Towards Scalable Delivery of Video Streams to Heterogeneous Receivers

Bashar Qudah
bqudah@wayne.edu

Nabil J. Sarhan
nabil@ece.eng.wayne.edu

Department of Electrical and Computer Engineering
Wayne State University
Detroit, MI 48202

ABSTRACT

The required real-time and high-rate transfers for multimedia data severely limit the number of requests that can be serviced concurrently by *Video-on-Demand* (VOD) servers. Resource sharing techniques can be used to address this problem. We study how VOD servers can support heterogeneous receivers while delivering data in a client-pull fashion using enhanced resource sharing. We propose three *hybrid solutions*. The first solution simply combines existing resource sharing techniques and deals with clients as two bandwidth classes. The other two solutions, however, classify clients into multiple bandwidth classes and service them accordingly by capturing the proposed ideas of *Adaptive Stream Merging* or *Enhanced Adaptive Stream Merging*, respectively. We also discuss how scheduling policies can be adapted to the heterogeneous environment so as to exploit the variations in client bandwidth. We evaluate the effectiveness of the proposed solutions and analyze various scheduling policies through extensive simulation.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols; C.4 [**Computer Systems Organization**]: Performance of Systems; I.6.5 [**Simulation and Modeling**]: Model Development

General Terms

Design, Performance

Keywords

Client heterogeneity, multimedia servers, stream merging, Video-on-Demand (VOD), video streaming.

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 and to motivate new applications and service models. Unfortunately, the number of concurrent video streams that can be supported is severely limited by the required real-time and high-rate transfers. Resource sharing techniques [4, 8, 9, 12, 5, 13, 18, 15, 17] face this challenge by utilizing the multicast facility. *Batching* [1, 8, 22], the simplest of these techniques, services all waiting requests for a video using one stream. To further reduce the delivery costs, stream merging techniques combine streams when possible. These techniques include *Stream Tapping/Patching* [5, 13], *Transition Patching* [4], and *Earliest Reachable Merge Target* (ERMT) [9, 11]. Whereas Batching and stream merging techniques deliver data in a *client-pull* fashion, periodic broadcasting techniques [12, 14, 16] employ the *server-push* approach. In particular, they divide each supported video into multiple segments and broadcast them periodically on dedicated channels. Thus, they can service unlimited numbers of customers but can be used only for the most popular videos, and they require customers to wait until the next broadcast times of the first segments. Moreover, server channels may become underutilized when videos are requested infrequently. In this paper, the focus is on the client-pull approach.

Providing successful VOD services over the Internet requires support for heterogeneous receivers. Measurements of Internet client bandwidth [3] 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 even the connection time, can play a significant role in the actual bandwidth.

Resource sharing has been studied extensively, but to our knowledge, no study has addressed the client heterogeneity issue in VOD servers employing the client-pull delivery approach. Batching requires each client to have one download channel at the video playback rate, whereas stream merging techniques (such as Patching, Transition Patching, and ERMT) generally require client bandwidth of double the video playback rate. One notable exception is *Bandwidth Skimming* [10], which may require client bandwidth less than double the video playback rate through special video encoding or complex data delivery, but still assumes receivers homogeneity. Client heterogeneity has not been addressed except by a few, recent studies, such as *HeRo* [2] and *BroadCatch* [21] that use the server-push approach.

MM'06, October 23–27, 2006, Santa Barbara, California, USA.

Besides inheriting the side effects of periodic broadcasting, both solutions are built around the assumption of very large client bandwidth. For example, when the server bandwidth allocated to broadcast a video is 11 times the video playback rate, a client with bandwidth a little below double the video playback rate may wait for service up to 25% of the video length with the first solution and up to 50% with the second.

In this paper, we study how VOD servers can support heterogeneous receivers while delivering data in a client-pull fashion. We propose three solutions: *Simple Hybrid Solution* (SHS), *Adaptive Hybrid Solution* (AHS), and *Enhanced Hybrid Solution* (EHS). SHS simply combines Batching with either Patching (SHS-P) or with ERMT (SHS-E). Batching is used for clients with bandwidth less than $2b$ (where b is the video playback rate) and Patching/ERMT is used for the rest. AHS and EHS, however, classify clients into multiple bandwidth classes and service them accordingly. To serve clients with bandwidth capacities between b and $2b$, AHS and EHS employ new stream types, called *adaptive stream* and *enhanced adaptive stream*, respectively. AHS and EHS use 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. The three proposed solutions do not assume any special video coding.

The request scheduling problem becomes complicated by the support for client heterogeneity. Thus, we introduce the concept of a *virtual queue* and discuss how scheduling policies can be adapted to operate on the virtual queues rather than the video waiting queues so as to capture the variations in client bandwidth.

We evaluate the effectiveness of the proposed schemes and analyze various scheduling policies through extensive simulation. 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 four performance metrics: customer defection probability, average waiting time, unfairness against unpopular videos, and unfairness against clients with low bandwidth classes.

The rest of this paper is organized as follows. Section 2 provides background information. Section 3 describes the proposed solutions and schemes. Section 4 discusses the performance evaluation methodology and main results.

2. BACKGROUND INFORMATION

The design of VOD servers involves two main aspects: resource sharing and request scheduling. Let us first discuss the main resource sharing techniques that can be combined to support heterogeneous clients: Batching, Patching, and ERMT.

While Batching services all waiting requests for a video using one full multicast video stream and requires only one client download channel at the video playback rate. Patching expands the multicast tree dynamically to include new requests. A new request joins the latest *regular* (i.e., full) stream for the object and receives the missing portion as a *patch*. Hence, it requires two download channels (each at the video playback rate) and additional client buffer space. 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* (Wr). Figure 1 further explains the concept in servicing a 2-hour video.

ERMT is a near optimal hierarchical stream merging technique. It also requires two download channels, but it makes each stream sharable and thus leads to a dynamic merge tree, as shown in Figure 2. A new client joins the closest reachable stream (*target*) and receives the missing portion by a new stream (*merger*). After the merger stream finishes and merges into the target, the later can get extended to satisfy the playback requirement of the new client(s), and this extension can affect its own merge target.

Let us now briefly discuss the issue of request scheduling. A VOD server maintains a waiting queue for every video and applies a scheduling policy to select an appropriate queue for service whenever it has available server resources. All requests in the selected queue can be serviced using the same set of server resources. In this study, we consider the following scheduling policies: *First Come First Serve* (FCFS) [8], *Maximum Queue Length* (MQL) [8], *Maximum Factored Queue Length* (MFQL) [1], and *Minimum Cost First Per Request* (MCF-P) [20]. 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-P is a recently proposed policy that captures the variations in stream lengths by selecting the queue requiring the least cost per request.

3. SUPPORTING CLIENT HETEROGENEITY

3.1 Proposed Resource Sharing Solutions

Let us now discuss the three proposed solutions for the client heterogeneity problem. In the discussion, we assume the video playback rate is b bits per second.

3.1.1 Simple Hybrid Solution (SHS)

Batching and all existing stream merging techniques assume client bandwidth homogeneity. Batching requires one client download channel at the video playback rate (b), whereas stream merging techniques generally require $2b$ in client bandwidth. A simple solution to support heterogeneous clients is to combine resource sharing techniques. We consider two alternatives: *Patching with Batching* (SHS-P) and *ERMT with Batching* (SHS-E). ERMT is the most efficient and Patching is the simplest among all stream merging techniques that require $2b$ in client bandwidth.

Figures 3 and 4 further explain how SHS-P and SHS-E work. Clients (or group of clients) with bandwidth capacities of $2b$ or greater ($2nd$, $6th$, and $7th$ in the figures) are served with Patching or ERMT, while those with lower bandwidth ($1st$, $3rd$, $4th$, and $5th$) are served using Batching. In fact, Batching streams can eliminate the need for regular (full) streams in Patching.

3.1.2 Adaptive Hybrid Solution (AHS)

SHS classifies clients into two bandwidth classes: clients with bandwidth equal to or higher than b but lower than

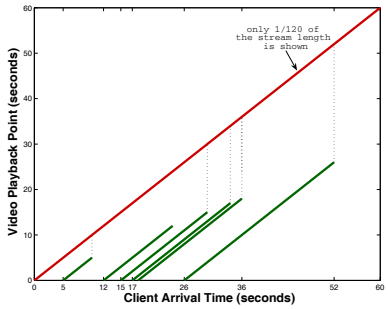


Figure 1: Patching

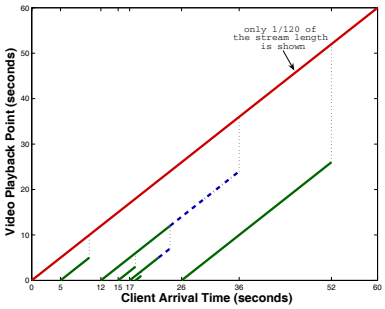


Figure 2: ERMT

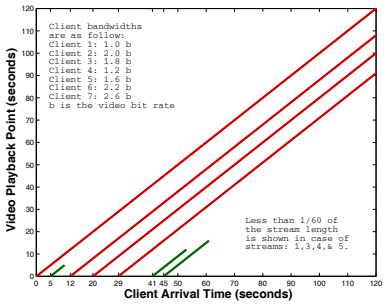


Figure 3: SHS-P

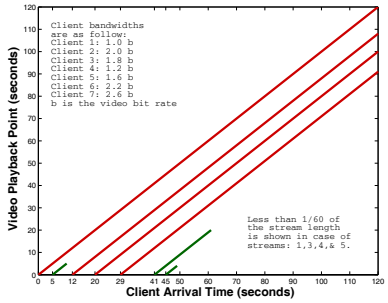


Figure 4: SHS-E

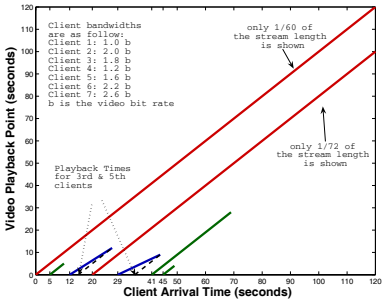


Figure 5: AHS-E

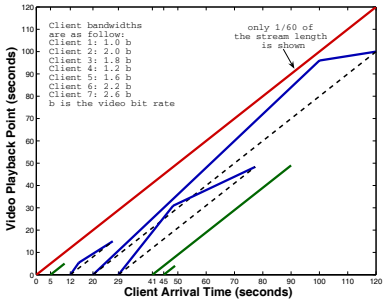


Figure 6: EHS-E

$2b$, and clients with bandwidth capacities equal to or higher than $2b$. SHS services the clients in the first class using Batching and the clients in the second class with Patching or ERMT, depending on the scheme used. Hence, it does not capture important opportunities for resource sharing in the first class. For the second class, utilizing more than double the playback rate (i.e., $2b$) is not expected to lead to worthwhile performance benefits. The results in [11] show that utilizing more than two channels (each at the playback rate) at each client leads to only low additional performance benefits when an optimal stream merging (or ERMT) is used. In the heterogeneous environment, not every client has more than $2b$ in bandwidth and thus the potential gain is even less significant. We discuss next how to utilize the extra client bandwidth in the first class.

We propose here two ideas: *Adaptive Stream Merging* and *Early Snooping*. The first introduces a new stream type, called *adaptive stream*, or succinctly *a-stream*, to serve those clients with bandwidth higher than b but lower than $2b$. The client uses b of bandwidth to listen to the latest Batching (full) 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 can 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 available client 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 can be attained.

Let us now discuss in more detail the adaptability to server load. Obviously, since the missed data is received at a slower rate than the video playback, clients need to buffer data for sometime before the playback can start. The additional waiting time due to buffering, W_{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 BW, $b < BW < 2b$, and $B = \frac{BW}{b}$. Then, W_{Buf} is given by

$$W_{Buf} = (\frac{2-B}{B-1})P. \quad (1)$$

Since W_{Buf} is predictable, customers can be encouraged to wait by informing them with the expected time of playback (which is roughly equal to W_{Buf}).

The required buffering has two conflicting effects. In particular, it leads to increasing data sharing among streams and thus servicing more customers and allowing resources to become available sooner for servicing new requests. It, however, increases the waiting time for some customers and may in turn cause them to defect. The overall server performance in terms of customer 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 customer waiting tolerance times a certain factor. This factor can take any positive real value, and it varies dynamically based on the variation in customer defection rate so as to achieve the lowest possible rate. If the triggering condition is not met, a full stream is delivered instead. As will be shown later, the dynamic approach reduces not only the defection probability but also the average waiting time.

To further reduce the additional waiting time caused by buffering, we propose the idea of *Early Snooping*. Basically, each client can start snooping on the latest full stream for the video as soon as it issues the request as opposed to

waiting till it gets admitted for service. In this case, P in Equation 1 becomes the individual client arrival time (with respect to the last full stream of the video) 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 b and $2b$. For other clients, it operates like SHS. Specifically, it applies Batching for clients with b bandwidth and Patching or ERMT for clients with bandwidth of $2b$ or higher. This leads to two alternative schemes: AHS-P (when Patching is used) and AHS-E (when ERMT is used). Figure 5 further explains how AHS-E works. We assume here that the buffering time cannot exceed 30 seconds. The $3rd$ and the $5th$ clients are serviced using a-streams instead of Batch streams, which reduces the cost from 28829 seconds in case of SHS-E down to 14459 seconds with AHS-E. The buffering times are 3 seconds for the $3rd$ client and 6 seconds for the $5th$. Servicing the $4th$ client using an a-stream would have resulted in 80 seconds of buffering time, which exceeds the assumed threshold. Therefore, it is serviced using Batching.

3.1.3 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 b and $2b$. 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 full 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 packet delay jitter.

Figure 7 further clarifies the concept of ea-streams. In this example, a client with bandwidth BW, where $b < BW < 2b$ (i.e., $B = \frac{BW}{b}$), arrived P seconds after the latest full 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., $W_{Buf} = 0$).

Let us now discuss how to provide the client with a continuous playback with minimum cost and without any additional waiting time due to buffering (W_{Buf}). Basically, two conditions must be satisfied. First, the last playback point (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 video. 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

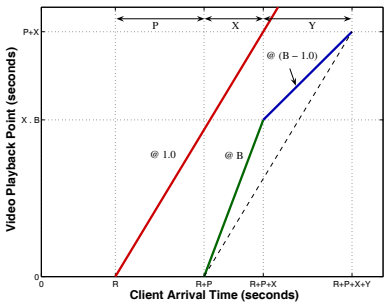


Figure 7: Enhanced Adaptive Streams

full stream, which is equal to $P + X$. These two conditions translate to the following two equations, respectively:

$$P + X = X + Y \quad and \quad X(B) + Y(B-1) = X + P.$$

It follows from these two equations that

$$Y = P \quad and \quad X = \frac{(2-B)}{(B-1)}P.$$

Hence, the cost for delivering the ea-stream in seconds of data can be determined by

$$Cost = P + X = \frac{P}{B-1}. \tag{2}$$

Equation 2 is valid only if the cost is less than the video length. Otherwise, a new full stream must be delivered instead.

The ea-streams can be 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 $2b$ or higher bandwidth. Figure 6 shows the earlier example with EHS-E. The cost is reduced to 7421.34 seconds of video data.

3.2 Tuning the Adaptive Streams

Equation 2 highlights the importance of minimizing the average value of time skewness P since the last full stream. Let us assume that for a certain D-second video, the request arrival rate is λ and the average time between two consecutive full streams is W_r. Thus, the number of requests arriving per video length can be given by $N = \lambda D$, and the number of full streams per video length is $\frac{D}{W_r}$. 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 can be given by

$$Cost = \sum_{i=1}^{N}(\frac{P_i}{B_i - 1}) + \frac{D}{W_r}D. \tag{3}$$

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 $\frac{W_r}{2}$, the total cost per video duration is given by

$$Cost = \sum_{i=1}^{N}(\frac{P_i}{B^* - 1}) + \frac{D^2}{W_r} \approx \frac{\lambda D W_r}{2(B^* - 1)} + \frac{D^2}{W_r}. \tag{4}$$

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

$$\frac{\partial}{\partial W_r}(\frac{\lambda D W_r}{2(B^*-1)}+\frac{D^2}{W_r})=0 \quad \Rightarrow \quad W_{r_{opt}}=\sqrt{\frac{2D(B^*-1)}{\lambda}}. \tag{5}$$

By substituting $W_{r_{opt}}$ in Equation 4,

$$Cost_{opt} \approx \frac{D\sqrt{\lambda D}}{\sqrt{2(B^*-1)}}+\frac{D\sqrt{\lambda D}}{\sqrt{2(B^*-1)}}. \tag{6}$$

Therefore, when the delivery cost is minimized, the total cost of ea-streams is equal to the total cost of full streams. With some approximation, this can be generalized to a-streams and ea-streams in both homogeneous and heterogeneous client environments and with both TVOD and NVOD services. Motivated by this conclusion, the server can simply trigger full streams dynamically by computing the total cost of a-streams or ea-streams for each video since the last full stream and initiating a full stream for servicing new requests for a video when the computed value exceeds the cost of a full stream.

3.3 Addressing Variations in Client Bandwidth

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 can deal much better with this problem because of the fast delivery period of ea-streams. These streams can 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 can be recomputed using the client bandwidth history, and thus the fast delivery period can 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 during the slow delivery period, where $0 \leq \delta_B \leq B-1$. The new fast delivery period $\overline{X}$ can be calculated using

$$\overline{X} = X + \frac{\delta_B}{B_{avg}-1}P, \tag{7}$$

where B_{avg} is the average value of B during the fast delivery period, and X is computed based on that average value. The additional cost is $\frac{\delta_B}{B_{avg}-1}P$. Consequently, the new value of Y ranges from 0 to P, depending on both δ_B and the actual drop in bandwidth. If the 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 full stream.

3.4 Scheduling in the Heterogeneous Environment

Supporting client heterogeneity complicates the scheduling issue. In this case, a video waiting queue may contain requests for clients varying in download bandwidth. So, 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. The corresponding clients have b, $1.5b$, and $2b$ bandwidth, respectively. The server can service all three requests by Batching or only R2 and R3 by the a-streams or ea-streams, or only R3 by Patching or ERMT. Thus, 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 $q_{v,i}$ has all clients in the v^{th} video queue (q_v) with bandwidth classes $\geq i$. A bandwidth class is a group of clients with bandwidth capacities within a specified range. For example, *Class 0* can be for clients with bandwidth within $[b, 1.05b[$, *Class 1* within $[1.05b, 1.10b[$, and so on. The step in this example is $0.05b$ and is called *bandwidth class width*.

Scheduling can be performed by dividing the requests in each video queue to 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. Consequently, the implementations of the existing scheduling policies may need to be adjusted so as to incorporate the new virtual queue concept. Because of their nature, MQL and FCFS will select for service *all* requests in a particular video queue and will not utilize the variations in client bandwidth. MFQL and MCF-P, however, can exploit these variations. They operate on virtual queues 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 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. PERFORMANCE EVALUATION

We analyze the effectiveness of the proposed schemes and analyze various scheduling policies through extensive simulation. 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 four performance metrics: customer defection probability, average waiting time, unfairness against unpopular videos, and unfairness against clients with low bandwidth classes. The defection probability is the most important and can be defined as the probability that customers leave the server 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. Video unfairness quantifies the bias against unpopular videos and is equal to $\sqrt{\sum_{i=1}^{M}(d-d_i)^2/(M-1)}$, where M is the number of videos, d_i is the i_{th} video defection rate, and d is the mean defection rate for all videos. We introduce class unfairness to measure the bias against clients with low bandwidth classes. It can be calculated using the previous equation by redefining M as the number of classes, d_i as the i_{th} class defection rate, and d as the mean defection rate for all classes. In TVOD, we compare the server bandwidth required to service all requests immediately.

4.1 Workload Characteristics

Like most prior studies, we assume that the arrival of the requests to a VOD server follows a Poisson Process with an

average arrival rate λ and the accesses to videos are highly localized and follow a Zipf-like distribution [6] with a skewness parameter $\theta = 0.271$. 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 [19, 22] 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 the exponential distribution. The default value of μ_{tol} is 1.5 minutes. To ensure high performance, each client is assumed to have 50% buffer space of the video data.

Table 1 summarizes the main workload characteristics and shows the default values. Unless otherwise mentioned, we consider a VOD 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) from $300b$ to $3000b$. Interactive operations can be supported using contingency channels [7]. Thus, the relative performance of various techniques and policies does not depend on these operations as long as the fraction of server channels 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 channels.

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 $\theta = 0.0$ and 0.5, and three truncated Normal distributions with mean values $\mu_B = 1.5b$, $2.0b$, and $2.5b$ and a standard deviation of $0.5b$. The first Zipf distribution represents the case when there is a high ratio of clients with bandwidth in the region just above b, and the other distributions cover other possible scenarios. The client bandwidth capacities range between b and $11b$ in all five distributions.

Table 1: Summary of Workload Characteristics

Parameter	**Model/Value(s)**
Request Arrival Model	Poisson Process
Request Arrival Rate (λ)	20 to 360, default: **40** Req/min
Server Capacity	$300b$ to $3000b$
Video Access	Zipf-Like ($\theta = 0.271$)
Number of Videos	60 to 1920, default: **120**
Video Length (D)	5 to 180, default: **120** minutes
Waiting Tolerance Model	Exponential, TSG
Waiting Tolerance Mean (μ_{tol})	0.5, **1.5** (default), 2.5 min
Client Bandwidth Model	Zipf-Like, Normal
Bandwidth Class Width	$0.05b$

4.2 Result Presentation and Analysis

4.2.1 Comparing Resource Sharing Schemes

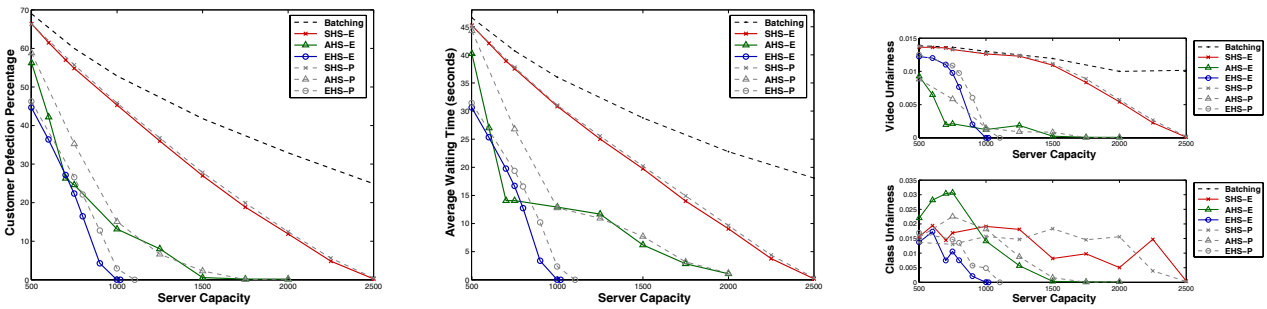
Let us start by comparing various resource sharing schemes that work for heterogeneous receivers (Batching, SHS-P, SHS-E, AHS-P, AHS-E, EHS-P, and EHS-E) in terms of defection rate, average waiting time, and video and class unfairness. Figures 8 and 9 depict the results under Normal and Zipf bandwidth distributions, respectively. At this stage, scheduling is performed using MQL. 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 $4800b$ in server bandwidth to achieve TVOD (i.e., 0 defections and 0 waiting time), SHS-E requires $2500b$, and EHS-E requires less than $1000b$. AHS-E can achieve 0 defections with $1700b$ server bandwidth, but it can never completely eliminate the waiting times due to the required, initial buffering time. Moreover, with $1000b$ 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 class unfairness in particular, the results are slightly different. Batching is blind towards bandwidth classes since it 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 $2b$ or higher is of less significance than changing the solution (and thus the method of service for clients with bandwidth between b and $2b$). (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 very close to, but slightly worse than EHS-E, AHS-E, and SHS-E, respectively. Batching is dropped because of its poor performance.

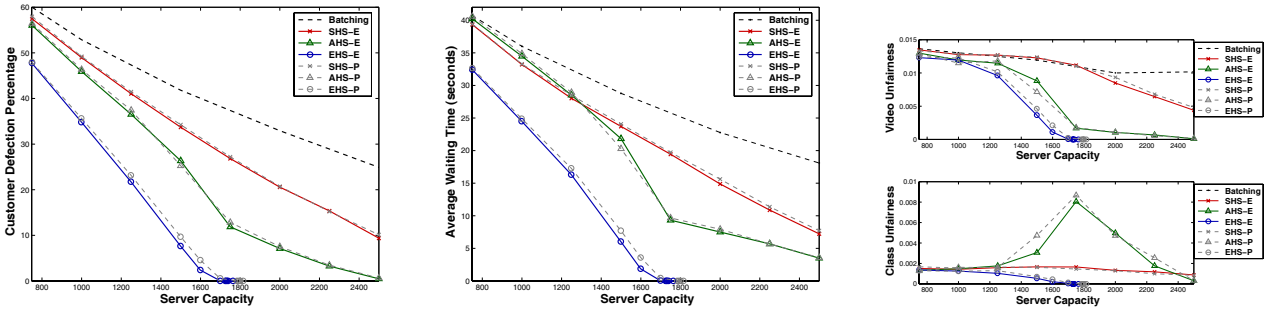
4.2.2 Comparing Scheduling Policies

Let us now discuss the impact of scheduling. Figures 10, 11, and 12 compare scheduling policies when applying EHS-E, AHS-E, and SHS-E, respectively. The results are shown under two bandwidth distributions: Zipf with $\theta = 0.0$ and Normal with $\mu_B = 2.0$. FCFS and MFQL in general perform worse than MCF-P and MQL in terms of defection rate and waiting time. Figure 13 compares scheduling policies in the two main metrics (defection rate and waiting time) under the remaining bandwidth distributions. 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 class unfairness since it considers the cost. 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 ma-



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

Figure 8: Comparing Different Schemes [Normal Bandwidth Dist. with $\mu_B = 2.0$, MQL]



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

Figure 9: Comparing Different Schemes [Zipf Bandwidth Dist. with $\theta = 0.0$, MQL]

jority of clients have low bandwidth (as in Zipf with $\theta = 0$ and Normal with $\mu_B = 1.5$), 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 $2b$. 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 the Normal($\mu_B = 1.5$), 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($\theta = 0.0$) because the small increase in defection rate compared with MQL is justified by a more significant reduction in waiting time.

4.2.3 Comparing Schemes with Best Scheduling

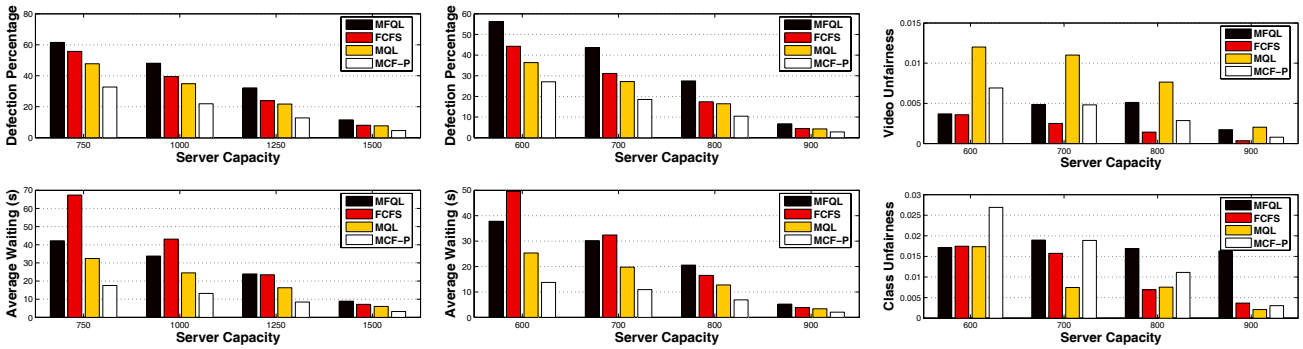
Figure 14 compares EHS-E, AHS-E, and SHS-E under five different bandwidth distributions. As expected, the performance gap increases when the distribution offers more clients that can benefit from ea-streams or a-streams. For example, when the average client bandwidth is $2.5b$, the majority of clients do have more than the $2b$ required by ERMT, while when the average client bandwidth is $1.5b$, 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 (b). Having large portion of clients with very high bandwidth motivates higher video quality.

4.2.4 Impact of Customer Waiting Tolerance

Let us now discuss the impact of customer waiting tolerance. Figure 15 plots the percentage improvements achieved by EHS-E and AHS-E compared with SHS-E in terms of the defection percentage and waiting times for different values of μ_{tol}. Four main conclusions can be drawn 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 its. 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 defection rates and waiting times.

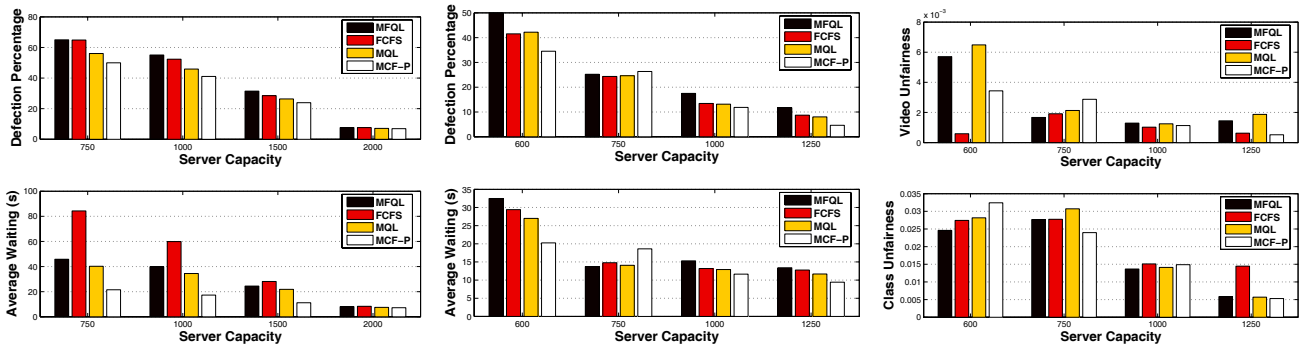
4.2.5 Impacts of Other Parameters

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 16, 17, and 18 compare the required server bandwidth to achieve TVOD by the two schemes versus request arrival rate, video length, and number of videos, respectively, under the two bandwidth distributions: Normal with $\mu_B = 2$ and Zipf with $\theta = 0$. The impact of the video length is similar to that of the request rate. The number of concurrent customers per video increases with each, and thus



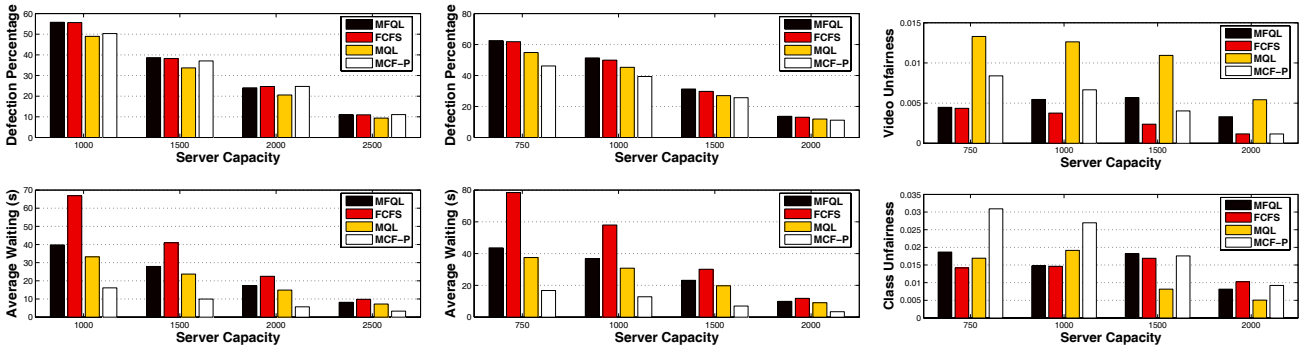
(a) Zipf ($\theta = 0.0$) (b) Normal ($\mu_B = 2.0$) (c) Normal ($\mu_B = 2.0$)

Figure 10: Comparing Scheduling Policies for EHS-E



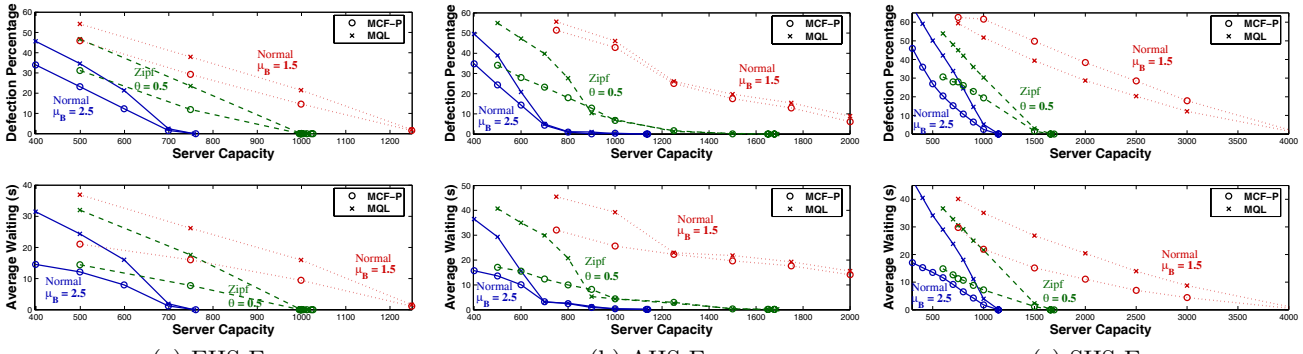
(a) Zipf ($\theta = 0.0$) (b) Normal ($\mu_B = 2.0$) (c) Normal ($\mu_B = 2.0$)

Figure 11: Comparing Scheduling Policies for AHS-E



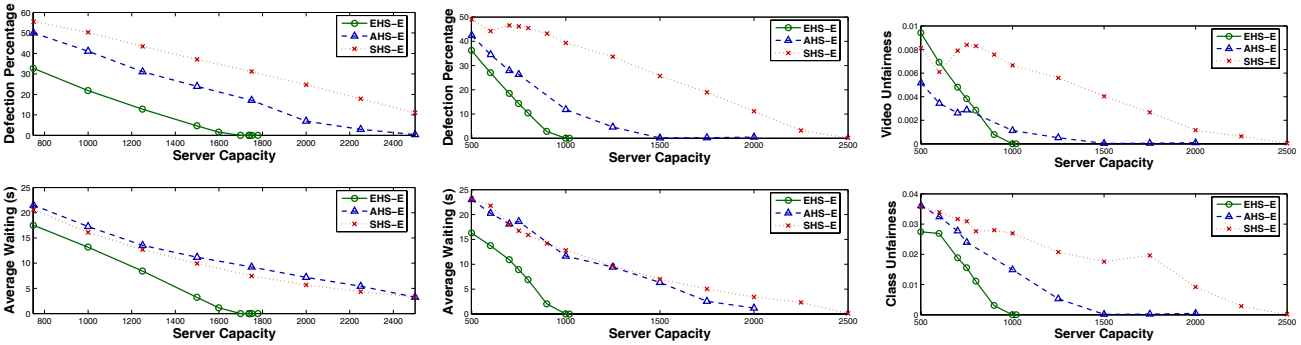
(a) Zipf ($\theta = 0.0$) (b) Normal ($\mu_B = 2.0$) (c) Normal ($\mu_B = 2.0$)

Figure 12: Comparing Scheduling Policies for SHS-E



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

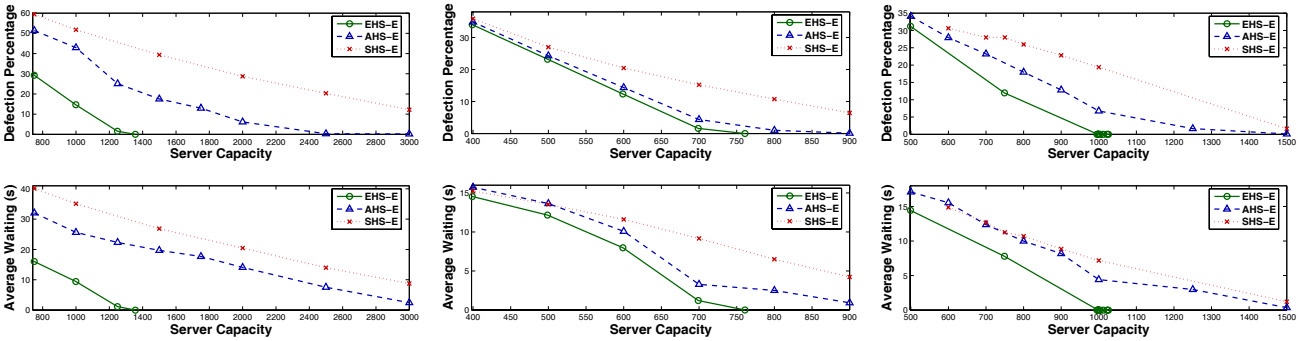
Figure 13: Comparing Scheduling Policies with Different Bandwidth Distributions



(a) Zipf ($\theta = 0.0$)

(b) Normal ($\mu_B = 2.0$)

(c) Normal ($\mu_B = 2.0$)

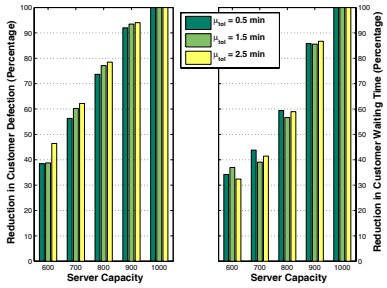


(d) Normal ($\mu_B = 1.5$)

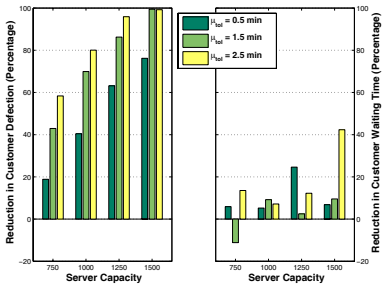
(e) Normal ($\mu_B = 2.5$)

(f) Zipf ($\theta = 0.5$)

Figure 14: Comparing the Schemes with Best Scheduling



(a) EHS-E vs. SHS-E



(b) AHS-E vs. SHS-E

Figure 15: Impact of Customer Waiting Tolerance (μ_{tol})

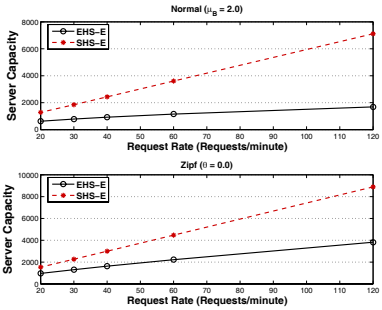


Figure 16: Impact of Request Rate (TVOD)

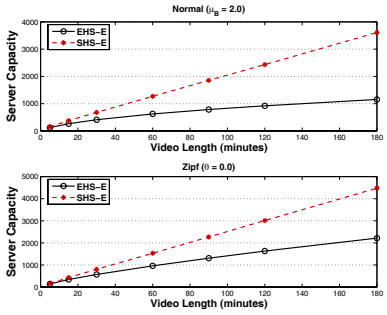


Figure 17: Impact of Video Length (TVOD)

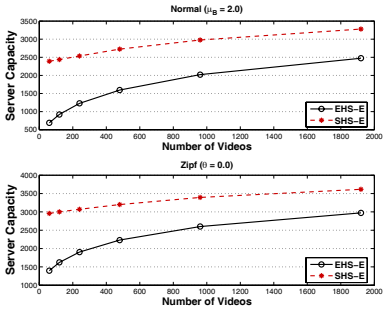


Figure 18: Impact of Video Number (TVOD)

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 increasing less popular and thus data sharing decreases.

5. CONCLUSION

We have studied the support for heterogeneous VOD receivers by using three proposed solutions: *Simple Hybrid Solution* (SHS), *Adaptive Hybrid Solution* (AHS), and *Enhanced Hybrid Solution* (EHS). 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 discussed how scheduling policies can be adapted so as to capture the variations in client bandwidth.

We have evaluated the effectiveness of the proposed schemes and have analyzed various scheduling policies through extensive simulation. The main results can be summarized as follows. (1) The proposed schemes can achieve high degrees of resource sharing. (2) 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. Besides, it can reduce the average request waiting time and tends to be more fair towards both unpopular videos and low bandwidth classes. (3) 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. (4) 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 $2b$ or higher is of less significance than changing the solution. (5) 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.

6. REFERENCES

[1] 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.

[2] 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.

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

[4] 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.

[5] 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 (ICCCN)*, pages 200–207, Sept. 1997.

[6] A. L. Chervenak. *Tertiary Storage: An Evaluation of New Applications*. PhD thesis, U.C. Berkeley, Dec. 1994.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] N. J. Sarhan and B. Qudah. Cost-based scheduling for scalable media streaming. Technical report, Wayne State University, Electrical and Computer Engineering, Detroit, Michigan, June 2006.

[21] 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), Sept. 2004.

[22] 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.