Optimizing Segment Storage and Retrieval for Distributed Video-on-Demand

by

ZHUOLIN XU

A Thesis Submitted to
The Hong Kong University of Science and Technology
in Partial Fulfillment of the Requirements for
the Degree of Master of Philosophy
in Computer Science and Engineering

August 2012, Hong Kong

Copyright © by Zhuolin Xu 2012
Authorization

I hereby declare that I am the sole author of the thesis.

I authorize the Hong Kong University of Science and Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the Hong Kong University of Science and Technology to reproduce the thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

XU, Zhulin

Zhuolin XU
Optimizing Segment Storage and Retrieval
for Distributed Video-on-Demand

by

ZHUOLIN XU

This is to certify that I have examined the above MPhil thesis
and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by
the thesis examination committee have been made.

Dr. Shueng-Han Gary Chan, Thesis Supervisor

Prof. Mounir Hamdi, Head of Department

Department of Computer Science and Engineering
21 August, 2012
Acknowledgements

I would like to express my great gratitude to my supervisor, labmates, friends and family. Without them, this thesis would not have been possible.

First of all, most deeply thanks my supervisor Prof. Gary Chan for his excellent guidance, encouragement, caring and patient polishing my thesis writing. He is the one who first leads me into the world of scientific research. I have learned a lot from him during my Mphil. study in these two year.

I am heartily thanks to Huang Huan and Xia Pengye (Henry). Whenever facing difficulties in research, I can always move on through discussion with you. Really thanks for patiently listening my problems and immediately giving me good suggestions. Special thanks to Zhaoqi Li, who assisted me with problem formulation and mathematical problems involved in the thesis.

Bo Zhang, Steve Zhuo, Jerry Huang and Tony Ren, my dear research comrades. I never forget the moment working with you day and night to meet the paper deadlines. Your assiduousness and intelligence on research always urge me to never give up and make me feel that I am not alone.

I would like to show my appreciation for Jack Lam, Joe Tsoi and Nelson Leung, who provided powerful desktops for me to run simulations. Especially thank Nelson and Xietong Lu. Thank them for helping me in every way when we are working together for developing our LP-SR system.

I am grateful to Prof. Jogesh K. Muppala and Prof. Lin Gu for serving in my thesis examination committees and providing insightful comments which make this thesis better.

Many thanks to all my friends, my parents and family for their support, advice and encouragement. Last but not least, thank all those are not mentioned above, that have taught, encouraged, inspired and helped me.
# TABLE OF CONTENTS

Title Page  
i  
Authorization Page  
ii  
Signature Page  
iii  
Acknowledgements  
iv  
List of Figures  
vi  
List of Tables  
vii  
Abstract  
viii  
Chapter 1  
Introduction  
1  
Chapter 2  
Related Work  
5  
Chapter 3  
Problem Formulation and Its Complexity  
7  
3.1  Problem Formulation  
7  
3.2  Time complexity  
11  
Chapter 4  
LP-based Segment Storage and Retrieval  
13  
4.1  LP-S: Relaxation to a Linear Program for Movie Storage  
14  
4.2  Asymptotically Optimal Segment Storage  
14  
4.3  LP-R: Optimal Segment Retrieval as a Linear Program  
16  
4.4  Run-time Complexity of LP-SR  
17  
Chapter 5  
Illustrative Simulation Results  
19  
5.0.1  Setup and Performance Metrics  
19  
5.1  Illustrative Results  
23  
Chapter 6  
Efficient Computation  
31  
6.0.1  Efficient Movie Grouping  
31  
6.0.2  Time Complexity  
33  
6.0.3  Illustrative Results  
33  
Chapter 7  
Conclusion  
37  
Bibliography  
39
LIST OF FIGURES

1.1 A distributed servers architecture for VoD service. 2
5.1 Movie popularity with the skewness of 0.6 among 100 movies. 20
5.2 Streaming cost model at a proxy server. 22
5.3 Total cost versus $k$ in LP-SR. 23
5.4 Cost versus average proxy storage. 24
5.5 Total cost versus request rate given different schemes. 25
5.6 Total cost versus the skewness of storage capacity given different schemes. 26
5.7 Total cost versus the skewness of movie popularity given different schemes. 27
5.8 Total cost versus the skewness of bandwidth capacity given different schemes. 27
5.9 Total cost versus the movie size given different schemes. 28
5.10 Server cost distribution given different schemes. 29
5.11 Movie cost for different schemes. 30
6.1 Efficient computation method on movie grouping in the default baseline. 32
6.2 Total cost versus request rate given different group sizes. 34
6.3 The running time of LP-SR given different group sizes and in no grouping case. 35
6.4 The tradeoff curve between total cost and running time for movie grouping. 35
LIST OF TABLES

3.1 Major symbols used in this paper. 8

4.1 Symbols for segment retrieval. 16

5.1 Baseline parameters used in our study. 20
Optimizing Segment Storage and Retrieval for Distributed Video-on-Demand

by

ZHUOLIN XU

Computer Science and Engineering
The Hong Kong University of Science and Technology

Abstract

In a distributed large-scale video-on-demand (VoD), a content provider often deploys local servers close to their users. A movie is partitioned into \( k \) segments which the servers collaboratively store and retrieve (\( k \geq 1 \)). A critical but challenging problem is how to minimize overall system deployment cost due to server bandwidth, server storage, and network traffic among servers. In this paper, we address this problem through jointly optimizing movie storage and retrieval in the server network.

We first formulate the optimization problem and show that it is NP-hard. To address the problem, we propose a novel, effective and implementable heuristic. The heuristic, termed LP-SR, decomposes the problem into two computationally efficient linear programs (LPs) for segment storage and retrieval, respectively. The strength of LP-SR is that it is asymptotically optimal in terms of \( k \), and \( k \) does not need to be high to achieve near optimality (around 5 to 10 in our study).
For large movie pool, we propose a movie grouping algorithm to further reduce the computational complexity without compromising much on the performance. Through extensive simulation study, LP-SR is shown to perform significantly the best as compared with other state-of-the-art and traditional schemes, reducing the deployment cost by a wide margin (by multiple times in many cases). It attains performance very close to the global minimum cost.
Introduction

In order to provide cost-effective video-on-demand (VoD) service scalable to large number of users, a content provider often deploys distributed servers placed close to user pools. These servers cooperatively replicate and retrieve movies given movie popularity. Such architecture is able to greatly reduce network load and scale up the streaming and storage capacity of the network. In this paper, we consider the critical and challenging problem of minimizing the system deployment cost through optimizing movie storage and retrieval in the servers. The cost model we use is general and comprehensive, capturing server storage, server bandwidth utilization and network traffic among the servers.

We show in Figure 1.1 a typical distributed and cooperative VoD network consists of a central server (or 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 possibly heterogeneous storage which may be able to replicate only a fraction of the movies. Each user has a home (or local) server to serve his request. If the request is a hit, the home server directly streams to the users. Otherwise (a miss), the home server pulls the content from a remote server (either a proxy server or the central server) to serve the request. In other words, the

\(^1\)In this paper, we use “client” and “user” interchangeably. We also use “movie,” “video” and “content” interchangeably.
bandwidth of the servers\(^2\) are used to stream not only its own home users (if any), but also remote servers requesting their contents.

The deployment cost of such a VoD network mainly consists of two major components, server cost due to the storage and bandwidth usage of the servers, and network cost due to streaming among servers to serve the misses [1, 2]. A challenging problem is hence which movies to store/replicate and where to access them in order to minimize the deployment cost.

For efficient server storage and retrieval, each movie is considered to be partitioned into \(k\) segments \((k \geq 1)\). We formulate the cost-minimization problem of optimizing movie storage and retrieval and show that it is NP-hard. To make it tractable, we propose a novel and efficient heuristic termed LP-SR which decomposes the problem into two linear programs (LPs) for segment storage and retrieval, respectively. The salient feature of LP-SR is that it is \textit{asymptotically optimal} in \(k\), i.e., as \(k\) increases, its performance approaches global optimum. Furthermore, our results show that \(k\) does not need to be large (say 5 − 10) for

\(^2\)In this paper, we use the term “servers” to collectively refer to the central and proxy servers.
the system to be closely optimal (within 6.5% deviation).

Our contributions are three-folds:

- **Comprehensive consideration of system deployment cost:** We consider a realistic, general and comprehensive model on the deployment cost of VoD, which includes server bandwidth utilization, storage and network transmission cost. (Previous work in VoD seldom considers all these factors together.) We formulate the joint optimization problem in movie storage and retrieval and show that it is NP-hard.

- **LP-SR: Achieving asymptotic optimality for video-on-demand:** We propose LP-SR which decomposes the original problem into two linear programming (LP) problems for segment storage and retrieval, respectively. These LPs can be efficiently solved in polynomial time. LP-SR is asymptotically optimal in $k$, i.e., the system cost approaches the exact minimum as $k$ increases. With LP-SR, the network is able to make the best use of limited server storage, efficiently utilize server bandwidth, and substantially save network traffic cost due to server access. To reduce further its computational complexity for large number of movies, we propose an efficient movie grouping algorithm which achieves close optimality but reduces the complexity by a factor of $O(g^3)$, where $g$ is the group size of the movies.

- **Extensive simulation study:** We conduct extensive simulation and comparison study of LP-SR with both state-of-the-art and traditional schemes. Our results show that LP-SR achieves substantially the lowest system cost, outperforming them by a wide margin (by multiple times in many cases). The results show that many existing heuristics are still far from the optimum, and LP-SR can achieve performance very close to such optimum.

This work is organized as follows. In Chapter 2, we briefly review the previous work. In Chapter 3, we formulate the joint optimization problem for VoD and show that it is a NP-hard. We present LP-SR in Chapter 4. In Chapter 5, we
present illustrative simulation results on the performance of LP-SR. In Chapter 6, we present how to further reduce the run-time complexity of LP-SR by movie grouping. We conclude in Chapter 7.
Related Work

We briefly discuss previous related work in this chapter. The joint problem of movie storage and retrieval is generally regarded as NP-hard [3–5]. As a result, many heuristics have been proposed to address the problem (e.g., [3–11]). It is often not clear how well they perform as compared with the optimum. In contrast, the proposed LP-SR is asymptotically optimal in $k$. Even under realistic condition that $k$ is far from large (5 to 10), its performance is very close to the optimum. Due to its highly optimal nature, LP-SR outperforms many state-of-the-art and traditional schemes by a wide margin of multiple times. In contrast with some previous algorithms based on iterations [8,12], LP-SR is based on LP formulations and hence guarantees to converge even for a large network.

The work in [13,14] considers how to support user interactivity through efficiently searching for movie segments. While the heuristics are strong and impressive, they have not considered cost optimization issue. For the work studying the cost issue for VoD [5,6,15–19], many of them have not sufficiently considered the general case with network access cost, storage constraint and bandwidth utilization of the servers. Our model captures all these elements, leading to a more complete, realistic and practical formulation. While some presents impressive measurement work on commercially deployed VoD systems [20,21], we address
the optimization aspect of a VoD network and propose a new scheme which is asymptotically optimal. Our optimality is shown to be significantly better than the state of the arts.

The recent VoD work in [22–24] seeks to maximize the sharing of peers to offload the server load. While the objective of these works is to utilize the uploading bandwidth as much as possible, ours is to minimize the deployment cost of VoD. The difference in objectives leads to different system design and operation.

A preliminary version of this work has been reported in [25]. The current work extends it by studying and proving the complexity of the problem, proposing a movie grouping algorithm which greatly reduces the time complexity of the algorithm when the movie pool is large. We also present substantially more simulation results to show the optimality of LP-SR.
Problem Formulation and Its Complexity

In this chapter, we present the joint cost-optimization problem of movie storage and retrieval to minimize deployment cost and show that it is NP-hard.

3.1 Problem Formulation

We show the important symbols used in Table 3.1.

The overlay network is modeled as an undirected graph $G = (V, E)$, where $V$ is the set of servers and repository and $E = V \times V$ is the set of overlay edges connecting nodes in $V$ (the extension to directed graph is straightforward given our current formulation). Let $M$ be the set of movies and $L^{(m)}$ be the movie length (in seconds). Let $p^{(m)}$ be the popularity of movie $m$, which is the probability that a user requests movie $m$, where $0 \leq p^{(m)} \leq 1$ and $\sum_{m \in M} p^{(m)} = 1$. 

7
Table 3.1: Major symbols used in this paper.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G(V, E)$</td>
<td>A undirected graph representing the overlay network topology</td>
</tr>
<tr>
<td>$V$</td>
<td>The set of servers (repository and distributed proxy servers)</td>
</tr>
<tr>
<td>$</td>
<td>V</td>
</tr>
<tr>
<td>$M$</td>
<td>The set of movies</td>
</tr>
<tr>
<td>$</td>
<td>M</td>
</tr>
<tr>
<td>$L^{(m)}$</td>
<td>The movie length (in seconds)</td>
</tr>
<tr>
<td>$p^{(m)}$</td>
<td>Access probability of movie $m$</td>
</tr>
<tr>
<td>$I_v^{(m)}$</td>
<td>0 or 1 variable indicating whether server $v$ stores movie $m$</td>
</tr>
<tr>
<td>$B_v$</td>
<td>Storage space of server $v$ (in seconds)</td>
</tr>
<tr>
<td>$r_{uv}^{(m)}$</td>
<td>The probability of streaming movie $m$ from server $u$ to server $v$</td>
</tr>
<tr>
<td>$\lambda_v$</td>
<td>Request arrival rate at server $v$ (requests per second)</td>
</tr>
<tr>
<td>$\alpha_m L^{(m)}$</td>
<td>Average holding (viewing) time of movie $m$ $\alpha_{m} \geq 0$</td>
</tr>
<tr>
<td>$\Gamma_{uv}$</td>
<td>Network transmission bandwidth from server $u$ to $v$ (bits/s)</td>
</tr>
<tr>
<td>$b$</td>
<td>Movie streaming rate (bits/s)</td>
</tr>
<tr>
<td>$R_v$</td>
<td>Uploading bandwidth of server $v$ for streaming to remote servers (bits/s)</td>
</tr>
<tr>
<td>$C_v^S$</td>
<td>Cost for server $v$ (per second)</td>
</tr>
<tr>
<td>$C^S$</td>
<td>Aggregated server cost (per second)</td>
</tr>
<tr>
<td>$C_{uv}^N$</td>
<td>Network cost due to traffic from server $u$ and $v$ (per second)</td>
</tr>
<tr>
<td>$C^N$</td>
<td>Total network cost (per second)</td>
</tr>
<tr>
<td>$C$</td>
<td>Total deployment cost ($= C^S + C^N$)</td>
</tr>
</tbody>
</table>
A server \( v \) has a certain storage space \( B_v \) (in seconds). Let

\[
I_v^{(m)} \in \{0, 1\}, \forall v \in V, m \in M, \quad (3.1)
\]

indicating whether server \( v \) stores movie \( m \). Note that for the repository, \( I_v^{(m)} = 1, \forall m \in M \). We obviously must have

\[
\sum_{m \in M} I_v^{(m)} I_v^{(m)} \leq B_v, \forall v \in V. \quad (3.2)
\]

Let

\[
0 \leq r_{uv}^{(m)} \leq 1, \forall u, v \in V, m \in M, \quad (3.3)
\]

be the probability of supplying movie \( m \) from server \( u \) to server \( v \). As the server cannot supply more than that it stores, we must have

\[
r_{uv}^{(m)} \leq I_u^{(m)}, \forall u, v \in V, m \in M. \quad (3.4)
\]

In other words, when there is a miss, a remote server can only supply the movie it locally stores. And by definition, \( r_{vv}^{(m)} = I_v^{(m)} \).

Each user retrieves data from the servers (including his home server), we hence must have

\[
\sum_{u \in V} r_{uv}^{(m)} = 1, \forall v \in V, m \in M. \quad (3.5)
\]

Let \( \lambda_v \) be the total movie request rate at server \( v \) (requests per second); the request rate for movie \( m \) at server \( v \) is hence \( p_v^{(m)} \lambda_v \). Further let \( \alpha_v^{(m)} L_v^{(m)} \) be the average holding (or viewing) time for movie \( m \), where \( \alpha_v^{(m)} \geq 0 \). Then the average data streamed is \( \alpha_v^{(m)} r_{uv}^{(m)} L_v^{(m)} \), as the actual amount of streamed data is assumed to be directly proportional to the viewing time.
Let $b$ be the movie streaming bitrate (bits/s). Hence, the data rate the server $v$ “pulls” from server $u$ for movie $m$ is $p^{(m)} \lambda_v \alpha^{(m)} r^{(m)}_{uv} L^{(m)} b$. Therefore, the total network transmission bandwidth (bits/s) from server $u$ to $v$ is

$$\Gamma_{uv} = \sum_{m \in M} p^{(m)} \lambda_v \alpha^{(m)} r^{(m)}_{uv} L^{(m)} b, \forall u, v \in V,$$  

(3.6)

for $u \neq v$, and, by definition, $\Gamma_{uu} = 0$.

Let $C_{uv}^N$ be a monotonically non-decreasing piece-wise linear function for network cost due to the traffic from server $u$ to $v$, i.e.,

$$C_{uv}^N = C_{uv}^N(\Gamma_{uv}), \forall u, v \in V,$$  

(3.7)

with $C_{uu}^N = 0$. The total network cost $C^N$ is hence

$$C^N = \sum_{u,v \in V} C_{uv}^N,$$  

(3.8)

The bandwidth used in a server to serve the other remote servers depends on where to store and how to retrieve a movie. For any server $v \in V$, the total rate (bits/s) that it serves other servers is given by

$$R_v = \sum_{u \in V, u \neq v} \Gamma_{vu}, \forall v \in V.$$  

(3.9)

The servers help each other using “cache and stream” model, i.e., a remote server streams to a user through his home server. Therefore, the total bandwidth of server $v$ to serve its local users is given by $\sum_{m \in M} p^{(m)} \lambda_v \alpha^{(m)} L^{(m)} b$. This is a fixed quantity given local traffic, and hence will not be considered in our cost optimization.

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.,

$$C^S_v = C^S_v(B_v, R_v), \forall v \in V. \quad (3.10)$$

In another words, the server cost is a function of its storage and streaming bandwidth. The aggregated server cost $C^S$ is hence

$$C^S = \sum_{v \in V} C^S_v. \quad (3.11)$$

Therefore, the total system deployment cost $C$ is

$$C = C^N + C^S. \quad (3.12)$$

We state our joint cost-optimization problem as follows:

**JOSR: Joint Optimization on Movie Storage 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 $\{C_{uv}^N\}$ and $\{C^S_v\}$, we seek to minimize the total cost given by Equation (3.12), subject to Equations (3.1) to (3.5). The output is movie storage in each server (i.e., $\{I_v^{(m)}\}$) and movie retrieval between servers (i.e., $\{r_{uv}^{(m)}\}$).

### 3.2 Time complexity

**Claim:** The JOSR problem is NP-hard.

**Proof:** We prove its NP-hardness by deriving a polynomial reduction from the Domatic Number Problem, whose NP-complete version is stated as follows. Given $G = (V, E)$ and a positive integer $K$ no larger than $|V|$, is the domatic number of $G$ at least $K$, i.e., can $V$ be partitioned into at least $K$ disjoint
Given $G$ and $K$, we construct an instance of our decision problem as follows. The network contains $K$ equal-sized movies with equal popularity everywhere, with each server being able to store only one movie. Consider the case that movie length is homogenous, average viewing time of a movie is the entire movie, request demand at each server is $D/|V|$, unit storage cost is zero and unit access cost of “pulling” a movie from a remote server (including the streaming cost of the remote server and the network cost from remote to the home server) is $i$ units ($i \in Z^+$. Note that such instance construction can obviously be done in polynomial time. Our decision problem hence becomes: Given the instance, is there a joint SR strategy which achieves total cost of at most $D \times (K − 1)/K$?

Clearly, at each node $v$, the minimum average access cost is $(K − 1)/K$ when one movie is stored locally and $(K − 1)$ movies are in the remote servers with 1-unit access cost, and the total cost collected from all the nodes is correspondingly $D \times (K − 1)/K$. As the overlay VoD network in consideration is a connected graph, we construct its subgraph $G'$ which maintains all the servers (i.e. nodes) and the edges with 1-unit access cost. This can also be done in polynomial time. We show that $G'$ can be partitioned into $K$ disjoint dominating sets if and only if there is such a strategy.

First, if there is such a strategy, any node $v$ can access a movie within 1–unit access cost. Then if we could separate $V$ into $K$ disjoint sets with each corresponding to a certain movie, it would result in $K$ dominating sets because the distance between a set and any node is either zero (i.e., contained in the set) or one unit (i.e., connected by an edge). Furthermore, if we have $K$ disjoint dominating sets, we can easily derive such a strategy by assigning each set a distinct movie to store. Therefore, we reduce the Domatic Number problem to our decision problem which proves that our optimization problem is NP-hard.
4

LP-based Segment Storage and Retrieval

In this chapter, we present our novel and efficient heuristic called LP-SR, which decomposes the optimization problem into two LPs for segment storage (LP-S) and retrieval (LP-R). LP-SR works as follows: we first relax the problem stated above to an LP which yields optimal movie storage (Chapter 4.1). Our discretization process for segment storage is asymptotically optimal (Chapter 4.2). Given the storage, we solve the optimal segment retrieval problem by another LP (Chapter 4.3). We end this chapter by analyzing the run-time complexity of LP-SR (in Chapter 4.4).
4.1 LP-S: Relaxation to a Linear Program for Movie Storage

In order to address the NP-hard problem, we relax the constraint in Equation (3.1) as

\[ 0 \leq I_v^{(m)} \leq 1, \forall v \in V, m \in M. \tag{4.1} \]

After such relaxations, our problem becomes an LP, where \( I_v^{(m)} \) refers to the fraction of movie \( m \) that server \( v \) stores.

Note that for any arbitrary linear functions of \( C_N^{uv} \) (Equation (3.10)) and \( C_S^v \) (Equation (3.7)) the relaxed problem becomes a linear programming (LP) problem which can be solved efficiently.

Due to the variable relaxations (\( I_v^{(m)} \) and \( r_{uv}^{(m)} \) above), our LP solution is expected to obtain lower cost solution than its original NP-hard formulation, i.e., \( C^{LP^*} \leq C^{NP^*} \). We call \( C^{LP^*} \) the super-optimum solution which is no worse than the NP-hard solution. We will see later in Chapter 5 that LP-SR asymptotically approaches the super-optimum, meaning that both the NP-hard solution and LP-SR are very close.

4.2 Asymptotically Optimal Segment Storage

The LP solution above is used for segment storage (and hence LP-S). We propose an asymptotically optimal segment storage algorithm here. Each movie is partitioned into \( k \) segments (\( k \geq 1 \)). In order to sufficiently utilize the fractional solution derived from LP-S, we place some of the \( k \) segments to match as closely as possible the optimal movie storage \( I_v^{(m)} \). The major issues are how many seg-
ments of a movie should be locally stored (i.e. segment space allocation) and which segments should be stored (i.e. segment placement).

1. Segment space allocation: The number of segments of movie $m$ that server $v$ stores is

$$n_v^{(m)} = I_v^{(m)}k, \forall v \in V, m \in M. \quad (4.2)$$

For finite $k$, $\{n_v^{(m)}\}$ needs to be discretized to integral values. We present below a simple discretization approach where each server tries to match the optimal LP solution as much as possible through integer rounding.

We first round down the result $\{n_v^{(m)}\}$ as obtained in Equation (4.2) to its closest integers $\{\lfloor n_v^{(m)} \rfloor\}$. For server $v$, it first allocates $\lfloor n_v^{(m)} \rfloor$ segment space according to these integers for movie $m$. This clearly does not violate its storage constraint (given in Equation (3.2)).

Note that after the above process, the “unmatched” portion (in minutes) of movie $m$ is given by

$$\left(n_v^{(m)} - \lfloor n_v^{(m)} \rfloor \right) \frac{L_v^{(m)}}{k}. \quad (4.3)$$

For the residual storage server $v$ then allocates space in decreasing order of the unmatched portion until until its total storage is exhausted.

Note that our segment storage asymptotically approaches the optimal solution as $k$ increases. It is because the rounding effect decreases with increasing $k$.

2. Segment placement: It is clear that after the above segment space allocation, each server stores integral number of movie segments. Each server then needs to place some of $k$ segments to store. The guiding principle of our placement algorithm is that all the segments of a movie should has similar number of replicates in the whole network. Accordingly, we use rarest first in segment placement. Specifically, when a server makes a segment
Table 4.1: Symbols for segment retrieval.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I^{(ms)}_v$</td>
<td>0 or 1 variable indicating whether server $v$ stores segment $s$ of movie $m$</td>
</tr>
<tr>
<td>$r^{(ms)}_{uv}$</td>
<td>The probability of streaming segment $s$ of movie $m$ from server $u$ to server $v$.</td>
</tr>
<tr>
<td>$S$</td>
<td>The set of segment indices of a movie, i.e. ${1, 2, ..., k}$</td>
</tr>
<tr>
<td>$\lambda'_v$</td>
<td>Request rate of segments at server $v$ (req./sec.)</td>
</tr>
<tr>
<td>$p^{(ms)}$</td>
<td>Access probability of segment $s$ of movie $m$</td>
</tr>
<tr>
<td>$L^{(ms)}$</td>
<td>Length of segment $s$ of movie $m$ (seconds)</td>
</tr>
</tbody>
</table>

placement for a movie, it selects the segment which is the least globally stored until the space budget of the movie is fully consumed.

4.3 LP-R: Optimal Segment Retrieval as a Linear Program

The optimal solution of $\{r^{(m)}_{uv}\}$ given by LP-S is no longer appropriate due to our segment storage. We hence need to formulate another LP (called LP-R) to derive optimal segment retrieval given segment storage above. The major variables used for segment retrieval is shown in Table 4.1.

Let $S = \{1, 2, ..., k\}$ be the set of segment indices of any movie. Let $I^{(ms)}_v \in \{0, 1\}$ indicates whether server $v$ stores segment $s$ of movie $m$, which has been derived the solution given in Chapter 4.1. We further let $r^{(ms)}_{uv}$ be the probability of requesting server $u$ from server $v$ segment $s$ of movie $m$. The segment retrieval problem can then be stated as follows:

- **Arrival rate**: A request for a movie leads to streaming of all its $k$ segments.

Therefore, the request rate for segments at server $v$, given movie request
rate $\lambda_v$, is

$$\lambda'_v = k\lambda_v, \forall v \in V.$$ \hfill (4.4)

- **Length**: A movie is equally divided into $k$ segments; hence we have

$$L^{(ms)} = \frac{L^{(m)}}{k}, \forall m \in M, s \in S.$$ \hfill (4.5)

- **Popularity**: The popularity of the segments of the same movie is given by

$$p^{(ms)} = \frac{p^{(m)}}{k}, \forall m \in M, s \in S.$$ \hfill (4.6)

Using the above, we can formulate optimal segment retrieval problem as an LP (LP-R), i.e.

$$\min \left( \sum_{u,v \in V} C_{uv}^N(\Gamma_{uv}) + \sum_{v \in V} C_v^S(B_v, R_v) \right)$$

subject to

$$0 \leq r^{(ms)}_{uv} \leq I^{(ms)}_v, \forall u, v \in V, m \in M, s \in S,$$

$$\sum_{u \in V} r^{(ms)}_{uv} = 1, \forall v \in V, m \in M, s \in S.$$

### 4.4 Run-time Complexity of LP-SR

To solve LP-S and LP-R, we may use CVX which implements the wide-region centering-predictor-corrector algorithm (an interior-point method) to solve this problem [26] [27]. It has been proven that it has $O(\sqrt{N})$ worst-case iteration bound and $O(N^3)$ overall time complexity, where $N$ is the number of variables.

Note that LP-S and LP-R has $O(|V|^2|M|)$ and $O(|V|^2|M|k)$ variables, respectively. Therefore, the time complexity of the two LPs is $O(|V|^6|M|^3)$ and $O(|V|^6|M|^3k^3)$, respectively. Furthermore, the time complexity of segment stor-
age is $O(|V||M|k)$, because all the servers, movies and their constituent segments are traversed to determine which server to store which segment.

Given the above, the overall time complexity of LP-SR is $O(|V|^6|M|^3k^3)$. Note that even though the run-time complexity of LP-SR is related to $k$, low $k$ (say 5) already achieves closely optimal solution.
5

Illustrative Simulation Results

In this chapter, we first present our simulation environment and performance metrics to study the performance of LP-SR, followed by illustrative results at steady state.

5.0.1 Setup and Performance Metrics

Movie popularity follows the Zipf distribution with skewness parameter $s$, i.e., the request probability of the $i$th movie is proportional to $1/i^s$. Figure 5.1 shows movie popularity with our baseline parameters of $s = 0.6$ and $m = 100$ movies, where the top 30% of the movies account for close to 60% (56.72%) of the traffic.

Requests arrive at each proxy server according to a Poisson process with total rate $\lambda$ (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. Unless otherwise stated, we use the default values as shown in Table 5.1 for our system parameters (the baseline case).
Table 5.1: Baseline parameters used in our study.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>5</td>
</tr>
<tr>
<td>Number of proxy servers</td>
<td>10</td>
</tr>
<tr>
<td>Number of movies</td>
<td>100</td>
</tr>
<tr>
<td>Average server storage</td>
<td>10 movies</td>
</tr>
<tr>
<td>Skewness of server storage</td>
<td>0.4</td>
</tr>
<tr>
<td>Average proxy server bandwidth capacity</td>
<td>160 Mbits/s</td>
</tr>
<tr>
<td>Skewness of server bandwidth</td>
<td>0 (i.e., same bandwidth)</td>
</tr>
<tr>
<td>Skewness of movie popularity</td>
<td>0.6</td>
</tr>
<tr>
<td>Movie length</td>
<td>90 minutes</td>
</tr>
<tr>
<td>Average movie holding time</td>
<td>Movie length (i.e., $\alpha(m) = 1$)</td>
</tr>
<tr>
<td>Movie streaming rate</td>
<td>1 Mbits/s</td>
</tr>
<tr>
<td>Total request rate in the network</td>
<td>0.3 req./s (equally distributed to the proxies)</td>
</tr>
<tr>
<td>$c_{uv}$ between central and proxy server</td>
<td>0.01 unit/s</td>
</tr>
<tr>
<td>$c_{uv}$ between proxies</td>
<td>Zipf with skewness 0.6 and mean 0.005 unit/s</td>
</tr>
</tbody>
</table>

Figure 5.1: Movie popularity with the skewness of 0.6 among 100 movies.
In the simulation, we consider the network cost function from server $u$ to server $v$ to be proportional to the bandwidth between them, i.e.,

$$C^N_{uv}(\Gamma_{uv}) = c_{uv}\Gamma_{uv}, \quad \forall u, v \in V. \quad (5.1)$$

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, modelled as

$$C^S_v = \sigma_B B_v + C_v(R_v), \quad \forall v \in V, \quad (5.2)$$

where $\sigma_B$ 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 5.2 streaming cost $C_v(R_v)$ 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. The cost increases with the bandwidth utilization at 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 $\sigma_S$ is some constant). As the consumed bandwidth $R_v$ approaches the bandwidth capacity $U_v$, the server cost increases sharply.

The performance metrics we are interested in are:

- **Total cost** (unit/s), which is the sum of server cost and network cost according to Equation (3.12). This is the total deployment cost of the network.
- **Server cost** (unit/s), which is the sum of its storage and streaming defined in Equations (3.11) and (5.2). We further examine the following cost components:
  - **Storage cost**, which is the total cost due to server storage; and
  - **Streaming cost**, which is the server bandwidth cost to support other servers.
Figure 5.2: Streaming cost model at a proxy server.

- **Network cost** (unit/s), which is network transmission cost defined by Equations (3.8) and (5.1).

- **Cost of each movie** (unit/s), which is the average cost to access movie \( m \) by any user.

We compare LP-SR with the following traditional and state-of-the-art 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** [6], which divides the movies into three categories, those popular ones which all servers store (full 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 a LP problem, it seeks to minimize network cost. As Local Greedy assumes homogenous access cost, we set its access cost to be equal to the average access cost between servers in our network.

In all the comparison schemes, upon a miss request, the home server $v$ chooses an available server $u$ which has the requested content with a 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.

5.1 Illustrative Results

We plot in Figure 5.3 the total cost versus $k$ in LP-SR. As $k$ increases, the network approaches the super-optimal case given by relaxing the integer constraints on
Figure 5.4: Cost versus average proxy storage.

movie storage \( \{J_n^m\} \). (Note that the curve optimum lies between the LP-SR and the super-optimum.) However, for humble value of \( k \), the performance is already very close to the optimum (less than 6.5\% deviation in our default setting). This shows that our network is highly efficient, with closely optimal performance even for the practical finite value of \( k \).

We show in Figure 5.4 the cost components and total cost versus the average storage space for servers. The total cost falls off initially but rises up again, showing a minimum. At the beginning when the proxy servers have little storage, all the traffic concentrates on the repository, leading to high overall streaming cost. As proxy storage increases, the repository load is reduced and hence the streaming and network transmission cost. As storage further increases, storage cost becomes a major component. It is clear that LP-SR can balance the cost between storage and bandwidth and achieve its optimality by provisioning optimal network resources.

We show in Figure 5.5 the total cost versus the request rate given different schemes. Total cost grows with request rate mainly due to the increase in network
traffic. LP-SR clearly achieves much lower total cost among all the schemes. In other words, given the same deployment budget, LP-SR 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 network cost optimization. LP-SR achieves by far the best performance because it achieves near optimality by capturing not only the network transmission cost but also the server storage and streaming cost.

We show in Figure 5.6 the total cost versus the skewness of the storage capacity given different schemes. The total cost in general increases with storage skewness. It is because skewed storage means that many requests have to be indirectly served by the remote servers of large storage space. This consumes much of server bandwidth, and hence the streaming cost dramatically increases. Having a substantially lower cost with a slower growth rate in cost, LP-SR is much more scalable and can make good movie placement by best utilizing server storage. MPF decreases with the skewness of storage because the proxies of large
storage share some load from the repository, hence reducing the streaming cost of the repository.

We plot in Figure 5.7 the total cost versus the skewness of movie popularity given different schemes. The total cost in general decreases with the popularity skewness. This is because skewed popularity means that more requests are concentrated on fewer popular movies. Consequently, there is lower miss rate, leading to lower streaming and network cost. LP-SR achieves substantially the lowest cost, even for low skewness (i.e. when the popularity is quite uniform). This shows that LP-SR makes good movie placement and retrieval decisions. Local Greedy performs better than MPF because it takes network cost into consideration. The cost of Random increases with skewness because it is popularity-blind. As a result, the popular movies, because of their copies not increasing with their popularity, suffer from high streaming and network cost.

We plot in Figure 5.8 the total cost versus bandwidth skewness of proxies given different schemes. In general, the total cost increases with the bandwidth skewness because skewed bandwidth means that the servers with low bandwidth
Figure 5.7: Total cost versus the skewness of movie popularity given different schemes.

Figure 5.8: Total cost versus the skewness of bandwidth capacity given different schemes.
but large storage cannot support much remote streaming to other servers. This
defeats the advantages on its locally stored movies. Furthermore, the servers with
lower bandwidth more easily run out of bandwidth, leading to a sharp increase
in streaming cost. LP-SR clearly has the lowest cost, beating the other schemes
by multiple times. It is also robust to system heterogeneity, and fully utilizes the
storage and bandwidth resource for cooperative streaming.

We show in Figure 5.9 the total cost versus total number of movies for
different schemes. The cost increases with the movie size because of additional
network load to stream movies. LP-SR enjoys substantially the lowest cost as it
makes the best decisions in movie storage and retrieval. MPF suffers from high
cost because the repository needs to stream those unpopular movies, leading to
high streaming cost.

We compare in Figure 5.10 the 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 our baseline), and the last server is the repository. It is
clear that LP-SR utilizes very well the storage and bandwidth resources of proxy
servers, leading to low repository streaming cost. All the other schemes suffer from high repository cost (note that log scale) due to misses in the proxies. The figure shows that LP-SR has strong server cooperation to achieve near-optimal performance. As MPF only stores the most popular movies at the proxy servers, it has lower proxy server cost but much higher repository cost (due to miss traffic). In MPF, the proxies barely contribute their bandwidth and storage to help each other. Local Greedy, with network cost optimization, outperforms Random in both proxy server cost and repository cost.

We compare in Figure 5.11 the cost to access a movie for different schemes. The movies are sorted according to their descending popularity. The popularity-based schemes (i.e., LP-SR, Local Greedy and MPF) tend to locally store the popular movies, and hence those movies enjoy lower cost. LP-SR makes much better decision by cooperatively storing the movies. LP-SR accomplishes much better optimality of a rather uniform movie cost, with the cost of unpopular movies strikingly much lower by orders of magnitude than the other schemes. For MPF, the costs of popular movies are negligible at much sacrifice of less popular ones. Random treats each movie equally and thus has the most uniform

Figure 5.10: Server cost distribution given different schemes.
cost distribution. The figure shows that LP-SR makes intelligent decisions on movie storage and retrieval to achieve low deployment cost.

Figure 5.11: Movie cost for different schemes.
In terms of the number of movies $|M|$, the run-time complexity of LP-SR is $O(|M|^3)$ (Section 4.4). As the number of movies increases, we need to compute the solution more efficiently.

In this chapter, we propose an efficient computation based on movie grouping. The movies are divided into groups of size $g$. Our algorithm achieves a polynomial reduction $O(g^3)$ in complexity without sacrificing much on performance. We illustrate its performance with simulation results.

6.0.1 Efficient Movie Grouping

Figure 6.1 illustrates the efficient computation method on movie grouping in the default baseline. The basic idea is to divide movies into groups such that the complexity of the LP-SR is reduced. We first pool the movies into fixed-length groups for server storage. Let $L$ be the length of a group (in seconds) for server storage. In order to accommodate a group in any server, $L$ should be chosen such
Movies are grouped according to their decreasing popularity. The most popular movies are put into the 1st group to make up the length $L$, fragmenting the last movie across groups if necessary. The remaining movies are then assigned to group 2, 3, and so on. This process is repeated until all the movies are grouped. Let $G$ be the set of groups and $G_i$ be the $i^{th}$ group. The movies are grouped such that:

- **Length**: Each group is of equal length $L$ and satisfies

$$L = \sum_{m \in G_i} L^{(m)}$$

for all $G_i \in G$. Additionally, it is required that

$$L \leq \min_{v \in V} B_v.$$  

(6.1)
• **Group popularity:** The group popularity $p^{(G_i)}$ is given by

$$p^{(G_i)} = \sum_{m \in G_i} p^{(m)}, \forall G_i \in \mathbb{G}. \quad (6.3)$$

Note that if a movie is fragmented across groups, the popularity of the movie in a group is proportionally adjusted based on its fragment length in the group.

After this movie grouping is done, LP-SR is run on the groups by treating them as “movies” with popularity $p^{(G_i)}$. In the phase of segment storage, a group is partitioned into $k$ group-segments and stored and accessed accordingly similar as before. The movie can be efficiently retrieved from a group-segment with a simple table lookup.

### 6.0.2 Time Complexity

The time complexity of movie grouping is clearly $O(|M|)$, as all the movies are traversed to be assigned to different groups. The total run-time complexity on obtaining the solution is hence $O(|V|^b|M|^3/g^3k^3)$, where $g$ is the average group size. In other words, the complexity is reduced by a factor of $O(g^3)$.

### 6.0.3 Illustrative Results

We conduct simulation to study the performance of our grouping algorithm. We use the same baseline parameters as given in Chapter 5.

We plot in Figure 6.2 the total cost versus request rate given different group sizes. Total cost rises up with the request rate mainly because the increase of network load. Large group size can reduce the time complexity of LP-SR but increase the performance deviation from the optimum. LP-SR with movie
grouping can still outperform Local Greedy by a wide margin (by multiple times in many cases).

We show in Figure 6.3 the running time of LP-SR given different group sizes and in no grouping case. The running time increases with the total number of movies. It is because LP-SR captures the information for every movie, and hence more movies introduces more information and decision variables for LP-SR to process. LP-SR with movie grouping can greatly reduce the time complexity (over triple times in the default baseline setting). The result shows that movie grouping is efficient computation for a large movie pool.

We plot in Figure 6.4 the tradeoff curve between total cost and running time for movie grouping. We also illustrate the figure the corresponding group size of each tradeoff point. As running time increases, LP-SR achieves better performance. Total cost first decreases sharply and then converges to some value (corresponding to the optimal value of no grouping). It is clear that a good operating point of the system is one with low running time and cost, which is around the “knee” of the around (with group size of 3 to 4.5 hours corresponding
Figure 6.3: The running time of LP-SR given different group sizes and in no grouping case.

Figure 6.4: The tradeoff curve between total cost and running time for movie grouping.
to 2 to 3 movies per group respectively).
Conclusion

In this work, we have studied optimal segment storage and retrieval to minimize VoD deployment cost with distributed proxy servers. The deployment cost captures the costs of server streaming, server storage and network transmission cost.

For efficient server storage and retrieval, each movie is partitioned into $k$ segments ($k \geq 1$). We first formulate the joint problem and show that it is NP-hard. To address this, we propose LP-SR, a novel and efficient heuristic which decomposes the problem into two linear programs (LP) for segment storage (LP-S) and retrieval (LP-R), respectively. In stark contrast with much of the previous work where heuristics are often proposed without knowing how they perform with respect to the optimum, our solution is asymptotically optimal in $k$, and $k$ does not need to be large to achieve near optimality ($k$ is around 5). To make our solution more efficient for large movie pool, we propose a movie grouping algorithm which achieves close to optimal performance with polynomial reduction in running time.

We have conducted extensive simulation to compare its performance with other traditional and state-of-the-art schemes. The results show that our scheme substantially outperforms the other schemes by a wide margin (multiple times in many cases). LP-SR achieves very close to optimality with much lower deploy-
ment cost.
Bibliography


