if server i decides to replace j, server k has to remove i from E_j^k. The expected access delay of segment j at server k (i.e., $\bar{d}_j^{ki}$ in Equation (2)) also changes. Denote this access delay change of segment j at server k as C_j^{ki}. Let $\bar{d}_j^{ki^*}$ and $\bar{d}_j^{ki}$ be the expected segment delay at server k with and without the entry of server i in E_j^k, respectively. Thus, we have

$$C_j^{ki} = \bar{d}_j^{ki} - \bar{d}_j^{ki^*}. \quad (5)$$

We further define the gain G_j^i as the sum of change of the average delay of segment j in server i's local region L^i. Given Equation (4), G_j^i is defined as

$$G_j^i = \sum_{k \in L^i} C_j^{ki}. \quad (6)$$

The server i first calculates the gain for each segment. Then it selects the segment j with the lowest G_j^i among all the replicated segments and the segment t with highest G_t^i among all the non-replicated segments. If $(G_t^i - G_j^i)/\sum_k G_j^k$ is larger than a certain threshold α, it continues to examine the next pair of segments. Otherwise, the server i stops segment replacement. Note that since $(G_t^i - G_j^i)/\sum_k G_j^k$ is always smaller than 1, α must be set smaller than 1; otherwise, the server will not replace any segment. CR-SS is guaranteed to converge, because it continuously reduces the average segment access delay through segment replacement.

C. Exchange Overhead and Computational Load

Let T be the set of segments, and $|T|$ be the number of segments. Note that the servers do not need to exchange the entire full routing tables. They only need to exchange two types of information. First, when a server i performs segment replacement, it only needs the information of the delay change of a neighbor k as given by Equations (5). Such information of the potential delay change can be efficiently calculated at neighbors in $O(|T|)$ time and sent to the server with size $O(|T|)$. Second, after the replacement, the server has to send its segment replication information (in a bitmap of $O(|T|)$ to each of its neighbors. Clearly, the exchange overhead of both types of information is rather small.

The computational load of CR-SS is rather low. As mentioned above, each neighbor computes potential delay change with time complexity $O(|T|)$. After obtaining all such information, a server sorts them in $O(|T| \log |T|)$ time. For segment replacement, we simply calculate the reduction of delay by removing the segment of lowest gain and replacing it with the segment of highest gain. We repeat this process until the reduction of delay is less than the threshold. The running time of the whole process is $O(|T|)$, because the segments have already been sorted. Hence the total running time of CR-SS for replacement is $O(|T| \log |T|)$.

IV. Simulation Environment and Illustrative Results

A. Simulation Environment and Metrics

We have conducted extensive simulations to study the performance of CR-SS on Internet-like topology generated by BRITE. Our BRITE topology is a router-only topology, which contains 3,072 routers and 10,850 underlay links. We will vary some of them to study their effect on system performance. Unless otherwise stated, we use the following baseline (default) parameters in simulation: link bandwidth 5 Gbits/s, number of servers 30, number of segments 1200, capacity 30, $s = 0.4$, streaming bitrate 5 Mbits/s, downloading bitrate 25Mbits/s, number of neighbors 30, segment replacement interval 12 minutes, $\lambda = 6$ requests/minute. (We have also conducted simulations on real Internet topologies obtained from an ISP (http://personalpages.manchester.ac.uk/staff/m.dodge/cybergeography/atlas/more_isp_maps.html). The results are qualitatively the same and omitted here.)

A certain number of servers are randomly attached to the routers in the topology. Each server has a fixed replication capacity and has a fixed number of nearest servers as its neighbors. All segments are of the same size and their popularities follow the Zipf distribution with skewness parameter s. When s is small (e.g., $s = 0$), all segments have similar popularities;

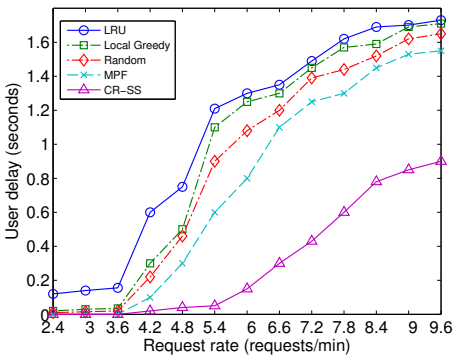

Fig. 1. Average user delay versus request rate given different schemes.

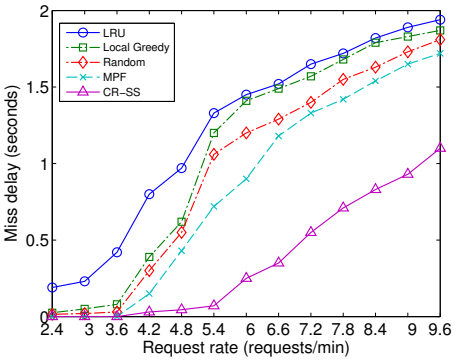

Fig. 2. Average miss delay versus request rate given different schemes.

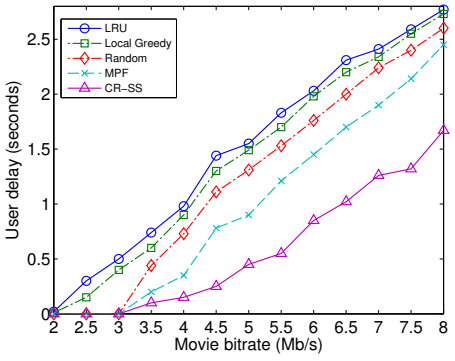

Fig. 3. Average user delay versus streaming bitrate given different schemes.

on the other hand, when s is high (e.g., $s = 1$), the distribution is more skewed.

We write our event-driven simulation in Java. Segment requests arrive at a server according to a poisson process with rate λ (requests/minute). For the server selection, we set U_j^{ik} to $1/d_j^{ik}$ in Equation (1). A server periodically replaces its cached segments. When a replacement decision is made, the server downloads the segment from another server with the segment according to the server selection algorithm. The downloading bitrate is higher than the streaming bitrate (2 times in our simulation). When the downloading is completed, the server informs its neighbors its new routing table and the neighbors update theirs accordingly. For each experiment, we run it till steady state, after which we take the statistics.

The performance metrics that we are interested in are:

- *User (interactive) delay*, which is defined as the delay from the request of a segment until the segment is streamed to the user. As a result, the delay is the sum of the path delay of the segment from the request and the waiting time for the end-to-end streaming bandwidth to be available in case the network and servers are congested.
- *Miss delay*, which is defined as the user delay for the requested segment not stored by the home servers. We are interested in both its average and distribution.
- *Wait probability*, which is the probability of a segment request which, due to network and server congestion, needs to wait for the available bandwidth before it is served. This is measured as the ratio of the number of requests that cannot be served immediately to the total number of requests.
- *Reneging rate*, due to the abortion of a segment request on impatience after waiting for a certain amount of time (10 seconds in our simulation). Reneging rate is calculated as the fraction of requests leaving the system without being served.

We compare CR-SS with the following schemes:

- *Random*, where each server randomly replicates segments without considering the segment popularity;
- *MPF (Most Popular First)*, where each server only replicates the most popular segments;
- *LRU (Least Recently Used)*, where each server only replicates the segments that are most recently requested at itself;
- *Local Greedy Replication*, where each server independently replaces segments to reduce access cost in accordance with [10]. The access cost takes into consideration of both segment access delay and segment popularity.

For all of the above comparison schemes, if a request cannot be served directly by its local storage, the home server will choose the nearest server as the streaming parent.

B. Illustrative Results

We present illustrative results in this section. Figure 1 plots the average user delay versus request rate given different schemes. User delay increases with request rate due to the increase in network and server load. CR-SS clearly achieves much lower user delay among all the schemes. LRU and Local Greedy do not perform well mainly because of their high frequency of segment replacement, which incurs large amount of download traffic in the network. Random replacement does not have any segment replacement, and hence achieves lower replacement traffic. MPF, due to its replication of only the most popular segments, does not take advantage of *content* replication. CR-SS achieves the best performance because it has low replacement overhead.

Figure 2 plots the average miss delay versus request rate given different schemes. We have similar observation here as the user delay. CR-SS achieves much lower miss delay as compared with all other schemes due to its better content replication and server selection. As the performance of miss delay is qualitatively the same as user delay, we will only show user delay in the following.

Figure 3 plots the average user delay versus the streaming bitrate given different schemes. User delay increases with the streaming bitrate, because network traffic increases with streaming bitrate. It is clear that CR-SS achieves much lower user delay as compared with all other schemes. Given the same requirement on user delay, CR-SS is able to support a much higher streaming bitrate and hence video quality.

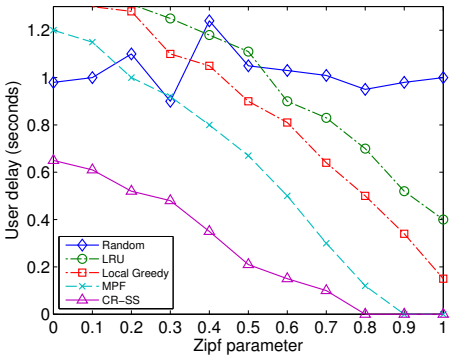

Fig. 4. Average user delay versus zipf parameter s given different schemes.

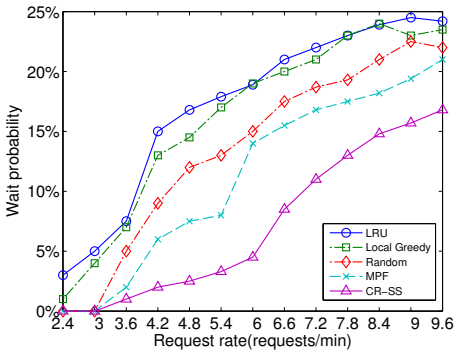

Fig. 5. Average wait probability versus request rate given different schemes.

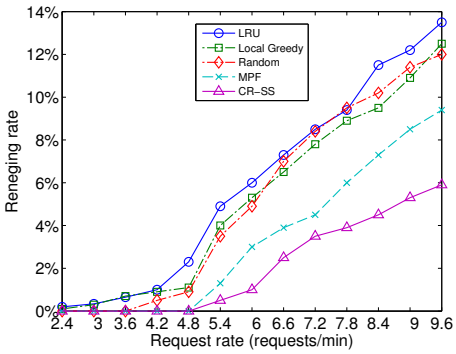

Fig. 6. Average reneging rate versus request rate given different schemes.

Figure 4 plots the average user delay versus zipf parameter s (i.e., skewness) given different schemes. User delay decreases with the zipf parameter. This is because a skewed popularity means that more requests are concentrated on fewer segments. Consequently, the miss rate decreases, leading to lower delay. We can see that CR-SS achieves substantially lower delay than the other schemes. This shows that CR-SS makes very good content replication and server selection decisions. When s is large, CR-SS achieves similar performance as MPF. This is because it makes effective decision as MPF by replication high-popularity segments at each of the servers.

Figure 5 plots wait probability versus request rate. The probability increases with request rate due to network congestion. CR-SS achieves the lowest wait probability due to its better replication and server selection algorithms. LRU and Local Greedy do not perform well because of their ineffective replication and server selection algorithm, leading to bottleneck link in the network. Random and MPF do not have segment replacement, hence enjoying lower wait probability.

Figure 6 plots the reneging rate versus request rate. The reneging rate increases with request rate due to network and server congestion. CR-SS has substantially lower reneging rate as compared with the other schemes because requests are served with much lower delay.

V. Conclusion

In this work, we study the provisioning of large-scale video-on-demand (VoD) services to distributed users. We address the joint optimization of Content Replication (CR) and Server Selection (SS) to achieve low user (interactive) delay. We propose a simple and fully distributed algorithm called CR-SS. With CR-SS, each server uses a simple routing table to store segment location information. CR-SS makes effective distributed replacement decision and uses probabilistic server selection to minimize user delay and network load.

We have conducted extensive simulation and comparison study of CR-SS on Internet-like topologies. The results show that CR-SS achieves much lower user delay, miss delay, wait probability and reneging rate than traditional (i.e., Random, LRU and MPF) and state-of-the-art schemes (i.e., Local Greedy). Given the same requirement on user delay, it is able to support much higher streaming bitrate and request rate to provide distributed VoD service.

References

[1] X. Zhang and H. Hassanein, "Video on-demand streaming on the internet — a survey," in *2010 25th Biennial Symposium on Communications (QBSC)*, 12-14 2010, pp. 88–91.

[2] Z. Chen, C. Lin, and X. Wei, "Enabling on-demand internet video streaming services to multi-terminal users in large scale," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 4, pp. 1988–1996, November 2009.

[3] U. C. Kozat, O. Harmanci, S. Kanumuri, M. U. Demircin, and M. R. Civanlar, "Peer assisted video streaming with supply-demand-based cache optimization," *IEEE Transactions on Multimedia*, vol. 11, pp. 494–508, April 2009.

[4] C. Huang, J. Li, and K. W. Ross, "Can internet video-on-demand be profitable?" in *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, New York, NY, USA, 2007, pp. 133–144.

[5] S.-H. Chan and F. Tobagi, "Distributed servers architecture for networked video services," *IEEE/ACM Transactions on Networking*, vol. 9, no. 2, pp. 125–136, Apr 2001.

[6] K. Suh, C. Diot, J. Kurose, L. Massoulie, C. Neumann, D. Towsley, and M. Varvello, "Push-to-Peer video-on-demand system: Design and evaluation," *IEEE Journal on Selected Areas in Communications,*, vol. 25, no. 9, pp. 1706–1716, December 2007.

[7] W.-P. K. Yiu, X. Jin, and S.-H. G. Chan, "VMesh: Distributed segment storage for peer-to-peer interactive video streaming," *IEEE Journal on Selected Areas in Communications (JSAC) special issue on Advances in Peer-to-Peer Streaming Systems*, vol. 25, no. 9, pp. 1717–31, Dec. 2007.

[8] W. Jiang, R. Zhang-Shen, J. Rexford, and M. Chiang, "Cooperative content distribution and traffic engineering," in *Proceedings of the 3rd international workshop on Economics of networked systems*, 2008, pp. 7–12.

[9] P. Xia, S.-H. G. Chan, M. Chiang, G. Shi, H. Zhang, L. Wen, and Z. Yan, "Distributed joint optimization of traffic engineering and server selection," in *Packet Video Workshop (PV), 2010 18th International*, December 2010, pp. 86–93.

[10] S. Borst, V. Gupta, and A. Walid, "Self-organizing algorithms for cache cooperation in content distribution networks," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 2, pp. 71–72, 2009.

[11] H. Huang, B. Zhang, S.-H. G. Chan, G. Cheung, and P. Frossard, "Coding and caching co-design for interactive multiview video streaming," in *Proc. of the 31th Annual IEEE Conference on Computer Communic ations (INFOCOM'12) mini-conference*, 2012.