The Pennsylvania State University

The Graduate School

Department of Computer Science and Engineering

AN INVESTIGATION OF SCHEDULING POLICIES FOR MULTIMEDIA SYSTEMS

A Thesis in

Computer Science and Engineering

by

Nabil J. Sarhan

© 2003 Nabil J. Sarhan

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2003

We approve the thesis of Nabil J. Sarhan.

Date of Signature

Chita R. Das
Professor of Computer Science and Engineering
Thesis Adviser

Vijaykrishnan Narayanan
Assistant Professor of Computer Science and Engineering

Raj Acharya
Professor of Computer Science and Engineering
Head of the Department of Computer Science and Engineering

Abstract

The interest in *Multimedia-on-Demand* (MOD) applications on the World Wide Web (WWW) has grown dramatically because of the exponential expansion of the Internet and the availability of fast Internet access at home and office. In contrast with broadcast-based systems (such as cable TV), MOD servers enable customers to select the multimedia contents they want to access at the times of their choosing. Moreover, they allow customers to apply VCR-like operations such as pause, resume, fast forward, and fast rewind.

The performance of MOD servers is highly constrained by the stringent requirements of multimedia data. Specifically, multimedia data must be presented continuously in time so that they can convey meaningful information. They also require high transfer rates. Without careful management of the limited system resources, both the available bandwidth of the storage subsystem and the available bandwidth to the network will run out rather quickly after supporting only a relatively small number of customers. Resource sharing strategies save precious resources by using the same set of resources in servicing multiple requests for the same multimedia content. Taking advantage of the high locality of reference in the access to multimedia data, resource sharing can significantly increase the numbers of customers that can be serviced concurrently.

The exploited degrees of resource sharing depend greatly on how MOD servers schedule the waiting requests. By scheduling the requests intelligently, a server can support more concurrent customers, can reduce their waiting times for service, and/or can meet some other performance objectives. Choosing an appropriate scheduling policy is very complicated by the presence of several design tradeoffs and the dependence of performance on customer waiting tolerance and server load.

This thesis addresses the scheduling problem in MOD servers. The main application of interest is *video-on-demand* (VOD). A VOD server maintains a waiting queue for every movie, selects an appropriate queue for service whenever the required set of resources becomes available, and services all requests in that queue together using only one stream. Scheduling policies for VOD servers include *First Come First Serve* (FCFS), *Maximum Queue Length* (MQL), and *Maximum Factored Queue Length* (MFQL). FCFS schedules the waiting requests based on their waiting times, whereas MQL and MFQL schedule them based on the queue lengths.

The main contributions of this thesis are providing a detailed investigation of scheduling policies and proposing two new policies, called *Quantized First Come First Serve* (QFCFS) and *Enhanced Minimum Idling Maximum Loss* (IML^+). QFCFS combines the benefits of FCFS and MQL/MFQL, whereas IML^+ improves the overall performance by exploiting minimum request waiting times. This study

compares, mainly through simulation, various policies in terms of several performance metrics and examines the impacts of customer waiting tolerance and server load. The results show that the proposed policies suite different patterns of waiting tolerance and provide significant performance benefits.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Multimedia-on-demand (MOD) systems enable customers to watch what they want when they want it and allow them to apply VCR-like operations such as pause, resume, fast forward, and fast rewind. These important features have encouraged motion picture studios and cable companies to look for ways to deliver MOD services. These services can also help these companies to create markets for their huge on-the-shelf media contents. Besides its use for entertainment, MOD has been of great importance in education and distant learning in particular. *Video-on-demand* (VOD) is the most common MOD application and is the application of interest in this study.

Unfortunately, the number of clients that can be supported concurrently by a VOD is highly constrained by the requirements of the real-time playback and the high transfer rates. A wide spectrum of techniques, therefore, has been developed to enhance the performance of VOD servers, including *resource sharing* and *request scheduling* [13, 21, 42, 2, 24, 25, 37], *admission control* [5, 26, 43, 44, 28], *disk striping* [38, 10], *data replication* [18, 10, 11], *disk head scheduling* [33, 29], *data block allocation* [32, 18], and *adaptive block rearrangement* [34].

The performance of VOD servers can be significantly improved by servicing multiple requests from a common set of resources. The classes of resource sharing strategies for VOD servers include *batching* [13, 15, 42, 2], *patching* [24, 36], *piggy-backing* [21, 20, 1], *broadcasting* [25, 30, 31], and *buffer management* through *interval caching* [14, 16]. Batching accumulates the requests for the same movies and services them together by utilizing the multicast facility. It, therefore, offloads the underlying storage subsystem and uses efficiently server bandwidth and network resources. In contrast with batching, patching expands the multicast tree dynamically to include new requests. Patching reduces the request waiting time and increases resource sharing, but it requires additional bandwidth and buffer space at the client. Like patching, piggy-backing services a request almost immediately, but it adjusts the playback rate so that the request catches up with a preceding stream, resulting in a lower-quality presentation of the initial part of the requested video. Broadcasting techniques divide each popular movie into multiple segments and broadcast each segment periodically. These techniques require relatively very high bandwidth and buffer space at the client. With interval caching, the server caches intervals between successive streams in its main memory. This technique shortens the request waiting time and increases server throughput without increasing the bandwidth or the space requirements at the client, but it increases the overall cost of the server.

The exploited degrees of resource sharing depend greatly on how VOD servers schedule the waiting requests. Through intelligent scheduling, a server can increase the number of customers serviced concurrently while reducing their waiting times for service. Batching systems rely entirely on scheduling to boost up their performance. VOD systems that employ other resource sharing techniques also benefit from intelligent scheduling. Note that only the most popular movies are broadcasted when a broadcasting technique is used.

Scheduling policies for VOD servers include *First Come First Serve* (FCFS) [13, 42], *Maximum Queue Length* (MQL) [13], *Maximum Factored Queue Length* (MFQL) [2], *Minimum Idling Maximum Loss* (IML) [37], and *Minimum Idling Maximum Queue Length* (IMQ) [37]. A VOD server maintains a waiting queue for every movie and services all requests in a queue together using only one stream. FCFS selects the queue with the oldest request, while MQL selects the longest queue, and MFQL selects the queue with the *largest factored length*. The factored length of a queue is defined as its length divided by the square root of the relative access frequency of its corresponding movie. Both IML an IMQ improve throughput by exploiting minimum request waiting times, but they differ in the selection criterion. IML selects from the set of eligible queues the queue that will otherwise incur the largest expected loss of requests till the next stream completion time, whereas IMQ selects the longest queue. *Longest Wait First* (LWF) [45, 17] is another common scheduling policy, but it was not studied in the context of VOD

servers. LWF selects the queue with the largest sum of request waiting times. To simplify the terminology, it is referred to as *FCFS-sum* in this study.

This thesis provides a detailed analysis of scheduling policies. Only small subsets of scheduling policies were investigated in prior works and only under limited models of customer waiting tolerance. Previous studies are also inconsistent with regard to the relative performance of MQL to FCFS. For example, [13] shows that MQL achieves better throughput than FCFS, whereas [2] and [37] indicate the opposite. These discrepancies are not due to any specific assumptions in these studies and are resolved in this thesis. Moreover, there is a misconception with regard to the ability of FCFS to provide time of service guarantees. In [42], it is stated that FCFS can provide time of service guarantees, which are either precise or later than the actual times of service. In contrast, this study shows that FCFS may violate these guarantees. Furthermore, previous studies on VOD unfairly discarded FCFS-sum without proper investigation although it was shown to perform very well in other contexts [45, 17, 3].

This thesis also proposes two scheduling policies: *Quantized FCFS* (QFCFS) and *Enhanced IML* (IML$^+$). QFCFS combines the benefits of FCFS and MQL/MFQL by scheduling the requests based on both their waiting times and queue lengths. It first translates waiting times into discrete levels and then selects the queue with the highest level. If there are multiple eligible queues, it chooses the longest queue

or the queue with the largest factored length. QFCFS can lead to the best compromise between FCFS and MQL/MFQL if the distance between two consecutive levels is adjusted appropriately. IML$^+$ efficiently exploits minimum request waiting times. The idea behind this policy is based on the observation that the expected request loss, which is examined by IML, is a discrete number with a typically small value, so multiple queues may have the same expected loss. Whereas IML selects just any one of these queues, IML$^+$ enhances performance by selecting the queue with the largest factored length. The choice between QFCFS and IML$^+$ depends on the pattern of the waiting tolerance exhibited by customers.

This thesis compares the performance of various policies through extensive simulation. The primarily investigated policies include FCFS, MQL, MFQL, FCFS-sum, QFCFS, IMQ, IML, and IML$^+$. The simulation study analyzes three objectives: the overall customer reneging (defection or turnaway) probability, the average customer waiting time, and unfairness. The first objective is the most important because it translates to the number of customers that can be serviced concurrently and to server throughput, whereas the second objective signifies the quality of service (QoS) and comes next in importance. Unfairness measures the bias against unpopular movies. This study also examines the impacts of customer waiting tolerance and server capacity (or server load) on the effectiveness of each policy. Moreover, this study compares the policies in terms of other objectives, such as implementation complexity, ability to prevent starvation, and ability to

provide (predictive) time of service guarantees. This study shows that the numerous scheduling objectives, which lead to several design tradeoffs, and the dependence of performance on the waiting tolerance and server load complicate the decision as to which policy to apply.

The main results can be summarized as follows.

- MQL and MFQL always achieve the shortest waiting times, and MQL can perform nearly as well as MFQL in terms of throughput, waiting times, and unfairness, even when assuming that MFQL has perfect knowledge of movie access frequencies. MQL is also simpler because it does not require periodic computations of movie access frequencies. Hence, contrary to [2], MQL may be preferred over MFQL.
- When the waiting tolerance follows a normal distribution, FCFS-sum and IML^+ perform almost identically in terms of various metrics and yield the highest throughput.
- When the tolerance follows an exponential distribution, MFQL and MQL yield the highest throughput.
- When customers exhibit minimum waiting times, IML^+ achieves the highest throughput. IML^+ also results in shorter waiting times than IML but longer than MQL, MFQL, and IMQ, and it is also fairer than MQL, MFQL, and IMQ.

- The distinct advantages of FCFS are simplicity, fairness, and ability to prevent starvations. When assuming that FCFS provides true time of service guarantees and that all other policies cannot influence customers to wait, FCFS leads to the highest throughput, but it results in the longest waiting times. Besides, all scheduling policies can motivate customers to wait to various degrees.
- When the tolerance follows an exponential distribution, QFCFS performs very well. In particular, with QFCFS, a server can support as many concurrent customers for high server capacities as MFQL and MQL and can start their service as immediately, while being fair, able to prevent starvations, and able to provide reasonably-accurate predictive time of service guarantees.

The rest of the thesis is organized as follows. Chapter 2 discusses the related work. Chapter 3 preliminarily analyzes various scheduling policies and Chapter 4 presents QFCFS and IML$^+$. Chapter 5 discusses the simulation platform, the workload characteristics, and the main simulation results. Finally, conclusions are drawn in the last chapter.

Chapter 2

Related Work

This chapter surveys the main classes of resource sharing techniques and discusses the main scheduling objectives and policies.

2.1 Resource Sharing Policies

The performance of VOD servers can be significantly improved by servicing multiple requests from a common set of resources. The main classes of resources sharing techniques include *batching* [13, 15, 42, 37, 2, 35], *patching* [24, 36, 6, 7], *piggy-backing* [21, 20, 1], *broadcasting* [25, 23, 30, 31, 22, 8], and *buffer management* through *interval caching* [12, 14, 16, 40, 27, 4, 35]. These techniques benefit greatly from the high locality of reference in the accesses to multimedia data. Namely, rental patterns indicate that the accesses to movies are highly localized, with only a small number of movies receiving most of the hits [9].

With batching, requests to same movies are accumulated and serviced together by utilizing the multicast facility. Batching, therefore, off-loads the underlying storage subsystem and uses efficiently server bandwidth and network resources. Fortunately, the multicast facility is already employed or can be easily employed in most enterprise and local area networks (LANs). Besides, it is supported by IPv4

and has incrementally been deployed over the Internet because of the development and standardization of pertinent protocols and the willingness of Internet Service Providers (ISPs) to provide a scalable architecture. In fact, a ubiquitous wide-scale deployment of native (non-tunneled) multicast across the Internet is becoming a reality [19]. For example, Sprint has already deployed native multicast across its Internet backbone [39].

In contrast with batching, patching expands the multicast tree dynamically to include new requests. When a request arrives, the server initiates a patching stream to service the trail of the requested video. Meanwhile, the server transmits the multicast data, and the client caches them. Patching reduces the request waiting time and improves resource sharing, but it requires additional bandwidth and buffer space at the client.

Like patching, piggy-backing services a request almost immediately, but unlike patching, the playback rate is adjusted so that the request catches up with a preceding stream. Obviously, reducing the waiting time and increasing resource sharing in this case comes at the expense of a lower-quality presentation of initial part of the requested video.

Broadcasting techniques divide each movie into multiple segments and broadcast each segment periodically on dedicated server channels. Thus, the client has to wait until the next broadcast of the first segment. Because multiple segments

of the same movie are transmitted concurrently, broadcasting requires relatively very high bandwidth and buffer space at the client.

Interval caching exploits the locality of reference by caching intervals between successive streams in the main memory of the server. With this technique, each playback request is paired (if possible) with an immediately preceding request for the same movie that is currently being serviced (either from disk or cache). The two streams are called the *following* stream and the *preceding* stream, respectively. When the server accepts a request for service from cache, it starts to cache the data of the preceding stream while it retrieves and transfers them to the client. The required cache space for servicing the following stream depends on the time interval between the two streams. Interval caching uses the available cache (memory) space efficiently by ensuring that only the shortest intervals are cached at any time. This is done by victimizing longer intervals for caching shorter ones. This technique shortens the request waiting time and increases server throughput without increasing the bandwidth or the space requirement at the client, but it increases the overall cost of the server.

The multicast facility is essential to maximize the benefits of resource sharing. In [41], however, a resource sharing technique that works in the absence of this facility was proposed. The proposed *layered range multicast* (LRM) technique reduces the required server bandwidth, accommodates client heterogeneity, and provides a unicast-based multicast implementation by employing overlay nodes

(i.e., application-level routers). The main disadvantages of this technique are requiring placement of LRM-enabled across the Internet and serving only as an intermediate solution until the multicast facility is fully deployed.

2.2 Scheduling Objectives and Policies

Resource sharing can be significantly improved through intelligent scheduling of the waiting requests. A VOD server maintains a waiting queue for every movie, routes incoming requests to their corresponding queues, selects for service an appropriate queue whenever it has an available *channel*, and services all requests in the selected queue using only one channel. A channel is a set of resources needed to deliver a multimedia stream. The number of channels in a server is referred to as *server capacity*.

2.2.1 Scheduling Objectives

All scheduling policies are guided by one or more of the following primary objectives.

1. Minimize the overall customer reneging (defection) probability.

2. Minimize the average request waiting time.

3. Prevent starvations.

4. Provide time of service guarantees.

5. Minimize unfairness.

6. Minimize implementation complexity.

The first objective is the most important because it corresponds directly to server throughput. The throughput, X, for a given request arrival rate, λ, and reneging probability, P_r, is given by

$$X = (1 - P_r) \times \lambda.$$

The second objective comes next in importance. The second, third, and fourth objectives are indicators of customer-perceived quality of service (QoS). By providing time of service guarantees, a VOD server can also influence customers to wait, thereby increasing server throughput. It is also usually desirable that VOD servers treat equally the requests for all movies. Unfairness measures the bias of a policy against cold (i.e., unpopular) movies and can be found by the following equation:

$$unfairness = \sqrt{\sum_{i=1}^{M} (r_i - \bar{r})^2 / (M - 1)},$$

where r_i is the reneging probability for the waiting queue i, $\bar{r}$ is the mean reneging probability across all waiting queues, and M is the number of waiting queues (and number of movies as well). Finally, minimizing the implementation complexity is a secondary issue in VOD servers for two reasons. First, the number of objects (movies) and the number of concurrent customers are relatively small compared

with those in other servers such as web servers. Second, the CPU and the main memory are not usually performance bottlenecks in VOD servers.

2.2.2 Scheduling Policies

Let us now discuss the common scheduling policies for VOD servers.

- *First Come First Serve* (FCFS) [13] - This policy selects the queue with the oldest request.

- *FCFS-n* [13] - This policy broadcasts periodically the n most common movies on dedicated channels and schedules the requests for the other movies on a FCFS basis. When no request is waiting for the playback of any one of the n most common movies, it uses the corresponding dedicated channel for the playback of one of the other movies.

- *Maximum Queue Length* (MQL) [13] - This policy selects the queue with the largest number of waiting requests.

- *Maximum Factored Queue Length* (MFQL) [2] - This policy attempts to minimize the mean request waiting time by selecting the queue with the *largest factored queue length*. The factored length of a queue is defined as its length divided by the square root of the relative access frequency of its corresponding movie. MFQL reduces waiting times optimally only if the server is fully

loaded and customers always wait until they receive service (i.e. no defections).

- *Group-Guaranteed Server Capacity* (GGSC) [42] - This policy preassigns server channel capacity to groups of requests in order to optimize the mean request waiting time. It groups objects that have nearly equal expected batch sizes and schedules requests in each group on a FCFS basis on the collective channels assigned to each group.

- *Maximum Batching* Schemes [37] - These schemes aggressively pursue batching by deliberately delaying requests. With these schemes, a queue is eligible for selection only if it has at least one request with a waiting time greater than or equal to a pre-specified batching threshold. The selection criterion of a queue from the eligible queues leads to two alternative policies: *Maximum Batching Maximum Queue Length* (BMQ) and *Maximum Batching Minimum Loss* (BML). BMQ selects the longest queue, while BML selects the queue that will incur – if not selected – the largest expected loss of requests till the next stream completion time.

- *Minimum Idling* Schemes [37] - These schemes pursue batching without minimum wait requirements. They partition the queues into a hot set ($\mathcal{H}$) and a cold set ($\mathcal{C}$). These schemes consider first the queues in $\mathcal{H}$. If $\mathcal{H}$ is empty, they select the queue in $\mathcal{C}$ with longest request waiting time. The selection

criterion of a queue in $\mathcal{H}$ leads to two alternative policies: *Minimum Idling Maximum Queue Length* (IMQ) and *Minimum Idling Minimum Loss* (IML). IMQ selects the longest queue, while IML selects the queue that will otherwise incur the largest expected loss of requests till the next stream completion time. IML and IMQ are discussed in more detail in Section 4.2.

Chapter 3

Preliminary Analysis

This chapter first compares various scheduling policies. Then, it discusses two variants of MQL and two variants of FCFS. Finally, it shows that FCFS may violate its time of service guarantees.

3.1 Comparing Various Policies

FCFS is the fairest and the easiest to implement. MQL and MFQL reduce the average request waiting times but tend to be biased against cold movies, which have relatively few waiting requests. Unlike MQL, MFQL requires periodic computations of access frequencies. FCFS can prevent starvations, whereas MQL and MFQL cannot. GGSC does not perform as well as FCFS in high-end servers [42], so I will not consider it further in this paper. Similarly, I will not analyze FCFS-n because [42] shows that it performs either as well as or worse than FCFS. Maximum batching and minimum idling schemes have relatively high implementation complexities and may cause starvations, but they can exploit minimum request waiting times. Minimum idling schemes require fewer channels to guarantee a given reneging percent than maximum batching schemes [37]. Hence, I will not consider maximum batching schemes further.

3.2 MQL Variants

There is a large discrepancy in the relative performance of MQL with respect to FCFS (and MFQL) in [13, 37, 2]. For example, [13] shows that MQL achieves better throughput than FCFS, whereas [2] and [37] indicate the opposite. I have identified that the discrepancy is caused by different possible implementations of MQL. Note that the length of a queue is a discrete number with a typically small value. Thus, there is a significant probability that multiple queues have the same length. The definition of MQL [13], however, does not specify the selection criterion among the longest queues.

I consider the following two alternative implementations: *MQL-u* and *MQL-f*. MQL-u selects the longest queue, and whenever there is more than one eligible queue, it selects the queue of the most popular movie among them. MQL-f is similar to MQL-u, but it selects the queue of the least popular movie among the eligible queues. An implementation of MQL can easily be MQL-f or MQL-u, depending, in many cases, merely on the order of examining the queues or the specification of the priority function (e.g., as $>$ or $\geq$).

MQL-u can be aggressively biased against cold movies, while MQL-f can be much fairer. As shown in Subsection 5.2.1, MQL-u and MQL-f vary considerably in terms of the achieved throughput and average request waiting time. I was able to reproduce the results in [13, 37, 2] by using one or the other of these two implementations. In the context of videotex systems, a policy called *Most Requests*

First Lowest (MRFL) was proposed in [17, 45]. This policy is essentially the same as MQL-f, but it was not considered in previous studies of VOD servers.

3.3 FCFS Variants

A variation of FCFS, called *Longest Wait First* (LWF), was investigated in the context of videotex systems [17, 45] and web servers with data broadcasting [3]. LWF selects the queue with the largest sum of request waiting times, and it was shown to perform very well in these contexts. In the context of VOD servers, this policy was discarded [13] without proper investigation just because of its high implementation complexity. In VOD systems, however, the number of objects (movies) is much smaller than the number of objects (pages) in web servers or videotex systems, and the number of concurrent customers is also much smaller. Furthermore, the CPU and the memory are not typically performance bottlenecks in VOD servers. To simplify the terminology, this policy is referred to as *FCFS-sum* in the subsequent analysis.

3.4 Providing Time of Service Guarantees

FCFS is believed to provide time of service guarantees. In [42], it is stated that FCFS can provide time of service guarantees, which are either precise or later than the actual times of service. I show, however, that FCFS may violate these guarantees. A server that provides time of service guarantees can be implemented

in several ways. The argument here is based on the concept rather than any specific implementation. The argument, therefore, remains valid regardless of which implementation is used.

Let us discuss first how a server may provide time of service guarantees. For now, let us assume the absence of VCR-like operations (pause, resume, fast forward, and fast rewind). In the absence of these operations, a VOD server knows exactly when each running stream will complete. A channel becomes available whenever a running stream completes, so the server can assign completion times of running streams as time of service guarantees to incoming requests. Obviously, the server should assign the closest completion times first. Thus, when a request comes and joins an empty waiting queue, the server grants that request a new time of service guarantee. If the incoming request, however, joins a queue that has at least one request, then the new request can be given the same time of service guarantee as the other request(s) waiting in the queue because of batch scheduling. Now, let us discuss the impact of VCR-like operations. Applying a VCR-like operation can be considered as an early completion if the corresponding client is the only recipient of the stream because VOD servers typically support interactive operations by using contingency channels [15]. Early completions lead to servicing some requests earlier than their time of service guarantees.

Figure 3.1 shows an implementation of FCFS for providing time of service guarantees. In the implementation, C++-like syntax is used. For simplicity, I

assume that the number of channels is greater than or equal to the number of movies, which is typically the case. A VOD server maintains a waiting queue (WQ) for each movie and one running queue (RQ) for all. All new requests are routed to the corresponding WQs. RQ keeps track of the currently running streams. To provide time of service guarantees, the server needs to maintain an index, $RQIndex$, which points to the next stream in RQ whose completion time has not been assigned yet. The current system time is represented by $currTime$. $WQi.guarantee$ denotes the time of service guarantee assigned to the waiting queue i. $RQ[0]$ is the first element of RQ, and it corresponds to the *newest* running stream. $RQ[RQindex].compTime$ denotes the completion time of the next running stream whose completion time has not been assigned yet. $RQIndex$ must be initialized to -1. Note that $RQIndex$ is decremented every time a schedule time is assigned and is incremented every time the only waiting request in a queue gets cancelled or when a queue is selected for service.

The following example explains why with FCFS, the server may violate time of service guarantees. Let us assume that $t1 < t2 < t3 < t4 < t5 < t6$ and that $i \neq j$. Let us also assume that at the current state of the server, $t5$ is the next stream completion time that has not yet been assigned, and $t6$ is the completion time that immediately follows. At time $t1$, a new request, $R1$, arrives and joins the empty waiting queue i. So, the server gives $R1$ the time of service guarantee $t5$. At time $t2$, a new request, $R2$, arrives and joins the empty waiting queue j.

```
Event: Initialization
   RQIndex = -1;
Event: Arrival of request R to WQi
    if (a channel is available){  // the server is not fully loaded
       Service the request immediately;
       RQIndex++;
    }
    else {  // the server is fully loaded
      if (WQi.empty()){
          WQi.guarantee = RQ[RQIndex].compTime;
          RQIndex−−;
      }
      Inform R of the guarantee WQi.guarantee;
    }
Event: Cancellation of request R in WQi
    if (R is the only request in WQi)
      RQIndex++;
Event: Completion of a running stream
    if (at least one request is waiting){
      Service the queue with the longest waiting request;
      RQIndex++;
    }
```

Fig. 3.1: An Implementation of FCFS for Providing Time of Service Guarantees

Hence, the server gives $R2$ the time of service guarantee $t6$. At time $t3$, a new request, $R3$, arrives and joins the waiting queue i, which already has the request $R1$. So, the server assigns $R3$ the time of service guarantee $t5$. Assume that at time $t4$, $R1$ reneges (probably because it was given a far time of service guarantee). Thus, using FCFS (which selects the queue with the oldest request), the server will service $R2$ before $R3$, although $R3$ was given a better time of service guarantee. Assuming that the time of service guarantee $t4$ is precise (i.e., equal to the time

when service starts for the request(s) assigned to it), the server will violate the time of service guarantee of $R3$! I have also observed violations of time of service guarantees during simulations that use the same waiting tolerance of customers used in [42].

Chapter 4

Proposed Policies

This chapter presents Quantized FCFS (QFCFS) and Enhanced IML (IML$^+$).

4.1 Quantized FCFS (QFCFS)

Basing the scheduling decisions entirely on waiting times (as in FCFS) may not be advantageous when the oldest requests in multiple queues differ only a little in age. Similarly, basing the decisions entirely on queue lengths (as in MQL and MFQL) causes starvations and undermines server's ability to provide good predictive time of service guarantees. Hence, the proposed *Quantized FCFS* (QFCFS) policy examines both waiting times and queue lengths, thereby serving as a good compromise between FCFS and MQL/MFQL.

QFCFS works as follows. First, it translates waiting times into discrete levels. The interval between consecutive levels is called the *quantization interval* (Q). Then, it selects for service the queue with the largest *quantized* waiting time. Note that there may be multiple eligible queues. The selection criterion of one queue among these queues leads to two variants of QFCFS: *QFCFS-m* and *QFCFS-f*. *QFCFS-m* chooses the longest queue, while *QFCFS-f* chooses the queue with the largest factored length. QFCFS is a generalized policy because if Q approaches

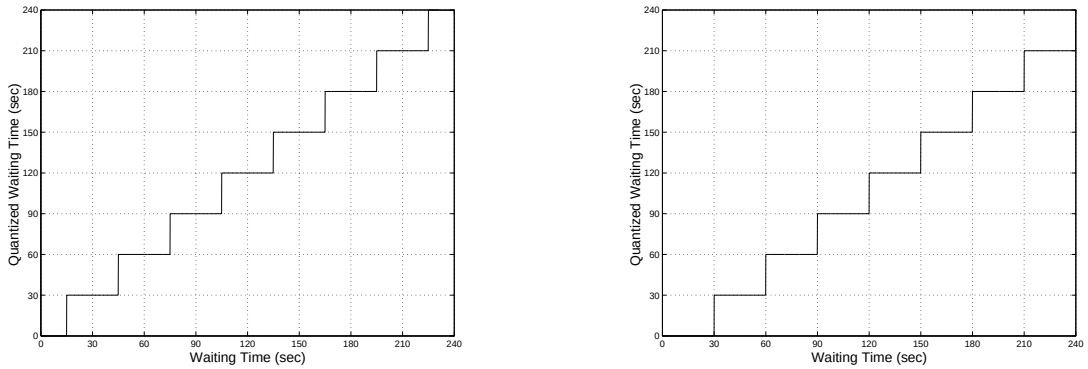
zero, then QFCFS becomes FCFS, and if Q approaches a sufficiently large number, QFCFS becomes MQL (or MFQL). The quantization can be performed by rounding or truncation. By rounding, a waiting time is translated to the closest level. By truncation, however, a waiting time is translated to the closest lower level. Figure 4.1 illustrates how quantization is performed in both variants when $Q = 30$ seconds.

Combining the two alternatives for the selection criterion and the two alternatives for quantization leads to four alternative implementations of QFCFS: *QFCFS-m-rnd*, *QFCFS-m-tnc*, *QFCFS-f-rnd*, and *QFCFS-f-tnc*.

QFCFS has a slightly higher implementation complexity than MQL or MFQL, depending on which variant of QFCFS is used. Note that in the actual implementation of QFCFS, the waiting times do not have to be computed every scheduling time. The scheduling can be performed based on the arrival times, which need to be quantized only once.

4.2 Enhanced IML (IML$^+$)

Minimum idling schemes were shown to be very effective in exploiting minimum waiting times [37]. Minimum waiting times can be estimated by the server either predictively or conservatively. These schemes partition the waiting queues into two sets: a hot set ($\mathcal{H}$) and a cold set ($\mathcal{C}$). A waiting queue belongs to $\mathcal{H}$ if it meets any of the following three conditions.



(a) QFCFS-rnd

(b) QFCFS-tnc

Fig. 4.1: The Quantization in QFCFS-rnd and QFCFS-tnc ($Q = 30$ sec)

- It corresponds to a popular movie. Because movies are numbered in decreasing order of their popularity, a movie is classified as popular if its number is less than a fixed number, τ. This fixed number is also referred to as the *hot movie threshold* in this study.
- It has more than one request.
- It has exactly one request and that request has been waiting longer than a pre-specified threshold, T.

A queue belongs to $\mathcal{C}$ if it does not meet any of these conditions. Minimum idling schemes give a higher priority to the queues in $\mathcal{H}$ and schedule the queues in $\mathcal{C}$ on a FCFS basis when $\mathcal{H}$ is empty. These schemes are not very sensitive to the value of τ because they can also recognize popular movies dynamically by examining the numbers of requests in the queues.

Minimum idling schemes include IML and IMQ, which differ in the selection criterion of a queue in $\mathcal{H}$. IML selects the queue with the largest expected loss. The loss of a queue is defined as the number of its requests that will exceed the minimum waiting times by the next scheduling time (next stream completion time) if they are not selected for service at the current scheduling time. In contrast, IMQ simply selects the longest queue.

Note that the expected loss of a queue is a discrete number with a typically small value. Therefore, more than one queue in $\mathcal{H}$ may meet the scheduling criterion of IML. In such situations, IML blindly chooses any of these queues. Conversely, the proposed *Enhanced IML* (IML^+) policy exploits these situations by selecting the queue with the largest factored length among the set of all eligible queues. (I have found that in these situations, choosing the longest queue is not as effective as choosing the queue with the largest factored length.) By combining the benefits of IML and IMQ, IML$^+$ can both improve throughput and reduce the mean request waiting time. IML$^+$ has a slightly higher implementation complexity than IML, whereas IMQ has the lowest complexity among the three.

Chapter 5

Performance Evaluation

This chapter analyzes the performance of FCFS, FCFS-sum, MQL-u, MQL-f, MFQL, IML, IMQ, QFCFS, and IML$^+$ through extensive simulation. I have developed a simulator for a VOD server that supports various scheduling policies and have validated the simulator by reproducing several graphs in previous studies. The simulated server starts with a state close to the steady state of the common case to accelerate the simulation. The simulation, therefore, starts with a server delivering its full capacity of running streams, and the remaining time for the completion of each of these streams is uniformly distributed between zero and the normal movie length. The simulation stops after a steady state analysis with 95% confidence interval is guaranteed. Next, I discuss the workload characteristics and then present the main results.

5.1 Workload Characteristics

Like most prior studies, I assume that the arrival of the requests to a VOD server follows a Poisson Process with an average arrival rate λ. Hence, the inter-arrival time is exponentially distributed with a mean $T = 1/\lambda$. I also assume, as in previous works, that the accesses to movies are highly localized and follow a

Zipf-like distribution [9]. With this distribution, the probability of choosing the n^{th} most popular of M movies is $C/n^{1-\theta}$ with a parameter θ and a normalized constant C. The parameter θ controls the skewness of movie access. Note that the skewness reaches its peak when $\theta = 0$, and that the access becomes uniformly distributed when $\theta = 1$. I assume that $\theta = 0.271$ in accordance with prior studies. I study a VOD server with 120 movies, each of which is 120-minute long, and I examine the server at different loads by fixing the request arrival rate at 40 requests per second and varying the number of channels (server capacity) generally from 500 to 4000. To keep the discussion focused, I assume that batching is the primary resource sharing technique and do not consider any VCR-like operations. These operations can be supported by allocating contingency channels [15]. I also assume that the server knows exactly the access frequency of each movie. This may lead to overestimating the performance of MFQL. The performance of minimum idling schemes is not sensitive to the value of the hot movie threshold τ because they also classify movies as hot or cold dynamically. The threshold τ is fixed at 20.

As in previous studies, I characterize the waiting tolerance of customers by two distributions: an exponential distribution with $\mu = 4$ minutes and a truncated normal distribution with $\mu = 4$ minutes and $\sigma = 1.33$ minutes. With truncation, the waiting times that are negative or greater than 15 minutes are excluded.

I characterize the waiting tolerance with IMQ, IML, and IML$^+$ by the following two distributions. The first is a normal distribution with $\mu = 4$ minutes

and $\sigma = 1.33$ minutes. In the second, which is referred to as *C+Exponential* in this thesis, customers exhibit a minimum waiting time of 3 minutes, followed by an exponential time with $\mu = 3$ minutes. These distributions were used in [37].

To study the effectiveness of providing time of service guarantees, I characterize the waiting tolerance by a *NormalG* distribution. In this distribution, the customers who receive time of service guarantee will wait for service if their waiting times will be less than 4 minutes; the waiting times of all other customers follow a truncated normal distribution with $\mu = 4$ minutes and $\sigma = 1.33$ minutes. This distribution was used in [42] (under a different name).

For ease of reference in the subsequent analysis, Table 5.1 summarizes the workload characteristics.

Table 5.1: A Summary of Workload Characteristics

Distribution of Request Arrival	*Poisson Process*
Request Arrival Rate (λ)	40 requests/second
Distribution of Movie Access	*Zipf-Like*
Skewness in Movie Access (θ)	0.271
Number of Movies	120
Movie Duration	120 minutes
Server Capacity	500 - 4000
Hot Movie Threshold (τ)	20
Distributions of Waiting Tolerance	*Exponential*, *Normal*, *C+Exponential*, and *NormalG*.

5.2 Result Presentation and Analysis

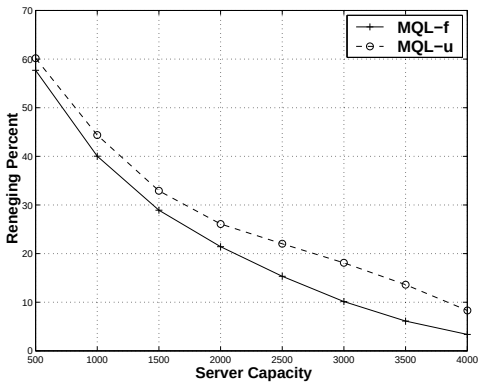
In this section, the various scheduling policies are compared, and then the main results are summarized.

5.2.1 Comparing MQL Schemes: MQL-f and MQL-u

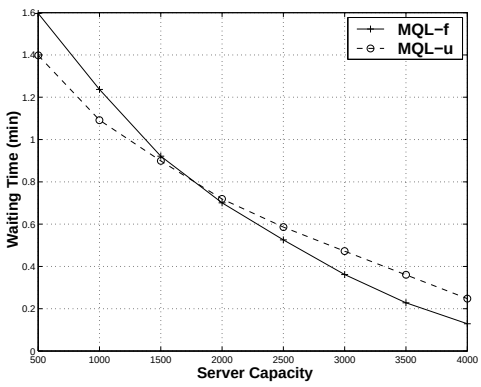
Let us first analyze the performance of MQL-f and MQL-u. Figures 5.1 and 5.2 compare MQL-f and MQL-u in terms of reneging percent, average request waiting time, and unfairness. The results are shown for the cases in which the waiting tolerance of customers follows an exponential and a normal distribution, respectively. These results show that MQL-f is not only significantly fairer than MQL-u, but it also yields higher throughput. MQL-u, however, reduces the waiting times better for high server capacities. The superior throughput with MQL-f comes from improved batching. In particular, the requests for more popular movies will be given the chance of accumulating for longer periods of time in the waiting queues. This also explains the increase in waiting times.

5.2.2 Comparing FCFS, FCFS-sum, MQL-f, and MFQL

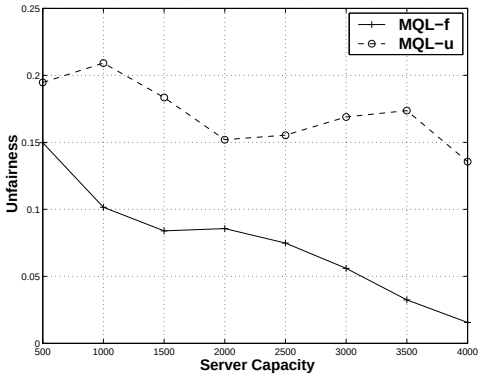
Let us now analyze the performance of FCFS, FCFS-sum, MQL-f, and MFQL. Figures 5.3 and 5.4 compare these policies in terms of the three performance metrics when the waiting tolerance follows an exponential and a normal distribution, respectively. The performance of RAND, which selects randomly a



(a) In Reneging Percent



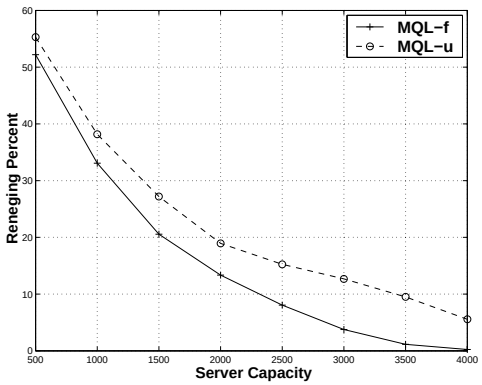
(b) In Waiting Time



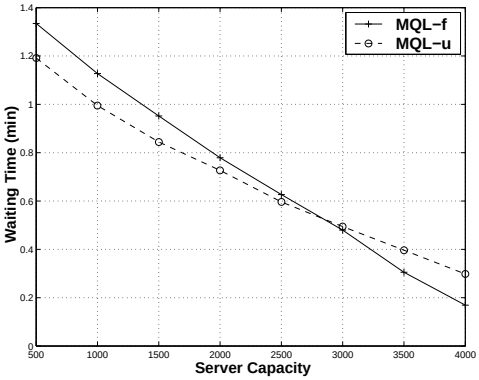
(c) In Unfairness

Fig. 5.1: Comparing Variants of MQL (Exponential Distribution)

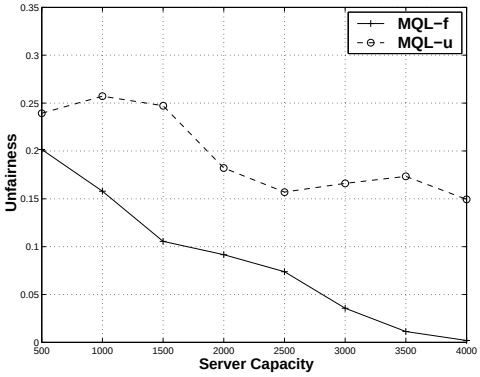
waiting queue for service, is also shown to demonstrate the effectiveness of the other policies. The results indicate that MQL-f and MFQL perform nearly as well in terms of the three performance metrics. The little performance edge of MFQL may diminish if the server has no perfect knowledge of the movie access frequencies. Moreover, MQL-f is much simpler as it does not require periodic computations of access frequencies. Hence, contrary to [2], MQL (MQL-f in particular)



(a) In Reneging Percent



(b) In Waiting Time

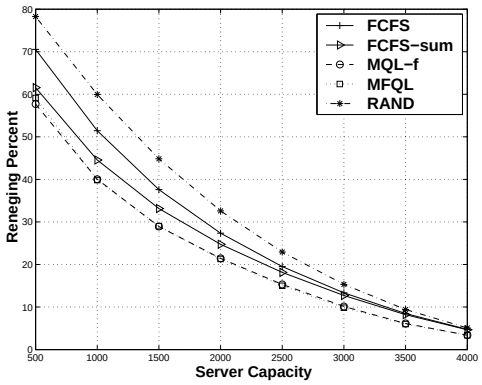


(c) In Unfairness

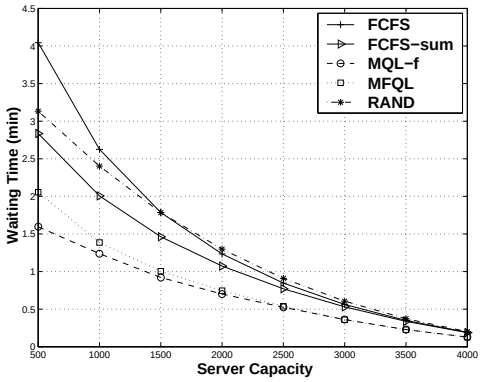
Fig. 5.2: Comparing Variants of MQL (Normal Distribution)

may be preferred over MFQL. The results also show that MQL-f and MFQL always perform the best in terms of the average waiting time. Interestingly, RAND outperforms FCFS in terms of the mean waiting time when the tolerance follows a normal distribution. Moreover, the results demonstrate that MQL-f and MFQL provide better throughput than FCFS and FCFS-sum when the waiting tolerance follows an exponential distribution. When the waiting tolerance follows a normal

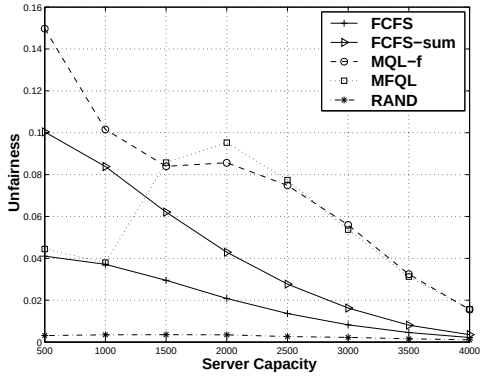
distribution, however, FCFS-sum yields the highest throughput, and FCFS performs nearly as well for high server capacities. FCFS-sum outperforms FCFS in terms of throughput and the mean waiting time because it considers the waiting times of all request in each queue. In terms of unfairness, RAND has the absolute edge, followed by FCFS and then FCFS-sum. As expected, all policies perform almost identically for very high server capacities.



(a) In Reneging Percent

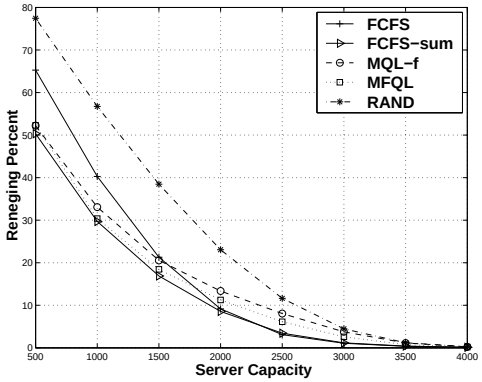


(b) In Waiting Time

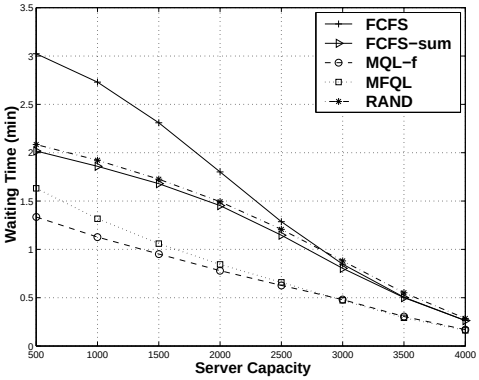


(c) In Unfairness

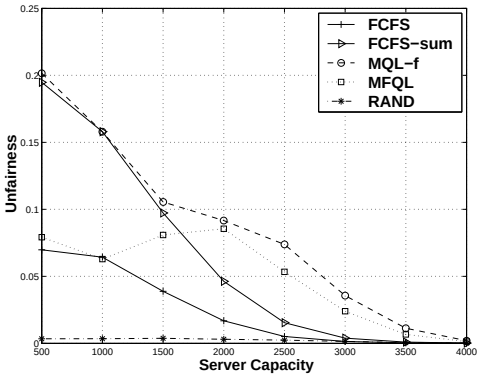
Fig. 5.3: Comparing FCFS, FCFS-sum, MQL-f, MFQL, and RAND (Exponential Distribution)



(a) In Reneging Percent



(b) In Waiting Time



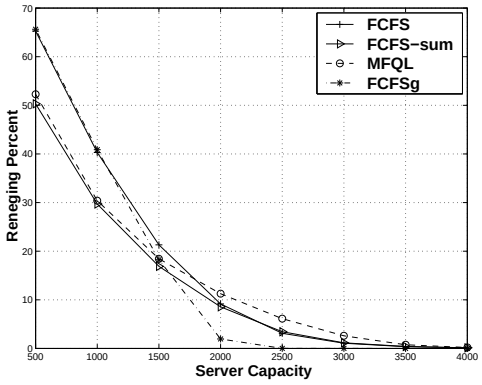
(c) In Unfairness

Fig. 5.4: Comparing FCFS, FCFS-sum, MQL-f, MFQL, and RAND (Normal Distribution)

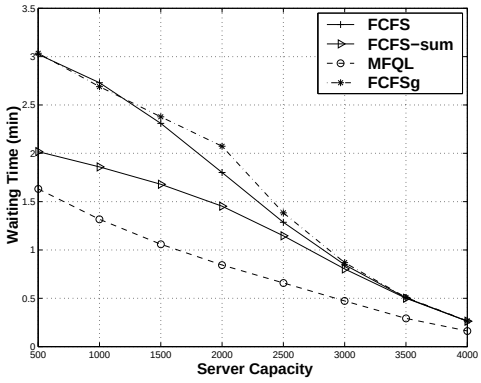
5.2.3 Providing Time of Service Guarantees

Let us now discuss the performance advantages of providing customers with time of service guarantees. FCFS that may provide time of service guarantees is referred to as FCFSg in this thesis. Figure 5.5 shows how well FCFSg performs

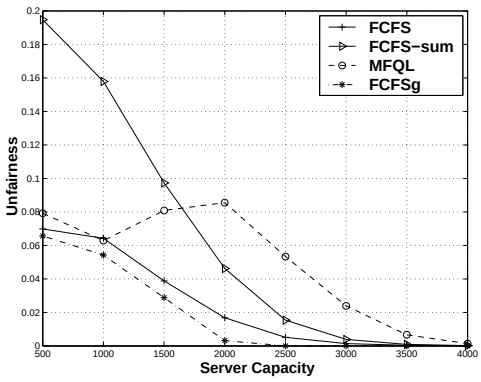
in comparison with FCFS, FCFS-sum, and MFQL when the waiting tolerance follows a normalG distribution (discussed in Section 5.1). The results here are based on the assumption that only FCFSg can encourage customers to wait. Note that FCFSg results in the best throughput and generally the best fairness, but it performs the worst in terms of the mean waiting time.



(a) In Reneging Percent



(b) In Waiting Time



(c) In Unfairness

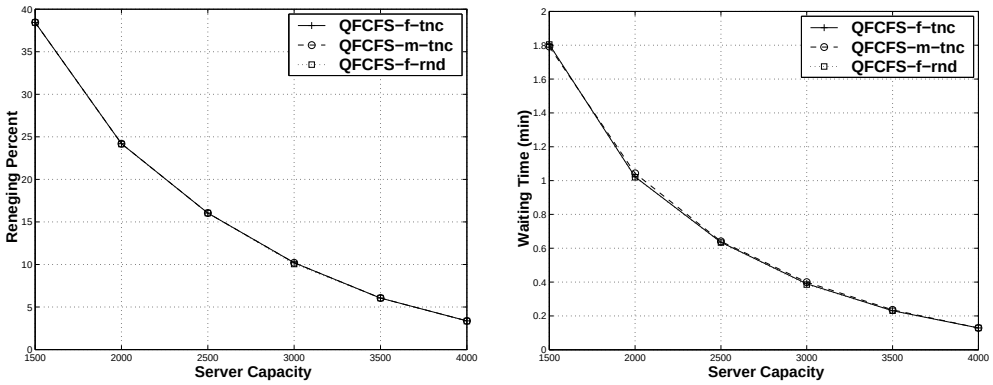
Fig. 5.5: Effectiveness of FCFSg (NormalG Distribution)

FCFSg provides superior throughput by motivating customers to wait, and the performance gains depend greatly on the resultant patterns of waiting tolerance. It is unclear, however, whether the normalG distribution can accurately characterize the real waiting tolerance because of the lack of any modeling study of servers that provide time of service guarantees. Moreover, FCFSg may violate its time of service guarantees, and it is not the only policy that can influence the waiting tolerance. All other scheduling policies can also motivate customers to wait to various degrees by providing them with expected starting times of service. The expected starting times of service can be estimated based on server load, the completion times of running streams, and the history of waiting times. Because the average waiting times differ widely from one queue to another, basing the scheduling decisions on each particular queue can produce more accurate estimates of the expected starting times of service.

5.2.4 Effectiveness of QFCFS

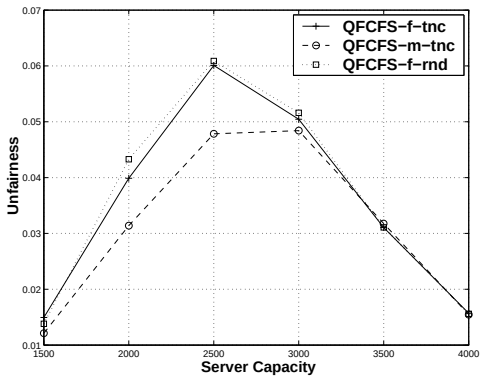
Let us now examine the performance of the proposed QFCFS policy. I consider here only the case when the waiting tolerance follows an exponential distribution. When the tolerance follows a normal distribution, FCFS already performs better than MQL/MFQL in terms of throughput, so the quantization in that case may not be advantageous. Figure 5.6 compares the performance of

three variants of QFCFS: QFCFS-f-tnc, QFCFS-m-tnc, QFCFS-f-rnd. (QFCFS-m-rnd is not shown because it performs almost exactly the same as QFCFS-m-tnc.) These results indicate that these variants perform nearly the same in terms of throughput and waiting times. QFCFS-m-tnc, however, is the fairest. Therefore, the subsequent analysis is limited to QFCFS-m-tnc, which also offers the simplest implementation.



(a) In Reneging Percent

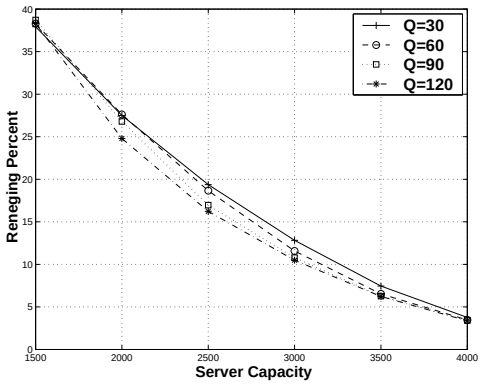
(b) In Waiting Time



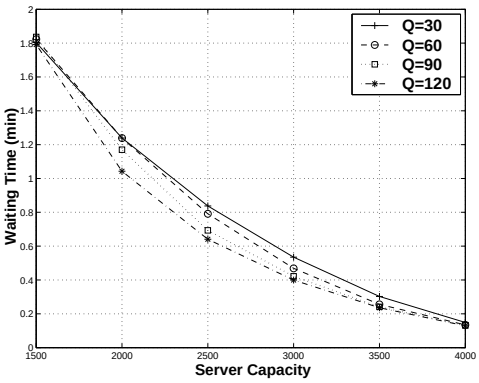
(c) In Unfairness

Fig. 5.6: Comparing Variants of QFCFS (Exponential Distribution, $Q = 120$ sec)

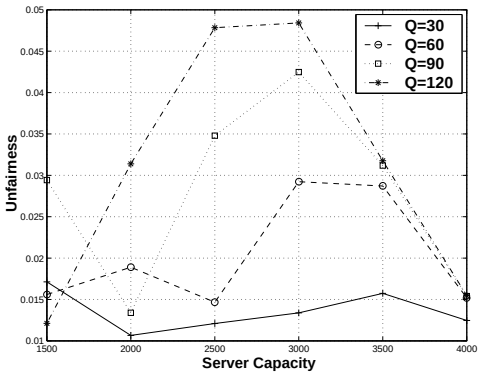
Figure 5.7 demonstrates the impact of the quantization interval (Q) on performance. As expected, both the throughput and the waiting times improve with larger values of Q, but unfairness increases.



(a) In Reneging Percent



(b) In Waiting Time



(c) In Unfairness

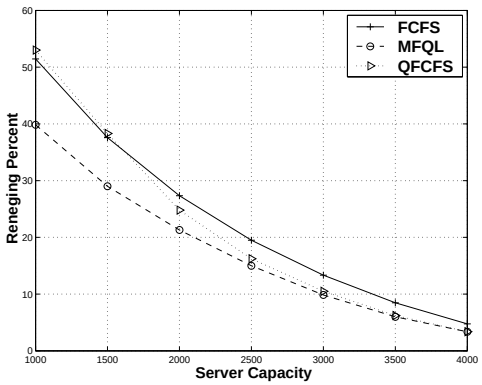
Fig. 5.7: Effect of Quantization Level (Exponential Distribution)

Figure 5.8 compares QFCFS with FCFS and MFQL. (MFQL is shown here instead of MQL-f because it results in slightly shorter waiting times when it has perfect knowledge of access frequencies.) Note that the performance of QFCFS in

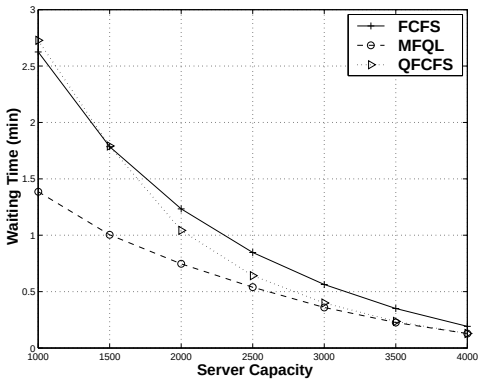
terms of three metrics approaches that of MFQL as the server capacity increases. Thus, QFCFS performs as well as MFQL (especially for high server capacities) while being able to prevent starvations. QFCFS can also provide reasonably-accurate predictive time of service guarantees because these guarantees can be based on next stream completion times. Note that as Q decreases, the accuracy of prediction increases, while the performance (in terms of throughput and waiting times) degrades. So, Q should be chosen as a tradeoff. I could not quantify the impact of providing predictive time of service guarantees because of the lack of any study that models the waiting tolerance in such cases.

5.2.5 Exploiting Minimum Waiting Times

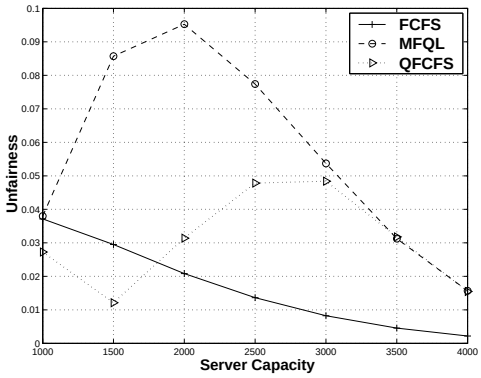
Finally, let us analyze the performance of the policies that exploit minimum request waiting times: IMQ, IML, and the proposed IML$^+$ policy. As discussed in Section 6.1, I characterize the waiting tolerance by two distributions: normal and c+exponential. As in [37], the threshold T in the case of a c+exponential distribution is set to to the minimum waiting time. By contrast, in the case of a normal distribution, where there is no fixed minimum waiting time, the best value of T should be found. I introduce the parameter H, which is defined as the ratio of the threshold T to the average request waiting time (which is mean of the normal distribution, μ).



(a) In Reneging Percent



(b) In Waiting Time

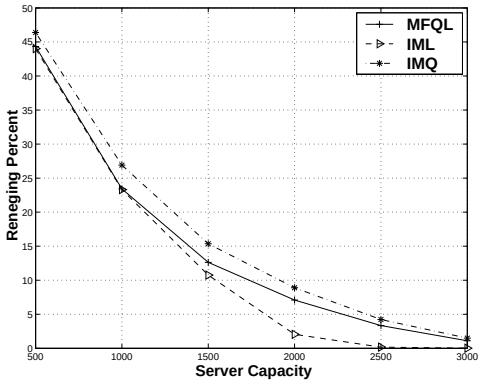


(c) In Unfairness

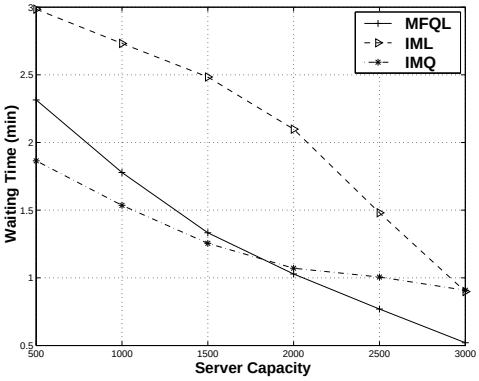
Fig. 5.8: Effectiveness of QFCFS (Exponential Distribution, $Q = 120$ sec)

Figure 5.9 compares the performance of IML, IMQ, and MFQL in the case of a c+exponential distribution. (FCFS-sum and FCFS do not perform as well as MFQL with this distribution.) The results in the case of a normal distribution are not shown because they follow a very similar behavior. Note that IML yields the highest throughput but leads to the longest waiting times. IML is also the fairest

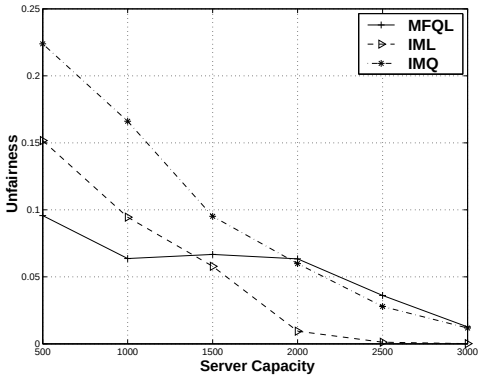
for high server capacities. In contrast, IMQ performs relatively well in terms of the waiting times but yields the lowest throughput.



(a) In Reneging Percent



(b) In Waiting Time

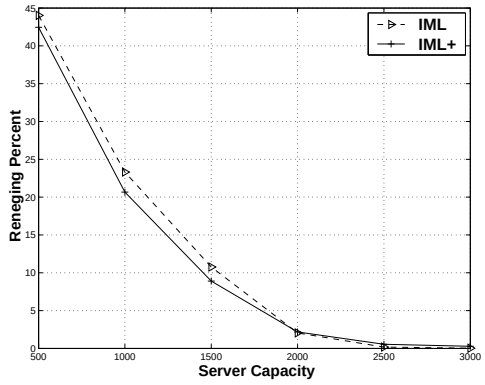


(c) In Unfairness

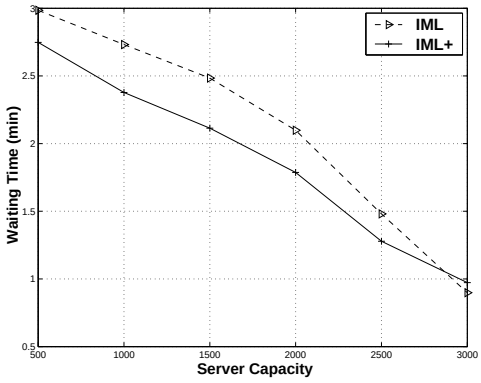
Fig. 5.9: Comparing IML, IMQ, and MFQL (C+Exponential Distribution)

Figures 5.10 and 5.11 show the effectiveness of the proposed IML^+ policy in comparison with IML when the waiting tolerance follows c+exponential and normal distributions, respectively. The results show that IML^+ achieves up to 21% better throughput than IML when the waiting tolerance follows a c+exponential

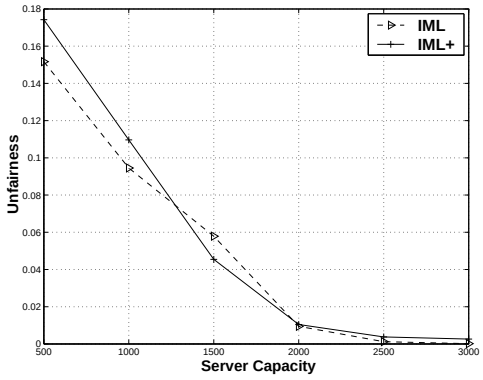
distribution and up to 15% better throughput when the tolerance follows a normal distribution. IML$^+$ also reduces the average waiting time by up to 18% when the tolerance follows a c+exponential distribution and by up to 21% in the case of a normal distribution. Each of IML$^+$ and IML is fairer than the other in certain regions of the curve.



(a) In Reneging Percent

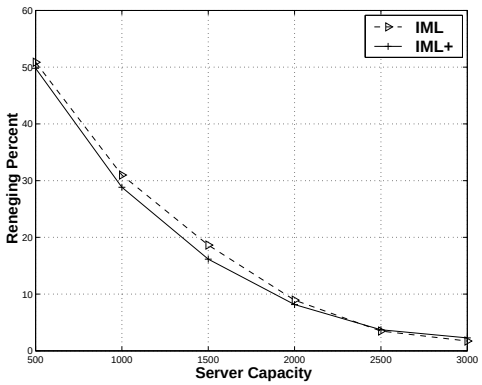


(b) In Waiting Time

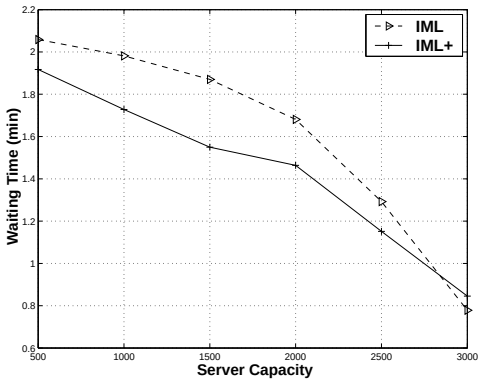


(c) In Unfairness

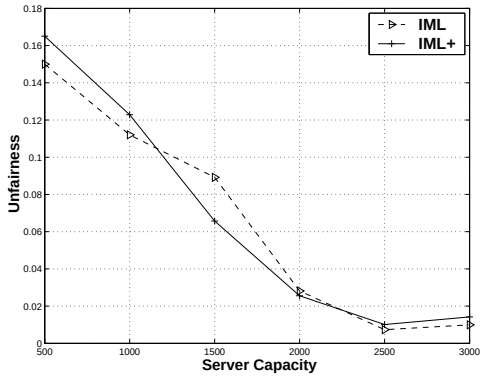
Fig. 5.10: Effectiveness of IML$^+$ (C+Exponential Distribution)



(a) In Reneging Percent



(b) In Waiting Time

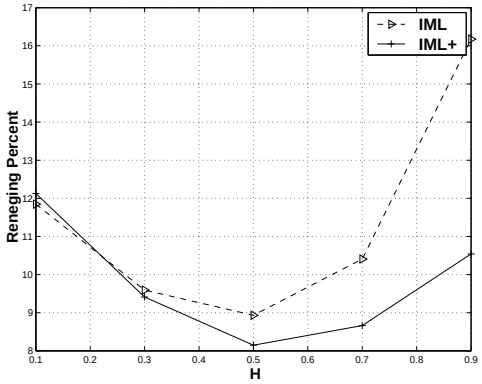


(c) In Unfairness

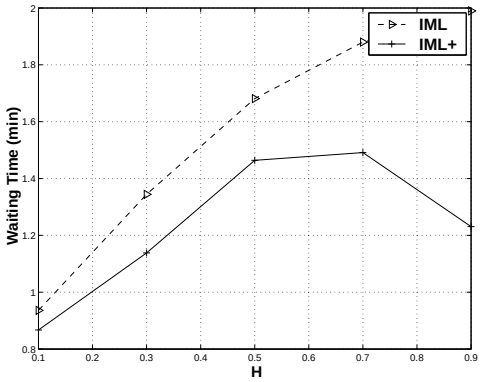
Fig. 5.11: Effectiveness of IML$^+$ (Normal Distribution, $H = 0.5$)

Figure 5.12 illustrates the impact of the selected value of H on the performance of IML$^+$ and IML. Note the reneging percent decreases with H until H reaches 0.5, and then, it starts to go up. In contrast, the waiting time generally increases with H, except when IML$^+$ is applied and H is greater than 0.7. The unfairness decreases with H until H reaches 0.7. Thus, the value of H may be selected as a compromise, but I believe that 0.5 is the best value because the throughput is

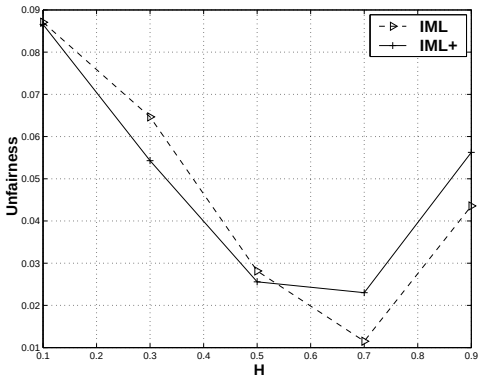
the most important performance metric. Also note that IML$^+$ almost invariably outperforms IML in terms of throughput and waiting times.



(a) In Reneging Percent



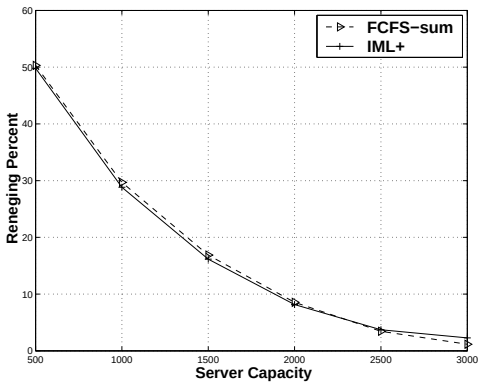
(b) In Waiting Time



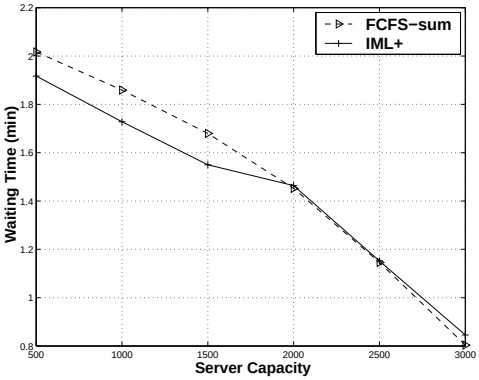
(c) In Unfairness

Fig. 5.12: Effect of Threshold H (Normal Distribution, 2000 Channels)

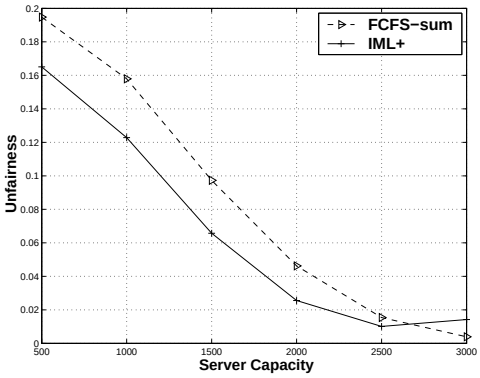
Interestingly, IML$^+$ also performs almost identically to FCFS-sum when the tolerance follows a normal distribution (where there is no fixed minimum waiting time) as shown in Figure 5.13.



(a) In Reneging Percent



(b) In Waiting Time



(c) In Unfairness

Fig. 5.13: Comparing FCFS-sum and IML$^+$ (Normal Distribution, $H = 0.5$)

5.2.6 Summary of Results

The relative performance of scheduling policies depends greatly on the waiting tolerance of customers. Thus, let us summarize the results for each tolerance model separately. Table 5.2 compares the best performers when the tolerance follows an exponential distribution. The numbers indicate the rank of the corresponding policies in terms of the specified performance metrics. In the context

of "starvations", a "1" means that the policy prevents starvations, while a "2" means the opposite. Interestingly, MQL-f performs nearly as well as MFQL in terms of throughput, waiting times, and unfairness, even when assuming that MFQL has perfect knowledge of movie access frequencies. In addition, MQL-f is simpler because it does not require periodic computations of access frequencies. Thus, contrary to [2], MQL (MQL-f in particular) may be preferred over MFQL. When the reneging percentage is less than 15, which should be the common case as much larger reneging percentages would be unacceptable, QFCFS performs very closely to MFQL and MQL-f in terms of throughput and waiting times, and the performance gap becomes negligible when the reneging percentage falls below 10. QFCFS can also prevent starvations, whereas MFQL and MQL-f cannot. Moreover, QFCFS is generally fairer and can provide more accurate time of service guarantees than MFQL and MQL-f.

Table 5.2: The Best Performers with an Exponential Distribution of Tolerance

Policy	Throughput	Waiting Time	Fairness	Starvations
MQL-f	1	1	2	2
MFQL	1	1	2	2
QFCFS	2	2	1	1

Table 5.3 compares the best performers when the tolerance follows a normal distribution. In this group, FCFS-sum and IML$^+$ perform generally better than FCFS, and they perform very closely to each other in terms of various metrics.

They also have comparable implementation complexities. IML$^+$ is only a little fairer than FCFS-sum.

Table 5.3: The Best Performers with a Normal Distribution of Tolerance

Policy	Throughput	Waiting Time	Fairness	Starvations
FCFS	2	2	1	1
FCFS-sum	1	1	3	2
IML$^+$	1	1	2	2

Table 5.4 compares the best performers when the tolerance follows a c+exponential distribution. In this case, IML$^+$ is the clear winner (Figures 5.9 and 5.10).

Table 5.4: The Best Performers with a C+Exponential Distribution of Tolerance

Policy	Throughput	Waiting Time	Fairness	Starvations
IML	2	2	1	2
IML$^+$	1	1	1	2

When assuming that FCFS provides true time of service guarantees and that all other policies cannot influence customers to wait, FCFS can lead to the best throughput, but it results in the longest waiting times. The performance gains of FCFS depend greatly on the resultant waiting tolerance of customers, but there is no modeling study of the waiting tolerance in the presence of time of service guarantees. Moreover, FCFS is not the only policy that can influence customers to wait. All other scheduling policies can motivate customers to wait to various degrees by providing them with the expected waiting times.

Chapter 6

Conclusions

The design and analysis of various alternatives to improve the performance of *multimedia-on-demand* (MOD) servers in general and *video-on-demand* (VOD) servers in particular has become a major research focus. Increasing the degrees of resource sharing through intelligent scheduling is one such avenue and is the theme of this thesis.

I have conducted an in-depth investigation of many existing scheduling policies, including FCFS, MQL, MFQL, IML, and IMQ. In contrast with [42], I have shown that FCFS may violate its time of service guarantees. I have also shown that the discrepancy in [13, 37, 2] with regard to the relative performance of MQL to FCFS is caused by alternative implementations of MQL. I have considered two implementations of MQL: *MQL-f* and *MQL-u*. MQL-f selects the queue of the least popular movie among the longest queues, while MQL-u selects the queue of the most popular movie. Moreover, I have studied the effectiveness of *Longest Wait First* (LWF or FCFS-sum) [17, 45] in VOD servers.

I have also proposed two scheduling policies: *Quantized FCFS* (QFCFS) and *Enhanced IML* (IML^+). QFCFS combines the benefits of FCFS and MQL/MFQL by scheduling requests based on both waiting times and queue lengths. IML^+

improves IML by capturing the situations in which multiple queues become eligible candidates for selection.

I have evaluated the effectiveness of various scheduling policies through extensive simulation. I have examined the impacts of customer waiting tolerance and server capacity (or server load) on the performance of each policy in terms of the overall customer reneging probability, the average customer waiting time, and unfairness against unpopular movies. The first objective is the most important because it translates to server throughput. Moreover, I have compared the policies in terms of other objectives, such as implementation complexity, ability to prevent starvations, and ability to provide (predictive) time of service guarantees. The results can be summarized as follows.

- MQL-f is not only fairer than MQL-u, but it also yields higher throughput, especially for high server capacities.
- MQL-f performs nearly as well as MFQL in terms of throughput, waiting times, and unfairness, even when assuming that MFQL has perfect knowledge of movie access frequencies. MQL-f is also simpler. Thus, contrary to [2], MQL (MQL-f in particular) may be preferred over MFQL.
- QFCFS is recommended when the waiting tolerance follows an exponential distribution and when the server is not very heavily loaded. With QFCFS, a server can support as many concurrent customers for high server capacities as

MQL-f and MFQL and can start their service as immediately, while being relatively fair and able to prevent starvations and provide reasonably-accurate predictive time of service guarantees. In contrast, MQL-f is recommended when the server is very heavily loaded. Because QFCFS is a generalized form of MQL/MFQL and FCFS, this behavior in the case of an exponential distribution motivates the use of a dynamic QFCFS policy that adjusts the quantization interval based on server load.

- When the waiting tolerance follows a normal distribution, FCFS-sum and IML^+ achieve the best overall performance. IML^+ tends to be a little fairer than FCFS-sum.
- When customers exhibit minimum waiting times, IML^+ achieves the best overall performance.

The results indicate that the waiting tolerance of customers determines the most appropriate scheduling policy. The policies that achieve the best overall performance are dynamic QFCFS (when the tolerance follows an exponential distribution) and IML^+ (when the server follows a normal distribution or when the customers exhibit minimum waiting times).

This thesis provides two main opportunities for future research. First, it highlights the importance of developing and analyzing scheduling policies that provide hard time of service guarantees as FCFS cannot provide such guarantees. In contrast with FCFS, these policies should schedule the requests based on

assigned schedule times rather than the waiting times for service. Second, the thesis demonstrates the importance of modeling customer waiting tolerance in VOD servers that provide hard time of service guarantees as well as those that provide predictive guarantees.

References

[1] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, On Optimal Piggyback Merging Policies for Video-On-Demand Systems. *In Proceedings of the ACM SIGMETRICS Conference on Measurements and Modeling of Computer Systems*, 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 Transactions on Computers*, 50(2): 97-110, February 2001.

[3] D. Aksoy and M. J. Franklin. Scheduling for Large-Scale On-Demand Data Broadcasting, *In Proceedings IEEE INFOCOM*, pages 651-659, April 1998.

[4] J. Almeida, D. Eager, and M. Vernon. A Hybrid Caching Strategy for Streaming Media Files. *In Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking*, January 2001.

[5] D. P. Anderson, Y. Osawa, and R. Govindan. A File System for Continuous Media. *ACM Transactions on Computer Systems*, 10(4): 311-337, November 1992.

[6] Y. Cai, K. A. Hua. An Efficient Bandwidth-Sharing Technique for True Video on Demand Systems. *In Proceedings of the ACM International Conference on Multimedia*, pages 211-214, 1999.

[7] Y. Cai, K. A. Hua, and K. Vu. Optimizing patching performance. *In Proceedings of the IS&T/SPIE Conference on Multimedia Computing and Networking*, pages 204-215, 1999.

[8] S. R. Carter, J.-F. Pâris, S. Mohan, and D. D. E. Long. A Dynamic Heuristic Broadcasting Protocol for Video-on-Demand. *In Proceedings of the International Conferencerence on Distributed Computing Systems (ICDCS 2001)*, pages 657-664, April 2001.

[9] A. L. Chervenak. *Tertiary Storage: An Evaluation of New Applications*. Ph.D. Thesis, U.C. Berkeley, December 1994. U.C. Berkeley Technical Report UDB/CSD 94/847, December 1994.

[10] A. L. Chervenak, D. A. Patterson, and R. H. Katz. Choosing the Best Storage Systems for Video Service. *In Proceedings of the ACM Conference on Multimedia*, pages 109-119, November 1995.

[11] C. Chou, L. Golubchik, J. C .S. Lui. Striping Doesn't Scale: How to Achieve Scalability for Continuous Media Servers with Replication. *In Proceedings of the International Conferencerence on Distributed Computing Systems (ICDCS 2000)*, pages 64-71, April 2000.

[12] A. Dan, and D. Sitaram. *Buffer Management Policy for an On-Demand Video Server*. Technical Report RC 19347, IBM Watson Research Center, January 1994.

[13] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling Policies for an On-Demand Video Server with Batching. *In Proceedings of the ACM Conference on Multimedia*, pages 391-398, October 1994.

[14] A. Dan, D. M. Dias, R. Mukherjee, D. Sitaram, R. Tewari. Buffering and Caching in Large-Scale Video servers. *In Digest of Papers. IEEE International Computer Conference*, pages 217-225, March 1995.

[15] 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, November 1995.

[16] A Dan and D. Sitaram. *Multimedia Caching Strategies for Heterogeneous Application and Server Environments*. Technical Report RC 20670, IBM Watson Research Center, December 1996.

[17] H. D. Dykeman, M. H. Ammar, and J. W. Wong. Scheduling Algorithms for Videotex Systems under Broadcast Delivery. *In Proceedings of IEEE International Conference on Communication*, pages 1847-1851, June 1986.

[18] R. Flynn, and W. Tetzlaff. Disk Striping and Block Replication Algorithms for Video File Servers. *In Proceedings of the International Conference on Multimedia Computing and Systems*, pages 590-597, June 1996.

[19] L. Giuliano. *Deploying Native Multicast across the Internet*. Online white paper at `http://www.sprintlink.net/multicast/whitepaper.html`.

[20] L. Golubchik, J. Lui, 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.

[21] L. Golubchik, J. C. S. Lui, and R. Muntz. Reducing I/O Demand in Video-On-Demand Storage Servers. *In Proceedings of the ACM SIGMETRICS Conference on Measurements and Modeling of Computer Systems*, pages 25-36, May 1995.

[22] A. Hu. Video-on-Demand Broadcasting Protocols: A Comprehensive Study. *In Proceedings of IEEE INFOCOM*, Vol 1, pages 508-517, April 2001.

[23] K. A. Hua, S. Sheu. Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand System. *In Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'97)*, pages 89-100, September 1997.

[24] K. A. Hua, Y. Cai, and S. Sheu. Patching: A Multicast Technique for True Video-on-Demand Services. *In Proceedings of ACM Multimedia*, pages 191-200, September 1998.

[25] L. Juhn and L. Tseng. Harmonic Broadcasting for Video-on-Demand Service. *IEEE Transactions on Broadcasting*, 43(3): 268-271, September 1997.

[26] S. Jamin, S. Shenkar, L. Zhang, and D. D. Clark. An admission Control Algorithm for Predictive Real-Time Service. *In Proceedings of the International*

Workshop on Network and Operating System Support for Digital Audio and Video, pages 349-356, November 1992.

[27] S. Kim, C. R. Das, and A. Sivasubramaniam. Performance Analysis of A Buffer Management Technique for Interactive Video-on-Demand. *In Proceedings of the International Conference on Multimedia Modeling*, 2000.

[28] F. Y. S. Lin. Optimal Real-time Admission Control Algorithms for Video-On-Demand (VOD) Service. *IEEE Transactions on Broadcasting*, 44(4): 402-408, December 1998.

[29] J. Nieh and M. S. Lam. The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications. *In Proceedings of the ACM Symposium on Operating Systems Principles*, pages 184-197, October 1997.

[30] J.-F. Pâris, S. W. Carter, and D. D. E. Long. Efficient Broadcasting Protocols for Video on Demand. *In Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 127-132, July 1998.

[31] J.-F. Pâris. A Fixed-Delay Broadcasting Protocol for Video-on-Demand. *In Proceedings of the International Conference on Computer Communications and Networks*, pages 418-423, October 2001.

[32] P. V. Rangan, and H. M. Vin. Designing File Systems for Digital Video and Audio. *In Proceedings of the ACM Symposium on Operating Systems Principles*, pages 81-94, October 1991.

[33] A. L. N. Reddy, J. Wyllie. Disk Scheduling in a multimedia I/O system. *In Proceedings of the ACM Conference on Multimedia*, pages 225-233, August 1993.

[34] N. J. Sarhan and C. R. Das. Adaptive Block Rearrangement Algorithms for Video-On-Demand Servers. *In Proceedings of International Conference on Parallel Processing*, pages 452-459, September 2001.

[35] N. J. Sarhan and C. R. Das. An Integrated Resource Sharing Policy for Multimedia Storage Servers Based on Network-Attached Disks. To appear in *Proceedings of the 23rd International Conference on Distributed Computing Systems* (ICDCS 2003), May 2003.

[36] S. Sen, L. Gao, J. Rexford, and D. Towsley. Optimal Patching Schemes for Efficient Multimedia Streaming. *In Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, June 1999.

[37] H. Shachnai and P. S. Yu. Exploiting Wait Tolerance in Effective Batching for Video-on-Demand Scheduling. *Multimedia Systems*, 6(6): 382-394, 1998.

[38] P. Shenoy, and V. Harric. Efficient Striping Techniques for Variable Bit Rate Continuous Media File Servers. *Performance Evaluation Journal*, 38(2): 175-199, December 1999.

[39] Sprint. *SprintLink Multicast*. Online document at `http://www.sprintlink.net/multicast/whitepaper.html`.

[40] R. Tewari, H. M. Vin, A. Dan, and D. Sitaram. Resource-based Caching for Web Servers. *Proceedings of the SPIE Conference on Multimedia Computing and Networking*, pages 191-204, January 1998.

[41] D. A. Tran, K. A. Hua, T. T. Do. Layered Range Multicast for Video on Demand. *In Proceedings of IEEE International Conference on Computer Communications and Networking*, pages 210-215, October 2002.

[42] A. K. Tsiolis and M. K. Vernon. Group-Guaranteed Channel Capacity in Multimedia Storage Servers. *In Proceedings of the ACM SIGMETRICS Conference on Measurements and Modeling of Computer Systems*, pages 285-297, June 1997.

[43] H. M. Vin and P. V. Rangan. Admission Control Algorithms for Multimedia-On-Demand Servers. *In Proceedings of the International Workshop for Digital Audio and Video*, pages 56-68, November 1992.

[44] H. M. Vin, P. Goyal, and A. Goyal. A Statistical Admission Control Algorithms for Multimedia Servers. *In Proceedings of the ACM Multimedia*, pages 33-40, October 1994.

[45] J. W. Wong and M. H. Ammar. Analysis of Broadcast Delivery in a Videotex System. *IEEE Transactions on Computers*, 34(9): 863-866, September 1985.