ENHANCED RESOURCE SHARING FOR SCALABLE VIDEO-ON-DEMAND SERVICES

by

BASHAR QUDAH

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

2009

MAJOR: COMPUTER ENGINEERING

Approved by:

_____

Advisor **Date**

_____

_____

_____

_____

ACKNOWLEDGEMENTS

This research would not have been possible without the support of many people. I would like first to thank Dr. Nabil Sarhan, my advisor, for his support and guidance through out this research. I would like also to thank my PhD committee members, Dr. Loren Schwiebert, Dr. Cheng-Zhong Xu, and Dr. Harpreet Singh for their valuable inputs. I would like to express my love and gratitude to my wife Ehsan; for her love, understanding, and endless support the duration of my study and research, my love for my daughter and son; Laila and Yousef, and my appreciation and love for my mother, father, brothers, and sisters; for all the support and motivation they provided.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Driven by the increasing expectations of customers for fully customizable and more convenient services, yet at low prices, *Video-on-Demand* (VOD) is expected to replace both broadcast-based TV programming and DVD movie rentals at stores. The market has already prepared for such a change, and some landmark steps have been taken. For example, many digital TV service providers offer VOD, and some TV channels, such as HBO, have launched VOD programming. Moreover, motivated by the availability of high client download bandwidth, many news networks (such as CNN and BBC) provide short, medium-quality, on-demand news clips on their web-sites. The popularity of video web-sites, such as YouTube [50], provides another clear indication of this direction in video delivery.

Unfortunately, the number of video streams that can be supported concurrently is highly constrained by the required real-time and high-rate transfers, which quickly consume server and network resources, including network bandwidth and disk I/O bandwidth. Resource sharing techniques face this challenge by utilizing the multicast facility. The main classes of these techniques are *Batching* [2, 17, 47], *stream merging* [9, 31, 10, 7, 8, 18, 20, 13, 1, 25, 26, 23], *periodic broadcasting* [38, 39, 22, 30, 33, 35, 34, 45, 44, 36, 46, 48], and *composite* techniques [24]. Batching services all waiting requests for a video using one stream. Stream merging techniques combine streams when possible to reduce the delivery costs. These techniques include *Patching* [31], *Transition Patching* [8], and *Earliest Reachable Merge Target (ERMT)* [18, 20]. Whereas Batching and stream merging techniques deliver data in a *client-pull* fashion, periodic broadcasting techniques, such as *Skyscraper Broadcasting (SB)* [33], *Greedy Disk-conserving Broadcasting (GDB)* [22], *Harmonic Broadcasting (HB)* [35], and *Pyramid Broadcasting (PB)* [48], employ the *server-push* approach. In particular, they divide each supported video into multiple segments and broadcast them periodically

on dedicated channels. Composite techniques, such as *Catching* [24] and *Selective Catching* [24], combine stream merging and periodic broadcasting.

VOD literature lacks detailed comparative analysis of various resource sharing strategies. The decision as to which class and particular technique to apply greatly impacts the overall system performance and the perceived quality-of-service (QoS). With the many available resource sharing techniques, it is unclear which class and particular technique is the best to use in a target environment. For example, it is unclear how ERMT performs in comparison with Selective Catching and Transition Patching and how periodic broadcasting techniques perform compared with stream merging techniques. Moreover, only limited performance metrics, workloads, and service models were considered.

Furthermore, there is very little work on cache management for reducing the demands on the disk I/O when resource sharing techniques are applied. This is, however, an important issue for designing cost effective and scalable VOD servers, especially because of the widening gap between technology improvements in the capacities and speeds of hard disk drives. In [5], the impact of caching was reported by actual measurements for only limited caching schemes and limited and simple resource sharing techniques. The results show that without caching, the disk I/O bandwidth may become a bottleneck.

Client heterogeneity poses another serious challenge to VOD servers. Measurements of Internet client bandwidth show that the bandwidth varies significantly, not only from one technology to another, or from one Internet Service Provider (ISP) to another, but also from one client to another within the same ISP [6]. Client physical location, and even the connection time, can play a significant role in the actual bandwidth. Moreover, the available client buffer space varies significantly, considering the wide variety of devices that support video streaming. The variability in the resources and capabilities of clients is referred to as client heterogeneity. Up to our knowledge, no study has addressed the client heterogeneity issue in systems employing the client-pull delivery approach.

In addition, the achieved resource sharing depends significantly on the user access patterns. The overwhelming majority of prior studies assumed simple workload, in which videos are accessed

sequentially from the beginning to the end (this pattern is referred to here as *Full-Content Access*). Recent characterization studies [12, 14], however, reveal that the actual workload is more complex and dynamic. In particular, a user may access a video from a point other than the beginning and may stop before the end. We refer to this generalized pattern as *Selected-Content Access*. This pattern is common, especially for long videos. Popular interactions (such as jump forward, jump backward, and pause) can be considered as a series of selected-content access periods. Unlike other prior work, the study [40] used such realistic workload but focused on reducing the data requested by clients in interactive operations through enhanced data buffering and relied on traditional stream merging techniques without changing their stream merging decisions.

This research addresses those challenges and limitations. The main objectives of this research can be summarized as follows:

- To provide a detailed analysis of resource sharing techniques, considering both the *True Video-on-Demand* (TVOD) and the *Near Video-on-Demand* (NVOD) models and two video workloads: *mixed-video* and *hot-video*. The first workload contains both hot (i.e., popular) and cold (i.e., unpopular) videos, whereas the second contains only hot videos. In TVOD, we compare the server resources required to service all requests immediately, whereas in NVOD, we fix the resources and compare the number of customers that can be serviced concurrently, the waiting times, and unfairness towards unpopular videos. We develop analytical models for optimal tuning of design parameters for some techniques, examine the impacts of many system and workload parameters, and introduce and applies a framework for comparing stream merging and periodic broadcasting techniques, considering the ability of the latter to provide maximum waiting time guarantees.

- To propose an efficient *Workload-Aware Hybrid Solution* (WAHS) that combines the advantages of stream merging and periodic broadcasting. This solution is highly adaptive to variations in the workload and the available system resources. It first identifies the videos to be served by each technique by estimating and comparing their requirements. It then allocates

resources intelligently to each technique by utilizing a proposed *prediction-based allocation* approach, which allocates resources by predicting the achieved customer defection (i.e., turn-away) probability and average waiting time by each technique. Finally, it divides the resources allocated for periodic broadcasting among the corresponding videos using a proposed *maximum-gain allocation*, which allocates channels, one at a time, to the video that gains the most from that channel in reducing its expected defection probability or waiting time. WAHS provides a promising generic solution and performs significantly better than *Selective Catching* [24], which is up to our knowledge the best existing hybrid scheme.

- To develop and analyze a *Statistical Cache Management* (SCM) approach, which computes periodically video access frequencies and determines the data to be cached based on these statistics. In addition to its performance effectiveness, it is easy to implement and incurs small overhead as updates are triggered only when the workload varies considerably. We derive analytical models for allocating cache space for various resource sharing techniques. This work differs significantly from previous work on proxy caching [37] (and references within), which has different objectives and factors affecting the allocation decisions. It also differs from low-level caching techniques, such as *Least Recently Used* (LRU), *Least Frequently Used* (LFU), and *Interval Caching* [15, 41], whereby the content of the cache changes frequently, incurring significant overheads.

- To study how to support heterogeneous receivers while delivering video streams in a client-pull fashion. We propose three solutions to address the variability of the download bandwidth among clients: *Simple Hybrid Solution* (SHS), *Adaptive Hybrid Solution* (AHS), and *Enhanced Hybrid Solution* (EHS). SHS simply combines Batching with either Patching (SHS-P) or ERMT (SHS-E). It classifies clients into two bandwidth classes. Clients with bandwidth capacities less than double the playback rate are serviced using Batching and the rest are serviced using Patching/ERMT. In contrast, AHS and EHS classify clients into multiple bandwidth classes and service them accordingly. They employ new stream types to service clients

with bandwidth capacities ranging between the video playback rate and double the video playback rate. These streams are called *adaptive streams* and *enhanced adaptive streams*, respectively. Whereas the streams in Batching, Patching, and ERMT are all delivered at the video playback rate, adaptive streams are delivered at variable rates that are lower than the playback rate. Similarly, enhanced adaptive streams are delivered at variable rates but in two transmission phases. They are delivered initially at rates higher than the playback rate and subsequently at rates lower than the playback rate. The specific rates at which these two stream types are delivered depend on the corresponding client bandwidth capacities. AHS and EHS employ adaptive streams or enhanced adaptive streams in conjunction with Batching and Patching or ERMT, leading to four possible schemes: AHS-P, AHS-E, EHS-P, and EHS-E. We also study how to address the variations in client bandwidth during a session. In addition, we study the support for the variability in the available buffer space among clients. Furthermore, we study how the waiting playback requests for different videos can be scheduled for service in the heterogeneous environment, capturing the variations in client bandwidth and buffer space.

- To study the impact of selected-content access on streaming servers delivering data in a client-pull fashion using stream merging techniques. We propose several enhancements to reduce server load and improve the client perceived quality-of-service (QoS).

- To investigate utilizing more advanced video coding, such as Layered Video Coding (LVC) and Multiple Description Coding (MDC) [49, 27, 21], with advance stream merging techniques for better serving heterogeneous receivers. We investigate the impact of a temporary reduction in video quality on reducing the overall delivery cost (amount of data delivered by the server). We answer two questions concerning such a reduction in quality: (i) when resort to this option? and (ii) for how long this temporary period can last? In answering the first question, we propose three approaches, and in answering the second question, we consider three options on limiting the reduced quality period. We evaluate the Cartesian product of

these approaches and limit options with both LVC and MDC and we compare them to each other and to an alternative system with multiple copies of every video each decoded at different rate.

The rest of this dissertation is organized as follows: Chapter 2 discusses background information and the related work. Chapter 3 analyzes resource sharing techniques and presents the *Workload-Aware Hybrid Solution* (WAHS) and the *Statistical Cache Management* (SCM) approach. Chapter 4 addresses how to support heterogeneous receivers and presents the proposed streaming solutions. Chapter 5 studies the impact of selected-content access with homogeneous receivers, and presents five enhancements. Finally, Chapter 6 investigates the combined impact of selected-content access and receiver heterogeneity, and utilizing LVC and MDC video coding techniques for better serving heterogeneous receivers,

CHAPTER 2

RELATED WORK

2.1 Resource Sharing

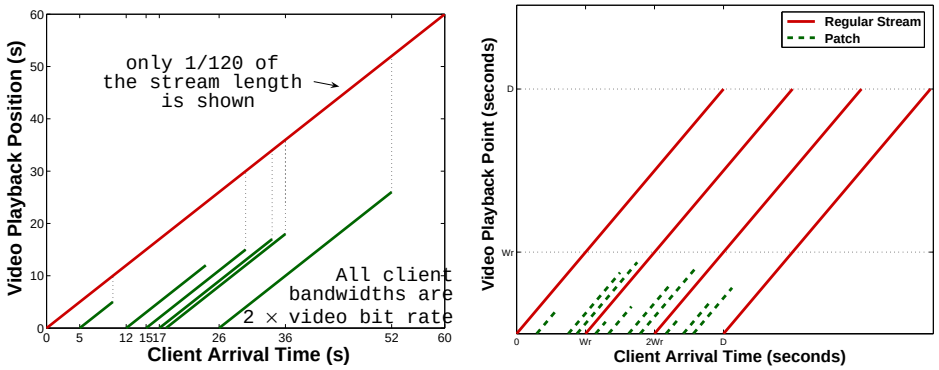
2.1.1 Batching

Batching simply services all waiting requests for a video using one full-length multicast stream, called *regular stream*. Thus, it requires only one download channel. Because streams are delivered at the video playback rate (r), the required client download bandwidth is r. The playback rate is measured in bits per second.

2.1.2 Stream Merging Techniques

These techniques reduce the delivery costs by combining streams. They include, in increasing order of complexity and performance, *Stream Tapping/Patching* [11, 9], *Transition Patching* [10], and *Earliest Reachable Merge Target (ERMT)* [18]. Each of these techniques requires two client download channels at the video playback rate.

Patching expands the multicast tree dynamically to include new requests. A new request joins the latest regular stream for the video and receives the missing portion as a *patch* stream at the video playback rate. Hence, it requires additional buffer space and two download channels, leading to a total required download bandwidth of $2r$. When the playback of the patch is completed, the client continues the playback of the remaining portion using the data received from the multicast stream and already buffered locally. To avoid the continuously increasing patch lengths, regular streams are retransmitted when the required patch length for a new request exceeds a pre-specified value, called *regular window* (W_{reg}). Figure 2.1(a) further explains the concept of Patching in servicing 7 requests for a specific 2-hour video. This figure shows the video playback positions delivered by different streams versus time. In the example, all clients have $2r$ of download bandwidth. The



(a) Numeric Example (b) Stream Types and Wr

Figure 2.1: Illustration of Patching

first client arrives at time 0 and starts receiving the video as a regular stream (2-hour long). The second client arrives after 5 seconds and starts to receive the existing regular stream immediately. Meanwhile, it receives the initial 5 seconds of the missing video data as a patch stream (5-second long). The third client arrives 12 seconds after the first, and receives the missing 12 seconds of data by another patch. The later clients are serviced in a similar manner. To avoid the continuously increasing patch lengths, regular streams are retransmitted when the required patch length exceeds a pre-specified value called *regular window* (Wr). Figure 2.1(b) further explains the concept of Wr in servicing one video of D seconds in length. (Typically, Wr is much smaller than D.)

Transition Patching allows some patches to be sharable by extending their lengths. It introduces another multicast stream, called *transition patch*. The threshold to start a regular stream is Wr as in Patching, and the threshold to start a transition patch is called the *transition window* (Wt). The transition patch length is equal to the difference between the starting times of the transition patch and the last regular stream plus $2Wt$, whereas the length of the patch is equal to the difference between the starting times of the patch and the latest transition stream or regular stream. Hence, the maximum possible patch length is Wt, and the maximum possible transition patch length is $Wr + 2Wt$. Figure 2.1.2 further illustrates the concept. A possible scenario for a client is to start

listening to its own patch and the transition patch, and when its patch is completed, it starts listening to the regular stream.

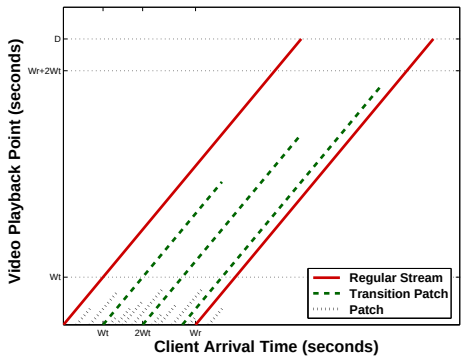


Figure 2.2: Illustration of Transition Patching

ERMT is a near optimal hierarchical stream merging technique. It also requires $2r$ of client download bandwidth and additional client buffer space, but it makes each stream sharable by later clients and thus leads to a dynamic merge tree (whereas Patching shares only regular streams with later clients). A new client joins the closest reachable stream (*target*) and receives the missing portion by a new stream (*merger*). A target stream is reachable if the client can merge with it before the target terminates. When more than one reachable stream is possible, the closest to the client's request arrival time is selected. After the merger stream finishes and merges into the target, the latter may get extended to satisfy the playback requirement of the new client(s), and this extension may affect its own merge target. For example, in Figure 2.1.2, Stream 3 length got extended twice; once when Stream 4 merged into it, and once when Stream 5 did. Specifically, it started as a 12-second patch, but when Stream 4 merged into it at time 18 seconds (i.e., Stream 4 ended and Stream 3 became the primary stream for the 4^{th} client), Stream 3 got extended by 6 seconds to cover for the missed data from the regular stream by the 4^{th} client. The same scenario is repeated again when Stream 5 merged into Stream 3 to cover for data missed by the 5^{th} and the 6^{th} clients.

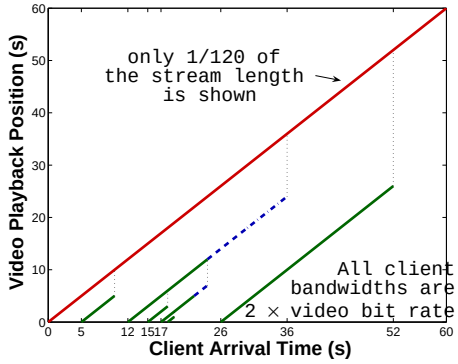


Figure 2.3: Illustration of ERMT

2.1.3 Periodic Broadcasting Techniques

Whereas stream merging suits a wider spectrum of video workloads, periodic broadcasting techniques can be more efficient in serving highly popular videos. These techniques divide each video into multiple segments and broadcast them periodically on dedicated channels. They include *Skyscraper Broadcasting* (SB) [33], *Greedy Disk-Conserving Broadcasting* (GDB) [22], *Fibonacci Broadcasting* (FB) [30], and *Harmonic Broadcasting* (HB) [35, 39]. In SB, GDB, and FB, the segments are of variable length and the channels have equal bandwidth. The client waits until the beginning of the next broadcast of the first segment and then receives data concurrently from two broadcast channels. The relative length of the n^{th} segment compared to the first segment is determined using a technique-specific partitioning function, as shown in Table 2.1. To illustrate the main concept, if a 30-minute video is divided into three segments by FB, the length of the first segment and thus the maximum waiting time are $30/(1+2+3) = 5$ minutes. Note that the maximum waiting times decreases with the number of segments at the expense of increasing server bandwidth.

In contrast, HB-based protocols have uniform-length segments and the channel bandwidth decreases with the segment number. Despite their high effectiveness in reducing server bandwidth, they require the client to have download bandwidth equal to the server bandwidth allocated for the video and partition each video into a relatively large number of segments (to reduce the maximum delay).

Table 2.1: Segment Partitioning in Periodic Broadcasting

Technique	**Partitioning Function**
Skyscraper Broadcasting (SB)	$f(n) = \begin{cases} 1 & n = 1, \\ 2 & n = 2, 3, \\ 2f(n-1)+1 & n \mod 4 = 0, \\ f(n-1) & n \mod 4 = 1, 3, \\ 2f(n-1)+2 & n \mod 4 = 2. \end{cases}$
Fibonacci Broadcasting (FB)	$f(n) = \begin{cases} n & n = 1, 2, \\ f(n-2)+f(n-1) & n > 2. \end{cases}$
Greedy Disk-Conserving Broadcasting (GDB)	$f(n) = \begin{cases} 1 & n = 1, \\ 2 & n = 2, 3, \\ 5 & n = 4, 5, \\ 12 & n = 6, 7, \\ 5f(n-4) & n > 7. \end{cases}$
Modified Greedy Disk-Conserving Broadcasting (MGDB) [Used in *Catching* and *Selective Catching*]	$f(n) = \begin{cases} 1 & n \leq 3, \\ 2 & n = 4, 5, \\ 5 & n = 6, 7, \\ 12 & n = 8, 9, \\ 5f(n-4) & n > 9. \end{cases}$

2.1.4 Combining Stream Merging and Periodic Broadcasting

Catching uses a modified version of GDB (called here MGDB) to deliver hot videos, but instead of making the client wait until the beginning of the next broadcast time, the client receives the small missed portion by a patch. A client initially listens to one broadcast channel and its own patch, and then to two broadcast channels. The partitioning function of MGDB is shown in Table 2.1. And if we define a function $h(m) = \Sigma_{i=1}^{m} f(i)$ then the length of the first segment is equal to $D/h(K)$,

h(m) is given by:

$$h(m) = \begin{cases} m & m = 1, 2, \text{ or } 3 \\ 5 & m = 4 \\ 7 & m = 5 \\ 5^{(m-6)/4} * 27/2 - 1.5 & m > 5, \text{ m mod 4=2} \\ 5^{(m-7)/4} * 37/2 - 1.5 & m > 5, \text{ m mod 4=3} \\ 5^{(m-8)/4} * 61/2 - 1.5 & m > 5, \text{ m mod 4=0} \\ 5^{(m-9)/4} * 85/2 - 1.5 & m > 5, \text{ m mod 4=1.} \end{cases} \quad (2.1)$$

Selective Catching extends Catching by delivering cold videos using *Controlled Multicast* [23], which works essentially as Patching. Another hybrid solution that combines Batching (an older multicast scheme without stream merging) with SB was proposed in [32]. In contrast with these techniques, our hybrid solution combines the best performers of stream merging and periodic broadcasting techniques and uses a comprehensive resource allocation approach.

2.1.5 Analyzed Techniques

In our work, we analyze Patching, Transition Patching, ERMT, Selective Catching, FB, SB, and GDB. To conduct fair comparisons and account for limitations in client bandwidth, we do not consider techniques that require client download bandwidth more than double the video playback rate, such as *Fast Broadcasting* [36] and *Enhanced Fast Broadcasting* [46].

2.2 Request Scheduling

To facilitate scheduling, a waiting request queue is maintained for each video. All requests in a selected queue are serviced using only one stream, whenever available resources become available. The main scheduling policies include *First Come First Serve* (FCFS) [17], *Maximum Queue Length* (MQL) [17], *Maximum Factored Queue Length* (MFQL) [2], and *Minimum Cost First* (MCF) [43].

FCFS selects the queue with the oldest request, whereas MQL selects the longest queue, and MFQL selects the queue with the *largest factored length*. The factored length is the queue length divided by the square root of the relative access frequency of its corresponding video. In contrast, MCF is a recently proposed policy that captures the variations in stream lengths by selecting the queue requiring the least cost to serve. The cost is measured as the number of seconds of video data to be delivered. We adopt this specific definition of the delivery cost and use the preferred implementation of MCF, called *MCF-P*, which considers the cost per request.

2.3 Video Coding

Traditionally, videos are compressed and encoded as one bit-stream or what is referred to as Single Description (SD). To address the different level of client download bandwidth, videos were encoded in different copies at different bit rates, each as a SD. We will refer to this scheme as *MultiCopy*. In this scheme, clients with low bandwidths served by the copy of video encoded at a low bit rate, and those with higher bandwidths served with copies encoded at higher bit rates.

Layered Video Coding (LVC) and Multiple Description Coding (MDC) [49, 27, 21] can provide an alternative in addressing client bandwidth heterogeneity, in addition they can address temporary drop in bandwidth, and can enhance error resilience, especially MDC. With MDC and LVC, the video is encoded into two or more descriptions/layers at equal or non-equal rates, and each description is delivered independently. The quality of decoded video is proportional on the number of description received. The main difference between MDC and LVC is in the importance of the different descriptions. With MDC, all descriptions are equally important and the client can decode the received data of the video regardless of which descriptions are received, as long as at least one is received. While with LVC, the descriptions (or layers) are with decreasing importance. The base layer is required to decode related date in the first enhancement layer, and the first enhancement layer is required to decode the second enhancement layer, and so on. Therefore, unless the delivery system can address the difference in importance among layers, its capability to enhance error resilience is limited.

However, to achieve a comparable quality of a SD video, both schemes add some redundancy (encoding overhead), otherwise, the distortion in the decoded video can be unacceptable. Quantifying the overhead is very complex and depends on several factors, including the acceptable distortion level (drop in quality), the encoding (compression) algorithm, the characteristics of video data, and the number of descriptions/layers. In addition to the encoding overhead, LVC and MDC introduce network overhead. The combined overhead is referred to collectively as *overhead*. In general, the overhead of MDC is higher than that of LVC. More information on the possible values of these overheads can be found in [21] (and references within).

CHAPTER 3

WORKLOAD-AWARE RESOURCE SHARING AND CACHE MANAGEMENT

3.1 Introduction

The decision as to which class and particular technique to apply greatly impacts the overall system performance and the perceived quality-of-service (QoS). With the many available techniques, it is unclear which one is the best to use in a target environment.

This chapter provides a detailed analysis of resource sharing techniques, considering both the *True Video-on-Demand* (TVOD) and the *Near Video-on-Demand* (NVOD) models and two video workloads: *mixed-video* and *hot-video*. The first workload contains both hot (i.e., popular) and cold (i.e., unpopular) videos, whereas the second contains only hot videos. In TVOD, we compare the server resources required to service all requests immediately, whereas in NVOD, we fix the resources and compare the number of customers that can be serviced concurrently, the waiting times, and unfairness towards unpopular videos. This study examines the impacts of many system and workload parameters and introduces and applies a framework for comparing stream merging and periodic broadcasting techniques, considering the ability of the latter to provide maximum waiting time guarantees.

Guided by this extensive analysis, this chapter proposes an efficient *Workload-Aware Hybrid Solution* (WAHS) that combines the advantages of stream merging and periodic broadcasting. This solution is highly adaptive to variations in the workload and the available system resources. It first identifies the videos to be served by each technique by estimating and comparing their requirements. It then allocates resources intelligently to each technique by utilizing a proposed *prediction-based allocation* approach, which allocates resources by predicting the achieved customer defection (i.e., turn-away) probability and average waiting time by each technique. Finally, it divides the resources

allocated for periodic broadcasting among the corresponding videos using a proposed *maximum-gain allocation*, which allocates channels, one at a time, to the video that gains the most from that channel in reducing its expected defection probability or waiting time. WAHS provides a promising generic solution and performs significantly better than *Selective Catching* [24], which is up to our knowledge the best existing hybrid scheme.

Moreover, this chapter considers other specific resource sharing aspects in terms of the disk I/O bandwidth and introduces this bandwidth as an additional dimension in comparing resource sharing techniques. Motivated by the widening gap between capacities and speeds of hard disk drives [28], cache management can be used to reduce further the cost of media streaming servers by minimizing the required disk I/O bandwidth. Unfortunately, only little work has been done on cache management when scalable resource sharing is applied. The study [5] analyzed the impact of caching by actual measurements, using limited caching schemes and simple resource sharing, and showed that the disk I/O bandwidth may become a bottleneck without proper caching. This chapter proposes a *Statistical Cache Management* (SCM) approach, which computes periodically video access frequencies and determines the data to be cached based on these statistics. In addition to its performance effectiveness, it is easy to implement and incurs small overhead as updates are triggered only when the workload varies considerably. We derive analytical models for allocating cache space for various resource sharing techniques. This work differs significantly from previous work on proxy caching [37] (and references within), which has different objectives and factors affecting the allocation decisions. It also differs from low-level caching techniques, such as *Least Recently Used* (LRU), *Least Frequently Used* (LFU), and *Interval Caching* [15, 41], whereby the content of the cache changes frequently, incurring significant overheads. The rest of this chapter is organized as follows. The proposed hybrid solution and cache management approach are presented in Sections 3.2 and 3.3, respectively. Section 3.4 addresses the tuning of design parameters, and Section 3.5 discusses the performance evaluation methodology and presents the main results. Finally, conclusions are drawn in the last section.

3.2 Workload Aware Hybrid Solution (WAHS)

We propose a *Workload-Aware Hybrid Solution* (WAHS) for scalable video streaming. The solution combines stream merging and periodic broadcasting. Without lack of generality, ERMT is used for stream merging and FB is used for periodic broadcasting. As will be shown later, each performs the best in its own class of techniques requiring only two download channels. The algorithm dynamically determines the videos to be served by each technique and allocates resources efficiently to the different techniques and also to the different videos served by periodic broadcasting. The main goals in decreasing order of importance are to minimize the customer defection probability and to minimize the average waiting time. (Customers defect without being served when their waiting time exceeds their waiting tolerance.) Table 3.1 summarizes the used symbols, and Table 3.2 shows a simplified algorithm. WAHS proceeds in the following three main steps.

Table 3.1: Symbols used in WAHS Algorithm

Parameter Name(s)	**Symbol(s)**
Number of videos served by FB/ERMT	V_{FB}, V_{ERMT}
Number of server channels	Ch
ERMT requirement to provide TVOD for videos s till t/for the i_{th} Video	$Ch_{ERMT:TVOD_{s:t}}$, $Ch_{ERMT:TVOD_i}$
FB requirement to provide zero defections for videos s till t/for the i_{th} video	$Ch_{FB:Zero_{s:t}}$, $Ch_{FB:Zero_i}$
Channels allocated for all FB videos/for the i_{th} video	Ch_{FB}, Ch_{FB_i}
Channels shared by all ERMT videos	Ch_{ERMT}
Request arrival rate for FB/ERMT videos (req./s)	$\lambda_{FB}, \lambda_{ERMT}$
Defection prob. for all videos/for the i_{th} video	DP, DP_i
Defection prob. for FB/ERMT videos	DP_{FB}, DP_{ERMT}
Average waiting time for all videos/for the i_{th} video	WT, WT_i
Average waiting time for FB/ERMT videos	WT_{FB}, WT_{ERMT}

3.2.1 Step 1: Identify the Videos to be Served by Each Technique

In this step, the videos served by each technique (ERMT or FB) are identified. This task translates to finding the first V_{FB} videos to be served by FB because the more popular the video is, the more likely it is to be suitable for service by FB. The algorithm first determines whether ERMT

Table 3.2: Simplified WAHS Algorithm

Step	Description
1	**Determine Number of Videos (V_{FB}) to be Served by FB:**
1.1	Estimate ERMT Channel Requirement ($Ch_{ERMT:TVOD_{1:V}}$) to provide TVOD:
	$V_{FB} = 0$; $\eta = 2.0$;
	$Ch_{ERMT:TVOD_{1:V}} = \sum_{i=1}^{V} \eta * log(N_i/\eta)$;
	if ($Ch_{ERMT:TVOD_{1:V}} < Ch$)
	proceed to Step 2;
1.2	Estimate FB Channel Requirement ($Ch_{FB:Zero_{1:V}}$) to provide zero defection;
	if ($Ch_{FB:Zero_{1:V}} < Ch$) {
	$V_{FB} = V$; *proceed to Step* 2; }
1.3	Estimate $Ch_{ERMT:TVOD_i}$ and $Ch_{FB:Zero_i}$ per video i
	for ($i = 1, i <= V, i++$) {
	if ($Ch_{FB:Zero_i} < Ch_{ERMT:TVOD_i}$) {$V_{FB}++$;}
	else {*break*;} }
2	**Determine Number of Channels (Ch_{FB}) for FB:**
2.1	Minimize Total Customer Defection Probability (DP):
	$Ch_{FB} = minimum(Ch \times \frac{\lambda_{FB}}{\lambda}, Ch_{FB:Zero_{1:V_{FB}}})$;
	$Ch_{ERMT} = Ch - Ch_{FB}$; $currentDP = DP$;
	while ($DP_{FB} > 0.0$ & $Ch_{ERMT} > 0$ & $DP < currentDP$) {
	$currentDP = DP$; $Ch_{FB}++$; $Ch_{ERMT}--$;
	re-predict DP_{FB}, DP_{ERMT}, *and* DP }
2.2	Minimize Total Customer Waiting Time (WT):
	$currentWT = WT$;
	while ($DP == 0.0$ & $Ch_{ERMT} > 0$ & $WT < currentWT$) {
	$currentWT = WT$; $Ch_{FB}++$; $Ch_{ERMT}--$;
	re-predict WT_{FB}, WT_{ERMT}, *and* WT }
3	**Distribute Broadcast Channels among FB Videos:**
3.1	Allocate minimum channels per video (*round robin from most popular to least*);
3.2	Minimize Total Customer Defection Probability (DP):
	while ($DP_{FB} > 0.0$ & $allocatedChannels < Ch_{FB}$) {
	/* *Pick the video to get the next channel* */
	$maxGain = 0.0$; $pickedVideo = 1$;
	for ($i = 1, i <= V_{FB}, i++$) {
	$gain = (DP_i(Ch_{FB_i}) - DP_i(Ch_{FB_i} + 1)) \times \lambda_i$;
	if ($gain > maxGain$) {
	$maxGain = gain$; $pickedVideo = i$; } }
	$Ch_{FB_{pickedVideo}}++$; $allocatedChannels++$; }
3.3	Minimize Total Customer Waiting Time (WT):
	while ($allocatedCh < Ch_{FB}$) {
	/* *Pick the video to get the next channel* */
	$maxGain = 0.0$; $pickedVideo = 1$;
	for ($i = 1, i <= V_{FB}, i++$) {
	$gain = (WT_i(Ch_i) - WT_i(Ch_{FB_i} + 1)) \times \lambda_i$;
	if ($gain > maxGain$) {
	$maxGain = gain$; $pickedVideo = i$; } }
	$Ch_{FB_{pickedVideo}}++$; $allocatedChannels++$; }

can serve all video requests immediately without any customer defections and then whether FB can serve all the requests without defections. If the number of channels required by any one of these techniques is within server capacity, then only that technique is used. Otherwise, the algorithm compares the channel requirements to serve each video by ERMT and FB and selects for that video the technique that requires the least.

The channel requirement by ERMT to serve all videos immediately (i.e., achieve TVOD) can be found by adding up the requirement for each video: $Ch_{ERMT:TVOD_{1:V}} = \sum_{i=1}^{V} Ch_{ERMT:TVOD_i}$. In ERMT, all videos share the same pool of resources. However, when the number of videos is large enough, $Ch_{ERMT:TVOD_i}$ can be substituted for the average server channel requirement for video i, which is derived in [19]. Thus,

$$Ch_{ERMT:TVOD_i} = \eta * \log(\lambda_i \times D_i/\eta), \tag{3.1}$$

where λ_i is the request arrival rate, D_i is the length (i.e., duration) of video i, and η is the deviation of the average number of channel requirements. η was found to be 1.62 for optimal stream merging [19], but for ERMT, we found that 2.0 is more accurate, as shown in Figure 3.1(a).

The channel requirement by FB is independent of the request rate. Moreover, FB has the advantage of encouraging customers to wait by providing them with highly accurate times of service. Hence, to achieve zero defections for a video, FB requires keeping the maximum waiting time (which is equal to the first segment length) for that video shorter than a certain value.

3.2.2 Step 2: Split the Resources

Once the videos to be served by each technique are determined, resources should be allocated intelligently to each technique. We propose a *prediction-based allocation approach*, which allocates resources by predicting the achieved customer defection probability and average waiting time by each technique. The algorithm first sets the number of channels for FB (Ch_{FB}) to the smaller value of the number of channels to achieve no defections and the number of channels based on its fair share in terms of the number of requests. The algorithm then tries to reduce the overall

defection probability by incrementing Ch_{FB} repeatedly as long as the overall defection probability continues to decrease and remains greater than 0. Subsequently, the algorithm tries to minimize the average waiting time if the defection probability is 0 by incrementing Ch_{FB} repeatedly as long as the waiting time continues to decrease and the defection probability remains 0.

The overall defection probability (DP) and waiting time (WT) can be estimated by the weighted sums of the corresponding technique-level metrics:

$$DP = DP_{FB} \times \frac{\lambda_{FB}}{\lambda} + DP_{ERMT} \times \frac{\lambda_{ERMT}}{\lambda} \tag{3.2}$$

and

$$WT = WT_{FB} \times \frac{\lambda_{FB}}{\lambda} + WT_{ERMT} \times \frac{\lambda_{ERMT}}{\lambda}. \tag{3.3}$$

The weighing is done based on the share of the number of requests. The technique-level metrics can be computed as the weighted sums of the corresponding per-video metrics. Hence, for FB,

$$DP_{FB} = \sum_{i=1}^{V_{FB}} DP_i \times \frac{\lambda_i}{\lambda_{FB}} \tag{3.4}$$

and

$$WT_{FB} = \sum_{i=1}^{V_{FB}} WT_i \times \frac{\lambda_i}{\lambda_{FB}}. \tag{3.5}$$

As discussed in Step 1, the estimations of the ERMT-level metrics are performed at the video level. Thus, DP_{ERMT} and WT_{ERMT} are computed as DP_{FB} and WT_{FB} in Equations (3.4) and (3.5) except for using the ERMT metrics and summing over the videos from $V_{FB} + 1$ and V.

Predicting Per-Video Metrics for FB

For FB, the defection probability for a video is a function of the waiting time, waiting tolerance model, and the number of channels allocated for that video. The first segment length (l_i), which represents the maximum waiting time for any customer requesting that video, can be easily calculated based on the number of allocated channels. Assuming equal probability for request arrival during

that period, $DP_i = \int_0^{l_i} F(x).dx$, where $F(x)$ is the cumulative distribution function of the customer waiting tolerance model. This model can be built dynamically based on recent history. Step 3 determines the number of channels allocated for each one of the videos served by FB. That step is called iteratively in this step (before it is finally executed in Step 3 to obtain the actual channel allocation). The per-video average waiting time is a function of only the allocated channels. Thus, WT_i is simply $l_i/2$. Figure 3.1(b) shows the estimation accuracy of the defection rate.

Predicting Per-Video Metrics for ERMT

The prediction here differs from that for FB in two main ways. First, because videos served by ERMT share the same pool of resources, the allocation of channels to different videos is hard to estimate. Under a fair scheduling policy, such as FCFS, each video is expected over a long period of time to receive a fair number of channels, which is a function of its ratio of the overall number of video requests. Thus, the number of channels for video i can be estimated by

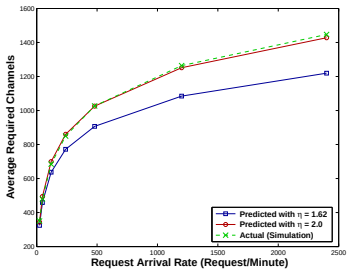
$$Ch_i = \frac{\log(\lambda_i \times D_i)}{\sum_{v=V_{ERMT}}^{V} \log(\lambda_v \times D_v)} \times Ch_{ERMT}. \tag{3.6}$$

The $\log$ represents the relation between the required channels and the request rate in case of ERMT, as shown in Equation (3.1). Under a biased policy, such as MQL, hot videos can receive more than their fair share. Second, the average video inter-service time should also be estimated. The inter-service time for video i (Δ_{S_i}) is the reciprocal of that video's stream initiation rate (λ_{S_i}), which can be estimated using Equation (3.1) as the maximum request arrival rate that can be served immediately by ERMT. The defection probability and waiting time can now be estimated as follows:

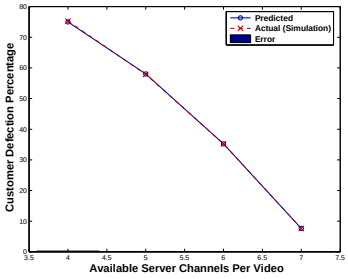
$$DP_i = \frac{\lambda_i - \lambda_{S_i}}{\lambda_i} \int_0^{\Delta_{S_i}} F(x).dx \tag{3.7}$$

and

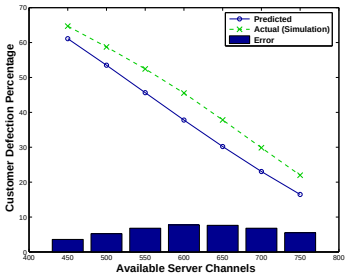
$$WT_i = \frac{\lambda_i - \lambda_{S_i}}{\lambda_i} \times \frac{\Delta_{S_i}}{2}. \tag{3.8}$$



(a) Average Server Channel Requirements by ERMT (TVOD)



(b) Defection Probability with FB



(c) Defection Probability with ERMT (120 Requests/Minute, FCFS)

Figure 3.1: Metric Prediction Accuracy [120 Videos, Video Length = 120 Minutes]

$F(x)$ with ERMT is likely to be different from that with FB because ERMT does not provide time of service guarantees. Multiplying by $\frac{\lambda_i - \lambda_{S_i}}{\lambda_i}$ is because ERMT does not initiate streams unless they are requested. This indicates that a number of clients equal to the number of the video streams must be served and are exempted from the defection prediction process. The equations assume $\lambda_i > \lambda_{S_i}$, which should be the reason why this step is reached. As depicted in Figure 3.1(c), the defection rate estimation is fairly accurate but not as accurate as that for FB due to the aforementioned complexities.

3.2.3 Step 3: Distribute the FB Resources

Instead of allocating the channels dedicated for FB uniformly to the corresponding videos or based on the fractions of request arrival rate, we propose a *maximum-gain allocation* approach, which allocates channels, one by one, to the video that gains the most from that channel in reducing its expected defection rate, weighted by its request rate. When all videos are expected to have zero defections, the allocation is based on the weighted gain in reducing the average waiting time.

3.3 Statistical Cache Management (SCM)

Cache management can be applied with resource sharing techniques to reduce further the cost of media streaming servers by minimizing the required disk I/O bandwidth. *Interval Caching* [15],

an efficient cache management technique, caches intervals between successive streams in the server main memory. It caches the data brought by a stream and reuses it in servicing a closely following stream. This technique incurs significant overhead and its effectiveness was not analyzed with scalable resource sharing techniques.

We propose and investigate a *Statistical Cache Management* (SCM) approach. It periodically computes video access frequencies and determines the data to be cached based on these statistics. In addition to its performance effectiveness, it is easy to implement and incurs small overhead as updates are triggered only when the workload varies considerably. We first derive the access distribution equations for videos served by various techniques and then use these equations to determine the optimal cache allocation. For ease of reference, Table 3.3 describes the main parameters used in the subsequent analysis.

Table 3.3: Symbols used in SCM Analysis

Parameter Name	**Symbol**
Request arrival rate for video j (req./s)	λ_j
Inter-arrival time for video j = 1/λ_j (s)	Δ_j
Data with playback position at time t in video j	$data(j,t)$
Access Distribution: the access frequency of data d (req./s)	$Fr(d)$
Cache size (s)	S
Size of cached portion of video j (s)	S_j
Number of videos	V
Duration of video j (s)	D_j
Regular window for video j (s)	Wr_j
Transition window for video j (s)	Wt_j
Number of requests per video length for video $j = \lambda_j \times D_j$	N_j

3.3.1 Video Access Distributions

We derive next the access distribution equations for each playback point in a video delivered using various techniques. We assume here TVOD except for periodic broadcasting. Hence, the average service rate of video j is the same as its average request arrival rate (λ_j). For NVOD, the video's stream initiation rate (λ_{S_j}) can be used instead. We also assume that $Fr(data(j,t))$ is the

rate (frequency) by which the t^{th} point of time of video j is requested from the disks when there is no cache. The derived equations were validated by simulation.

Patching

Because of having two types of streams in Patching and having a limit on the maximum patch length (Wr_j), the video can be divided into two regions: $[0, Wr_j]$ and $[Wr_j, D_j]$. Since every regular stream delivers the full video data, all points in the second region will be requested with an equal rate: $1/Wr_j$. The situation is different for the first region because the patches vary in length. The closer the point to the beginning of the video, the more likely it will be missed and requested by patches. The access rate approaches the video request rate (λ_j) as the point becomes closer to the beginning. Consequently, the access distribution can be formulated as follows:

$$Fr(data(j,t))=\begin{cases} \lambda_j - (\frac{\lambda_j}{Wr_j} - \frac{1}{(Wr_j)^2})t & 0 \leq t < Wr_j, \\ \frac{1}{Wr_j} & Wr_j \leq t \leq D_j. \end{cases} \tag{3.9}$$

Transition Patching

The stream types here are patches with maximum length of Wt_j, transition patches with lengths between $3Wt_j$ and $Wr_j + 2Wt_j$, and regular streams with full video length. Therefore, the region $[Wr_j + 2Wt_j, D_j]$ is delivered by only regular streams. As in Patching, the rate for region $[0, Wt_j]$ decreases linearly from λ_j to $1/Wt_j$. The region $[Wt_j, 3Wt_j]$ is delivered by every transition patch and regular stream, $[3Wt_j, 4Wt_j]$ is delivered by every regular stream and transition patch but the first transition patch in every regular window (Wr_j), and so on. Consequently, the access

distribution can be formulated as follows:

$$Fr(data(j,t))=\begin{cases}\lambda_j-(\frac{\lambda_j}{Wt_j}-\frac{1}{(Wt_j)^2})t & 0\le t<Wt_j,\\ \frac{1}{Wt_j} & Wt_j\le t<3Wt_j,\\ \frac{1}{Wt_j}-\frac{n}{Wr_j} & 3Wt_j\le t<Wr_j+2Wt_j,\\ \frac{1}{Wr_j} & Wr_j+2Wt_j\le t\le D_j,\end{cases} \tag{3.10}$$

where $n=\lfloor\frac{t}{Wt_j}\rfloor-2$.

ERMT

By studying the average behavior of ERMT over a long period of time with N_j requests per video length, it can be noticed that almost every stream delivers independently the first part of the video from the beginning to around Δ_j, and almost $N_j-N_j/2$ streams deliver the data from approximately Δ_j to $4\times\Delta_j$ per video length, and so on. In general, the access distribution is given by

$$Fr(data(j,t))\approx\frac{\lambda_j}{2^{(1+\lfloor\log_2(t/\Delta_j+2)/3\rfloor)}}. \tag{3.11}$$

Periodic Broadcasting

For each video segment, the access distribution function here is simply the inverse of the segment repeat time. For GDB, SB, and FB, the segment repeat time is the segment length.

Hybrid Techniques

As with periodic broadcasting, the access frequency with Catching for data in any segment other than the first is equal to the inverse of the segment length. Data in the first segment receives additional access due to the patches. The additional access frequency decreases linearly up to the

last point in the first segment. Consequently, the access distribution is given by

$$Fr(data(j,t))=\begin{cases}\lambda_j + \frac{h(K_j)}{D_j} - \lambda_j \frac{h(K_j)}{D_j} t & 0 \leq t \leq \frac{D_j}{h(K_j)}, \\ \frac{h(K_j)}{f(n)D_j} & \frac{D_j}{h(K_j)} < t \leq D_j, \end{cases} \tag{3.12}$$

where $h(m) = \Sigma_{i=1}^{m} f(i)$, $f(i)$ is the i_{th}-segment's relative size compared to the first segment in MGDB partition function (Table 2.1), and n can be found using the inverse of function $h(m)$ as $n = h^{-1}(\frac{h(K_j)}{D_j} t)$ (Equation 2.1).

For Selective Catching, Equation (3.12) can be used for hot videos, whereas Equation (3.9) can be used for cold videos.

For WAHS, Equation (3.11) can be used for the ERMT videos, whereas the calculations in Subsection (3.3.1) apply for FB videos.

3.3.2 Cache Allocation Model

Two conditions must be met for optimal cache allocation:

$$S = \sum_{j=1}^{V} S_j \quad \text{and} \quad Fr(d_{in}) \geq Fr(d_{out}), \tag{3.13}$$

$\forall d_{in} \in cached\text{-}data$ and $\forall d_{out} \notin cached\text{-}data$. Because the derived access distribution models exhibit the property $Fr(data(j,t_1)) \geq Fr(data(j,t_2))$ for all $t_1 < t_2$ in the j^{th} video, if the optimal cache size for the j^{th} video is found to be S_j, this implies that the first S_j of data from the beginning of the video must be cached. The allocation can be performed by proceeding with the video blocks (across all videos) in decreasing order of access frequency and caching them one at a time until the cache reaches its maximum capacity.

3.4 Tunings of Design Parameters

Server performance depends greatly on how design parameters are tuned. In this section, we derive equations for determining optimally the values of W_r and W_t for Transition Patching. For

Selective Catching, we present an approach for deciding on what videos to serve with Catching and what videos with Controlled Multicast when the server resources are limited. No approach was provided in [24] for such situations. For the other techniques, we highlight important results reported in [29, 24, 23].

3.4.1 Transition Patching

The objective here is to derive simple equations for Wr and Wt, which minimize the number of required server channels. The number of required channels (ch_j) with any technique is given by

$$ch_j = \int_0^{D_j} Fr(data(j,t))dt, \tag{3.14}$$

where $Fr(data(j,t))$ is the request distribution for the chosen technique. For Transition Patching,

$$ch_j = \frac{1}{2}\lambda_j Wt_j + \frac{1}{2}\frac{Wr_j}{Wt_j} + \frac{D_j}{Wr_j} - 2\frac{Wt_j}{Wr_j} + 1. \tag{3.15}$$

By taking the first derivative of Equation (3.15) with respect to Wr_j and equating it with zero, we get the optimal value of Wr_j:

$$Wr_j = \sqrt{2Wt_j(D_j - 2Wt_j).} \tag{3.16}$$

Since the transition patch cannot exceed the video length (i.e., $Wr_j + 2Wt_j < D_j$) and $Wt_j < Wr_j$, it follows that $Wt_j < \frac{1}{3}D_j$.

Similarly, to find the optimal value of Wt_j, we take the first derivative of Equation (3.15) with respect to Wt_j and equate it with zero, and then substitute the value of Wr_j from Equation (3.16):

$$Wt_j = \sqrt[3]{2D_j/\lambda_j^2} \times 1/\sqrt[3]{1 - 2Wt_j/D_j} = \sqrt[3]{2D_j\Delta_j^2}/\sqrt[3]{1 - 2Wt_j/D_j}. \tag{3.17}$$

Typically, $D_j >> 2Wt_j$, and thus

$$Wt_j \approx \sqrt[3]{2\Delta_j^2 D_j}. \tag{3.18}$$

Alternatively, we can substitute this value in Equation (3.17) and get the following equation that is more accurate:

$$Wt_j \approx \sqrt[3]{2\Delta_j^2 D_j} / \sqrt[3]{1 - 2\sqrt[3]{2\Delta_j^2 / D_j^2}}. \tag{3.19}$$

To avoid a negative value of Wt_j, $N_j = \lambda_j D_j > 4$. (It can be shown that for Transition Patching to be more cost effective than Patching, $N_j = \lambda_j D_j \geq 12$.)

3.4.2 Patching and Controlled Multicast

In [29], an optimal value for Wr_j in Patching/Controlled Multicast was derived:

$$Wr_j = \sqrt{2D_j/\lambda_j} = \sqrt{2\Delta_j D_j}. \tag{3.20}$$

Based on that, the number of required channels can be given by

$$ch_j = \sqrt{2\lambda_j D_j} - 1/2. \tag{3.21}$$

A slightly different equation was reached in [23], but our analysis favors Equation (3.20). Equations (3.21) and (3.20) assume $N_j = \lambda_j D_j > 1$; otherwise, no two streams will overlap and a unicast is the only option.

3.4.3 Catching and Selective Catching

In [24], the required number of server channels for Catching was shown to be

$$ch_j = K_j + \lambda_j F S_j / 2, \tag{3.22}$$

where FS_j is the first segment length of video j ($FS_j = D_j / \Sigma_{i=1}^{K_j} f(i)$) and K_j is the number of its broadcast channels. This equation can be optimized with respect to K_j to find the optimal value K_j^*.

In Selective Catching, a key issue is how to decide whether a video is hot or cold. One possible approach is to compare the numbers of required channels to serve the video with Catching (Equation (3.22)) and with Controlled Multicast (Equation (3.21)) [24]. The video is considered hot if the first number is smaller and cold otherwise. However, when the number of server channels is not sufficient to satisfy the Catching requirement for every hot video, we need to determine the subset of the hot videos to be served by Catching. We introduce the following *additional* test: video j is served by Catching if

$$ServerChannels \times \frac{\lambda_j}{\lambda} \geq ch_j. \tag{3.23}$$

The idea here is to serve by Catching, among the hot videos, only the videos whose numbers of required channels are not greater than their fair share, determined by their contributions to the total number of requests.

3.5 Performance Evaluation and Main Results

We have developed a validated simulator for streaming servers. It uses optimal tuning of design parameters for Transition Patching and Selective Catching based on our derived analytical models provided in Section 3.4. Simulation stops after a steady state analysis with 95% confidence interval is reached.

We consider two service models (TVOD and NVOD) and two video workloads (*mixed-video* and *hot-video*), leading to four target environments. In the TVOD model, we compare the server resources required to service all requests immediately, whereas in NVOD, we fix the server resources and compare the customer defection probability, average waiting time, and unfairness towards cold videos. The first metric translates directly to the number of customers that can be serviced concurrently. The two considered server resources are *server channels* and *disk channels*, which correspond to the network bandwidth and disk I/O bandwidth requirements, respectively. A channel can support only one stream at a time. Without caching, the numbers of these channels are equal. The mixed-video workload contains both hot and cold videos. The hot-video workload is used mainly to evaluate periodic broadcasting techniques, which are designed mainly for hot videos.

3.5.1 Workload Characteristics

We assume that the arrival of requests follows a Poisson Process and that the accesses to videos follow a Zipf-like distribution with skewness parameter θ. The mixed-video workload contains 120 2-hour videos, whereas in the hot-video workload, we consider three numbers of 2-hour videos: 1, 10, and 20. Table 3.4 shows the values of the input parameters in both workloads. For scheduling, we use MQL and FCFS and do not show the results of MFQL because it does not perform well in the stream merging environment.

We consider two models of customer waiting tolerance. In general, we assume as in prior work that the tolerance follows the exponential distribution with a mean of 2 minutes. For periodic broadcasting techniques, we borrow and apply the *Time-of-Service Guarantee* (TSG) model [3] to capture the ability of these techniques to provide time-of-service guarantees by upper limiting the waiting times. With TSG, customers who receive time-of-service guarantees shorter than 2 minutes will wait for service, whereas the waiting tolerance of all other customers follows the exponential distribution with a mean of 2 minutes.

Table 3.4: Summary of Workload Characteristics

Parameter	**Mixed-Video**	**Hot-Video**
Request arrival rate (req./min)	5 - 240	60 - 2400
Number of videos	120	$1, 10, 20$
Video duration (min)	120	
Video skewness (θ)	0.271	
Cache size	5% of total size of videos	
Scheduling policy	MQL	MQL, FCFS
Waiting tolerance model	Exponential, TSG	

To study the effectiveness of WAHS in handling dynamic variations in the request arrival rate and video popularity, we use the following dynamic workload. The request rate has a 24-hour cycle with a maximum (peak) and a minimum period, separated by 12 hours, and the minimum request rate being half of the maximum. The rate changes gradually in 3-hour steps by dividing/multiplying by $2^{\frac{1}{4}}$. The video popularity has a daily pattern and a day-to-day pattern. Within the day, when the system goes from the maximum to the minimum rate, each video popularity drops one rank every

90 minutes and the least popular becomes the most popular, and when going from the minimum to the maximum rate, each video gains a rank every 180 minutes and the most popular becomes the least popular. Hence, most videos loose 4 ranks from a day to the next. This workload evaluates WAHS in comparison with ERMT and FB when significant changes in the request rate and video popularity happen in short periods of time. It is synthetic and meant to be much worse than that expected in reality, where changes are more gradual and do not happen to all videos simultaneously.

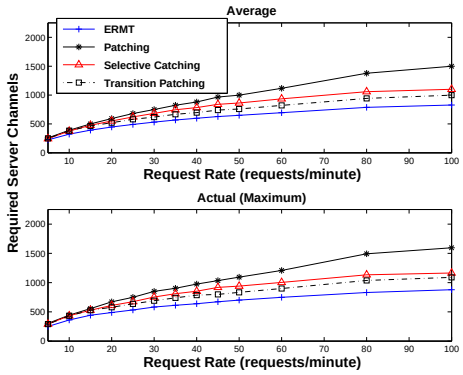
3.5.2 Analysis of Existing Techniques

Environment I: Mixed-Video and TVOD

This environment helps in comparing stream merging and composite techniques in terms of the number of server and disk channels required to achieve TVOD (i.e. zero defection and waiting time). We vary here the request arrival rate from 5 to 100 requests per minute and examine the average and the maximum server channels required by the different techniques. The average results represent the average requirements over long periods, while the maximum results represent the actual required number of channels for providing TVOD for all requests. Figure 3.2(a) plots these results. As expected, the gaps among the techniques widen as the arrival rate increases because the workload offers increasingly higher degrees of resource sharing, which are exploited to different levels by different techniques. Generally, ERMT has the lowest requirements, and Patching the highest. Interestingly, Selective Catching does not perform relatively well. We have experimented with using ERMT instead of Patching in Selective Catching for serving cold videos but the performance benefits are low.

Environment II: Hot-Video and TVOD

In this environment, we compare the performance of various periodic broadcasting techniques (SB, GDB, and FB) and the best performer in Environment I (ERMT). Since periodic broadcasting cannot provide a zero waiting time, we assume that a maximum waiting time below 2 seconds constitutes a TVOD service.



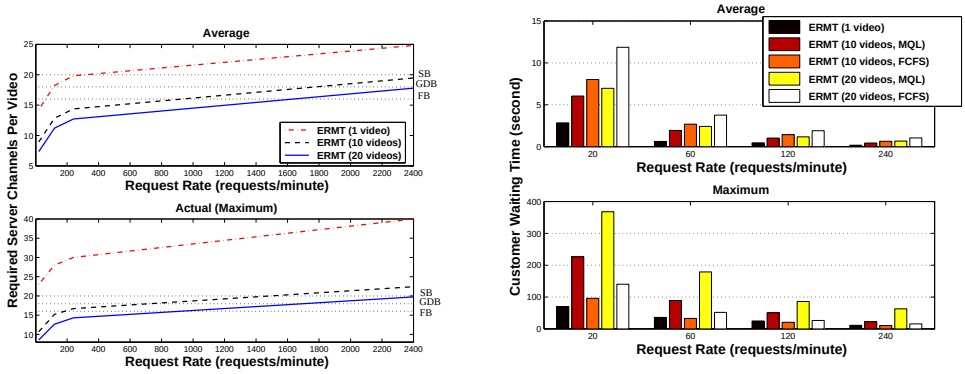
(a) Effect of Request Rate on Required Channels

Figure 3.2: Environment I: Mixed-Video and TVOD

Figure 3.3(a) plots the average and maximum required server channels per video to provide TVOD by the different techniques. Unlike periodic broadcasting, the number of server channels per video with ERMT depends on the arrival rate and the number of supported videos. Whereas periodic broadcasting techniques reserve channels for each video, ERMT deals with all the resources as one pool and distributes them dynamically to the requests. While the average helps in understanding the overall system load, the maximum indicates the actual bandwidth to be reserved to achieve TVOD. The distinction between the two metrics was always ignored in previous studies, where only the average results were usually reported. The results demonstrate that ERMT scales very well with the increase in request rate even in terms of both the average and actual (maximum) required bandwidth. For a workload of 20 2-hour videos, the request rate must be constantly over 900 requests/minute for the best periodic broadcasting technique (FB) to be a better choice than ERMT. Even when that is the case, while ERMT requires more guaranteed bandwidth, it does not consume it all the time, which may enable the system to face short periods of drops in server capacity. Since the required bandwidth with ERMT is a function of the number of requests per video length (i.e., $N = \lambda D$), the breakpoint is higher than 900 requests/minute when the videos are shorter.

To highlight the effect of the difference between the average and actual required number of channels, Figure 3.3(b) plots the average and maximum waiting times if only the average required

bandwidth is provided. Therefore, the number of provided channels varies with the request rate and with the number of videos as well. The service is obviously not TVOD, especially for the relatively low request rates because some customers had to wait few minutes for service. Since scheduling policies can impact the performance of ERMT compared with periodic broadcasting, both MQL and FCFS are examined. MQL achieves higher throughput and shorter average waiting time, whereas FCFS is fairer and can significantly reduce the maximum waiting time.



(a) Effect of Request Rate on Required Channels

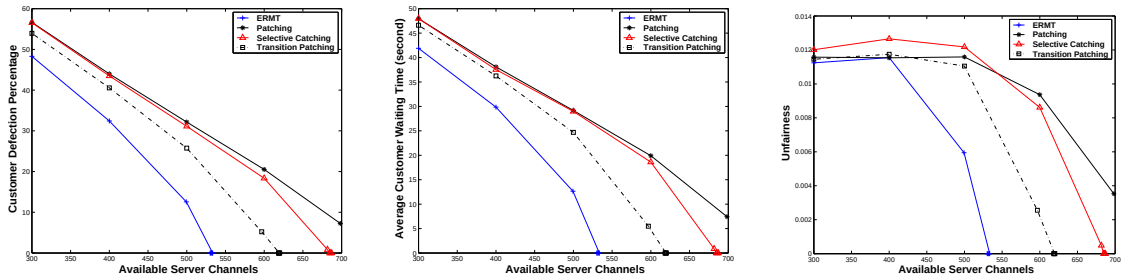
(b) Effect of Request Rate on Waiting Time (Average Required Channels Provided)

Figure 3.3: Environment II: Hot-Video and TVOD

Environment III: Mixed-Video and NVOD

By varying the number of server channels, we can use this environment to compare stream merging and composite techniques in terms of the achieved average customer defection probability, waiting time, and unfairness. Figure 3.4 plots these three metrics versus the number of server channels for various techniques. The results here demonstrate once again that ERMT performs the best and Patching the worst among stream merging and composite techniques. There is a clear correlation among the three metrics. The correlation between unfairness and the other two metrics is due to using MQL. MQL selects for service the longest video queue, trading off fairness for performance. The more limited the number of channels becomes, the longer customers have to

wait, and the faster the waiting queues build up. Consequently, both the defection probability and the bias against cold videos increase.



(a) Effect of Server Channels on Defection Probability

(b) Effect of Server Channels on Average Waiting Time

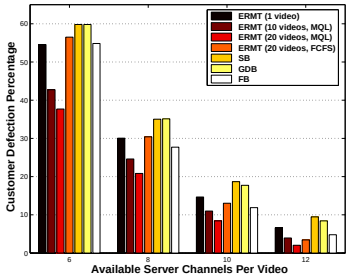
(c) Effect of Server Channels on Unfairness

Figure 3.4: Environment III: Mixed-Video and NVOD [30 Requests/Minute]

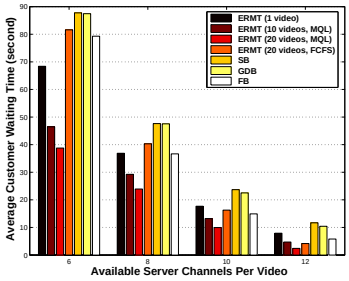
Environment IV: Hot-Video and NVOD

In this environment, we also compare the performance of various periodic broadcasting techniques with ERMT, considering two waiting tolerance models: exponential and TSG. Figures 3.5(a) through 3.5(d) depict the defection probability and average waiting time under the exponential model for different server capacities, request rates, and numbers of videos. These results confirm that ERMT is very competitive, compared with the best periodic broadcasting technique, even in servicing very hot videos and under very limited resources.

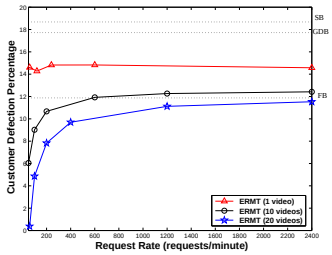
Figures 3.5(e) and 3.5(f) compare various techniques in terms of the defection probability and average waiting time under the TSG model, which captures the ability of the broadcasting techniques in providing waiting time guarantees. The results demonstrate that broadcasting techniques can achieve much lower defection probabilities than ERMT, especially when the available server bandwidth is not very limited. With only 8 server channels per video, all broadcasting techniques cause no defections, whereas ERMT causes more than 20% defections. However, ERMT achieves shorter average waiting time for those clients who were serviced, partially due to the high defection percentage.



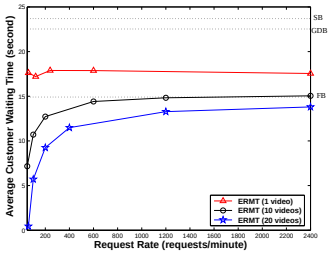
(a) Effect of Server Channels per Video on Defection Probability (Exponential)



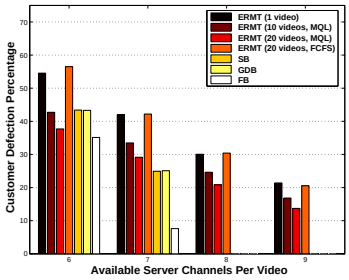
(b) Effect of Server Channels per Video on Average Waiting Time (Exponential)



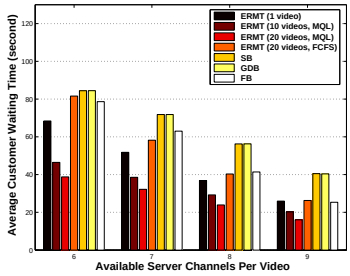
(c) Effect of Request Rate on Defection Probability (Exponential)



(d) Effect of Request Rate on Average Waiting Time (Exponential)



(e) Effect of Server Channels per Video on Defection Probability (TSG)



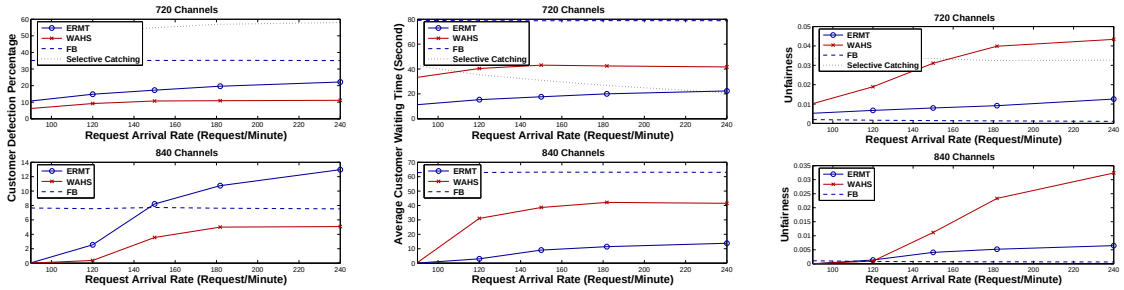
(f) Effect of Server Channels per Video on Average Waiting Time (TSG)

Figure 3.5: Environment IV: Hot-Video and NVOD [Defaults are 240 Requests/Minute, 10 Channels per Video, MQL]

3.5.3 Effectiveness of the Proposed Hybrid Solution

Under the mixed-video workload, Figure 3.6 demonstrates that WAHS outperforms both ERMT and FB in throughput, the most important metric, and its performance in the average waiting time lies between ERMT and FB. In addition, WAHS provides the majority of customers with time-of-service guarantees, whereas ERMT does not. As expected, FB is the best in fairness (the least important metric). Selective Catching performs poorly and thus its results are only shown for 720 channels. Its relative performance is worse with 840 channels.

Let us now discuss the impact of dynamic workload. Figure 3.7 explains how the per-video channel allocation transition is handled when the number of allocated channels for a video is changed due to workload changes. The example shows a switch from 6-channel allocation to 5-channel for



(a) Defection (b) Waiting (c) Unfairness

Figure 3.6: WAHS Compared to ERMT and FB [MQL, TSG]

a *2-hour* video. The required 5 channels in the new allocation will be required all the time, during the transition period and afterward, but an additional channel is needed for short periods during the transition. (The additional number of needed channels is the difference between the two allocations. These channels are required to change the allocation without impacting existing clients.) Although each additional channel can be used for several video transitions, to be conservative, a maximum of two transitions were allowed to share such a channel. Under the dynamic workload, Figure 3.8 demonstrates that WAHS still keeps its lead in the defection rate.

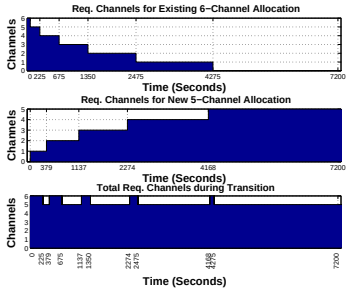
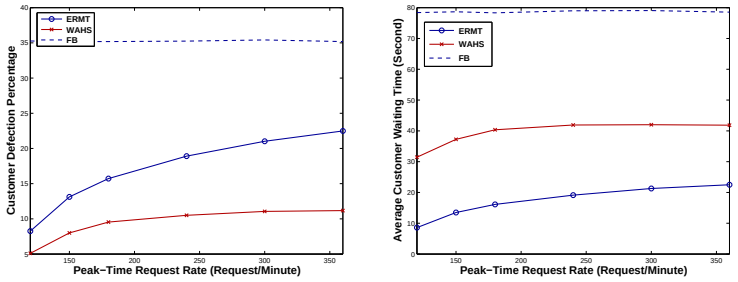


Figure 3.7: Illustration of Support for Per-Video Channel Allocation with FB (6 to 5 Channels)



(a) Defection (b) Waiting

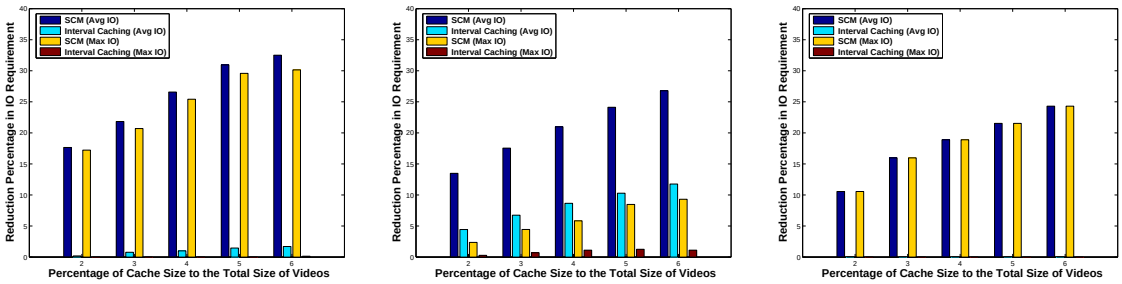
Figure 3.8: WAHS Compared to ERMT and FB under Dynamic Workload [720 Channels, MQL, TSG]

Table 3.5: Percentage of Customers Receiving Time-of-Service Guarantees with WAHS [720 Channels, MQL]

Request Arrival Rate (Requests/Minute)	Received TSG (Percentage)	No TSG (Percentage)
90	63%	37%
120	74%	26%
150	83%	17%
180	91%	9%

3.5.4 Effectiveness of the Proposed Cache Management Scheme

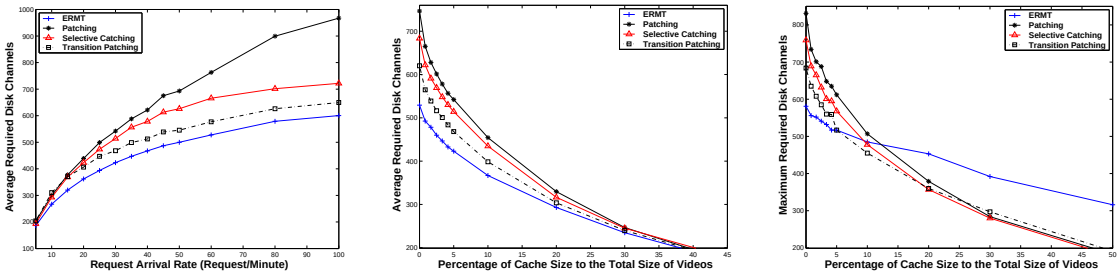
Finally, let us evaluate the effectiveness of SCM. Figure 3.9 compares SCM and Interval Caching in terms of the reduction in the average and maximum I/O requirements when using WAHS, ERMT, and FB. The results show that SCM performs significantly better than Interval Caching. With a cache size of 2% of the total video size, the reductions achieved by SCM are up to 18%. The reductions in the maximum I/O requirements in the case of ERMT are exceptionally low (but still much better than Interval Caching) because of the dynamic nature of ERMT and the irregularity in resource consumption and allocation among videos. Interval Caching causes no gain with FB because it exploits the data overlap between streams, and no two streams that coexist in FB carry the same data.



(a) WAHS (b) ERMT (c) FB

Figure 3.9: Effectiveness of SCM vs. Interval Caching [720 Channels, 150 Requests/Minute, MQL, TSG]

Figure 3.10 compares the effectiveness of SCM with other resource sharing techniques, including Patching, Transition Patching, and Selective Catching. ERMT requires the lowest average disk I/O channels, and Patching the highest. However, as expected, the effectiveness of caching decreases, and the gaps among various techniques diminish as we keep increasing the cache size. Interestingly, whereas ERMT requires the smallest actual number of disk channels with no cache, the situation changes when the cache is introduced as shown in Figure 3.10(c). With a cache size greater than 5% of the total size of videos, Transition Patching starts requiring fewer disk channels than ERMT. With a cache size greater than 12%, all the other techniques require fewer disk channels than ERMT. ERMT benefits the least from caching and Patching the most because of the access distribution curves in Subsection 3.3.1.



(a) Effect of Request Rate on Average Disk Channels

(b) Effect of Cache Size on Average Disk Channels

(c) Effect of Cache Size on Maximum Disk Channels

Figure 3.10: Impact of Caching [Environment I, 30 Requests/Minute, SCM]

3.6 Conclusions

The main results can be summarized as follows. (1) In workloads containing both hot and cold videos, ERMT generally performs the best among existing techniques. (2) In workloads containing only hot videos, FB performs better than both SB and GDB in terms of the waiting times. For NVOD, FB achieves higher throughput than ERMT, assuming customers are encouraged to wait when informed with their times of service. ERMT, however, achieves shorter waiting times. It is also more capable of providing TVOD. (3) WAHS outperforms both ERMT and FB in server throughput. It can eliminate more than half the defections compared to the best alternative among

ERMT and FB, and more than 70% compared to the other. WAHS also outperforms FB in the average customer waiting time. ERMT can lead to shorter waiting times, but it does not provide time-of-service guarantees while WAHS provides these guarantees to the majority of customers. (4) SCM is very effective in further reducing the disk I/O bandwidth requirements. The reductions can be up to 18% and 30%, when the cache sizes are 2% and 6% of the total size of videos, respectively.

Consequently, WAHS is the best choice under a wide range of workloads and server resources. SCM is highly recommended with WAHS to reduce further the disk I/O requirements.

CHAPTER 4

SUPPORTING HETEROGENEOUS RECEIVERS

4.1 Introduction

As described earlier, resource sharing techniques include *Batching* [2, 17, 47], *Stream Merging* [9, 10, 18], and *Periodic Broadcasting* [38, 33, 35]. Whereas Batching and stream merging techniques deliver data in a *client-pull* fashion, periodic broadcasting techniques employ the *server-push* approach. Thus, they significantly reduce the server bandwidth requirements but can be used only for the most popular videos, and they require clients to wait until the next broadcast times of the first segments. Moreover, server channels become underutilized when videos are requested infrequently. This chapter (and all remaining chapters) considers the client-pull approach.

Efficient support for a wide spectrum of clients with varying download bandwidth and buffer space is required so as to provide successful video streaming services over the Internet. Measurements of Internet client bandwidth [6] show that the bandwidth varies significantly not only from one technology (e.g., Cable, DSL, and Dial-up) to another, or from one Internet Service Provider (ISP) to another, but also from one client to another within the same ISP. Client physical location and time of connection also impact the actual available bandwidth. Moreover, the available client buffer space varies significantly, considering the wide variety of devices that support video streaming. The variability in the resources and capabilities of clients is referred to as client heterogeneity.

Resource sharing has been studied extensively, but to our knowledge, no study has addressed the client heterogeneity issue in systems employing the client-pull delivery approach. Batching requires each client to have one download channel at the video playback rate, whereas stream merging techniques generally require two channels each at the video playback rate. Hence, if the video playback rate is r (measured in bits per seconds), Batching and stream merging require the client download bandwidth to be at least r and $2r$, respectively. *Bandwidth Skimming* [19], is an exception among

stream merging techniques. It allows client download bandwidth to be lower than double the video playback rate, through special video encoding or complex data delivery, but still assumes receivers homogeneity in terms of download bandwidth. Client heterogeneity has not been addressed except by a few, recent studies, such as HeRo [4], BroadCatch [45], and OHPB [44], which use the server-push approach. However, clients with bandwidth capacities lower than double the video playback rate are likely to face significant startup delay times compared to the video length with all three techniques.

This chapter studies how to support heterogeneous receivers while delivering video streams in a client-pull fashion. We propose three solutions to address the variability of the download bandwidth among clients: *Simple Hybrid Solution* (SHS), *Adaptive Hybrid Solution* (AHS), and *Enhanced Hybrid Solution* (EHS). SHS simply combines Batching with either Patching (SHS-P) or ERMT (SHS-E). It classifies clients into two bandwidth classes. Clients with bandwidth capacities less than double the playback rate are serviced using Batching and the rest are serviced using Patching/ERMT. In contrast, AHS and EHS classify clients into multiple bandwidth classes and service them accordingly. They employ new stream types to service clients with bandwidth capacities ranging between the video playback rate and double the video playback rate. These streams are called *adaptive streams* and *enhanced adaptive streams*, respectively. Whereas the streams in Batching, Patching, and ERMT are all delivered at the video playback rate, adaptive streams are delivered at variable rates that are lower than the playback rate. Similarly, enhanced adaptive streams are delivered at variable rates but in two transmission phases. They are delivered initially at rates higher than the playback rate and subsequently at rates lower than the playback rate. The specific rates at which these two stream types are delivered depend on the corresponding client bandwidth capacities. AHS and EHS employ adaptive streams or enhanced adaptive streams in conjunction with Batching and Patching or ERMT, leading to four possible schemes: AHS-P, AHS-E, EHS-P, and EHS-E. We also study how to address the variations in client bandwidth during a session. In addition, we study the support for the variability in the available buffer space among clients. Furthermore, we study how

the waiting playback requests for different videos can be scheduled for service in the heterogeneous environment, capturing the variations in client bandwidth and buffer space.

We evaluate the effectiveness of the proposed schemes and analyze various scheduling policies through extensive simulation. We study the impacts of many parameters, including client bandwidth and buffer space distributions, server capacity (i.e., server upload bandwidth), request arrival rate, customer waiting tolerance, number of videos, and video length. We consider two service models: *True Video-on-Demand* (TVOD) and *Near Video-on-Demand* (NVOD). In TVOD, we compare the server bandwidth required to service all requests immediately. In NVOD, however, we fix server resources and consider primarily five performance metrics: customer defection (i.e., turn-away) probability, average waiting time, unfairness against unpopular videos, unfairness against clients with low bandwidth, and unfairness against clients with low buffer space.

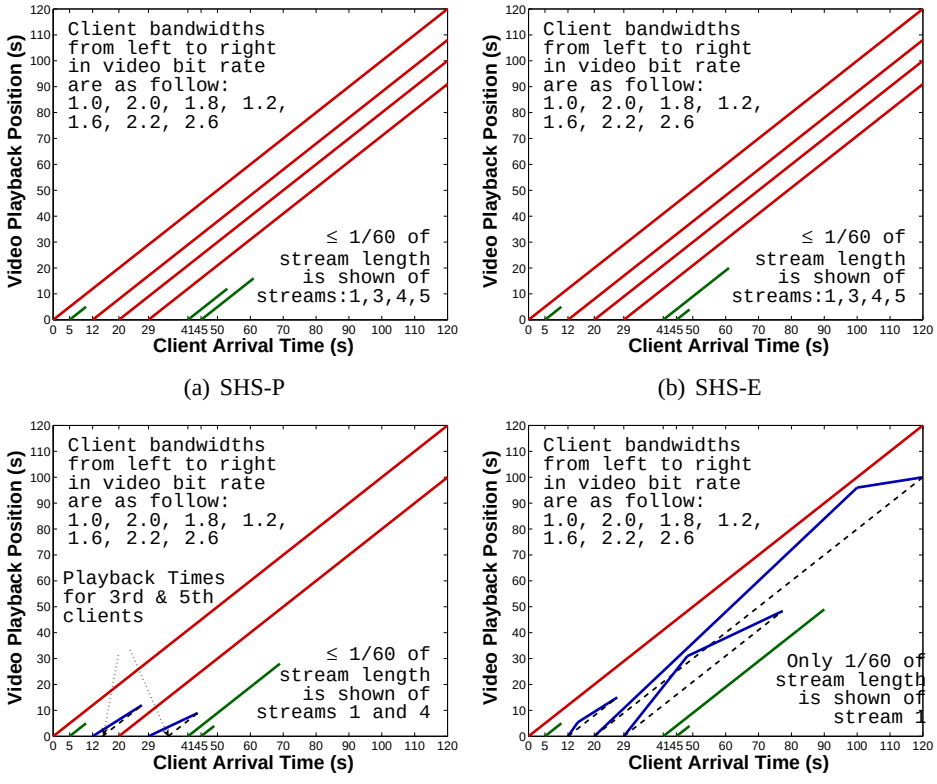
The rest of this chapter is organized as follows. Sections 4.2 and 4.3 discuss the proposed support for heterogeneous client download bandwidth and available buffer space, respectively. Subsequently, Section 4.4 discusses request scheduling in the heterogeneous environment. Finally, Section 4.5 discusses the performance evaluation methodology and main results.

4.2 Supporting Variations in Download Bandwidth among Clients

We discuss next three proposed solutions for supporting variations in download bandwidth among clients. Subsequently, we discuss how to control optimally regular streams in the proposed schemes and how to handle variations in the client download bandwidth during the session's lifetime.

4.2.1 Simple Hybrid Solution (SHS)

Batching and all existing stream merging techniques treat clients as if they all have the same download bandwidth. Batching requires the client download bandwidth to be at least r, whereas stream merging techniques generally require at least $2r$. A simple and straightforward solution to support heterogeneous clients is to combine different resource sharing techniques. Hence, the proposed *Simple Hybrid Solution* (SHS) classifies clients into two bandwidth classes: clients with



(a) SHS-P (b) SHS-E

(c) AHS-E (d) EHS-E

Figure 4.1: Illustrations of Proposed Hybrid Solutions

bandwidth equal to or higher than r but lower than $2r$, and clients with bandwidth capacities equal to or higher than $2r$. SHS services the clients in the first class using Batching and the clients in the second class with Patching or ERMT. This leads to two alternatives: *SHS-P* (when Patching is used) and *SHS-E* (when ERMT is used). ERMT is the most efficient and Patching is the simplest among all stream merging techniques that require client bandwidth of $2r$, as shown in Chapter 3.

Figures 4.1(a) and 4.1(b) further explain how SHS-P and SHS-E work. Clients (or group of clients) with bandwidth capacities of $2r$ or greater (2^{nd}, 6^{th}, and 7^{th} in the figures) are serviced with Patching or ERMT, while those with lower bandwidth (1^{st}, 3^{rd}, 4^{th}, and 5^{th}) are serviced using Batching. Batching streams eliminate the need for some regular streams in Patching and reduces the average stream lengths in both Patching and ERMT, which results in a lower cost of service than a solution using two separate systems, each supporting a different class of clients.

4.2.2 Adaptive Hybrid Solution (AHS)

As discussed earlier, SHS classifies clients into two bandwidth classes: clients with bandwidth equal to or higher than r but lower than $2r$, and clients with bandwidth capacities equal to or higher than $2r$. Hence, it does not capture important opportunities for resource sharing within the first class because all clients in that class are dealt with in the same manner. For the second class, utilizing higher bandwidth than double the playback rate (i.e., $2r$) is not expected to lead to worthwhile performance benefits. The results in [20] show that utilizing more than two channels (each at the playback rate) at each client leads to only low additional performance benefits when optimal stream merging (or ERMT) is used. Figure 4.2 shows the lower bound for server bandwidth requirement in servicing a single video in a TVOD fashion to homogeneous clients using stream merging. In the heterogeneous environment, not every client has more than $2r$ in bandwidth and thus the potential gain is even less significant.

The question now arises as to how to utilize the extra client bandwidth in the first class of clients (with bandwidth capacities equal to or higher than r but lower than $2r$). This can be achieved by using streams with bit rates lower than the video playback rate and setting their rates based on

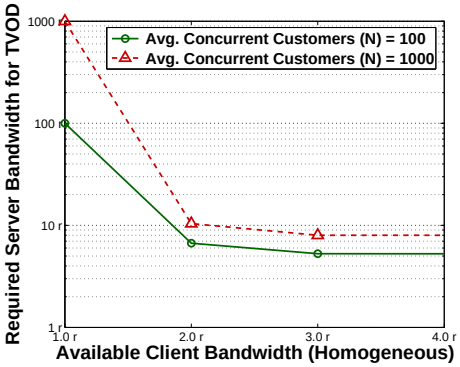


Figure 4.2: Lower Bound of Server Bandwidth Requirement [TVOD, Single Video]

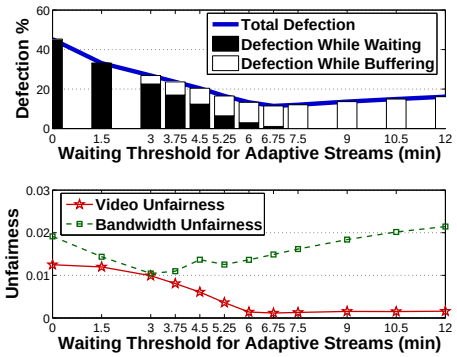


Figure 4.3: Effect of AHS Waiting Threshold [MQL, Server Capacity = $1000r$]

the corresponding clients. Thus, we propose here the idea of *Adaptive Stream Merging*. A new stream type, called *adaptive stream*, or succinctly *a-stream*, is introduced to service those clients with bandwidth higher than r but lower than $2r$. The client uses r of bandwidth to listen to the latest batching stream for the video and uses its remaining bandwidth to receive the missed portion as an a-stream at a slower rate than the video playback. The a-stream will be multicast when more than one request is waiting for the video. This stream replaces the costly batching stream used for such a client or a group of clients in the previous solution. Adaptive streams are adaptive to both the client available resources and server load. To achieve the adaptability to client resources, the delivery rate of an a-stream is determined based on the client with the lowest bandwidth in the group that has been selected for service. The adaptability to server load is achieved by triggering a-streams only when global performance optimization will be attained.

Let us now discuss in more detail the adaptability to server load. Obviously, since the missed data are received at a slower rate than the video playback, clients need to buffer data for some time before the playback. The additional waiting time due to buffering, T_{buf}, depends on the size of data to be delivered and the delivery rate. Let us assume that a group of clients are admitted to the system P seconds after the last batching stream, the lowest bandwidth in the group is b, $r < b < 2r$, and

$B = \frac{b}{r}$. Then, T_{buf} is given by

$$T_{buf} = (\frac{2 - B}{B - 1}).P. \tag{4.1}$$

Since T_{buf} is predictable, clients will be encouraged to wait by informing them with the expected time of playback (which is roughly equal to T_{buf}).

The required buffering has two conflicting effects. In particular, it leads to increasing data sharing among streams and thus servicing more clients and allowing resources to become available sooner for servicing new requests. It, however, increases the waiting time for some clients and may in turn cause them to defect. The overall server performance in terms of client defection rate and average waiting time depends on the combined effect. Hence, Adaptive Stream Merging triggers an a-stream only when the longest total client waiting time (including the required buffering time) will not exceed a certain *threshold*, which is equal to the mean client waiting tolerance times a certain factor. This factor takes only positive real values and varies dynamically based on the variation in client defection rate so as to achieve the lowest possible defection rate. If the triggering condition is not met, a batching stream is delivered instead. As will be shown later, the dynamic approach reduces not only the defection probability but also the average waiting time. Figure 4.3 further illustrates the impact of the waiting threshold on system performance under the default workload discussed in Section 4.5 and summarized in Table 4.1. The available server capacity is assumed to be $1000r$.

To further reduce the additional waiting time caused by buffering, the clients perform "early snooping" on batching streams. Basically, each client starts snooping on (i.e., listening to, or joining) the latest batching stream for the video as soon as the client issues the request as opposed to waiting until admission for service. In this case, P in Equation 4.1 becomes the individual client arrival time rather than the time of admission for service.

The proposed *Adaptive Hybrid solution* (AHS) applies Adaptive Stream Merging with early snooping for clients with bandwidth between but not equal to r and $2r$. For other clients, it operates like SHS. Specifically, it applies Batching for clients with r bandwidth and Patching or ERMT for

client with bandwidth of $2r$ or higher. This leads to two alternative schemes: AHS-P (when Patching is used) and AHS-E (when ERMT is used).

Figure 4.1(c) further explains how AHS-E works. We assume here that the buffering time cannot exceed 30 seconds. The 3^{rd} and the 5^{th} clients are serviced using a-streams instead of batching streams, which reduces the cost from 28829 seconds in case of SHS-E to 14459 seconds with AHS-E. The buffering times are 3 seconds for the 3^{rd} client and 6 seconds for the 5^{th}. Servicing the 4^{th} client using an a-stream would have resulted in 80 seconds of buffering time, which exceeds the assumed threshold, and thus, it is serviced by Batching.

4.2.3 Enhanced Hybrid Solution (EHS)

The Adaptive Hybrid Solution (AHS) requires additional delay due to buffering time, which may not be suitable for True Video-on-Demand (TVOD). Thus, we propose the concept of *enhanced adaptive streams*, or succinctly *ea-streams*, which are a generalization of the adaptive streams in AHS. Like a-streams, they are used for clients with bandwidth between but not equal to r and $2r$. These two stream types, however, vary significantly. The basic idea of ea-streams can be explained as follows. Initially, the server delivers the requested video at a rate higher than the playback rate, using the full client bandwidth. Then, the client starts to listen to the latest regular stream while using the remaining bandwidth to receive a slow patch stream at a rate lower than the playback rate. Therefore, an ea-stream has two phases: fast transmission at a rate higher than playback rate, followed by a slow transmission at a rate lower than the playback rate. These two phases take X and Y seconds, respectively. The initial transmission of the video at a higher rate than the playback rate eliminates the need for any extra delay in addition to the waiting time for server resource to become available and the buffering time for smoothing out the variation in the end-to-end packet delay (delay jitter).

Figure 4.4(a) further clarifies the concept of ea-streams. In this example, a client with bandwidth b, where $r < b < 2r$ (i.e., $B = \frac{b}{r}$), arrived P seconds after the latest regular stream (the first stream in the figure). The second stream is the ea-stream. The dashed line depicts the actual playback

line. The sign @ denotes the transmission rate in the unit of the playback rate. X and Y must be controlled so as to achieve zero delay caused by buffering (i.e., $T_{buf} = 0$).

The question now arises as to how to provide the client with a continuous playback with minimum cost and without any additional waiting time due to buffering (i.e., $T_{buf} = 0$). Basically, two conditions must be satisfied. First, the last playback position (frame) $P + X$ in the ea-stream must arrive at the scheduled time of playback. Specifically, it is required for playback exactly after $P+X$ seconds of the playback start time. This time must be equal to the required time to receive that point of the video: $X + Y$ seconds. Second, the data that is delivered by the ea-stream must be equal to the missed data from the last regular stream, which is equal to $P + X$. These two conditions translate to the following two equations, respectively:

$$P + X = X + Y \tag{4.2}$$

and

$$X.B + Y.(B - 1) = X + P. \tag{4.3}$$

It follows from these two equations that

$$Y = P \tag{4.4}$$

and

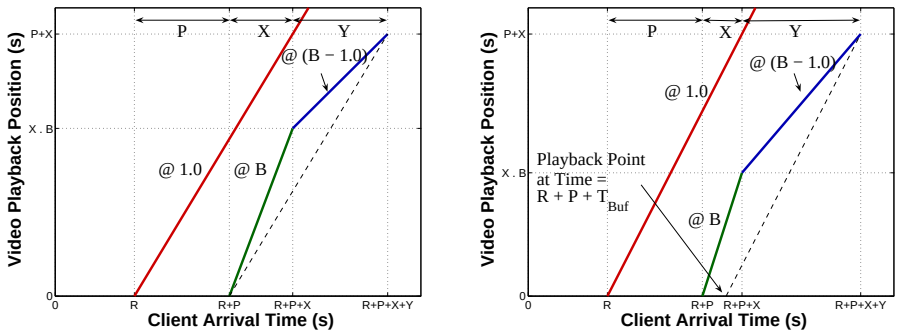
$$X = \frac{(2 - B)}{(B - 1)}P. \tag{4.5}$$

Hence, the cost for delivering the ea-stream in seconds of video data is determined by

$$Cost = P + X = \frac{P}{B - 1}. \tag{4.6}$$

Equation 4.6 is valid only if the cost is lower than the video length. Otherwise, a new regular stream must be delivered instead.

We also propose a generalized version of ea-streams that allows for some *controlled* buffering time (T_{buf}) to reduce the size of the delivered data. This controlled buffering time may be suitable



(a) With No Controlled Buffering Time ($T_{max} = 0$)

(b) With Controlled Buffering Time ($T_{max} > 0$.)

Figure 4.4: Illustration of Enhanced Adaptive Streams (ea-streams)

for a Near Video-on-Demand (NVOD) service. Let us now assume that T_{max} is the targeted maximum client waiting time, and a client has already waited T_{org} seconds before being admitted for service. Subsequently, $T_{buf_{max}} = max(T_{max} - T_{org}, 0)$. In the generalized version, if an a-stream can achieve a smaller T_{buf} than $T_{buf_{max}}$, then it is used instead of an ea-stream. Thus, $X = 0$, $Y = \frac{P}{B-1}$, and the cost is simply P. Otherwise, an ea-stream is used. In this case, T_{buf} is set to $T_{buf_{max}}$ and the following two equations must be satisfied, as illustrated in Figure 4.4(b):

$$X + Y - T_{buf} = X + P \tag{4.7}$$

and

$$X.B + Y.(B - 1) = X + P. \tag{4.8}$$

Subsequently, Y and X can be found as follows:

$$Y = P + T_{buf} \tag{4.9}$$

and

$$X = \frac{(2 - B)}{(B - 1)}.P - T_{buf}. \tag{4.10}$$

Finally, the cost for the ea-stream can be found by

$$Cost = P + X = \frac{P}{B-1} - T_{buf}. \tag{4.11}$$

Again, the last equation is valid only if the cost is lower than the video length. Otherwise, a new regular stream must be delivered instead.

The use of ea-streams is combined in a hybrid solution with Batching and Patching or ERMT to support clients with different bandwidth classes. We refer to this solution as *Enhanced Hybrid Solution* (EHS). It has two variants: EHS-P and EHS-E, depending on whether Patching or ERMT is used for clients with $2r$ or higher bandwidth. Figure 4.1(d) shows the prior example with EHS-E. The cost here is reduced to 7421.34 seconds of video data.

4.2.4 Optimal Control of Regular Streams

Equations 4.6 and 4.11 highlight the importance of minimizing the average value of time skewness P since the last regular stream. Let us assume that for a certain D-second video, the request arrival rate is λ and the average time between two consecutive regular streams is W_{reg}. Thus, the number of requests arriving per video length is given by $N = \lambda D$ and the number of regular streams per video length is $\frac{D}{W_{reg}}$. When $T_{buf} = 0$, the cost per ea-stream is $\frac{P}{B-1}$, and thus the total cost for servicing a heterogeneous group of clients arriving within a video duration is given by

$$Cost = \sum_{i=1}^{N}(\frac{P_i}{B_i - 1}) + \frac{D}{W_{reg}}.D. \tag{4.12}$$

For simplicity, let us assume a homogeneous group of clients, each with bandwidth B^* (in multiples of the video playback rate). Since the average value of P_i for a long period of time is $\frac{W_{reg}}{2}$, the total cost per video duration is given by

$$Cost = \sum_{i=1}^{N}(\frac{P_i}{B^* - 1}) + \frac{D^2}{W_{reg}} \approx \frac{\lambda D W_{reg}}{2(B^* - 1)} + \frac{D^2}{W_{reg}}. \tag{4.13}$$

$W_{reg_{opt}}$ can be found as follows:

$$\frac{\partial}{\partial W_{reg}}(\frac{\lambda D W_{reg}}{2(B^*-1)} + \frac{D^2}{W_{reg}}) = 0. \quad \Rightarrow \quad W_{reg_{opt}} = \sqrt{\frac{2D(B^*-1)}{\lambda}}. \tag{4.14}$$

By substituting $W_{reg_{opt}}$ in Equation 4.13,

$$Cost_{opt} \approx \frac{D\sqrt{\lambda D}}{\sqrt{2(B^*-1)}} + \frac{D\sqrt{\lambda D}}{\sqrt{2(B^*-1)}}. \tag{4.15}$$

Therefore, when the delivery cost is minimized, the total cost of ea-streams is equal to the total cost of regular streams. This conclusion is valid for both a-streams and ea-streams in both homogeneous and heterogeneous client environments and with both TVOD and NVOD service models. Motivated by this conclusion, the server can simply trigger regular streams dynamically by computing the total cost of a-streams or ea-streams for each video since the last regular stream and initiating a regular stream for servicing the new requests for the video if the computed value exceeds the cost of a regular stream.

4.2.5 Handling Variations in Client Bandwidth over Time

The variation in client bandwidth during the service time makes streaming challenging, especially when resource sharing is employed. Thus, the server should be conservative in estimating the client bandwidth to avoid some pauses in the playback. Compared with SHS and AHS, EHS deals much better with this problem because of the fast delivery period of ea-streams. These streams adapt easily to drops in the average bandwidth below the estimated value during the fast delivery period, as long as the average remains higher than b. Before the end of the fast delivery period, X and Y should be recomputed using the client bandwidth history, and thus the fast delivery period will be extended if necessary. This also helps in having a more accurate estimate for the more sensitive, slow delivery period. X can be computed conservatively to deal with further unexpected drops in the average bandwidth during the slow delivery period. To cover for a maximum drop of δ_B in

average bandwidth, where $0 \leq \delta_B \leq B-1$, the new fast delivery period $\overline{X}$ is calculated using

$$\overline{X} = X + \frac{\delta_B}{B-1}.(P + T_{Buf}). \tag{4.16}$$

The additional cost is $\frac{\delta_B}{B-1}(P+T_{Buf})$. Consequently, the new value of Y ranges from 0 to $P+T_{Buf}$, depending on both δ_B and the actual drop in bandwidth. If the average bandwidth drops beyond δ_B during the slow delivery period, the server can switch the service back to the fast delivery mode (when the server resources become available for the increased delivery rate). If the client bandwidth is significantly unstable, the fast delivery is recommended to continue until the merge with a regular stream. A drop in bandwidth below b for batching clients (or any other clients) cannot be solved unless a layered video coding is used and the video quality is reduced for some time to fit the client available bandwidth.

4.3 Dealing with Variations in Available Buffer Space among Clients

The variation in the buffer space among clients is another concern, especially with the wide variety of devices capable of streaming videos. For all stream merging techniques, including those proposed in this paper, the required client buffer space is $P+T_{buf}$. Recall that P is time skew of the request arrival from the last regular stream and that T_{buf} is the buffering time. As discussed earlier, $T_{buf} = 0$ for all stream types other than a-streams and ea-streams with controlled buffering time (i.e., $T_{max} > 0$).

Buffer space comes next in importance after the download bandwidth as long as some buffer space is available in each client. This is due to the following two reasons. First, the download bandwidth is generally the bottleneck and is decided by more than the receiving device. Second, while scalable resource sharing techniques (such as Patching and ERMT) require strict download bandwidth at almost every client, they are more tolerant of smaller buffer spaces. For example, using Patching, a client with bandwidth *slightly* less than the required $2r$ can only be serviced using regular streams. Therefore, such a client will either defect or wait up to W_{reg} seconds for the next regular stream ($W_{reg}/2$ seconds on the average). Alternatively, the server needs to initiate regular

streams earlier, but in this case, Patching may effectively lose its advantage over Batching if the clients with such bandwidth capacities are common. The situation is different when it comes to buffer space. In particular, a client with buffer space of S seconds of data that is slightly less than the required buffer space of W_{reg} seconds of data can be serviced as long as its request arrives within S seconds from the last regular stream. Otherwise, it must wait up to ($W_{reg} - S$) seconds for the next regular stream, or the next regular stream must be initiated earlier.

In SHS, AHS, and EHS, the expected waiting time is usually smaller than in Patching because regular streams (batching streams) will be delivered as well to support clients in the lowest bandwidth class, which results in a smaller effective W_{reg}. The option of initiating a regular stream earlier is generally preferred over forcing clients to wait, given that the server resources permit such an earlier service, but this leads to additional complications in request scheduling. In this paper, we use this preferred option.

Handling the variations in buffer space is relatively easy to implement in the proposed hybrid resource sharing solutions but they along with the variations in download bandwidth lead to significant complications in request scheduling. In the next section, we discuss how request scheduling can be performed and implemented efficiently in the heterogeneous client environment.

4.4 Request Scheduling for Heterogeneous-Client Environment

Supporting client heterogeneity complicates the scheduling issue. In this case, a video waiting queue may contain requests for clients varying in download bandwidth and buffer space. Therefore, there are different subgroups of requests that can be serviced concurrently. For example, assume that three requests R1, R2, and R3 for video v are waiting for service and that the corresponding clients have r, $1.5r$, and $2r$ in download bandwidth, respectively. Assuming sufficient buffer space in all three, the server has three service options for video v: *(i)* servicing all three requests by Batching, *(ii)* servicing only R2 and R3 by a-streams or ea-streams, and *(iii)* servicing only R3 by Patching or ERMT. Thus, a scheduling policy in the heterogeneous environment has to select not only a specific video among all videos that have waiting requests but also the specific subset of requests in that video.

To implement scheduling in the heterogeneous environment, we introduce the concept of a *virtual queue*. A virtual queue is a subgroup of the waiting requests for a certain video. In particular, virtual queue $q_{v,i,j}$ has all requests for video v by clients with bandwidth classes $\geq i$ and buffer space classes $\geq j$. A bandwidth class is a group of clients with bandwidth capacities within a specified range. Similarly, a buffer space class is a group of clients with buffer space within a specified range. For example, assuming a bandwidth class width of $0.05r$, *Class 0* is for clients with bandwidth within $[r, 1.05r[$, *Class 1* for clients with bandwidth within $[1.05r, 1.10r[$, and so on. The same applies for buffer space classes. Assuming a buffer space class width of 30 seconds, *Class 0* is for clients with buffer space within $[0, 30[$, *Class 1* for clients with buffer space within $[30, 60[$, and so on.

We propose to perform scheduling by dividing the requests in each video queue into virtual queues. Instead of selecting a video queue as in the homogeneous case, a scheduling policy selects a virtual queue in the set of all possible virtual queues (among all videos).

Existing scheduling policies can be used in the heterogeneous environment but with possible modifications to incorporate the virtual queue concept. Because of their nature, MQL and FCFS will still act the same as in the homogeneous case. In particular, they will always select for service *all* requests in a particular video queue and will not utilize the variations in client bandwidth and buffer space. The selected virtual queue is basically the the one that contains all waiting requests in the video queue because it is longest (in the case of MQL) or the one containing the oldest request (in the case of FCFS). MFQL and MCF, however, exploit these variations by operating at the virtual queue level, and thus not all waiting requests for a video are necessarily selected. Our implementation of MFQL for the heterogeneous environment divides the virtual queue length by the square root of the relative access frequency of clients with similar or lower bandwidth and buffer space classes requesting the same video. Hence, it tries to reduce the bias against both lower classes and unpopular videos, in the same way the homogeneous MFQL tries to do for unpopular videos.

4.5 Performance Evaluation Methodology and Results

We have developed a simulator for a video streaming server that supports various resource sharing techniques and scheduling policies. The simulator has been validated by reproducing several graphs in several previous studies and by checking results against those predicted by analytical models when possible. The simulation stops after a steady state analysis with 95% confidence interval is reached.

We analyze the effectiveness of the proposed schemes and analyze various scheduling policies through extensive simulation. We consider two service models: TVOD and NVOD. In NVOD, we fix the server bandwidth and consider primarily five performance metrics: *customer defection probability*, *average waiting time*, *video unfairness*, *client-bandwidth unfairness*, and *buffer-space unfairness*. The defection probability is the most important and can be defined as the probability that customers leave the system without being serviced because of waiting times exceeding their tolerance. The average waiting time comes next in importance. It includes the client waiting time for admission and any possible buffering time. The unfairness metrics quantify the bias against unpopular videos, clients with lower available download bandwidth, and clients with smaller available buffer space, respectively. They can be found as the standard deviation in the defection rate among various videos, bandwidth classes, or buffer space classes, respectively. The used class widths are shown on Table 4.1. In TVOD, we compare the server bandwidth required to service all requests immediately.

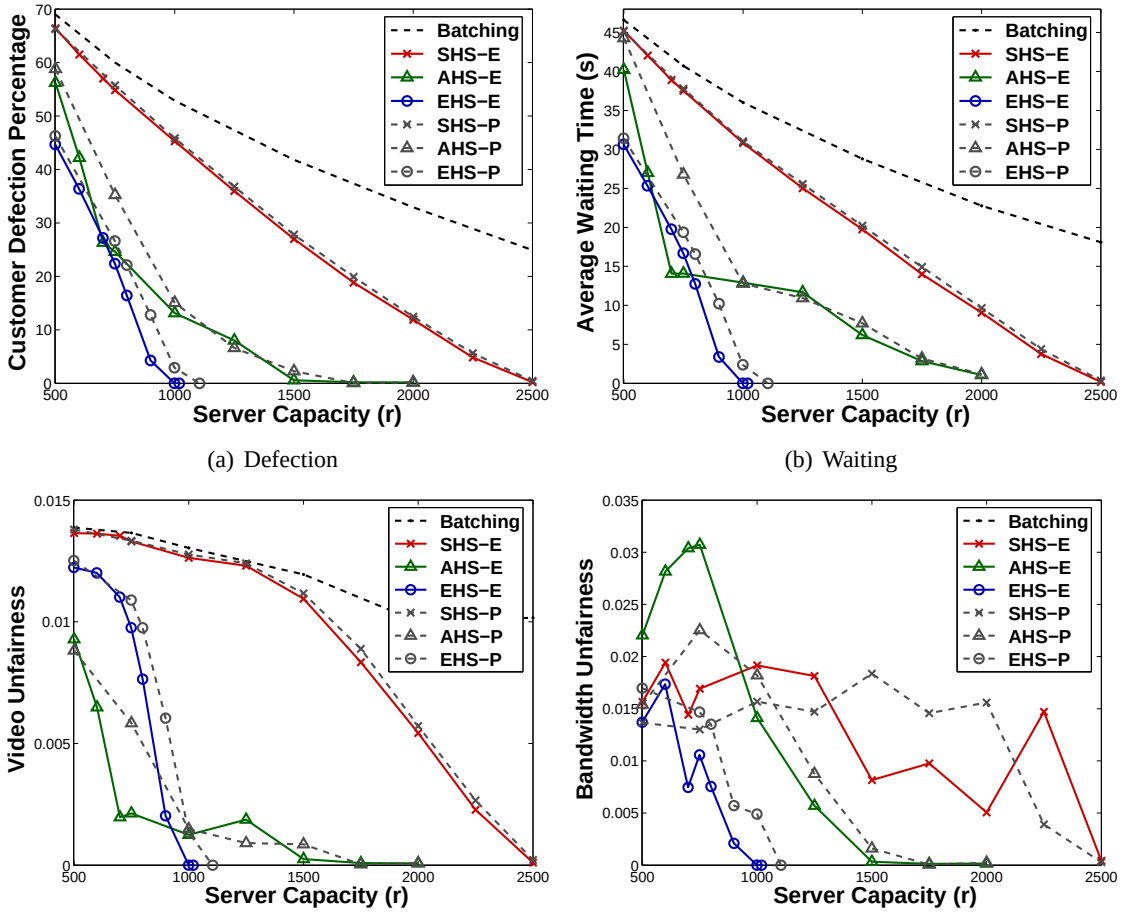
4.5.1 Workload Characteristics

Like most prior studies, we assume that the arrival of the requests follows a Poisson Process with an average arrival rate λ and the accesses to videos are highly localized and follow a Zipf-like distribution with skewness parameter $\theta_v = 0.271$. Generally, we characterize the waiting tolerance of customers by an exponential distribution with mean μ_{tol}. However, with AHS, we utilize the *time-of-service guarantee* (TSG) model [3, 42, 47] whenever the actual playback time is known and guaranteed. With this model, if the current waiting time plus the additional buffering time is less

than μ_{tol}, the customer waits; otherwise, the waiting tolerance follows an exponential distribution with mean μ_{tol}. The default value of μ_{tol} is 1.5 minutes. EHS is implemented without requiring any additional waiting time due to buffering (i.e., $T_{buf} = 0$) and thus TSG is not used. Unless otherwise indicated, each client is assumed to have 60 minutes of buffer space to ensure high performance.

Table 4.1 summarizes the main workload characteristics and shows the default values. Unless otherwise indicated, we consider a server with 120 videos, each of which is 120-minute long. We examine the server at different loads by fixing the request arrival rate at 40 requests per minute and varying the server bandwidth (server capacity) generally from $300r$ to $3000r$. Since interactive operations can be supported using contingency channels [16], the relative performance of various techniques and policies does not depend on these operations as long as the fraction of server bandwidth used for these operations is kept the same (which is typically the case). To isolate the impact of these operations, the reported server capacity in this study includes only non-contingency bandwidth.

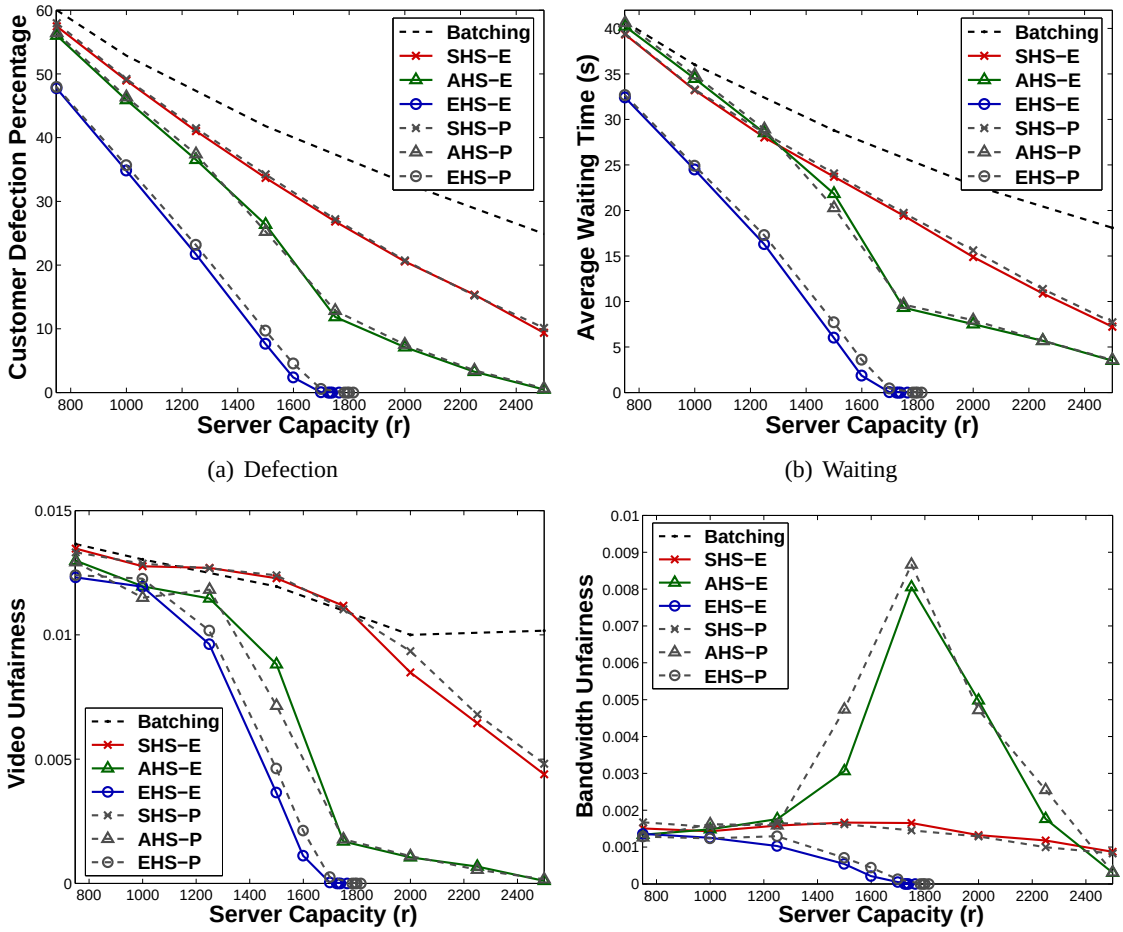
Because of the lack of a workload characterization study of client bandwidth, we generally use five bandwidth distributions: two Zipf-like distributions with skewness parameters θ_B = 0.0 and 0.5, and three truncated Normal distributions with mean values μ_B = 1.5, 2.0, and 2.5 and a standard deviation of 0.5, all in the unit of the video playback rate r. The first Zipf distribution represents the case when there is a high ratio of clients with bandwidth in the region just above r, and the others cover other possible scenarios. The client bandwidth capacities range between r and $11r$ in all five distributions. Similarly, we characterize the buffer space distribution by six distributions: Fixed 60 minutes for all clients, Uniform, Zipf (with parameter $\theta_s = 0$), and three Normal distributions with μ_S = 10, 30, and 60 minutes of video. The standard deviation of the Normal distributions is 15 minutes.



(a) Defection (b) Waiting

(c) Video Unfairness (d) Bandwidth Unfairness

Figure 4.5: Comparing the Performance of Different Resource Sharing Schemes Supporting Client Bandwidth Heterogeneity [Normal Bandwidth Dist. with $\mu_B = 2.0r$, MQL]



(a) Defection (b) Waiting

(c) Video Unfairness (d) Bandwidth Unfairness

Figure 4.6: Comparing the Performance of Different Resource Sharing Schemes Supporting Client Bandwidth Heterogeneity [Zipf Bandwidth Dist. with $\theta_B = 0.0$, MQL]

Table 4.1: Summary of Workload Characteristics

Parameter	**Model/Value(s)**
Request Arrival Model	Poisson Process
Request Arrival Rate (λ)	20 to 360, default: **40** Requests/min.
Server Capacity	$300r$ to $3000r$
Video Access	Zipf-Like ($\theta_v = 0.271$)
Number of Videos	60 to 1920, default: **120**
Video Length (D)	5 to 180, default: **120** min.
Waiting Tolerance Model	Exponential with mean μ_{tol} Time-of-Service Guarantee with threshold μ_{tol}
Waiting Tolerance Mean (μ_{tol})	0.5, **1.5** (default), 2.5 min
Targeted maximum client waiting (T_{max})	**0.0** (default), 1.5, 3.0 min
Client Bandwidth Model	Zipf-Like with θ_B = 0 and 0.5 Normal with mean μ_B = $1.5r$, **2.0r** (default), and $2.5r$, $\sigma_B = 0.5r$
Client Bandwidth Range	r to $11r$
Bandwidth class width	$0.05r$
Client Buffer Space Model	Fixed **60** min (default) Uniform Zipf (with $\theta_s = 0$) Normal with mean μ_S = 10, 30, and 60 min, $\sigma_S = 15$ min
Buffer Space Range	5 to 120 min
Buffer space class width	10 sec

4.5.2 Result Presentation and Analysis

Comparing the Performance of Resource Sharing Schemes

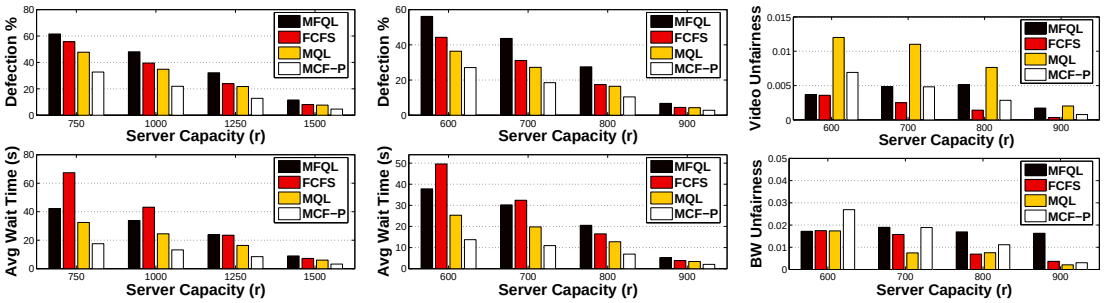
Let us start by comparing various resource sharing schemes that work for heterogeneous receivers (Batching, SHS-P, SHS-E, AHS-P, AHS-E, EHS-P, and EHS-E) in terms of defection rate, average waiting time, and video and bandwidth class unfairness. Figures 4.5 and 4.6 depict the results under Normal and Zipf-Like bandwidth distributions, respectively. At this stage, scheduling is performed using MQL, and no controlled buffering time is done with EHS (i.e., $T_{max} = 0$).

The results demonstrate the advantage of utilizing ea-streams in EHS and a-streams in AHS, with a noticeable lead for the former, regardless of the workload and bandwidth distribution. For example, under the Normal distribution, while Batching requires $4800r$ in server bandwidth to achieve TVOD (i.e., 0 defections and 0 waiting time), SHS-E requires $2500r$, and EHS-E requires less than

$1000r$. AHS-E achieves 0 defections with $1700r$ server bandwidth, but it can never completely eliminate the waiting times due to the required, initial buffering time. Moreover, with $1000r$ server bandwidth, when EHS-E achieves TVOD, AHS-E, SHS-E, and Batching cause defections of approximately 13%, 45%, and 53%, respectively. Of course, when it comes to the less important metrics, video and bandwidth class unfairness in particular, the results are slightly different. Batching treats all clients as homogeneous receivers, but among the other three solutions, EHS is the fairest towards lower bandwidth classes. For video unfairness, EHS and AHS are the fairest among all schemes (including Batching) towards unpopular videos. The better fairness of AHS in some cases towards unpopular videos is because the defections here are of two types: defections caused by waiting times till admission and defections caused by waiting times after admission due to buffering. The unfair nature of MQL makes the first type dominated by unpopular video clients and the second dominated by popular video clients with low bandwidth. This in a way balances the gap in defection rates among videos.

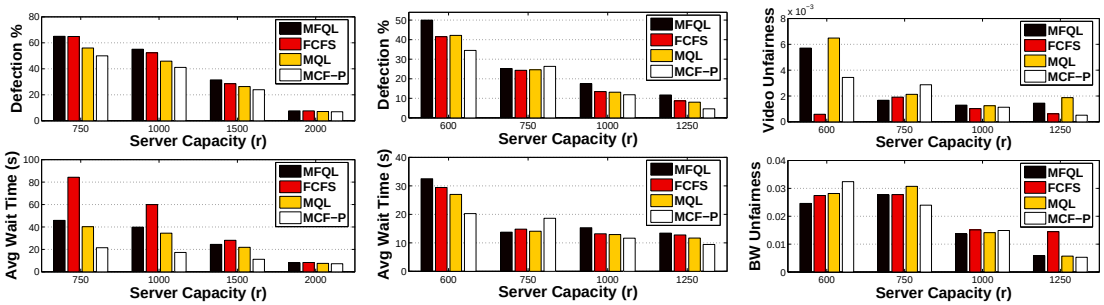
There are two interesting observations. (1) The three proposed solutions vary significantly in performance, whereas the two variants of each perform close to one other. In other words, using ERMT instead of Patching for serving clients with bandwidth capacities of $2r$ or higher is of less significance than changing the solution (and thus the method of service for clients with bandwidth between r and $2r$, noninclusive). (2) Although some additional buffering time is imposed by a-streams in AHS, a significant gain is achieved even in the average waiting time, compared to SHS and Batching.

From this point on, only the ERMT variants of the three solutions are considered: EHS-E, AHS-E, and SHS-E. EHS-P, AHS-P, and SHS-P are dropped because they perform close to, but slightly worse than EHS-E, AHS-E, and SHS-E, respectively. Batching is dropped because of its poor performance.



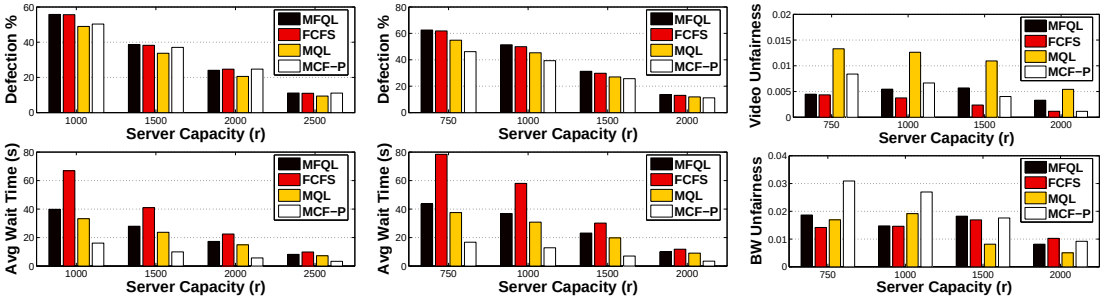
(a) Zipf ($\theta_B = 0.0$) (b) Normal ($\mu_B = 2.0r$) (c) Normal ($\mu_B = 2.0r$)

Figure 4.7: Comparing the Performance of Scheduling Policies for EHS-E



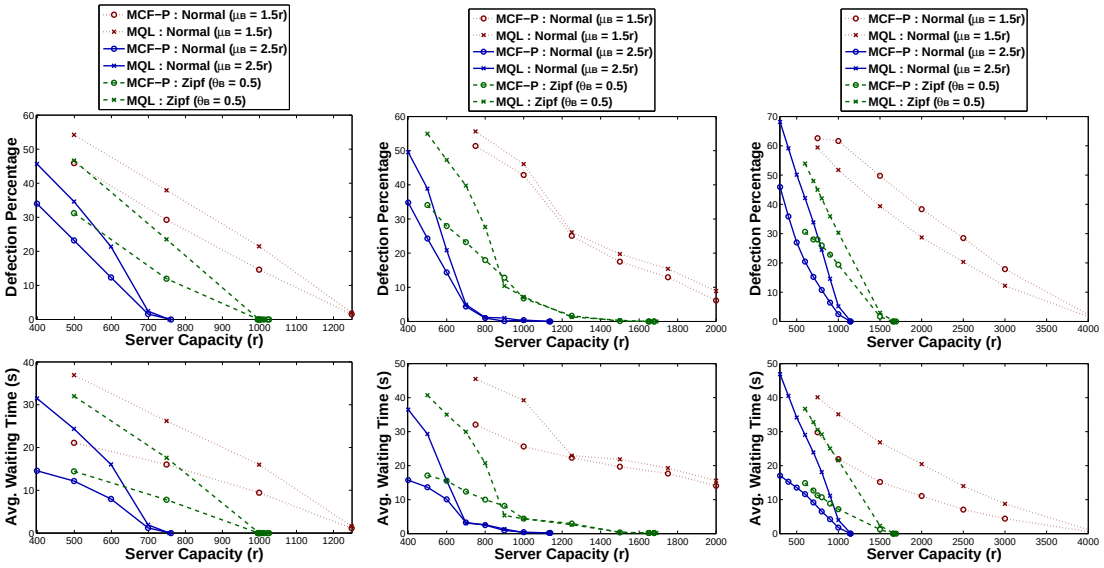
(a) Zipf ($\theta_B = 0.0$) (b) Normal ($\mu_B = 2.0r$) (c) Normal ($\mu_B = 2.0r$)

Figure 4.8: Comparing Scheduling Policies for AHS-E



(a) Zipf ($\theta_B = 0.0$) (b) Normal ($\mu_B = 2.0r$) (c) Normal ($\mu_B = 2.0r$)

Figure 4.9: Comparing the Performance of Scheduling Policies for SHS-E



(a) EHS-E (b) AHS-E (c) SHS-E

Figure 4.10: Comparing the Performance of Scheduling Policies with Different Bandwidth Distributions

Comparing the Performance of Scheduling Policies

Let us now discuss the impact of scheduling. Figures 4.7, 4.8, and 4.9 compare scheduling policies when applying EHS-E, AHS-E, and SHS-E, respectively. The results are shown under two bandwidth distributions: Zipf with $\theta_B = 0.0$ and Normal with $\mu_B = 2.0r$. FCFS and MFQL in general perform worse than MCF-P and MQL in terms of defection rate and waiting time. Figure 4.10 compares scheduling policies under the remaining bandwidth distributions but in only the two main metrics (defection rate and waiting time) to reduce the number of figures. To keep the figure less crowded, FCFS and MFQL are excluded.

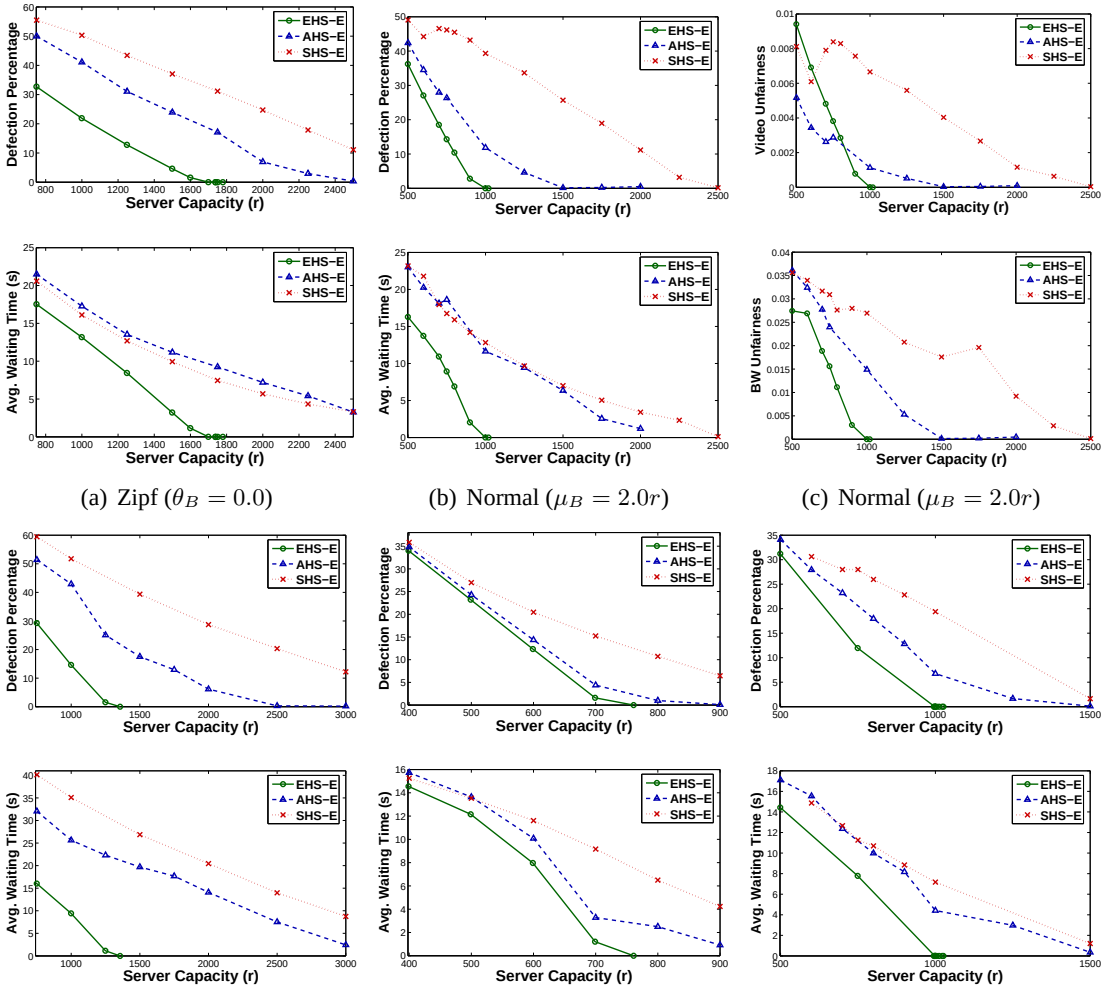
With EHS and AHS, MCF-P achieves the best results in the two most important metrics, compared with all considered policies, regardless of the bandwidth distribution. It also has better video fairness than its closest competitor (MQL). MCF-P, however, does not perform well in bandwidth class unfairness because it schedules requests for service based on their delivery costs, which depend on their available download bandwidth. Overall, MCF-P is generally the best choice with EHS and

AHS. The results are quite different with SHS-E. The best two candidates are still MQL and MCF-P, based on the two most important metrics. While MQL causes fewer defections than MCF-P when the majority of clients have low bandwidth (as in Zipf with $\theta_B = 0$ and Normal with $\mu_B = 1.5r$), MCF-P performs better when clients have higher bandwidth capacities on the average. The high defection rate of MCF-P compared with MQL in the first case is due to its bias against Batching clients, who require the highest cost of service. When SHS is used, Batching clients include every client with bandwidth lower than $2r$. MCF-P, however, performs better than MQL in the average waiting time.

Next, we use the best scheduling policy for each resource sharing scheme. For SHS-E, MQL is used in case of Normal($\mu_B = 1.5r$), and MCF-P is used under the other distributions. For EHS-E and AHS-E, MCF-P is always used. MCF-P is chosen for SHS-E under Zipf($\theta_B = 0.0$) because the small increase in defection rate compared with MQL is justified by a more significant reduction in waiting time.

Comparing the Performance of Resource Sharing Schemes with Best Scheduling

Figure 4.11 compares EHS-E, AHS-E, and SHS-E under five different bandwidth distributions. Only the two most important metrics are analyzed here to reduce the number of figures. As expected, the performance gap increases when the distribution offers more clients that can benefit from ea-streams or a-streams. For example, when the average client bandwidth is $2.5r$, the majority of clients do have more than the $2r$ required by ERMT, while when the average client bandwidth is $1.5r$, the majority can be serviced by ea-streams or a-streams. However, EHS-E keeps its lead (followed by AHS-E) under all bandwidth distributions. In practical systems, the average client bandwidth is not expected to be very large compared to the video playback rate (r) because having a large portion of clients with very high bandwidth usually motivates a higher video quality.



(a) Zipf ($\theta_B = 0.0$) (b) Normal ($\mu_B = 2.0r$) (c) Normal ($\mu_B = 2.0r$)

(d) Normal ($\mu_B = 1.5r$) (e) Normal ($\mu_B = 2.5r$) (f) Zipf ($\theta_B = 0.5$)

Figure 4.11: Comparing the Performance of the Different Hybrid Solutions with Best Scheduling

Impact of T_{max} with EHS-E

As discussed in Section 4.2, EHS-E can be implemented with some additional buffering time (T_{buf}). Figure 4.12 shows the results for three values of targeted maximum client waiting time (T_{max}): 0.0, 1.5, and 3.0 minutes. Although a small buffering time might slightly increase server throughput, it has a much larger negative impact on the waiting time. Large buffering times, however, negatively impact both throughput and waiting time. Thus, it is preferred not to introduce any controlled waiting time with EHS and to always target 0.0 waiting time. Interestingly, the average waiting time is smaller for $T_{max} = 3.0$ minutes compared to $T_{max} = 1.5$ minute. This is because only serviced customers are considered in the average waiting time. Controlled buffering time in the case of ea-streams affects every client almost equally (unlike a-streams). Therefore, a relatively large buffering time, compared to the average waiting tolerance, can result in the defection of the majority of clients serviced by ea-streams.

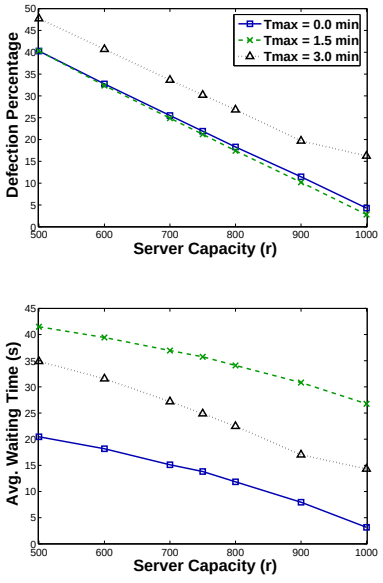
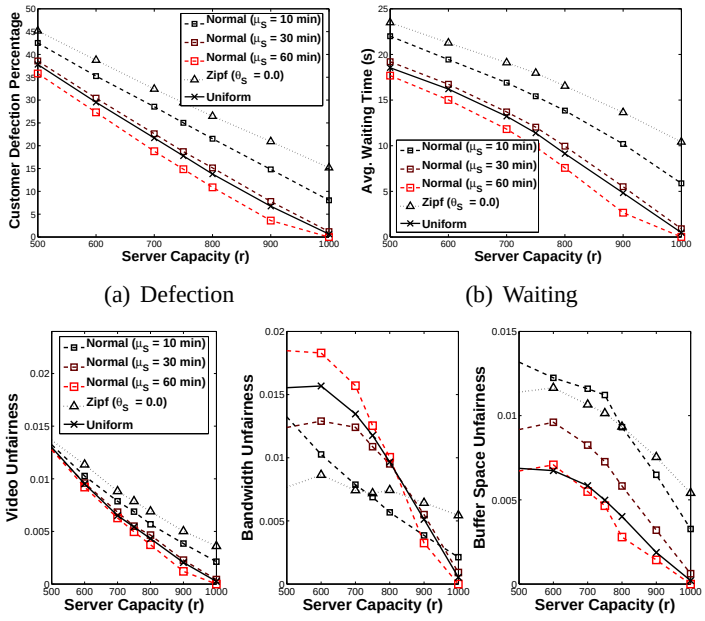


Figure 4.12: Impact of T_{max} with EHS-E [Normal Bandwidth Dist. with $\mu_B = 2.0r$, Normal Buffer Space Dist. with $\mu_S = 20$ min, MCF-P]



(a) Defection (b) Waiting

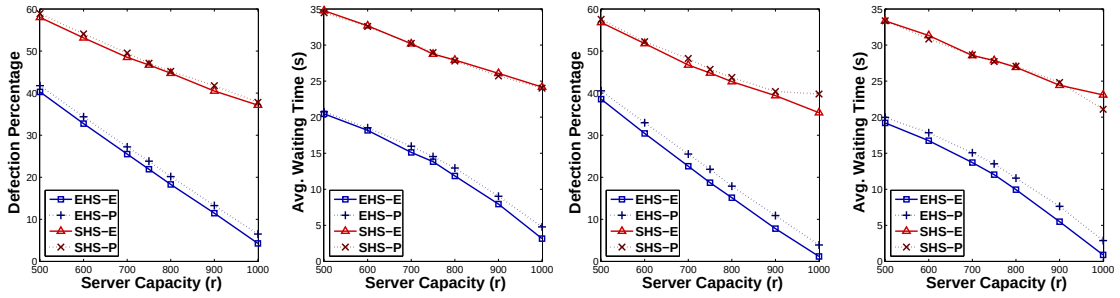
(c) Unfairness

Figure 4.13: Impact of Buffer Space Distribution on EHS-E [Normal Bandwidth Dist. with $\mu_B = 2.0r$, MCF-P]

Impact of Available Client Buffer Space

Let us now study the impact of available client buffer space on the relative performance of the proposed resource sharing schemes. As demonstrated in Figure 4.14, limited buffer space and its diversity do not change the relative performance of various schemes. It narrows, however, the performance gaps among them, especially between the two alternative implementations of each solution (such as EHS-E and EHS-P).

Let us now examine in more detail the impact of the available client buffer space on the performance of EHS-E under different download bandwidth distributions. As expected, Figure 4.13 demonstrates that system performance increases with the available buffer space, but the change is not dramatic even with a significant change in the available buffer space. For example, increasing the average available buffer space by a factor of 6 (under Normal Distribution and Server Capacity of $500r$) reduces the defection probability by only 16% from 43% to 36%, and the average waiting time by only 18% from 22 to 18 seconds. As expected, video unfairness is not very sensitive to buffer space availability. The impact on bandwidth class unfairness is moderate and is inversely proportional to the buffer space. This is due to forcing more regular (batching) streams which indirectly helps the clients with lower bandwidths.



(a) μ_S = 20 min (b) μ_S = 30 min

Figure 4.14: Impact of Buffer Space on Various Schemes [Normal Bandwidth Dist. with $\mu_B = 2.0r$, Normal Buffer Space Dist., MCF-P]

Table 4.2: Comparative Results of the Impacts of Server Capacity, Client Bandwidth, and Client Buffer Space [Normal Bandwidth Dist. with Default $\mu_B = 2.0r$, Normal Buffer Space Dist. with Default $\mu_S = 60$ min, EHS-E, MCF-P]

Input Parameters				Output		
Client Bandwidth Mean (r)	Buffer Space Mean (min)			Server Capacity (r)		
	From	To	%	From	To	%
1.5	10	20	100	1330.9	1294.7	2.72
2.0	10	20	100	1115.9	1049.5	5.95
2.5	10	20	100	996.6	908.7	8.82
Buffer Space Mean (min)	Client Bandwidth Mean (r)			Server Capacity (r)		
	From	To	%	From	To	%
10	1.5	2.0	33.3	1330.9	1115.9	16.15
30	1.5	2.0	33.3	1259.4	992.2	21.22
60	1.5	2.0	33.3	1230	933.3	24.12
Server Capacity (r) [Client Bandwidth Mean = $2.0r$]	Buffer Space Mean (min)			Defection Probability (%)		
	From	To	%	From	To	%
500	10	20	100	42.5	40.3	5.3
750	10	20	100	25.0	21.9	12.4
1000	10	20	100	8.0	4.3	46.9
Server Capacity (r) [Buffer Space Mean = 10 min]	Client Bandwidth Mean (r)			Defection Probability (%)		
	From	To	%	From	To	%
500	1.5	2.0	33.3	45.2	35.8	20.8
750	1.5	2.0	33.3	28.6	14.9	48.1
1000	1.5	2.0	33.3	13.4	0.0	100.0

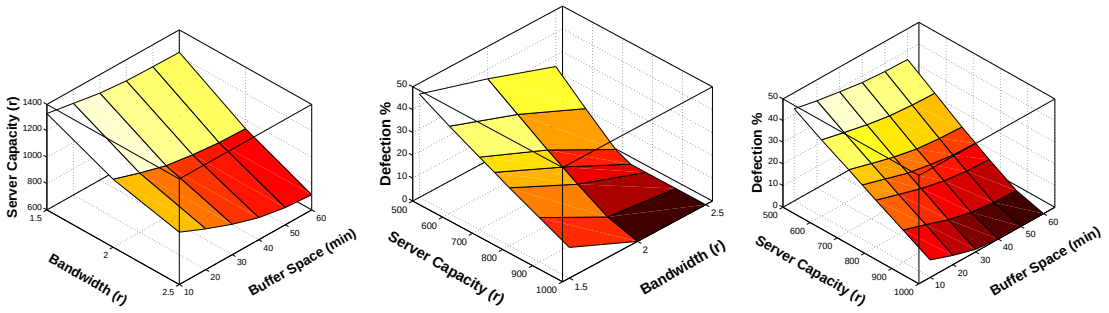
Comparing the Impacts of Server Capacity, Client Bandwidth, and Client Buffer Space

After evaluating in isolation the impacts of different client and server resources on system performance and requirements, let us now compare the impacts of the three main resources: server capacity, client bandwidth, and client buffer space. Comparing the impacts of these resources is very helpful in taking informative decisions to face any degradation in system performance. Figure 4.15 plots the results under different combinations of buffer space availability levels for these three main resources. The main results can be summarized as follows:

(1) Server and client bandwidth capacities, in general, have larger impacts on performance than the available client buffer space, as long as some buffer space is available in most clients. For example, when the average download bandwidth is $1.5r$, the availability of no buffer space leads to requiring $4800r$ in server bandwidth to achieve TVOD, while a 10-minute buffer space reduces

the requirement to approximately $1331r$. Doubling that average buffer space, however, reduces this requirement to $1295r$ (i.e., by only 3%). Even increasing it to six times that average buffer space reduces the requirement to $1230r$ (only 7.5%). In contrast, increasing the average download bandwidth from $1.5r$ to $2.0r$ (approximately 33%) reduces the required server capacity to $1116r$ (approximately 11%). Similarly in NVOD, when the server capacity is $500r$, increasing the buffer space from 10 minutes to 60 reduces the defection rate approximately from 42.5% to 36%, while increasing the server capacity to $600r$ results in a defection rate of 35%.

(2) Server and client bandwidth capacities both have comparable effects on system performance. For example increasing the average download bandwidth from 1.5 to 2.5 (67%) reduces the defection approximately from 45% to 25% when the server capacity is $500r$. To achieve comparable results without requiring that much client bandwidth, the server capacity needs to be increased to more than $800r$ (more than a 60% increase). The defection rate at $800r$ is 25%.



(a) Impact of Buffer Space and Bandwidth on Server Capacity

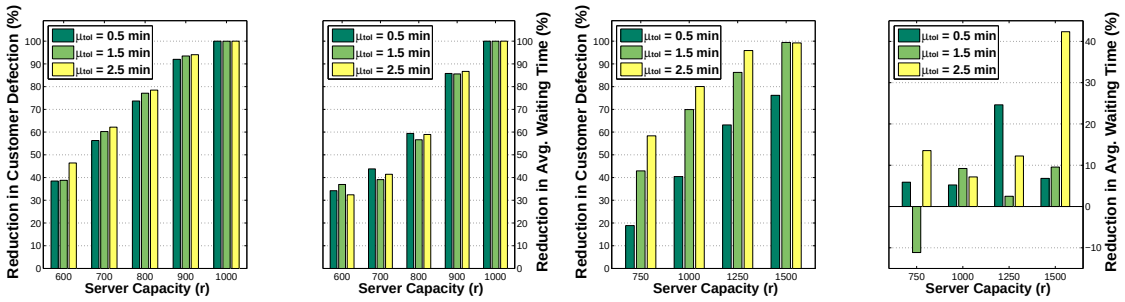
(b) Impact of Bandwidth and Server Capacity on Defection

(c) Impact of Buffer Space and Server Capacity on Defection

Figure 4.15: Impact of Client Bandwidth, Client Buffer Space, and Server Capacity on EHS-E Performance [Normal Bandwidth Dist. with Default $\mu_B = 2.0r$, Normal Buffer Space Dist. with Default $\mu_S = 60$ min, MCF-P]

Impact of Customer Waiting Tolerance

Let us now discuss the impact of customer waiting tolerance. Figure 4.16 plots the percentage improvements achieved by EHS-E and AHS-E compared with SHS-E in terms of the defection percentage and the waiting time for different values of μ_{tol}. Four main conclusions can be drawn



(a) EHS-E vs. SHS-E (b) AHS-E vs. SHS-E

Figure 4.16: Impact of Customer Waiting Tolerance (μ_{tol})

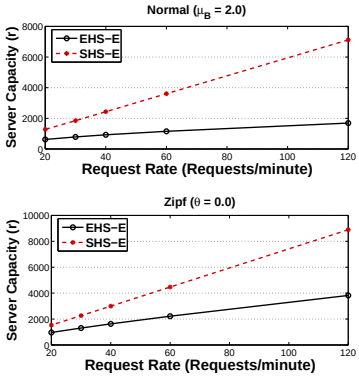


Figure 4.17: Impact of Variable Request Rate (TVOD)

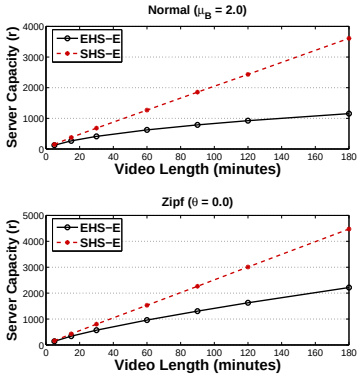


Figure 4.18: Impact of Variable Video Length (TVOD)

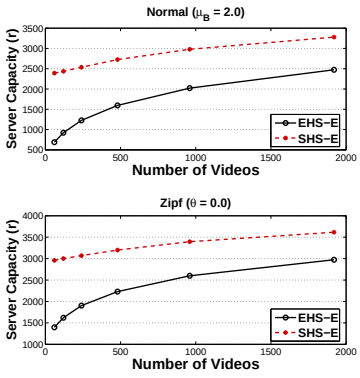


Figure 4.19: Impact of Variable Video Number (TVOD)

here. (1) The higher the customer waiting tolerance, the more efficient AHS becomes compared with SHS because longer waiting tolerance allows more clients to be serviced by a-streams even when longer buffering times are required. EHS, on the other hand, is less sensitive to the customer waiting tolerance. 2) Even with very low customer tolerance, such as 30 seconds, AHS achieves a significant improvement over SHS and can eliminate the majority of the defections compared to it. 3) Although the main advantage of AHS over SHS is reducing the defection rate and servicing a larger number of customers, it also results in a shorter average waiting time. 4) Regardless of the average waiting tolerance of customers, EHS keeps its lead over the other solutions, especially in terms of the defection rate and waiting time.

Impacts of Request Arrival Rate, Video Length, and Number of Videos

We discuss here the effectiveness of EHS-E and SHS-E in providing TVOD. (Recall that AHS cannot provide such service because of the required waiting time for buffering.) Figures 4.17, 4.18, and 4.19 compare the required server bandwidth to achieve TVOD by the two schemes versus request arrival rate, video length, and number of videos, respectively, under two bandwidth distributions: Normal with $\mu_B = 2r$ and Zipf with $\theta = 0$. The impact of the video length is similar to the that of the request rate. The number of concurrent customers per video increases with each, and thus data sharing will be enhanced. This behavior explains why EHS becomes increasingly more efficient than SHS. EHS can achieve TVOD with a fraction of the requirement of SHS and it scales well with the increase in concurrent customers. For example, as the request rate increases 6 folds (from 20 to 120 requests/minute), the required server bandwidth to provide TVOD service by EHS-E increases only 2.5 folds under the Normal bandwidth distribution and only 4 folds under the Zipf distribution, whereas the server bandwidth with SHS increases 5.6 to 5.8 folds, with the two bandwidth distributions, respectively. The impact of the number of videos is different. Increasing the number of videos while fixing the other two parameters reduces the number of concurrent customers per video, thereby reducing the chances for data sharing and narrowing the performance gap between EHS and SHS. In other words, as the number of videos increases while fixing the total request arrival rate, the videos become increasingly less popular and thus data sharing decreases.

4.6 Conclusion

We have studied scalable on-demand delivery of video streams to heterogeneous receivers. We have proposed three solutions to address the variation in the download bandwidth among clients: *Simple Hybrid Solution* (SHS), *Adaptive Hybrid Solution* (AHS), and *Enhanced Hybrid Solution* (EHS). SHS simply combines existing resource sharing techniques and deals with clients as two bandwidth classes. AHS and EHS, however, classify clients into multiple bandwidth classes and service them accordingly. Depending on whether Patching or ERMT is used for clients with bandwidth capacities of double the video playback rate or higher, the three solutions lead to six schemes:

SHS-P, SHS-E, AHS-P, AHS-E, EHS-P, and EHS-E. Moreover, we have studied the support for the variability the buffer space among clients. Furthermore, we have discussed how request scheduling policies can be adapted to capture the variations in client bandwidth and buffer space.

We have evaluated the effectiveness of the proposed schemes and have analyzed various scheduling policies through extensive simulation. We have considered two service models: *True Video-on-Demand* (TVOD) and *Near Video-on-Demand* (NVOD). The main results can be summarized as follows.

- The proposed schemes can achieve high degrees of resource sharing.

- AHS significantly outperforms SHS and Batching in the considered performance metrics. For example, it can provide a service with no customer defections, while SHS and Batching cause more than 27% and 41% defections, respectively. Additionally, it can reduce the average request waiting time and tends to be more fair towards both unpopular videos and low bandwidth classes.

- EHS achieves significant improvements over AHS under all workloads and service models. In addition, it can provide a TVOD service (i.e., zero waiting time), whereas AHS cannot. EHS is also more tolerant to variations in client bandwidth during service. With the same server bandwidth, EHS-E can achieve zero defections, while AHS-E, SHS-E, and Batching cause 13%, 45%, and 53% defections, respectively.

- The three proposed solutions vary significantly in performance, whereas the two variants of each perform close to one other. In other words, using ERMT instead of Patching for serving clients with bandwidth capacities of $2r$ or higher is of less significance than changing the solution (and thus the method of service for clients with bandwidth capacities higher than r but lower than $2r$).

- Client available buffer space, in general, has less impact on system performance than client bandwidth, but its impact can be very significant if it becomes the main bottleneck.

- The performance depends greatly on the applied scheduling policy. For example, with EHS-E, choosing the fair FCFS instead of the efficient, cost-oriented MCF-P can increase the number of defected customers by more than 50%. In certain situations, MQL performs better than MCF-P but only when SHS is used.

We conclude that EHS-E and EHS-P are the best overall performers. EHS-E performs slightly better but is much more complicated due to the high implementation complexity of ERMT. When any one of these two schemes is employed, it is best to schedule the waiting requests using MCF-P.

CHAPTER 5

GENERALIZED ACCESS PATTERNS: HOMOGENEOUS RECEIVERS

5.1 Introduction

Resource sharing has been studied extensively when videos are accessed sequentially from the beginning to the end. We refer to this access pattern as *Full-Content Access*. Actual workloads, however, suggest a different pattern, referred to as *Selected-Content Access* in this chapter. It is a generalized pattern that captures the fact that a considerably large number of sessions start from a position other than the beginning of the video and finish before reaching the end [12, 14]. This pattern is common, especially for long videos. Popular interactions (such as jump forward, jump backward, and pause) can be considered as a series of selected-content access periods. Unlike other prior work, the study [40] used such realistic workload but focused on reducing the data requested by clients in interactive operations through enhanced data buffering and relied on traditional stream merging techniques without changing their stream merging decisions.

In this chapter, we study the impact of selected-content access on streaming servers delivering data in a client-pull fashion using stream merging techniques, such as *Patching* [31] and *Earliest Reachable Merge Target (ERMT)* [18]. The *server-push* approach is not discussed because it is not suitable for selected-content access, can be used only for popular videos, may lead to server underutilization, and imposes waiting time on the majority of requests. Moreover, we propose five enhancements to reduce server load and improve the client perceived quality-of-service (QoS). We evaluate the effectiveness of the proposed enhancements through simulation.

The rest of this chapter is organized as follows. Section 5.2 presents the proposed enhancements. Section 5.3 discusses the performance evaluation methodology and main results.

5.2 Support for Selected-Content Access

Selected-content access has two conflicting effects on how much data is to be delivered by the streaming server. The shorter session lengths reduce the required data per client. However, the overlap of data and thus data sharing are also reduced because not all requests for the same video start from the same position.

This environment complicates resource sharing since later streams do not always access later data in the video. In Patching, for example, it is no longer satisfactory to consider only the last regular stream when servicing a new request. Instead, the server must examine all regular streams as potential merge targets. In this context, we define a *regular stream* as any non-patch stream, which may start from a position other than the beginning of the video and may finish before reaching the end.

For this environment, we propose three enhancements in stream merging and a new class of scheduling policies.

5.2.1 Proposed Stream-Merging Enhancements

The following proposed enhancements are applicable to all stream merging techniques.

Adopt Earlier Streams and Their Children (AESnC)

Traditionally, stream relationships (between a merger and a merge target) are determined when a merger stream starts or when merging occurs. In the new environment, a start of a potential target stream may require revising some of the previous decisions. For example, there were no potential targets for Stream 1 when it was initiated, as shown in Figure 5.1. Similarly, Stream 2 had no target when it started since it needed later data. With the original stream merging algorithms (such as Patching and ERMT), both streams will be regular streams (i.e., they will continue until possibly the end of the video and do not merge with other streams). A possible optimization is to downgrade Stream 1 to a patch and use Stream 2 as its merge target. In other words, Stream 2 *adopts* Stream 1 (which is an earlier stream) and all its possible children. A similar situation happens for Streams 4 and 5.

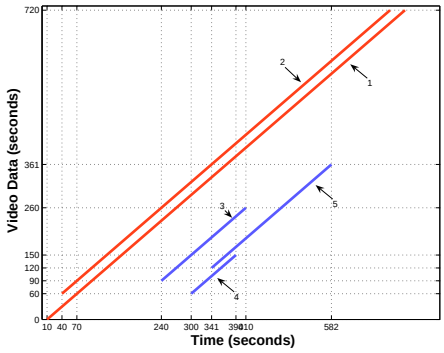


Figure 5.1: Illustration of ERMT with Selected-Content Access [No Enhancement]

Serve Later Clients at No Cost (SLC@NC)

This relatively simple enhancement lets waiting requests to be served by an existing regular stream when their requested starting position (i.e., playback point) is reached by that stream. Hence, these requests will be served by Batching at no extra server cost.

Admit Clients with Active Wait (ACwAW)

With this enhancement, the server admits the client to an existing or a new stream when the current position delivered by the stream is earlier than the requested starting position by no more than AW_{max}. The client watches the earlier data while actively waiting for its requested data. For customer satisfaction, AW_{max} must be set to a small value.

5.2.2 Proposed Fuzzy Queue Length (FzQL) Scheduling

Selected-content access complicates scheduling. A video waiting queue may contain requests for clients varying in their desired starting positions. So, there are different subgroups of requests that can be serviced concurrently. Thus, we introduce the *virtual-queue* concept and define it as a subgroup of the waiting requests for a certain video with starting positions within AW_{max} from the one with the earliest starting position. MQL scheduling selects the queue (or virtual queue in this context) with the largest number of requests. The objective function that MQL tries to maximize is

simply the virtual queue length. The proposed *Fuzzy Queue Length* (FzQL) policy, however, utilizes the $SLC@NC$ enhancement by favoring the streams that are more likely to be joined later by other waiting requests. In computing the objective function for a certain virtual queue vq, it considers not only the requests in vq but also the waiting requests for all starting positions further than those in vq. This is because these requests are likely to join the stream serving vq if they have to wait longer for service. FzQL is applicable only to regular streams and thus conventional MQL is used for other streams.

We present two alternative implementation of FzQL, called *Maximum Fuzzy Queue Length (MFzQL)* and *Weighted Fuzzy Queue Length (WFzQL)*. MFzQL works as follows. Each request in a virtual queue vq has full membership and thus will be counted as one. The other requests (belonging to other virtual queues) will be counted as fractions inversely proportional to how far their requested positions are away from vq's earliest starting position (Ps_{vq}). Let us assume that for request i with starting position Ps_i, $\Delta_i = Ps_i - Ps_{vq}$. For a video of Length L, let us assume that the count of requests with starting positions in $[t_s, t_e[$ is $C(t_s, t_e)$. Thus for vq, the objective function is given by

$$F_m(vq) = \sum_{i=1}^{C(Ps_{vq},L)} 1.0/max(1, \lceil \Delta_i / AW_{max} \rceil),$$

where $AW_{max} > 0$. The counting process is illustrated in Figure 5.2. The symbol x indicates a waiting request for a certain starting position.

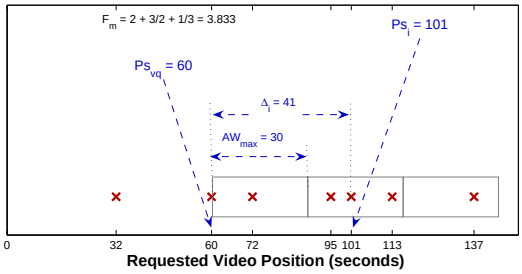


Figure 5.2: Illustration of Computing the MFzQL Objective

The WFzQL implementation favors the queues that are less likely to be served at no cost in the future by weighting F_m with the time T_{NoCost} required to serve the group at no cost using a regular stream. Hence, the objective function is given by $F_w(vq) = F_m(vq) \times min(T_{NoCost}, L)$.

5.3 Evaluation

We study the impact of selected-content access and the effectiveness of the proposed enhancements (including FzQL) through extensive simulation. Only a subset of the results is shown for space limitation. We consider five performance metrics: customer defection (i.e., turn-away) probability, average waiting time for service, average waiting time to view the requested data, unfairness against unpopular videos, and unfairness against unpopular requested starting positions. The metrics are listed in order of importance. The last metrics is introduced here to quantify the bias against unpopular positions, whose requests may have higher defection rates. It is computed as the standard deviation of the defection rate for various position intervals in the video. The average waiting time to view the requested data includes the waiting time for service and the subsequent time to reach the requested position in the video.

5.3.1 Workload Characteristics

We assume the request arrival follows a Poisson distribution and the access to videos is highly localized and follows a Zipf-like distribution. The access pattern is based loosely on [14, 40]. Table 6.5 summarizes the main workload characteristics.

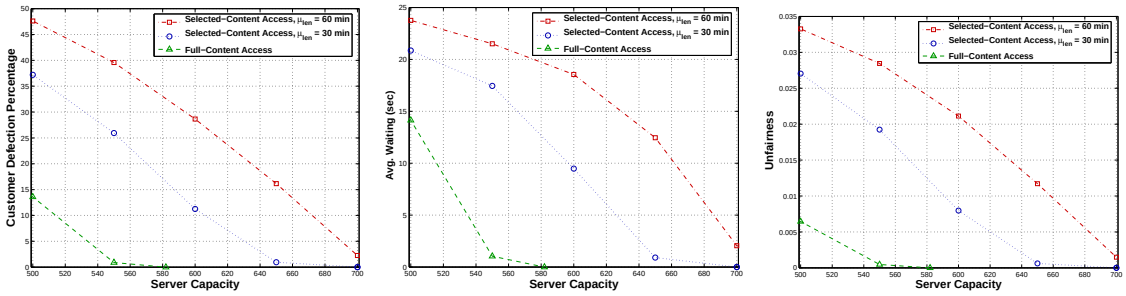
5.3.2 Result Presentation and Analysis

Figure 5.3 demonstrates the impact of the video access pattern on performance when ERMT is used. The same behavior happens with Patching and thus its results are not shown. Two means for the session length are examined ($\mu_{len} = 60$ and 30) in the selected-content access. Interestingly, although with full-content access, the session length is equal to the entire video length, it leads

Table 5.1: Summary of Workload Characteristics

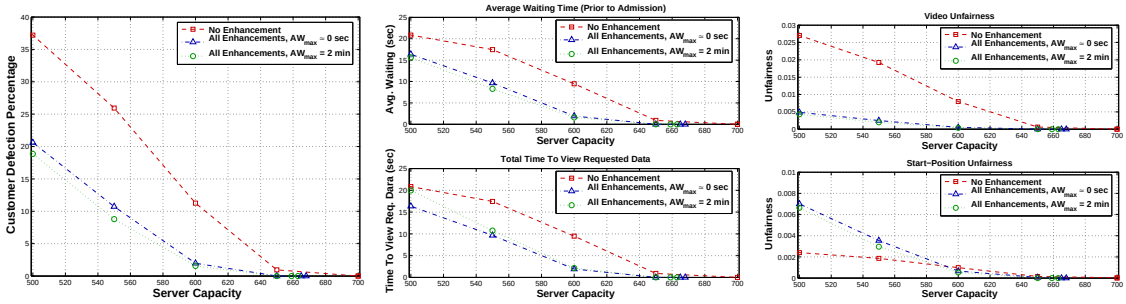
Parameter	**Default Value(s)**
Arrival Rate (λ)	30 Req/min
Number of Videos (N)	120
Video Length (L)	120 min
Video Popularity	Zipf-like with skewness θ = 0.271
Waiting Tolerance Model	Exponential with μ_{tol} = 2 min
Session Length Distribution	Normal with μ_{len} = 30 or 60 min
Session Starting Pos. Dist.	Zipf-like with skewness = 0.0
% Sessions Starting at Pos. 0	20%
Max. Active Wait (AW_{max})	2 min

to better performance than the selected-content access. The reason is due to the reduction in data sharing, caused by the variation in the requested starting position.



(a) Defection Rate (b) Average Waiting Time (c) Video Unfairness

Figure 5.3: Impact of Access Pattern [ERMT]

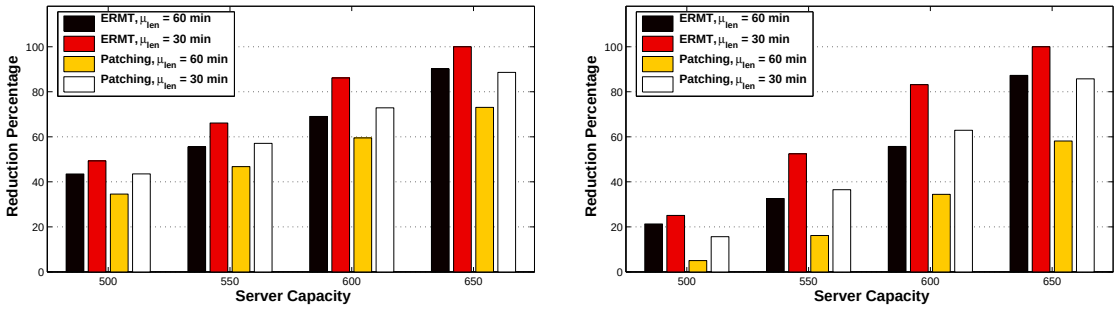


(a) Defection Rate (b) Average Waiting Time (c) Unfairness

Figure 5.4: Impact of Proposed Enhancements [ERMT, Selected-Content Access, μ_{len} = 30 minutes]

Figure 5.4 shows the effectiveness of the proposed stream-merging enhancements and WFzQL when ERMT is applied. The "No Enhancements" case uses MQL and $AW_{max} \approx 0$. The "All Enhancements" cases use WFzQL and the three enhancements, but the $ACwAW$ enhancement is effectively disabled when $AW_{max} \approx 0$. The individual effects of various policies and enhancements (except for $ACwAW$) are not shown to save space. The results for MFzQL are also not shown because it is outperformed by WFzQL. The results indicate that applying the first two enhancements and WFzQL improves the service significantly based on every metric, except for the starting-position unfairness when the server capacity is low. Applying ACwAW improves the service further in terms of the two most important metrics, but as expected, it introduces additional delay until the desired position in the video is reached.

Figure 5.5 depicts the improvements in the two most important metrics achieved by the proposed schemes (including WFzQL) for two different means of session length. The results for both Patching and ERMT are shown. These results demonstrate that the proposed schemes improve the server throughput and reduce the waiting time significantly. The reductions in customer defections and waiting times are $35\% - 100\%$ and $21\% - 100\%$, respectively.



(a) Defection Rate

(b) Average Waiting Time

Figure 5.5: Achieved Reductions in Defection Rate and Waiting Time [$AW_{max} = 2$ minutes]

5.4 Conclusion

We have studied the impact of selected-content access on stream-merging video streaming servers and have proposed five enhancements to reduce the load on the server and to improve

customer-perceived QoS. We have evaluated the effectiveness of the proposed enhancements through simulation, considering five performance metrics. The results show that selected-content access reduces significantly data sharing among requests and that the proposed enhancements are very effective in improving the sharability and reducing the waiting times.

CHAPTER 6

GENERALIZED ACCESS PATTERNS: HETEROGENEOUS RECEIVERS

6.1 Introduction

As shown in earlier chapters, the achieved resource sharing depends significantly on the user access patterns as well as the available server, client, and network resources. Selected-content access and client heterogeneity have been addressed individually in chapters 5 and 4, respectively. This chapter analyzes the combined impact of both aspects on resource sharing. As shown, the server design and overall system modeling are very challenging even when only one of these aspects is considered and thus considering both is much more challenging. This chapter aims at filling this gap in literature. In particular, it seeks to assess the achieved resource sharing when both these aspects are considered. It also develops new enhancements to increase the resource sharing and improve the client perceived quality-of-service (QoS). Some of these enhancements are guided by a novel gain and risk analysis presented in the chapter.

In addition, we investigate in this chapter utilizing more advanced video coding with advance stream merging techniques for better serving heterogeneous receivers. In particular, we investigate Layered Video Coding (LVC) and Multiple Description Coding (MDC) [49, 27, 21]. While with these advanced video codings, the objective is that each client receives the maximum number of descriptions its bandwidth allows, a temporary reduction in video quality can help significantly reducing the overall delivery cost (amount of data delivered by the server). We answer two questions concerning such a reduction in quality; when to resort to this option? and for how long this temporary period can last? In answering the first question, we propose three approaches, and in answering the second question, we consider three options on limiting the reduced quality period. We evaluate the Cartesian product of these approaches and limit options with both LVC and MDC and we

compare them to each other and to an alternative system with multiple copies of every video each decoded at different rate.

The effectiveness of the proposed enhancements and approaches and impacts of various workload parameters are analyzed through extensive simulation, considering both True Video-on-Demand (TVOD) and Near Video-on-Demand (NVOD) service models.

The rest of this chapter is organized as follows. Section 6.2 analyzes the shareability of multimedia data, Section 6.3 presents the proposed description level enhancements, while Section 6.4 addresses utilizing advanced video coding. Section 6.5 provides practical considerations in implementing scheduling efficiently in realistic environments with both heterogeneous receivers and selected-content access. Section 6.6 discusses the performance evaluation methodology and presents the main results.

6.2 Analysis of Data Shareability with Selected-Content Access and Heterogeneous Receivers

In this section, we analyze multimedia data shareability among various requests and how it is related to the system resources required to provide TVOD. We will also discuss the combined impact of selected-content access and receiver heterogeneity. Subsequently, we present a novel risk analysis of early delivery of data by resource sharing techniques. Table 6.1 summarizes the used symbols.

6.2.1 Analysis of Multimedia Data Shareability

The average rate at which a video data point $Data(v, d, t)$ is delivered can be found by

$$Deliver(v, d, t) = \frac{\lambda(v, d, t)}{Share(v, d, t)}, \tag{6.1}$$

where $\lambda(v, d, t)$ is the average rate at which $Data(v, d, t)$ is requested, and $Share(v, d, t)$ is the number of clients known at the delivery time of that data to require it in the future and are capable of receiving and storing it for later use. $Data(v, d, t)$ is the data in description d of video v with playback time t seconds from the beginning of the video. The average required server bandwidth to

Table 6.1: Symbols used in Analysis

Parameter	**Description**
$Data(v,d,t)$	Data point in description d of video v with playback time t seconds from the beginning of the video
$Len(v)$	Video v length ($Seconds$)
$Rate_{enc}(v)$	Encoding rate for the best quality single description of video v (bps)
$Rate_{enc}(v,d)$	Encoding rate for description d of video v (bps)
$\lambda(v)$	Average request arrival rate for video v ($Requests/Second$)
$\lambda(v,d)$	Average request arrival rate for description d of video v ($Requests/Second$)
$\lambda(v,d,t)$	Average request arrival rate for $Data(v,d,t)$ ($Requests/Second$)
$Share(v,d,t)$	Number of clients known at delivery time of $Data(v,d,t)$ to require that data in the future and are capable of receiving and storing it for later use
$Deliver(v,d,t)$	The average frequency at which $Data(v,d,t)$ is delivered by the system ($Second^{-1}$)
$BW_{serv}(v)$	Average required server bandwidth to serve all requests for video v immediately (TVOD) (bps)
$BW_{serv}(v,d)$	Average required server bandwidth to serve all requests for description d of video v immediately (TVOD) (bps)
$BW_{cli}(v)$	Allocated client bandwidth for video v (bps)
$BW_{cli}(v,d)$	Allocated client bandwidth for description d of video v (bps)
$Rate_{delv}(s,t)$	Data delivery rate of stream s at time t from its start time (bps)
$Pos_{delv}(s)$	The being delivered position (from beginning of video) of of stream s ($Seconds$)
$Pos_{adv}(s)$	The play back position of the most advanced listener of stream s ($Seconds$)
$Pos_{gap}(s)$	The gap between stream s being delivered position and the position being played back by the most advanced listener at the time of risk evaluation ($Seconds$) [$Pos_{gap}(s) = Pos_{delv}(s) - Pos_{adv}(s)$]
$Remain(s)$	Remaining from the stream delivery position till the end of the video ($Seconds$) [$Remain(s) = Len(v) - Pos_{delv}(s)$]
$Prob_{stop}(s,t)$	The probability that all listeners of stream s leave it t seconds from the time of risk evaluation
$Gain(s,t)$	The gain (saving in delivered data) by stream s compared to an alternative regular stream that matches the play back of the most advance listener, if all listeners leave the stream t seconds from the time of risk evaluation (with negative gain means loss) ($Seconds$)
$Gain_{exp}(s)$	The expected overall gain by stream s ($Seconds^2$)

serve all video requests immediately (TVOD) can be found by

$$BW_{serv}(v,d) = Rate_{enc}(v,d) \int_0^{Len(v)} Deliver(v,d,t).dt, \tag{6.2}$$

where $Len(v)$ is the video length in seconds.

With a full-content access workload, $\lambda(v,d,t)$ is simply equal to $\lambda(v,d)$, for all data points within the video description, where $\lambda(v,d)$ is the average request arrival rate for the video description. Without utilizing the multicast facility (i.e., only unicast is used), $Share(v,d,t) = 1$. Therefore, it follows under such conditions that

$$BW_{serv}(v,d) = Rate_{enc}(v,d) \int_0^{Len(v)} \lambda(v,d).dt = Rate_{enc}(v,d)\lambda(v,d)Len(v). \tag{6.3}$$

Let us now discuss what happens when the multicast facility is utilized through aggressive resource sharing and assuming unlimited server and client resources and no predictive data pre-fetching (i.e., without fetching data before the request is even issued based on a prediction that the client might want to watch the video some time soon in the future). $Data(v,d,t)$ in such a case is scheduled for delivery t seconds in advance and every client requiring that data is known t seconds in advance of its playback and thus $Data(v,d,t)$ is shareable by every client arriving in the t-second window prior to its delivery requesting that video description. Hence, $Share(v,d,t) = \lambda(v,d)t + 1$. Consequently,

$$Deliver(Data(v,d,t)) = \frac{\lambda(v,d)}{\lambda(v,d)t + 1} \tag{6.4}$$

and

$$BW_{serv}(v,d) = Rate_{enc}(v,d) \int_0^{Len(v)} \frac{\lambda(v,d)}{\lambda(v,d)t + 1}.dt, \tag{6.5}$$

or

$$BW_{serv}(v,d) = Rate_{enc}(v,d) log_e(\lambda(v,d)Len(v) + 1). \tag{6.6}$$

This conclusion is consistent with the findings in [20].

With selected-content access, however, not every client requires every data point in the video, and thus $Data(v,d,t)$ is not necessarily scheduled for delivery t seconds in advance, and the clients requiring it are not necessarily known t seconds in advance. Moreover, the limited client bandwidth and buffer space indicates that not every client that is known in advance to require the data is actually

capable of receiving and storing it until its playback time. Hence, the estimations of $\lambda(v, d, t)$ and $Share(v, d, t)$ become complex and potentially less accurate.

The aforementioned discussion suggests that improving the resource sharing, and thus reducing the required server bandwidth can be achieved by increasing $Share(v, d, t)$. (Reducing $\lambda(v, d, t)$ results in lower required server bandwidth, but, of course, it is not desirable to reduce the popularity of the video or the quality received by clients.) Obviously, $Share(v, d, t)$ is negatively affected by both interactive operations (or selected-content access) and by limited client resources. One way to improve $Share(v, d, t)$, however, is by having a better earlier prediction of clients requiring the video data point $Data(v, d, t)$, even before a video playback request is issued. Such prediction can be based on client's own history, its social network, pattern similarities, and other methods. The discussion and evaluation of this dimension is beyond the scope of this study and left for future work.

6.2.2 Risk Analysis of Early Delivery

Resource sharing can (intentionally or as a result of changes in some conditions) deliver data earlier than needed. For example, in EHS, an ea-stream delivers data ahead of time during the fast delivery period to increase the potential shareability. Another example is when a regular stream is initially started to serve a client or a group of clients and is later joined by clients of merging streams. Under selected-content access, there is a chance that the initial client or group of clients that started the regular stream leave the service before the end of the stream. Since the stream, however, is needed by the merging client(s), it cannot be terminated. Under such a situation, the stream delivers data before being needed from playback point of view. In both examples, if all listeners leave at a later time, the stream will end up delivering data that will never be used (even if it is terminated when the last listener leaves). A risk analysis is required based on gain/loss prediction to decide when it is good to deliver data ahead of time. Assume stream s is delivering data for description d

of video v. The expected overall gain of such a decision can be calculated by

$$Gain_{exp}(s) = \int_{Pos_{delv}(s)}^{Len(v)} Gain(s,t) Prob_{stop}(s,t).dt, \tag{6.7}$$

where $Gain(s,t)$ is the gain (i.e., the achieved saving in the amount of delivered video data in the unit of second) compared to an alternative regular stream matching the playback of the most advanced listener, if all listeners leave t seconds from the time of risk evaluation. The most advanced listener is the client that is playing back the furthest position from the beginning of the video among all current listeners. In the equation, $Prob_{stop}(t)$ is the probability that all listeners stop listening after t seconds, and $Pos_{delv}(s)$ is the delivered data position by the stream at time of risk evaluation. A negative gain means loss. Delivering data in advance is good only if $Gain_{exp}(s) >= 0$. Considering Figure 4.4(a), the gain for ea-streams can be found as

$$Gain(s,t) = \begin{cases} (1-B)t & t \leq X, \\ (2-B)(t-(X+Y)) & X < t \leq X+Y, \\ t-(X+Y) & X+Y < t \leq Remain(s), \\ Remain(s)-(X+Y) & Remain(s) < t, \end{cases} \tag{6.8}$$

where

$$Remain(s) = Len(v) - Pos_{delv}(s), \tag{6.9}$$

$$B = \frac{Rate_{delv}(s,0)}{Rate_{enc}(v,d)}, \tag{6.10}$$

and $X$and Y are the *fast* and *slow* delivery periods (in seconds) of the ea-stream, respectively. The gain is negative for the duration of the ea-stream $t \leq X+Y$ and is positive afterwards. Therefore, $Gain_{exp}$ can be negative if $Prob_{stop}(t < X+Y)$ is relatively high, which is more likely in very active environments, or if $(X+Y)$ is large compared to $Len(v)$, or if $Pos_{delv}(s)$ is close to the end of the video ($Len(v)$). Under such conditions, when the client is highly expected to leave before the scheduled end time of the ea-stream, an alternative regular stream would be more efficient.

Figure 6.1 further illustrates $Gain(s,t)$ for an ea-stream evaluated for a client with description download bandwidth of $1.6Rate_{enc}(v,d)$. Two options are shown in the figure, an ea-stream that eventually merges with its target (unless it is terminated earlier), and an alternative regular stream initiated specifically for the same client. In the example, other more advanced listeners are assumed to listen to the older merge target. The first option delivers data earlier than needed during the lifetime of the ea-stream to allow the client to receive data free of cost at later stages from the merge target. Therefore, if the client's session lasts beyond the lifetime of the ea-stream, an overall gain is achieved. If the client, however, leaves service before that, the data delivered in advance will be lost, and the alternative stream would have been a cheaper option, because no matter which option is chosen, the stream will be terminated when it has no listener. In contrast, for regular streams, the gain can be found by

$$Gain(s,t) = \begin{cases} -t & t \leq Pos_{gap}(s), \\ -Pos_{gap}(s) & Pos_{gap}(s) < t \leq Remain(s) - Pos_{gap}(s), \\ t - Remain(s) & Remain(s) - Pos_{gap}(s) < t \leq Remain(s), \\ 0 & Remain(s) < t, \end{cases} \tag{6.11}$$

where $Pos_{gap}(s)$ is the data gap between the delivered stream data and the data played back by the most advanced listener. Consequently, the gain for regular streams can never be positive. Therefore, an alternative regular stream that matches the playback of the most advanced client is better than a regular stream that delivers data much earlier. Further insights on the gain and risk analysis are provided by an example in Subsection 6.3.1.

6.3 Description Level Enhancement

6.3.1 Enhancement 1: Yield and Resume

As shown by Equation (6.11), it is not a good idea for a regular stream to deliver data before it is actually needed. One simple yet effective solution is to put such streams on a pause for a period

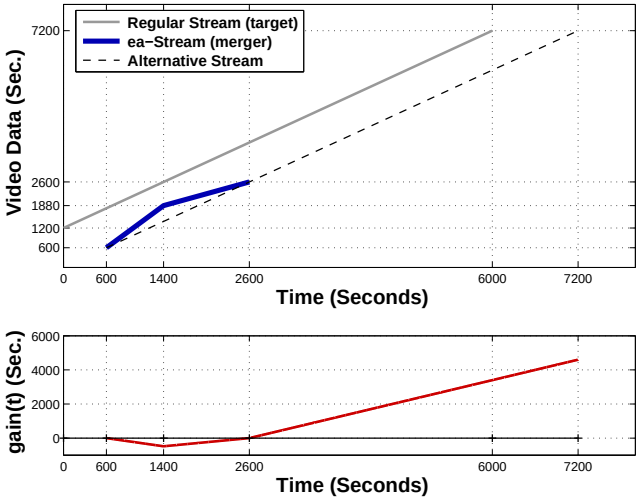


Figure 6.1: Illustration of Risk Analysis of Early Delivery of Data by *ea-Stream*

of time sufficient to bridge the gap between the delivered and needed data, and then to resume the stream as the data to-be delivered next becomes close enough (within threshold δ_{resume}) to the playback point of the most advanced listener. Because the server, however, needs to guarantee the capability of resuming the stream, the stream might have to be resumed earlier than optimal. δ_{resume} can be statistically set to the smallest value that ensures with high probability that the server will have the required resources to resume the stream within it. This enhancement reduces the server load by minimizing the wasted data, which is delivered without ever being used. For example in Figure 6.2, for the Regular Stream (which starts at time 0 with start position 600 seconds), $P_{gap}(s) = 0$ while the most advanced listener is Client 1, however, without "Yield and Resume", $P_{gap}(s)$ becomes 800 seconds after Client 1 leaves service and Client 2 becomes the most advanced listener. Without "Yield and Resume", by the time Client 2 leaves the service, the Regular Stream would have delivered the data up to playback point 4200 seconds, while Client 2 only viewed up to playback point 3400 seconds. In this example, "Yield and Resume" saved 800 seconds worth of data out of the 3600 seconds original length, a 22% saving.

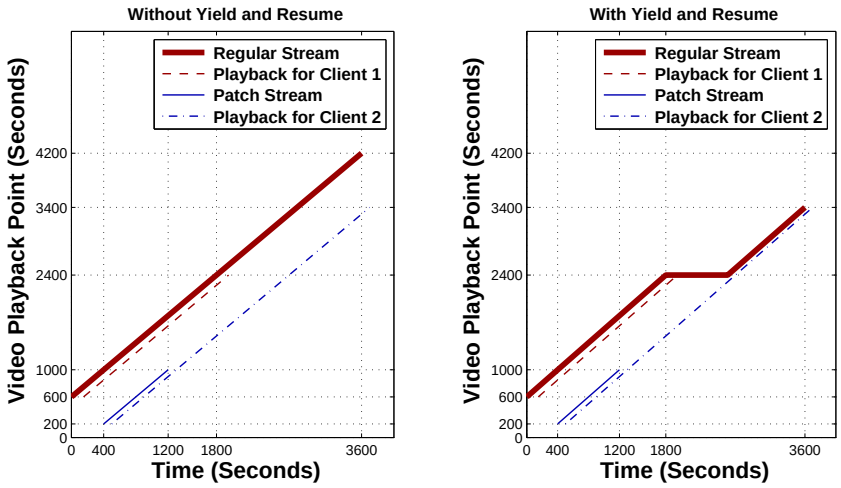


Figure 6.2: Illustration of Enhancement 1: Yield and Resume

6.3.2 Enhancement 2: Admit and Wait

Admit and Wait aims at increasing server throughput by increasing both data shareability and client waiting tolerance. The idea can be described as follows. Instead of waiting to admit the client when a stream meets its requested data as in *Serve Later Clients at No Cost* in Chapter 5 (Section 5.2), the client is admitted and then made wait if a stream does exist in the system that will deliver the client's requested data in a reasonable time ($aWait_{thr}$). $aWait_{thr}$ has to be tuned carefully to avoid the defection of such client. By *Admit and Wait* rather than *Wait and Admit*, the client is encouraged to wait longer through providing *time-of-service guarantees* [42, 47]. In addition, the request is removed from the waiting queue and from competition with other requests which might have no such chance of being served free of cost.

6.3.3 Enhancement 3: Adopt Earlier Streams and Their Children (AESnC)

This enhancement was discussed in Chapter 5 (Section 5.2), but its implementation is altered to fit the heterogeneity of receivers. For an adoption to be possible, all current and anticipated (due to merge operations) listeners must have the resources required for the new adoption. These resources include bandwidth and buffer space. Obviously, the chance for such adoptions becomes very limited

because it is very unlikely that a regular stream for a popular video description has all listeners with description download bandwidth of $2Rate_{enc}(v, d)$ and enough buffer space to hold the increased gap in data. The required buffer space in a client is equal to the gap between the client's playback point and the data being delivered by the root stream of the merging tree (i.e., the regular stream the client is expected to listen to eventually if remained in service).

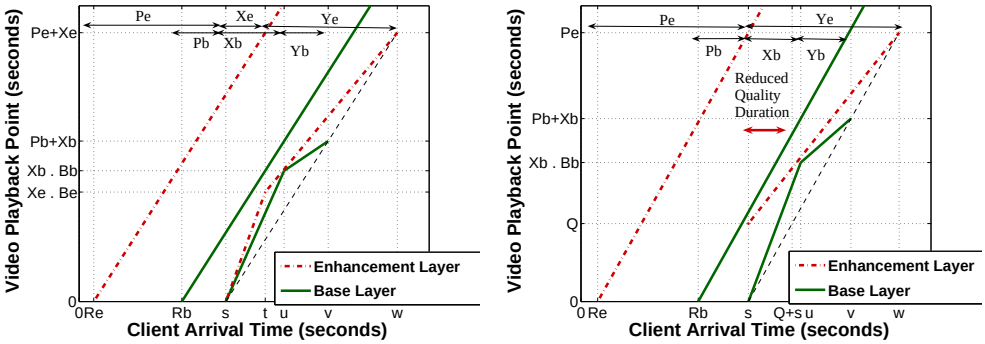
The implementation of MCF-P scheduling has to consider the impact of this enhancement when determining the cost. In other words, the cost of a regular stream should be reduced by the achieved saving in other streams due to adoption.

6.4 Utilizing Advanced Video Coding

Though EHS and AHS solutions do not assume any particular video coding, however to better support client heterogeneity, they can be used with multi description coding (MDC) and layered video coding (LVC). They can operate simply on each description/layer independently which allows supporting clients with a wider range of bandwidth, including those with fewer than one channel. (From now on, we will use the generic word description to refer to a description in the MDC or a layer in LVC, unless it is necessary to distinguish them). In addition, using multi description or layered video coding can provide other more aggressive options in servicing clients with bandwidth capacities higher than the playback rate, such as reducing video quality for a very short period of time, which will be discussed in the next Subsection. Figure 6.3 shows two examples of utilizing LVC in providing a TVOD service.

6.4.1 Reducing Quality: When and For How Long?

While with these advanced video codings, the objective is that each client receives the maximum number of descriptions its bandwidth would have allowed in a replacement unicast system with the same video library with sufficient server bandwidth. However, as mentioned earlier, when resource sharing techniques are used, a temporary reduction in video quality can help significantly reducing the overall delivery cost (amount of data delivered by the server). In this discussion, we try answer



(a) TVOD with Full Video Quality (b) TVOD with Short Period of Reduced Quality

Figure 6.3: Options for Supporting Layered Video Coding (LVC)

two questions concerning such a reduction in quality; when to resort to this option? and for how long this temporary period can last?

In answering the first question, we propose three approaches; (a) never reduce quality (referred to as *Full Quality*), (b) reduce quality to replace a full stream (referred to as *Moderate*), or (c) reduce quality whenever a gain can be achieved in reducing the delivered data (referred to as *Aggressive*). And in answering the second question, we consider three options on limiting the reduced quality period; (a) the reduced quality period has a *fixed* upper limit, (b) the period upper limit is *dynamic*, or (c) such a period is *unlimited*. We evaluate the Cartesian product of these approaches and limit options with both LVC and MDC and we compare them to each other and to an alternative system with multiple copies of every video each decoded at different rate (referred to as *MultiCopy*).

To compare quality across the different alternatives, we introduce a quality metric defined as the average percentage of video data received by each client compared to the best quality the same client can receive in a unicast system with unlimited server bandwidth serving the exact same video library.

More details on selecting the serving method for each description are shown in Tables 6.2, 6.3, and 6.4. With LVC, deciding which layers the client should eventually receive is straight forward, and a function of only the client bandwidth. However, the case is trickier with MDC. Instead of

selecting descriptions randomly till the client bandwidth is fully utilized or all descriptions are selected, we start by the description that costs the least to serve. Starting with the ones that can be served for free by *Admit and Wait* (described in Subsection 6.3.2), in ascending order of required waiting time. Followed by the ones with potential full stream targets in ascending order of the data distance of its target (target current position to required start position). Then randomly among the remaining descriptions. The selection of full stream target indicates either the expected cost if a stream merging technique is used (ERMT, Patching, or ea-stream), or the reduced quality period. Even with ERMT, the summation of the patch length and all extensions equals that distance.

Table 6.2: Selection of Description Serving Technique: Full Quality

```
Map<VideoDescription,ServiceCategory> fullQuality (Request request,
SortedSet<VideoDescription> selectedDescriptions,
int bandwidth, int bufferspace) {

    Map<VideoDescription,ServiceCategory> serviceDecision =
    new HashMap<VideoDescription,ServiceCategory>();

    for(VideoDescription description:selectedDescriptions) {
        StreamMerging sm;

        if(admitAndWait(request,description)) {
            serviceDecision.put(description, ServiceCategory.FreeRide);
        } else if( (sm =
            streamMerging(request,description,bandwidth,bufferspace)) != null) {
            serviceDecision.put(description, ServiceCategory.StreamMerging);
            bandwidth -= sm.getMinRequiredBandwidth();
            bufferspace -= sm.getRequiredBufferspace();
        } else {
            serviceDecision.put(description, ServiceCategory.Batching);
        }
    }

    return serviceDecision;
}
```

Table 6.3: Selection of Description Serving Technique: Moderate

```
Map<VideoDescription,ServiceCategory> moderate (Request request,
SortedSet<VideoDescription> selectedDescriptions,
int bandwidth, int bufferspace) {

    Map<VideoDescription,ServiceCategory> serviceDecision =
    new HashMap<VideoDescription,ServiceCategory>();
    int delayCount = 0;

    for(VideoDescription description:selectedDescriptions) {
        StreamMerging sm;

        if(admitAndWait(request,description)) {
            serviceDecision.put(description, ServiceCategory.FreeRide);
        } else if( (sm =
            streamMerging(request,description,bandwidth,bufferspace)) != null) {
            serviceDecision.put(description, ServiceCategory.StreamMerging);
            bandwidth -= sm.getMinRequiredBandwidth();
            bufferspace -= sm.getRequiredBufferspace();
        } else if(isDelyable(request,description,delayCount)) {
            serviceDecision.put(description, ServiceCategory.Delay); //Reduced Quality
            delayCount++;
        } else {
            serviceDecision.put(description, ServiceCategory.Batching);
        }
    }

    return serviceDecision;
}
```

6.4.2 Client Bandwidth Allocation Among Descriptions

For descriptions served with other than stream merging (ea-stream, patching, or ERMT), the bandwidth reserved for each description is decided when the service method is decided, and it is exactly the description encoding rate. However, with more than one description served by stream merging, the allocation of bandwidth is not trivial. We discuss here the optimal allocation of bandwidth in such case to minimize the overall delivered data by the server (cost). Let us assume n descriptions to be served by stream merging. Lets assume $Data_i$ is the missed data (in bits) from the targeted regular stream for description i at admission time. And lets assume description i is

Table 6.4: Selection of Description Serving Technique: Aggressive

```
Map<VideoDescription,ServiceCategory> aggressive (Request request,
SortedSet<VideoDescription> selectedDescriptions,
int bandwidth, int bufferspace) {

    Map<VideoDescription,ServiceCategory> serviceDecision =
    new HashMap<VideoDescription,ServiceCategory>();
    int delayCount = 0;

    for(VideoDescription description:selectedDescriptions) {
        StreamMerging sm;

        if(admitAndWait(request,description)) {
            serviceDecision.put(description, ServiceCategory.FreeRide);
        } else if(isDelyable(request,description,delayCount)) {
            serviceDecision.put(description, ServiceCategory.Delay); //Reduced Quality
            delayCount++;
        } else if( (sm =
            streamMerging(request,description,bandwidth,bufferspace)) != null) {
            serviceDecision.put(description, ServiceCategory.StreamMerging);
            bandwidth -= sm.getMinRequiredBandwidth();
            bufferspace -= sm.getRequiredBufferspace();
        } else {
            serviceDecision.put(description, ServiceCategory.Batching);
        }
    }

    return serviceDecision;
}
```

encoded at $Rate_i$. And lets assume BW is the client bandwidth available to serve all streams in question and BW_i is the allocated for the i_{th} description.

Cost for the n descriptions is

$$Cost = \sum_{i=1}^{n} Cost_i = \sum_{i=1}^{n} \frac{Data_i \times Rate_i}{BW_i - Rate_i} \tag{6.12}$$

subject to

$$\sum_{i=1}^{n} BW_i \leq BW \tag{6.13}$$

For simplicity, lets assume 2 descriptions and the total bandwidths is utilized, that is $BW \leq 4 \times r$:

$$Cost = \frac{Data_1 \times Rate_1}{BW_1 - Rate_1} + \frac{Data_2 \times Rate_2}{BW_2 - Rate_2} \tag{6.14}$$

subject to

$$BW_1 + BW_2 = BW \tag{6.15}$$

To achieve minimum cost, we take the first derivative of Cost in Equation (6.14) with respect to BW_1 and equating it with zero, we get the optimal value of BW_1:

$$\frac{Data_1 \times Rate_1}{(BW_1 - Rate_1)^2} = \frac{Data_2 \times Rate_2}{(BW_2 - Rate_2)^2} \tag{6.16}$$

or

$$BW_1 = \frac{\frac{Rate_1}{\sqrt{Data_1 \times Rate_1}} + \frac{BW - Rate_2}{\sqrt{Data_2 \times Rate_2}}}{(\frac{1}{\sqrt{Data_1 \times Rate_1}} + \frac{1}{\sqrt{Data_2 \times Rate_2}})}, \tag{6.17}$$

or

$$BW_1 = \frac{Rate_1\sqrt{Data_2 \times Rate_2} + (BW - Rate_2)\sqrt{Data_1 \times Rate_1}}{\sqrt{Data_1 \times Rate_1} + \sqrt{Data_2 \times Rate_2}}. \tag{6.18}$$

And

$$BW_2 = BW - BW_1. \tag{6.19}$$

Obviously, descriptions receive bandwidth below double the description encoding rate has to be served by ea-streams, while those received double the description encoding rate can be served by either Patching or ERMT depend on the scheme used.

6.5 Request Scheduling

As discussed earlier, a scheduling policy selects a video for service when an available channel becomes available and all requests for the selected video are serviced together by one stream. The implementation of scheduling is simple when assuming homogeneous receivers and full-content access. In this case, the maximum number of candidate groups to select from is equal to the number of videos. With realistic workload, however, the situation is significantly different. In particular, it is not always possible or desired to service all waiting requests for the same video at once because of variations in the available client download bandwidth, available client buffer space, and the requested starting position. When a variation in only one of these factors is considered, such as the client bandwidth in Chapter 4, or selected-content access in Chapter 5, the formulation of virtual queues is somewhat harder but not much complicated. In the first case, a virtual queue contains a client's request and all other requests for the same video from clients with the same or higher bandwidth. In the second case, it contains all requests for the same video with close-by but later starting positions. When variations in both these factors are considered in addition to variation in the buffer space, such simple rules cannot be applied. For example, Client A may have higher bandwidth than client B but less buffer space and may have an earlier requested starting position.

To simplify the situation, we present the following procedure for formulating the virtual queues: (1) Start by forming large groups based on the requested data. In particular, group requests for the same video with requested starting positions within $aWait_{thr}$. The available resources are to be ignored at this stage. (2) Within each group, form a sub-group of requests from clients having sufficient buffer space for stream merging. This step is not complicated because the suitability of buffer space for stream merging techniques is not a function of the technique or the stream type but is a function of the playback position of the request (starting position in this case) and the data being delivered by the merging tree root, which is the regular stream into which the entire tree will eventually merge. (It is is the closest regular stream with delivered data later than the latest starting position in the group.) (3) Within each group in Step 2, form virtual queues based on the available download bandwidth. (4) Add to the set of virtual queues in Step 3 the larger groups formed in Step

1 (but not the ones in Step 2). Note that if the a group in Step 1 has more requests than its sub-group in Step 2, the resulting large virtual queue can be serviced by only a regular stream.

LVC, MDC, and Multi-Copy systems complicate further the scheduling. In particular, Step 1 should form the groups based not on the requested data but based on the quality,as well. That is the each large group should include those request with close by start positions and with similar targeted video quality based on their available bandwidths.

Figure 6.4 further explains these steps by an example for a single video waiting queue. The queue contains six requests (A to F) which vary in the requested starting position (SP), download bandwidth (BW), and buffer space (BS). We assume that $aWait_{threshold} = 90$ seconds and that one stream for the video is currently being serviced and delivering data at position 600 seconds. In this example, three high-level virtual queues (VQs) are formed, with the first containing requests A to D. Because of their BSs, requested SPs, and the currently delivered data, requests C and D cannot be served by a stream merging technique. Two additional VQs can be formed from this group: the first containing A and B (VQ 1.1) and the other containing only A (VQ 1.2) because A has higher bandwidth than B. VQ 1.0 can only be served by a regular stream starting at position 0.0, VQ 1.1 can be served by an ea-stream delivering data at rate $1.6r$ during the fast period, whereas VQ 1.2 can be served by a regular patch using ERMT/Patching. The objective function (Obj.) for each VQ is calculated as number of requests in the VQ divided by the cost of service in seconds. VQ 1.1 has the largest objective, but has to compete with the other possible VQs for the other videos (not shown). VQ 3.1 was produced from VQ 3.0 but has the same members as VQ 2.1 and thus is deleted.

6.6 Performance Evaluation

6.6.1 Terminology

EHS-E combined with all the enhancements in this paper is called *Interactive EHS-E* or succinctly *iEHS-E*, whereas EHS-E combined with the enhancements in Chapter 5 is called *iEHS-E (-)*. The minus sign indicates that the new enhancements in this paper are not incorporated.

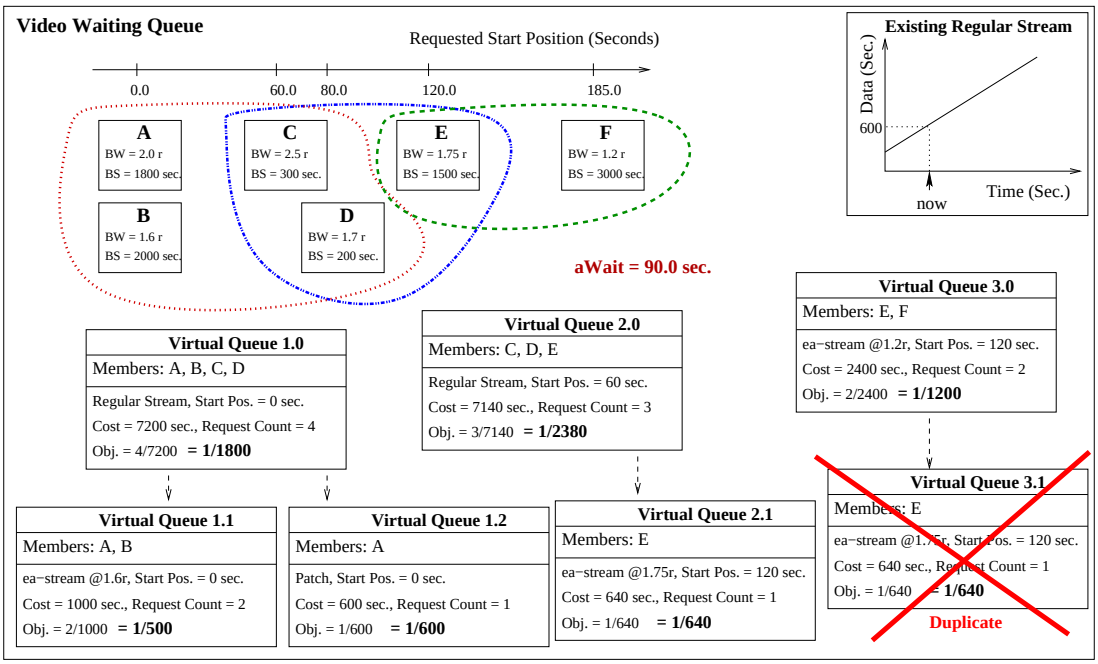


Figure 6.4: Illustration of Scheduling in Realistic Environments using MCF-P [Single Video, Single Description]

6.6.2 Methodology

We study the impacts of different workload parameters on resource sharing and analyze the effectiveness of SHS-E, EHS-E, iEHS-E(-), and the proposed iEHS-E scheme through extensive simulation. Patching variations (such as SHS-P, EHS-P, iEHS-P, and iEHS-P (-)) are not considered because of their lower performance compared with the variations using ERMT. We consider two service models: True Video-on-Demand (TVOD) and Near Video-on-Demand (NVOD). In NVOD, we fix the server bandwidth and consider primarily six performance metrics: *customer defection probability*, *average waiting time*, *video unfairness*, *starting-position unfairness*, *client-bandwidth unfairness*, and *buffer-space unfairness*. The defection probability is the most important and can be defined as the probability that customers leave the system without being serviced because of waiting times exceeding their tolerance. The average waiting time comes next in importance. It includes the client waiting time for admission and any possible post-admission waiting time. The unfairness

metrics quantify the bias against unpopular videos, unpopular requested video starting positions, clients with lower available download bandwidth, and clients with smaller available buffer space, respectively. They can be found as the standard deviation in the defection rate among various videos, starting-position classes, available bandwidth classes, or available buffer space classes, respectively. The used class widths are shown on Table 6.5. In TVOD, we compare the server bandwidth required to service all requests immediately. In addition, when evaluation the utilization of LVC and MDC, we introduce a new quality metric, defined as the average percentage of video data received by each client compared to the best quality the same client can receive in a unicast system with unlimited server bandwidth serving the exact same video library.

6.6.3 Workload Characteristics

Table 6.5 summarizes the main workload characteristics and shows the default values. Like most prior studies, we assume that the arrival of the requests follows a Poisson Process with an average arrival rate λ and the accesses to videos are highly localized and follow a Zipf-like distribution with skewness parameter $\theta_v = 0.271$. Generally, we characterize the waiting tolerance of customers by an exponential distribution with mean μ_{tol}. However, with iEHS-E, we utilize the *time-of-service guarantee* (TSG) model [3, 42, 47] whenever the actual playback time is known and guaranteed. With this model, if the current waiting time plus the additional buffering time is less than μ_{tol}, the customer waits; otherwise, the waiting tolerance follows an exponential distribution with mean μ_{tol}. The default value of μ_{tol} is $L/80$, where L is the video length. Unless otherwise indicated, we consider a server with 40 videos, each of which is 120-minute long.

We examine the server at different loads by fixing the request arrival rate at 60 requests per minute and varying the server bandwidth (server capacity) generally from $300r$ to $700r$. We also examine three levels of average client resources: low, medium, and high. Bandwidth and buffer space are assumed to follow truncated normal distributions with average values of $1.5r$ and $0.1L$ for the low resource set, $2.0r$ and $0.25L$ for the medium, and $2.5r$ and $0.5L$ for the high. In addition, we examine different levels of client interactivity by changing both the percentage of requests starting

from beginning of the video and the average viewing period. The higher the interactivity is, the lower is the percentage of requests starting from the beginning of the video, and the shorter the average viewing period. The LVC and MDC overheads are 10% and 20%, respectively. These include encoding and network overhead, and their choice are guided by values reported in [21] (and references within).

6.6.4 Result Presentation and Analysis

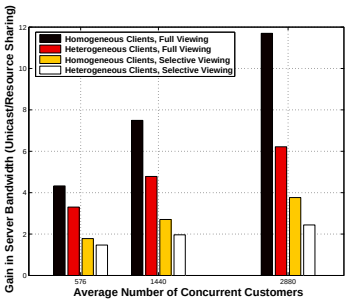
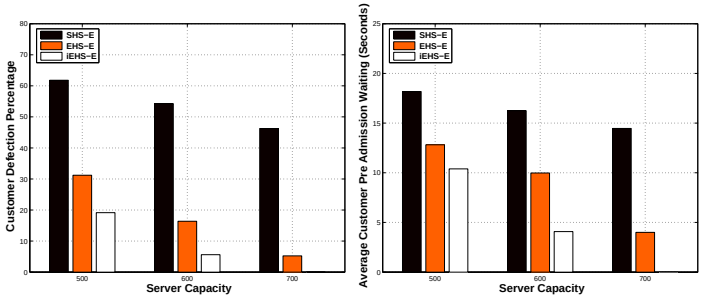


Figure 6.5: Impact of Workload on Resource Sharing [TVOD, EHS-E/ERMT]



(a) Defection (b) Pre-Admission Waiting

Figure 6.6: Effectiveness of Efficient Handling of Workload [Selected-Content Access, Heterogeneous Receivers]

Figure 6.5 shows the impact of the video access patterns (full-content access and selected-content access) and receiver types (homogeneous and heterogeneous) on performance. EHS-E is used in the heterogeneous case and ERMT otherwise. These results show that the gain from using EHS-E/ERMT compared to unicast in reducing the server bandwidth required for achieving TVOD is affected negatively by both client interactivity and receiver resources heterogeneity. Whereas aggressive resource sharing can reduce the required server bandwidth by up to 91% under full-content access and homogeneous receivers, the reduction is only around 50% under the more realistic setting with selected-content access and heterogeneous receivers. Although the savings in server bandwidth are much lower with realistic settings, these savings remain significant. The results also show that the access pattern has a stronger impact on performance than the type of receivers.

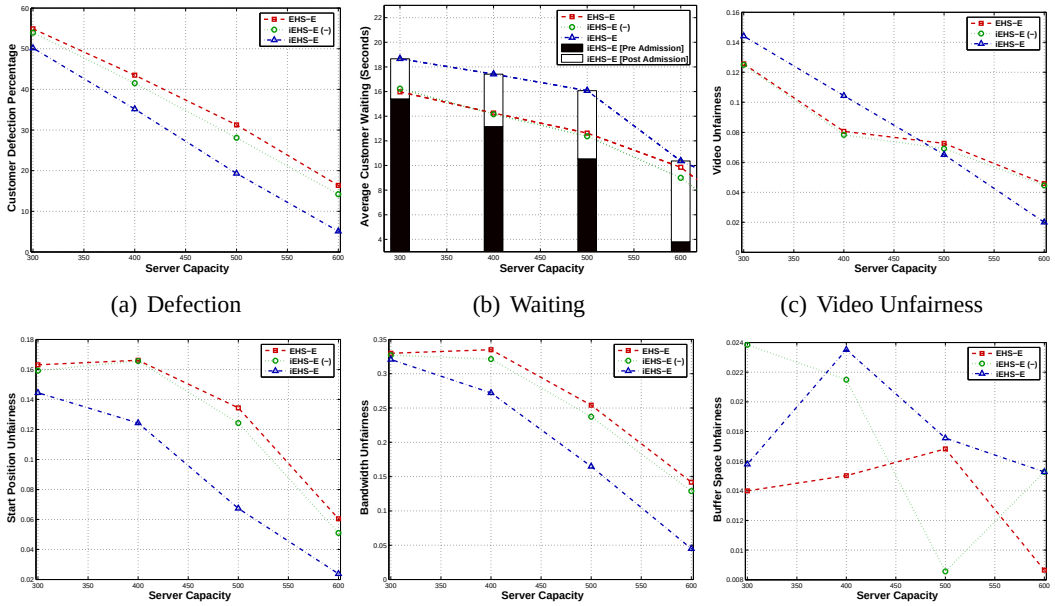
Table 6.5: Summary of Workload Characteristics

Category	**Parameter**	**Model/Value(s)**
Request Arrival	Model	Poisson Process
	Rate (λ)	40, 60, and 120, default: 60 Requests/min.
Server	Capacity	$300r$ to $700r$, default: $500r$
	Streaming Solution	SHS-E, EHS-E, iEHS-E(-) and iEHS-E
	Scheduling Policy	MCF-P
Design Parameters	Admit and Wait $aWait_{thr}$	$L/80$
	Yield and Resume δ_{resume}	2 seconds
	Maximum Reduced Quality Window W_Q	0, 5 minutes, $unlimited$, and $dynamic$
Video	Popularity	Zipf-Like ($\theta_v = 0.271$)
	Number (V)	40
	Length (L)	120 minutes
Waiting Tolerance	Model	Exponential with mean μ_{tol} = $L/80$ Time-of-Service Guarantee with threshold μ_{tol}
Client Bandwidth	Model	Normal with mean μ_{bw} = $1.5r$, $2.0r$(default), and $2.5r$, σ_{bw} = $0.5r$(default), and r (advanced video coding)
	Range	r **to** $11r$(default) and $0.5r$ to $11r$ (advanced video coding)
	Class width	$0.05r$
Client Buffer Space	Model	Normal with mean μ_{bs} = $0.1L$, $0.25L$ (default), and $0.5L$, σ_{bs}=$0.25L$
	Range	0.1L to L min
	Class width	$L/24$
Starting Position	Percentage at 0.0	10%, 20% (default), and 40%
	Earliest	0.0 seconds
	Latest	$0.9 \times L$
	Class width	$L/240$
	Class Popularity	Zipf (with $\theta_{sp} = 0$)
	In-Class Dist.	Uniform
Viewing Period	Model	Normal with mean μ_{vp} = $0.1L$, $0.2L$ (default), and $0.4L$, $\sigma_{vp} = 0.2L$
	Shortest	$L/100$
	Longest	L
Video Coding	Default	Single Description (SD) with Single Copy encoded at rate r
	Others	SD: Two Copies at rates $0.5r$ and r LVC: Two Layered, rates $0.55r$ each, (10% overhead) MDC: Two Descriptions, rates $0.6r$ each, (20% overhead)

** r is the encoding rate of the best quality single description copy of the video.*

Figure 6.6 demonstrates the importance of dealing intelligently with client heterogeneity and selected-content access. The considered metrics here are customer defection percentage and pre-admission waiting time. A distinction is made between post-admission and pre-admission because

the clients know precisely when to begin playback after admission. The overall waiting time will be discussed later in the section. SHS-E and EHS-E do not deal efficiently with selected-content access pattern, but EHS-E has the advantage of efficiently supporting heterogeneous receivers and thus performs significantly better than SHS-E. The proposed iEHS-E scheme deals efficiently with both heterogeneous receivers and selected content access, thereby outperforming the other two schemes.



(a) Defection (b) Waiting (c) Video Unfairness

(d) Start Position Unfairness (e) Bandwidth Unfairness (f) Buffer Space Unfairness

Figure 6.7: Effectiveness of iEHS-E [Selected-Content Access, Heterogeneous Receivers]

Figure 6.7 compares EHS-E, iEHS-E(-), and iEHS-E for different server capacities. As mentioned earlier, iEHS-E(-) is iEHS-E without the enhancements introduced in this paper. Thus, iEHS-E(-) still includes the enhancements that deal with selected-content access, particularly those introduced in Chapter 5. iEHS-E results in the highest server throughput and the lowest pre-admission waiting time, but has the longest overall waiting time. With iEHS-E, clients might need to wait after admission for few more seconds on the average before they start watching the desired content. iEHS-E generally reduces the unfairness against requests with less popular starting positions and clients with lower download bandwidth, and in some cases it also decreases the unfairness

against less popular videos. The results on unfairness against clients with small buffer space are non-conclusive; all three techniques show no real bias against clients with small buffer space. The output values are very small and variations in them can be attributed to the tolerance in simulation accuracy.

iEHS-E(-) performs slightly better than EHS-E because "Adopt Earlier Stream" loses most of its potential benefits due to client heterogeneity. As mentioned earlier, unlike the case with homogeneous receivers (as in Chapter 5), it is only possible to adopt a smaller subset of earlier streams because of the restrictions caused by lower client resources.

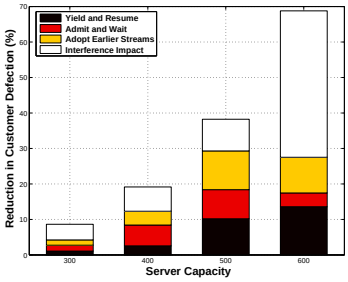
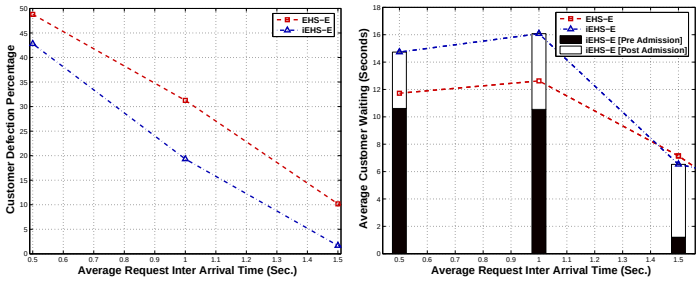


Figure 6.8: Contributions of Various Enhancements on Throughput [iEHS-E, Selected-Content Access, Heterogeneous Receivers]

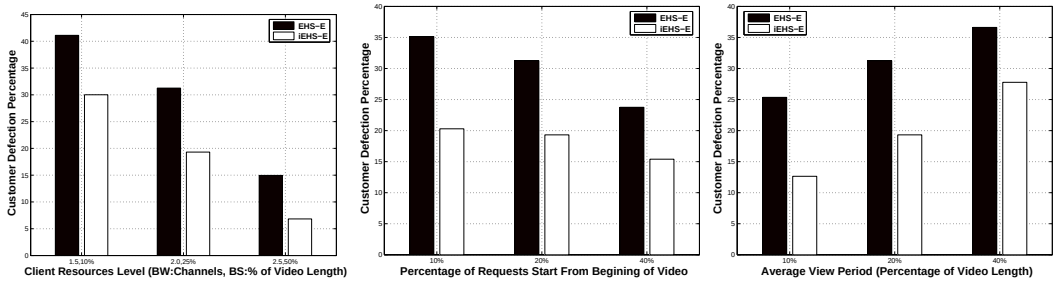


(a) Defection (b) Waiting

Figure 6.9: Impact of Request Rate [Selected-Content Access, Heterogeneous Receivers]

Figure 6.8 details the impacts of the individual iEHS-E enhancements. Although "Yield and Resume" tends to have the strongest impact among the other enhancements, when applied alone, the strongest impact is actually the interference caused by applying all enhancements together. In other words, the sum of improvements when each enhancement is applied individually is less than the improvement when all of them are applied together. For example, admitting the client early and making it wait prevents that stream from yielding even if its initiating request leaves after that. Without "Admit and Wait" it would have yielded and eliminated the possibility of cost free service for the waiting client.

Figure 6.9 compares EHS-E, iEHS-E(-), and iEHS-E for different request rates. The results are similar to those in Figure 6.7. iEHS-E keeps its lead in server throughput and pre-admission waiting, but it increases the overall service latency.



(a) Client Resources (b) Start Position (c) Viewing Period

Figure 6.10: Impact of Workload Parameters [Selected-Content Access, Heterogeneous Receivers]

Figure 6.10 demonstrates the advantage of using iEHS-E over EHS-E for different workload parameters. Figure 6.10(a), in particular, compares the performance under three different level of client resources: Low, medium, and high. In the three levels, clients are heterogeneous but the average values of resources are changed. These results clearly demonstrate that iEHS-E's provides the best overall performance regardless of the level of receiver resources. Figures 6.10(b) and 6.10(c) compare the performance under different levels of client interactivity. Obviously, the higher the interactivity is, the higher is the gain achieved by using iEHS-E, but iEHS-E remains a better choice under all levels of interactivity.

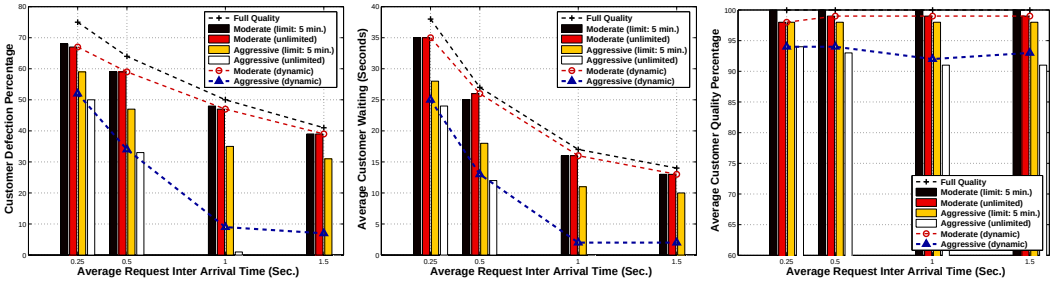
Figures 6.11 to 6.13 compare the results when utilizing LVC. As expected, the aggressive approach in general, achieves the highest throughput, the lowest service latency, but on the expense of lower average video quality, especially when no upper limit is imposed on the reduced quality period. The *aggressive(unlimited)* is able to increase throughput and to reduce service latency significantly. Compared to the *Full Quality*, it is able, in some cases, to double the throughput and fully eliminate a service latency that exceeds 15 seconds, at the same time. Interestingly, with the unlimited aggressive the quality never dropped below 90%, and it achieves so without even fully utilize the server bandwidth. The server bandwidth usage with the unlimited aggressive alternative

ranges between 72% and 100% of that used by the Full Quality alternative. In other words, the same server was able to serve double the requests, with less server bandwidth, while each client received only 10% less data in average. The unlimited aggressive alternative is the best alternative when increasing throughput and lowering the server bandwidth usage are the two main objectives of the system. The dynamic aggressive is an alternative that provides close but slightly lower throughput, but it tries to utilize the server bandwidth to provide better video quality in regions closer to TVOD. The impact of limiting the reduced quality period is greater with the aggressive approach than with the moderate approach due to the larger number of opportunities of quality reduction with former.

Figures 6.14 to 6.16 compare the results when utilizing MDC (for the same seven alternatives studied with LVC). Similar to the situation with LVC, both aggressive (dynamic) and aggressive (unlimited) reduce the defection probability and average waiting time significantly compared to the other five alternatives, but on the expense of the average played quality. Again, the dynamic alternative provides an acceptable compromise when taking all factors together. It results in slightly higher defection compared to the unlimited, but it provides much better quality, especially for regions of operation closer to TVOD. But the unlimited aggressive is the choice if throughput and load on server are the main concern and justify small reduction in played quality.

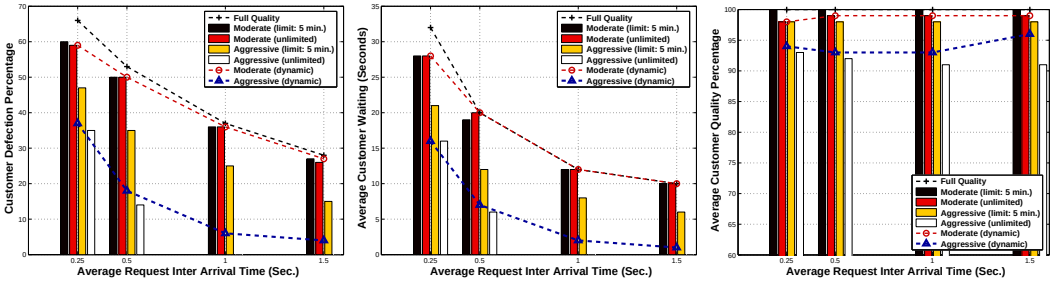
Figures 6.17 to 6.19 compare the best of LVC and MDC alternatives, as well as, the Full Quality alternatives in all three coding options. Clearly, the two considered aggressive LVC alternatives are the best performers. They both result in significantly lower defection rate and average waiting time compared to all other combinations. And as discussed earlier, un-limiting the quality reduction period improves throughput and service latency, while imposing a dynamic limit based on workload and load on server can improve the average played quality, but slightly hurts throughput and service latency. Unsurprisingly, the best two alternatives are, in general, the best in all unfairness metrics. The more the efficiency of the approach, the lesser the competition in the system between the different classes of clients. Therefore, the lower the unfairness.

Interestingly, while MDC provides more freedom in choosing which descriptions to serve a particular client, while with LVC layers has to be picked in order, LVC performed much better than



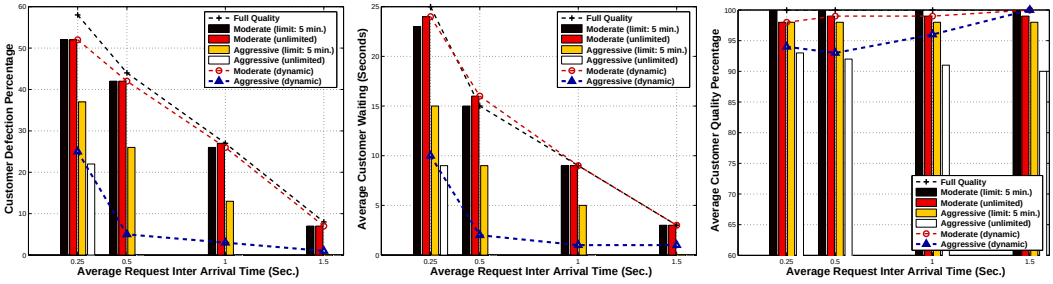
(a) Defection (b) Waiting (c) Quality

Figure 6.11: Impact of Utilizing LVC [iEHS-E, Selected-Content Access, Heterogeneous Receivers, Client Bandwidth($\sigma_{bw} = r$, range: $0.5r$ to $11r$), 300 Server Channels]



(a) Defection (b) Waiting (c) Quality

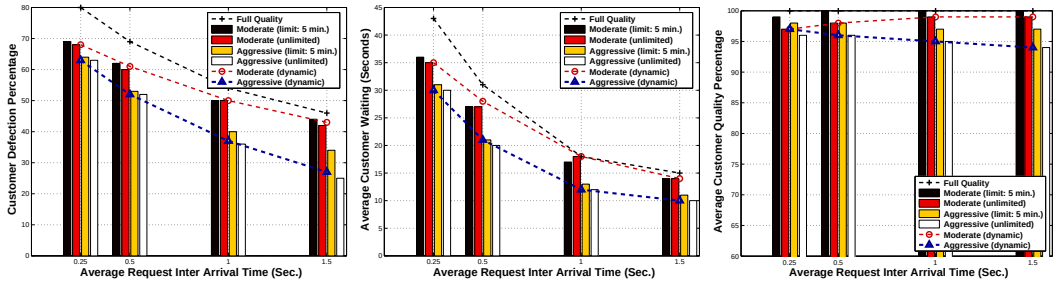
Figure 6.12: Impact of Utilizing LVC [iEHS-E, Selected-Content Access, Heterogeneous Receivers, Client Bandwidth($\sigma_{bw} = r$, range: $0.5r$ to $11r$), 400 Server Channels]



(a) Defection (b) Waiting (c) Quality

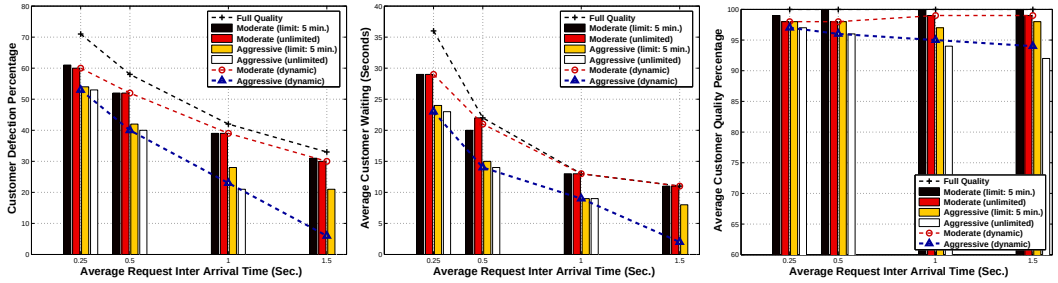
Figure 6.13: Impact of Utilizing LVC [iEHS-E, Selected-Content Access, Heterogeneous Receivers, Client Bandwidth($\sigma_{bw} = r$, range: $0.5r$ to $11r$), 500 Server Channels]

MDC. The difference is not only due to the larger overhead in MDC, but also due to the nature of resource sharing. While more freedom seems more attractive in reducing the cost serving a single request, it can hurt the global cost. With stream merging techniques, the relation between TVOD



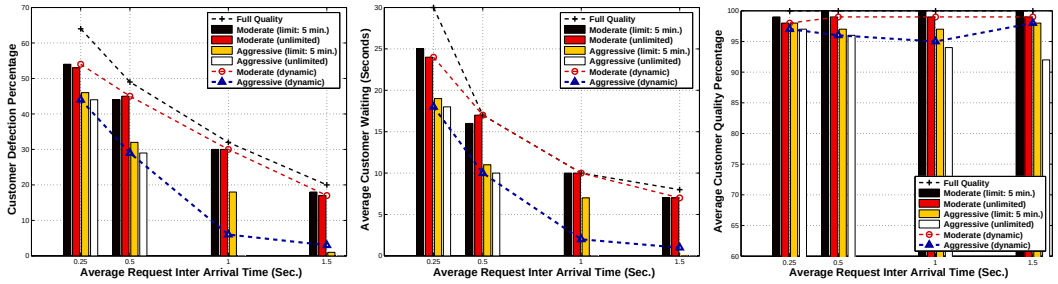
(a) Defection (b) Waiting (c) Quality

Figure 6.14: Impact of Utilizing MDC [iEHS-E, Selected-Content Access, Heterogeneous Receivers, Client Bandwidth($\sigma_{bw} = r$, range: $0.5r$ to $11r$), 300 Server Channels]



(a) Defection (b) Waiting (c) Quality

Figure 6.15: Impact of Utilizing MDC [iEHS-E, Selected-Content Access, Heterogeneous Receivers, Client Bandwidth($\sigma_{bw} = r$, range: $0.5r$ to $11r$), 400 Server Channels]

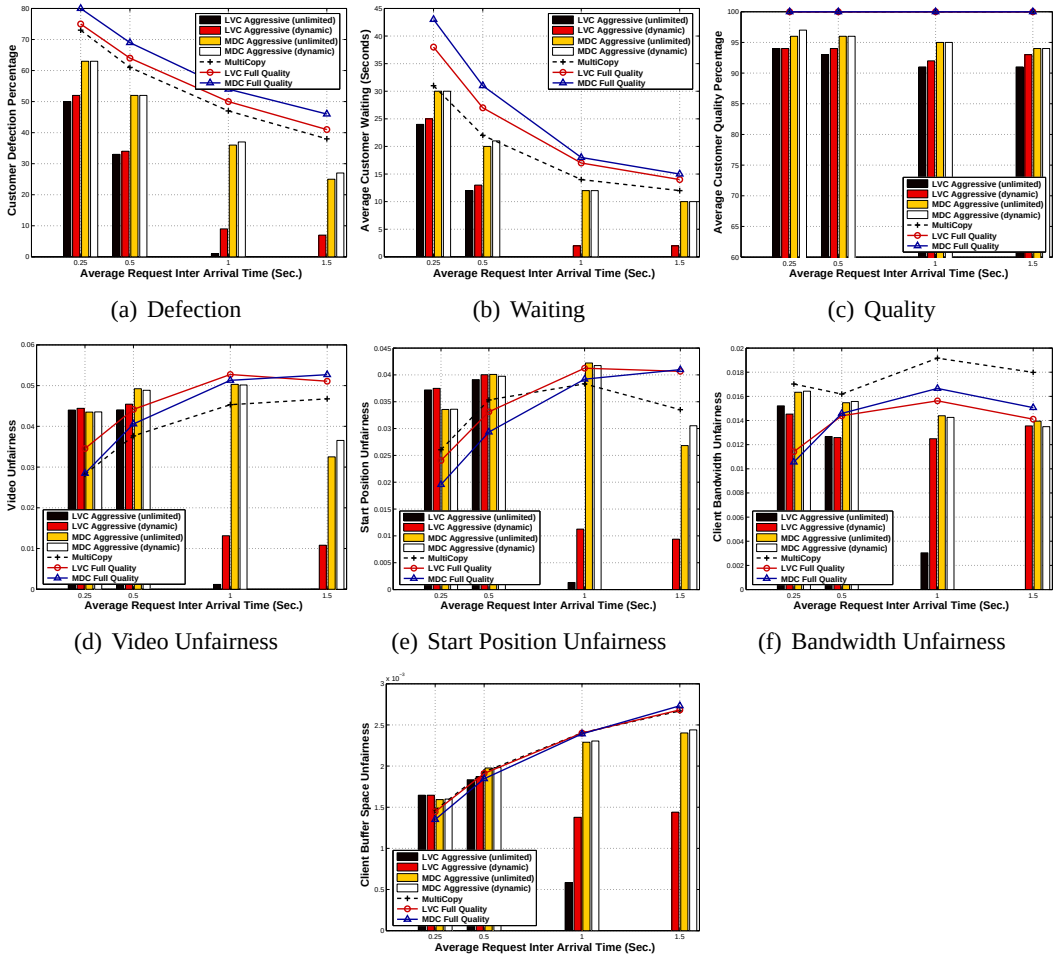


(a) Defection (b) Waiting (c) Quality

Figure 6.16: Impact of Utilizing MDC [iEHS-E, Selected-Content Access, Heterogeneous Receivers, Client Bandwidth($\sigma_{bw} = r$, range: $0.5r$ to $11r$), 500 Server Channels]

server bandwidth needed to serve a description and the number of concurrent requests receiving that description is a *square root* with Patching and ea-streams, and *logarithmic* with ERMT. Therefore, increasing skewness in the number of concurrent customers across videos or descriptions, while

keeping the total number, increases the data shareability and, as a result, reduces the required server bandwidth to provide TVOD (or reduces the defection and service latency in NVOD). So for a given video, the highest cost is when all descriptions of that video receive the same number of concurrent customers (of course, unless it is the only option when all customers capable of receiving all descriptions), and that what MDC tries to achieve over LVC. LVC is skewed by nature towards lower layers due to the dependency of a higher layer on the slower ones.

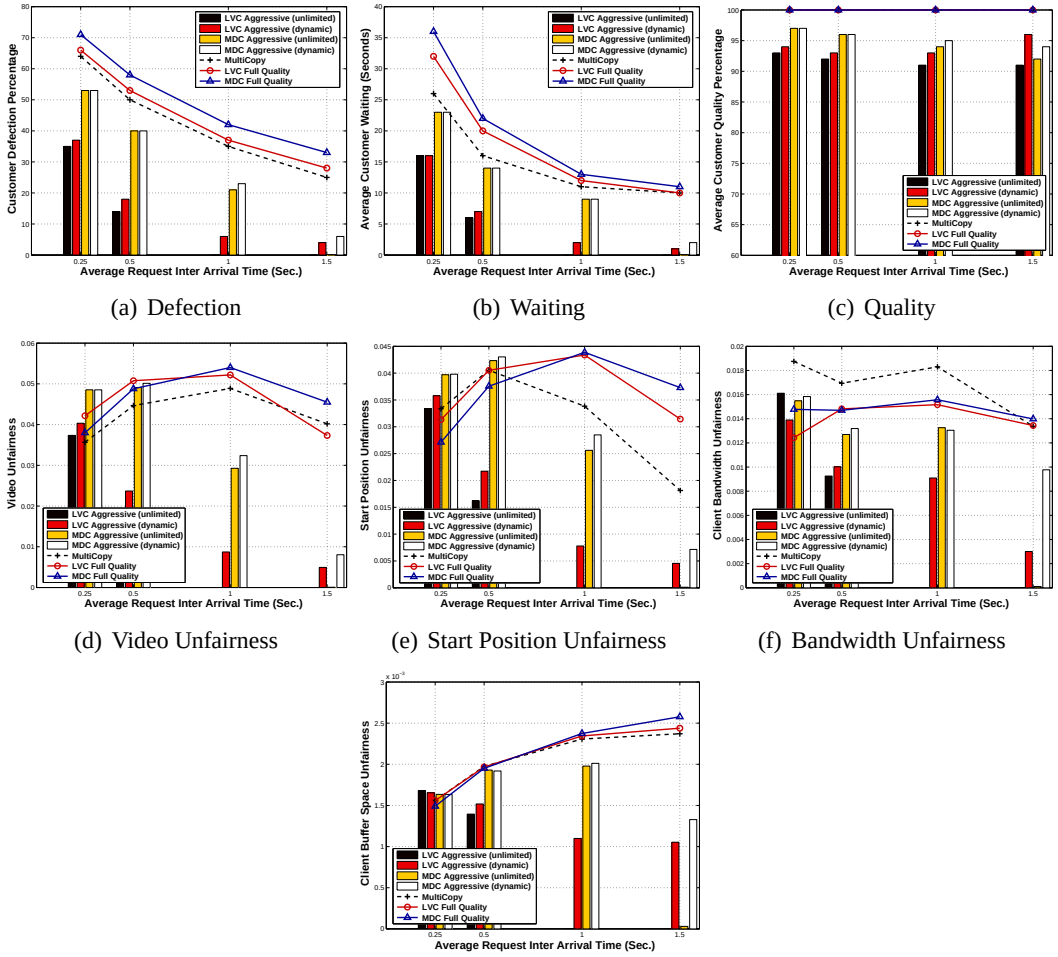


(a) Defection (b) Waiting (c) Quality

(d) Video Unfairness (e) Start Position Unfairness (f) Bandwidth Unfairness

(g) Buffer Space Unfairness

Figure 6.17: Impact of Utilizing LVC & MDC [iEHS-E, Selected-Content Access, Heterogeneous Receivers, Client Bandwidth($\sigma_{bw} = r$, range: $0.5r$ to $11r$), 300 Server Channels]



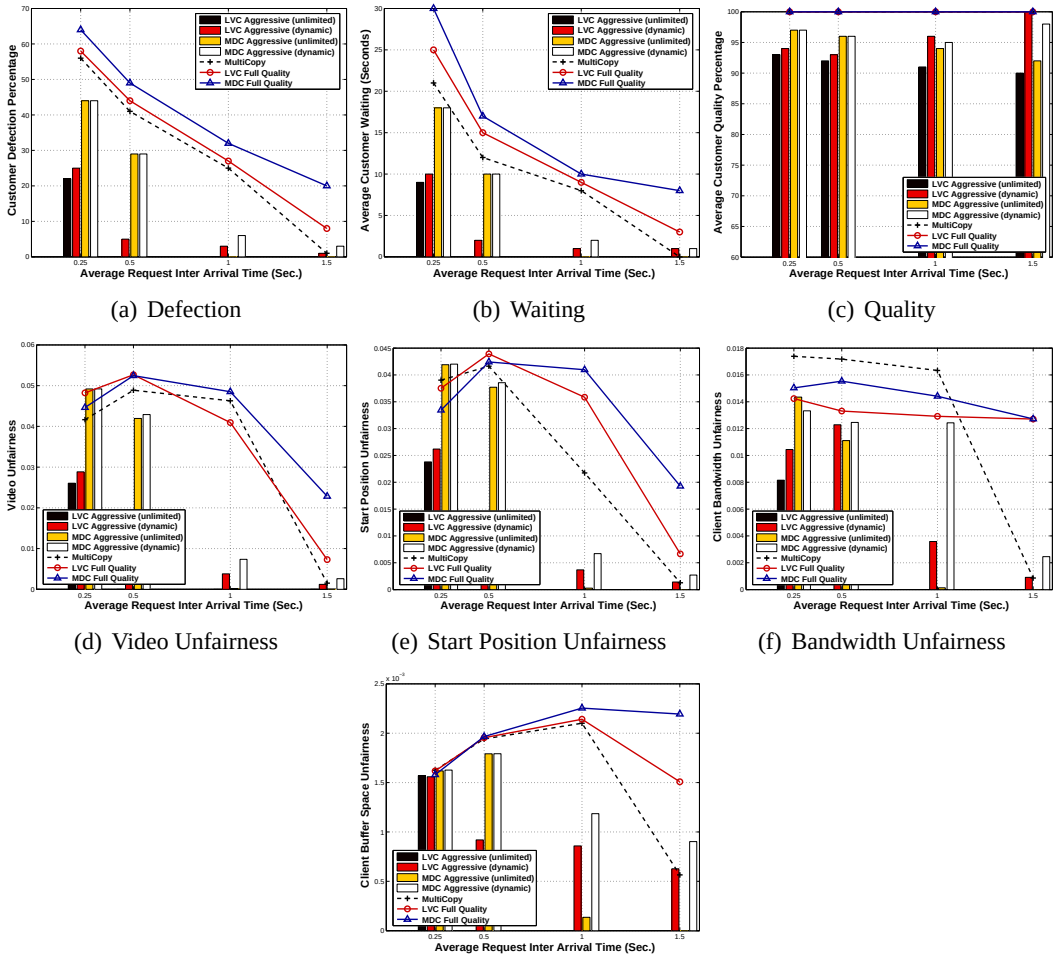
(a) Defection (b) Waiting (c) Quality

(d) Video Unfairness (e) Start Position Unfairness (f) Bandwidth Unfairness

(g) Buffer Space Unfairness

Figure 6.18: Impact of Utilizing LVC & MDC [iEHS-E, Selected-Content Access, Heterogeneous Receivers, Client Bandwidth($\sigma_{bw} = r$, range: $0.5r$ to $11r$), 400 Server Channels]

Finally, Figure 6.20 compares the alternatives in a TVOD service. Interestingly, despite its encoding overhead, LVC, combined with the unlimited aggressive approach, is capable of reducing the required server bandwidth to provide TVOD service by 52% to 58%, while the average received data by each client is reduced by only 8% to 10%.



(a) Defection (b) Waiting (c) Quality

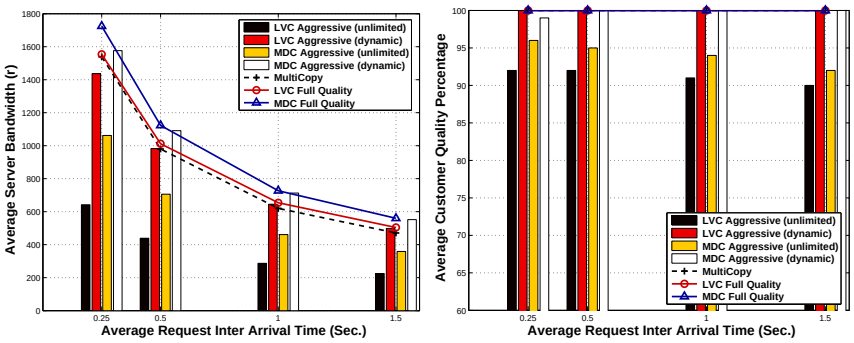
(d) Video Unfairness (e) Start Position Unfairness (f) Bandwidth Unfairness

(g) Buffer Space Unfairness

Figure 6.19: Impact of Utilizing LVC & MDC [iEHS-E, Selected-Content Access, Heterogeneous Receivers, Client Bandwidth($\sigma_{bw} = r$, range: $0.5r$ to $11r$), 500 Server Channels]

6.7 Conclusion

We have analyzed the impact of client heterogeneity and selected-content access on the shareability of multimedia data and have proposed three enhancements to utilize such opportunities. In addition, we have investigated utilizing more advanced video coding with stream merging techniques for better serving heterogeneous receivers, and we have proposed several approaches in doing so. The main results can be summarized as follows. (1) The combined impact of client



(a) Server Capacity (b) Quality

Figure 6.20: Impact of Utilizing LVC & MDC [iEHS-E, Selected-Content Access, Heterogeneous Receivers, Client Bandwidth($\sigma_{bw} = r$, range: $0.5r$ to $11r$), TVOD]

heterogeneity and selected-content access can reduce the shareability of multimedia data by a factor of six. (2) The access pattern has a stronger impact on resources sharing than the type of receivers. (3) Resource sharing that deals properly with client heterogeneity and interactivity can significantly improve the quality-of-service compared to traditional resource sharing. (4) iEHS-E can serve up to 70% of those clients who would have defected when EHS-E is used. (5) iEHS-E significantly increases the server throughput for all studied server capacities, request rates, client resource levels, and client interactivity levels. (6) While iEHS-E reduces the pre-admission waiting times, it introduces some post-admission waiting, resulting in longer average total waiting. (7) Utilizing advanced video coding intelligently, can significantly improves throughput, reduces service latency, and reduces unfairness across different client classes. (8) When LVC or MDC are utilized, the *Aggressive* approach is the best alternative when load on the server and throughput are the most important metrics. It can, in some cases, double the throughput and fully eliminate the service latency, while keeping average quality above 90% compared to the *Full Quality* alternative. (9) With the aggressive approach, un-limiting the quality reduction period achieves the highest throughput and the lowest service latency, while imposing a dynamic limit based on workload and load on server can improve the average played quality, but slightly hurts throughput and service latency, compared to the unlimited period. (10) While MDC provides more freedom compared to LVC in choosing

which descriptions to serve a particular client, LVC performs much better than MDC. The difference is not only due to the larger overhead in MDC, but also due to the nature of resource sharing. In fact, the skewness of LVC towards lower layers increases the data shareability and reduces, as a result, the required server bandwidth to provide TVOD (or reduces the defection and service latency in NVOD). (11) Despite its encoding overhead, LVC combined with the unlimited aggressive approach is capable of reducing the required server bandwidth to provide TVOD service by 52% to 58%, while the average received data by each client is reduced by only 8% to 10%.

CHAPTER 7

SUMMARY AND FUTURE WORK

In the this study, we provided a detailed analysis of resource sharing techniques, considering both the *True Video-on-Demand* (TVOD) and the *Near Video-on-Demand* (NVOD) models and two video workloads: *mixed-video* and *hot-video*. Then we proposed an efficient *Workload-Aware Hybrid Solution* (WAHS) that combines the advantages of stream merging and periodic broadcasting, which is highly adaptive to variations in the workload and the available system resources. We also developed and analyzed a *Statistical Cache Management* (SCM) approach, which easy to implement and incurs small overhead as updates are triggered only when the workload varies considerably.

In addition, we studied how to support heterogeneous receivers while delivering video streams in a client-pull fashion. We proposed three solutions to address the variability of the download bandwidth among clients: *Simple Hybrid Solution* (SHS), *Adaptive Hybrid Solution* (AHS), and *Enhanced Hybrid Solution* (EHS). Based on whether Patching or ERMT is used for serving clients with download bandwidths equal or greater than double the video playback rate, the three solutions result in six schemes (variants): SHS-P, SHS-E, AHS-P, AHS-E, EHS-P, EHS-E. We studied how to address the variations in client bandwidth during a session, and we studied the support for the variability in the available buffer space among clients.

Moreover, we studied the impact of selected-content access on streaming servers delivering data in a client-pull fashion using stream merging techniques. We proposed several enhancements to reduce server load and improve the client perceived quality-of-service (QoS).

Furthermore, we investigated utilizing Layered Video Coding (LVC) and Multiple Description Coding (MDC) with advance stream merging techniques for better serving heterogeneous receivers.

We also studied how the waiting playback requests for different videos can be scheduled for service in all mentioned environments, capturing the variations in client bandwidth, buffer space, and requested start position.

The main results can be summarized as follows.

- In environments with *homogeneous* receivers and *full-content access*, and among existing techniques, ERMT generally performs the best with workloads containing both hot and cold videos, while FB generally performs the best with workloads containing only hot videos.

- The proposed WAHS outperforms both ERMT and FB in server throughput. It can eliminate more than half the defections compared to the best alternative among ERMT and FB, and more than 70% compared to the other. WAHS also outperforms FB in the average customer waiting time. ERMT can lead to shorter waiting times, but it does not provide time-of-service guarantees while WAHS provides these guarantees to the majority of customers.

- SCM is very effective in further reducing the disk I/O bandwidth requirements. The reductions can be up to 18% and 30%, when the cache sizes are 2% and 6% of the total size of videos, respectively.

- In environments with *heterogeneous* receivers, AHS significantly outperforms SHS and Batching, however EHS achieves significant improvements over AHS under all workloads and service models. In addition, EHSE can provide a TVOD service (i.e., zero waiting time), whereas AHS cannot. EHS is also more tolerant to variations in client bandwidth during service. With the same server bandwidth, EHS-E can achieve zero defections, while AHS-E, SHS-E, and Batching cause 13%, 45%, and 53% defections, respectively. The three proposed solutions for heterogeneous receivers vary significantly in performance, whereas the two variants of each perform close to one other. In other words, using ERMT instead of Patching for serving clients with bandwidth capacities of $2r$ or higher is of less significance than changing the solution (and thus the method of service for clients with bandwidth capacities higher than r

but lower than $2r$). Client available buffer space, in general, has less impact on system performance than client bandwidth, but its impact can be very significant if it becomes the main bottleneck. The performance depends greatly on the applied scheduling policy. For example, with EHS-E, choosing the fair FCFS instead of the efficient, cost-oriented MCF-P can increase the number of defected customers by more than 50%. In certain situations, MQL performs better than MCF-P but only when SHS is used.

- When *selected-content access* is considered, in addition to the receiver *heterogeneity*, EHS-E combined with the proposed enhancement to deal with selected-content access (*interactive* EHS-E or iEHS-E) can serve up to 70% of those clients who would have defected when EHS-E is used without these enhancements. iEHS-E significantly increases the server throughput for all studied server capacities, request rates, client resource levels, and client interactivity levels.

- Utilizing advanced video coding intelligently can significantly improve throughput, reduce service latency, and reduce unfairness across different client classes. In particular, when LVC or MDC are utilized with the *Aggressive* quality reduction approach (with un-limiting the quality reduction period or imposing a dynamic limit), they can, in some cases, double the throughput and fully eliminate the service latency, while keeping the average quality above 90% compared to the *Full Quality* alternative.

- While MDC provides more freedom compared to LVC in choosing which descriptions to serve a particular client, LVC performs much better than MDC. The difference is not only due to the larger overhead in MDC, but also due to the nature of resource sharing. In fact, the skewness of LVC towards lower layers increases the data shareability and reduces, as a result, the required server bandwidth to provide TVOD (or reduces the defection and service latency in NVOD).

- Despite its encoding overhead, LVC combined with the unlimited aggressive approach, is capable of reducing the required server bandwidth to provide TVOD service by 52% to 58%, while the average received data by each client is reduced by only 8% to 10%.

We conclude that when all receivers have download bandwidth capacities of double the video playback rate or higher and when videos are always watched from the beginning to the end(i.e., full-content access), WAHS is the best alternative. However, in all other environments (i.e., heterogeneous receivers and/or selected-content access) iEHS-E is the best overall performer. Note that iEHS-E converges to EHS-E if used in an environment of full-content access with heterogeneous receivers, converges to iERMT (interactive ERMT) if used in an environment of selected-content access with homogeneous receivers, and converges to ERMT if used in an environment of full-content access with homogeneous receivers. When iEHS-E is employed, it is best to schedule the waiting requests using MCF-P. SCM can further further reduce the required disk I/O bandwidth.

In future work, we plan to study video streaming to wireless, portable, and mobile devices. Besides the impact of their download bandwidth and its characteristics on stream merging, especially during mobility, we plan to investigate the impact of stream merging solutions on power consumption and battery lifetime.

REFERENCES

[1] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. On optimal piggyback merging policies for Video-On-Demand systems. In *Proc. of ACM SIGMETRICS*, pages 200–209, May 1996.

[2] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. The maximum factor queue length batching scheme for Video-on-Demand systems. *IEEE Trans. on Computers*, 50(2):97–110, Feb. 2001.

[3] M. Alsmirat, M. Al-Hadrusi, and N. J. Sarhan. Analysis of waiting-time predictability in scalable media streaming. In *Proc. of ACM Multimedia*, pages 791 – 794, Sept. 2007.

[4] O. Bagouet, K. A. Hua, and D. Oger. A periodic broadcast protocol for heterogeneous receivers. In *Proc. of Multimedia Computing and Networking Conf. (MMCN)*, Jan. 2003.

[5] M. Bradshaw, B. Wang, S. Sen, L. Gao, J. Kurose, P. Shenoy, and D. Towsley. Periodic broadcast and patching services: Implementation, measurement and analysis in an Internet streaming media testbed. In *Proc. of ACM Multimedia*, pages 280–290, Oct. 2001.

[6] BroadBandReports. `http://broadbandreports.com/`.

[7] Y. Cai, K. Hua, and K. Vu. Optimizing patching performance. In *Proc. of Multimedia Computing and Networking Conf. (MMCN)*, pages 204–215, Jan. 1999.

[8] Y. Cai and K. A. Hua. An efficient bandwidth-sharing technique for true video on demand systems. In *Proc. of ACM Multimedia*, pages 211–214, Oct. 1999.

[9] Y. Cai and K. A. Hua. Sharing multicast videos using patching streams. *Multimedia Tools and Applications journal*, 21(2):125–146, Nov. 2003.

[10] Y. Cai, W. Tavanapong, and K. A. Hua. Enhancing patching performance through double patching. In *Proc. of 9th Intl Conf. on Distributed Multimedia Systems*, pages 72–77, Sept. 2003.

[11] S. W. Carter and D. D. E. Long. Improving Video-on-Demand server efficiency through stream tapping. In *the International Conference on Computer Communication and Networks (IC-CCN)*, pages 200–207, Sept. 1997.

[12] L. Cherkasova and M. Gupta. Characterizing locality, evolution, and life span of accesses in enterprise media server workloads. In *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video (NOSSDAV)*, pages 33–42, May 2002.

[13] E. Coffman, P. Jelenkovic, and P. Momcilovic. The dyadic stream merging algorithm. *Journal of Algorithms*, 43(18):120–137, April 2002.

[14] C. Costa, I. Cunha, A. Borges, C. Ramos, M. Rocha, J. Almeida, and B. Ribeiro-Neto. Analyzing client interactivity in streaming media. In *Proc. of The World Wide Web Conf.*, pages 534–543, May 2004.

[15] A. Dan, D. M. Dias, R. Mukherjee, D. Sitaram, and R. Tewari. Buffering and caching in large-scale video servers. In *Digest of Papers. IEEE Int'l Computer Conf.*, pages 217–225, March 1995.

[16] A. Dan, P. Shahabuddin, D. Sitaram, and D. Towsley. Channel allocation under batching and vcr control in movie-on-demand servers. *Journal of Parallel and Distributed Computing*, 30(2):168–179, Nov. 1995.

[17] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling policies for an on-demand video server with batching. In *Proc. of ACM Multimedia*, pages 391–398, Oct. 1994.

[18] D. L. Eager, M. K. Vernon, and J. Zahorjan. Optimal and efficient merging schedules for Video-on-Demand servers. In *Proc. of ACM Multimedia*, pages 199–202, Oct. 1999.

[19] D. L. Eager, M. K. Vernon, and J. Zahorjan. Bandwidth skimming: A technique for cost-effective Video-on-Demand. In *Proc. of Multimedia Computing and Networking Conf. (MMCN)*, pages 206–215, Jan. 2000.

[20] D. L. Eager, M. K. Vernon, and J. Zahorjan. Minimizing bandwidth requirements for on-demand data delivery. *IEEE Trans. on Knowledge and Data Engineering*, 13(5):742–757, Sept. 2001.

[21] F. Fitzek, B. Can, R. Prasad, and M. Katz. Overhead and quality measurements for multiple description coding for video services. In *Proc. of Wireless Personal Multimedia Communications*, pages 524 – 528, Sept. 2004.

[22] L. Gao, J. Kurose, and D. Towsley. Efficient schemes for broadcasting popular videos. In *Proc. of the Int'l Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, July 1998.

[23] L. Gao and D. Towsley. Supplying instantaneous Video-on-Demand services using controlled multicast. In *Proc. of IEEE Multimedia Computing and Systems*, pages 117–121, June 1999.

[24] L. Gao, Z.-L. Zhang, and D. F. Towsley. Catching and selective catching: efficient latency reduction techniques for delivering continuous multimedia streams. In *Proc. of ACM Multimedia*, pages 203–206, Oct. 1999.

[25] L. Golubchik, J. C. S. Lui, and R. Muntz. Reducing I/O demand in Video-On-Demand storage servers. In *Proc. of ACM SIGMETRICS*, pages 25–36, May 1995.

[26] L. Golubchik and R. Muntz. Adaptive piggybacking: A novel technique for data sharing in Video-On-Demand storage servers. *ACM Multimedia Systems Journal*, 4(3):140–155, 1996.

[27] V. K. Goyal. Multiple description coding: Compression meets the network. *IEEE Signal Processing Magazine*, pages 74–93, September 2001.

[28] J. Gray and P. Shenoy. Rules of thumb in data engineering. In *Proc. of the IEEE International Conference on Data Engineering*, Feb 2000.

[29] C. Griwodz, M. Liepert, M. Zink, and R. Steinmetz. Tune to Lambda Patching. *ACM Performance Evaluation Review*, 27(4):20–26, March 2000.

[30] A. Hu. Video-on-Demand broadcasting protocols: A comprehensive study. In *Proc. of IEEE INFOCOM*, April 2001.

[31] K. A. Hua, Y. Cai, and S. Sheu. Patching: A multicast technique for true Video-on-Demand services. In *Proc. of ACM Multimedia*, pages 191–200, 1998.

[32] K. A. Hua, J. Oh, and K. Vu. An adaptive video multicast scheme for varying workloads. *Multimedia Systems*, 8(4):258–269, 2002.

[33] K. A. Hua and S. Sheu. Skyscraper broadcasting: A new broadcasting scheme for metropolitan Video-on-Demand system. In *Proc. of ACM SIGCOMM*, pages 89–100, Sept. 1997.

[34] C. Huang, R. Janakiraman, and L. Xu. Loss-resilient on-demand media streaming using priority encoding. In *Proc. of ACM Multimedia*, pages 152–159, Oct. 2004.

[35] L. Juhn and L. Tseng. Harmonic broadcasting for Video-on-Demand service. *IEEE Trans. on Broadcasting*, 43(3):268–271, Sept. 1997.

[36] L. Juhn and L. Tseng. Fast data broadcasting and receiving scheme for popular video service. *IEEE Trans. on Broadcasting*, 44(1):100 – 105, March 1998.

[37] J. Liu and J. Xu. Proxy caching for media streaming over the Internet. *IEEE Communications Magazine*, 42(8):88–94, Aug. 2004.

[38] J.-F. Pâris. A fixed-delay broadcasting protocol for Video-on-Demand. In *Proc. of the Int'l Conf. on Computer Communications and Networks*, pages 418–423, 2001.

[39] J.-F. Pâris, S. W. Carter, and D. D. E. Long. Efficient broadcasting protocols for video on demand. In *Proc. of the Int'l Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 127–132, July 1998.

[40] M. Rocha, M. Maia, I. Cunha, J. Almeida, and S. Campos. Scalable media streaming to interactive users. In *Proc. of ACM Multimedia*, pages 966–975, Nov. 2005.

[41] N. J. Sarhan and C. R. Das. Caching and scheduling in NAD-based multimedia servers. *IEEE Trans. on Parallel and Distributed Systems*, 15(10):921–933, Oct. 2004.

[42] N. J. Sarhan and C. R. Das. A new class of scheduling policies for providing time of service guarantees in Video-On-Demand servers. In *Proc. of the 7th IFIP/IEEE Int'l Conf. on Management of Multimedia Networks and Services*, pages 127–139, Oct. 2004.

[43] N. J. Sarhan and B. Qudah. Efficient cost-based scheduling for scalable media streaming. In *Proc. of Multimedia Computing and Networking Conf. (MMCN)*, page 65040C, Jan./Feb. 2007.

[44] P. Sessini, L. Shi, A. Mahanti, Z. Li, and D. L. Eager. Scalable streaming for heterogeneous clients. In *Proc. of ACM Multimedia*, pages 337–346, Oct. 2006.

[45] M. A. Tantaoui, K. A. Hua, and T. T. Do. Broadcatch: A periodic broadcast technique for heterogeneous Video-on-Demand. *IEEE Trans. on Broadcasting*, 50(3):289–301, Sept. 2004.

[46] Y.-C. Tseng, C.-M. Hsieh, M.-H. Yang, W.-H. Liao, and J.-P. Sheu. Data broadcasting and seamless channel transition for highly-demanded videos. In *Proc. of IEEE INFOCOM*, pages 727–736, March 2000.

[47] A. K. Tsiolis and M. K. Vernon. Group-guaranteed channel capacity in multimedia storage servers. In *Proc. of ACM SIGMETRICS*, pages 285–297, June 1997.

[48] S. Viswanathan and T. Imielinski. Pyramid broadcasting for video on demand service. In *Proc. of Multimedia Computing and Networking Conf. (MMCN)*, 1995.

[49] Y. Wang, A. R. Reibman, and S. Lin. Multiple description coding for video delivery. *Proceedings of the IEEE*, 93(1):57–70, January 2005.

[50] YouTube. `http://youtube.com`.

ABSTRACT

ENHANCED RESOURCE SHARING FOR SCALABLE VIDEO-ON-DEMAND SERVICES

by

BASHAR QUDAH

May 2009

Advisor: Dr. Nabil Sarhan

Major: Computer Engineering

Degree: Doctor of Philosophy

The interest in scalable video streaming has increased dramatically. Unfortunately, the number of video streams that can be supported concurrently is highly constrained by the required real-time and high-rate transfers, which quickly consume server and network resources, including network bandwidth and disk I/O bandwidth. Resource sharing techniques face this challenge by utilizing the multicast facility. The main classes of these techniques are *Batching*, *stream merging*, *periodic broadcasting*, and *composite* techniques . Resource sharing techniques face this challenge by utilizing the multicast facility.

The decision as to which class and particular technique to apply greatly impacts the overall system performance and the perceived quality-of-service (QoS). With the many available techniques, it is unclear which one is the best to use in a target environment. In addition, the achieved resource sharing depends significantly on the user access patterns as well as the available server, client, and network resources. Unfortunately, the overwhelming majority of prior studies assumed simple workload, in which videos are accessed sequentially from the beginning to the end (this pattern is referred to here as *Full-Content Access*) and clients have homogeneous resources (particularly available download bandwidth and buffer space).

This thesis provides a detailed analysis of resource sharing techniques, considering both the *True Video-on-Demand* (TVOD) and the *Near Video-on-Demand* (NVOD) models and two video workloads: *mixed-video* and *hot-video*. Guided by this extensive analysis, this work proposes an

efficient *Workload-Aware Hybrid Solution* (WAHS) that combines the advantages of stream merging and periodic broadcasting. Moreover, we propose a *Statistical Cache Management* (SCM) approach, which computes periodically video access frequencies and determines the data to be cached based on these statistics. In addition to its performance effectiveness, it is easy to implement and incurs small overhead as updates are triggered only when the workload varies considerably.

In addition, we study how to support heterogeneous receivers while delivering video streams in a client-pull fashion. We propose three solutions to address the variability of the download bandwidth among clients: *Simple Hybrid Solution* (SHS), *Adaptive Hybrid Solution* (AHS), and *Enhanced Hybrid Solution* (EHS). We also study how to address the variations in client bandwidth during a session. In addition, we study the support for the variability in the available buffer space among clients. Furthermore, we study how the waiting playback requests for different videos can be scheduled for service in the heterogeneous environment, capturing the variations in client bandwidth and buffer space.

Moreover, we study the impact of selected-content access on streaming servers delivering data in a client-pull fashion using stream merging techniques. We propose several enhancements to reduce server load and improve the client perceived quality-of-service (QoS).

Finally, we investigate utilizing more advanced video coding, such as Layered Video Coding (LVC) and Multiple Description Coding (MDC), with advance stream merging techniques for better serving heterogeneous receivers.

AUTOBIOGRAPHICAL STATEMENT

Bashar Qudah is a Ph.D. candidate at the Department of Electrical and Computer Engineering at Wayne State University. Mr. Qudah received his B.S. (97) in Electrical Engineering from the University of Jordan and his M.S. (03) in Computer Engineering from the University of Michigan. Mr. Qudah joined the Wayne State Media Research Laboratory in 2004, since then, he has been conducting his research on video streaming and designing multimedia servers. He has several publications in top conferences and journals including ACM Multimedia and IEEE Transactions on Circuits and Systems for Video Technology (TCSVT). In addition Mr. Qudah has more than 10 years of professional experience in software engineering and development and several related professional certificates.