A Detailed Study of Request Scheduling in Multimedia Systems[1]

Nabil J. Sarhan *Chita R. Das*

Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16802
Phone: (814) 865-0194
E-mail: {sarhan,das}@cse.psu.edu

Abstract

The investigation of various alternatives to improve the performance of *multimedia-on-demand* (MOD) servers has become a major research focus because of the rapidly growing interest in MOD services, especially on the Internet. The performance of MOD servers can be enhanced significantly through resource sharing. The exploited degrees of resource sharing depend greatly on how these servers schedule the waiting requests. By scheduling the requests intelligently, a server can support more concurrent customers and can reduce their waiting times for service. In this paper, we provide a detailed analysis of existing scheduling policies and propose two new policies, called *Quantized First Come First Serve* (QFCFS) and *Enhanced Minimum Idling Maximum Loss* (IML$^+$). We demonstrate the effectiveness of these policies through simulation and show that they suite different patterns of customer waiting tolerance.

Keywords: Multimedia-on-demand (MOD), multimedia systems, performance evaluation, scheduling, video-on-demand (VOD).

1 Introduction

Multimedia-on-demand (MOD) systems enable customers to access the multimedia contents they want at the times of their choosing. They also allow customers to apply VCR-like operations such as pause, resume, fast forward, and fast rewind. Therefore, motion picture studios and cable companies have started looking 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 server is highly constrained by the requirements of the real-time playback and the high transfer rates. A wide spectrum

[1]This research was supported in part by NSF grants CCR-9900701, CCR-0098149, CCR-0208734, and EIA-0202007.

of techniques, therefore, has been developed to enhance the performance of VOD servers, including *resource sharing* and *scheduling* [5, 11, 25, 1, 12, 13, 22], *admission control* [14, 26], *disk striping* [23, 4], *data replication* [9, 4], *disk head scheduling* [19, 15], *data block allocation* [18, 9], and *adaptive block rearrangement* [20].

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* [5, 7, 25, 1], *patching* [12, 21], *piggy-backing* [11], *broadcasting* [13, 17], and *interval caching* [6]. With batching, requests to same movies are accumulated over a time window and serviced together by utilizing the multicast facility. Batching, therefore, off-loads the storage subsystem and uses efficiently server bandwidth and network resources. Patching expands the multicast tree dynamically to include new requests, so it reduces the request waiting time but requires additional bandwidth and buffer space at the client. Piggy-backing services a request almost immediately but adjusts the playback rate so that the request catches up with a preceding stream, resulting in a lower-quality initial presentation. Broadcasting techniques divide each movie into multiple segments and broadcast each segment periodically. Thus, broadcasting requires relatively very high bandwidth and buffer space at the client. With interval caching, the server caches intervals between successive streams. This technique shortens the request waiting time without increasing the bandwidth or the space requirement 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 initial waiting times. 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 the broadcasting technique is used.) This paper focuses on VOD servers that employ batching as the primary resource sharing technique. As most prior studies, this paper assumes the availability of the multicast facility. Multicast 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 [10]. For

example, Sprint has already deployed native multicast across its Internet backbone [24].

Scheduling policies for VOD servers include *First Come First Serve* (FCFS) [5, 25], *Maximum Queue Length* (MQL) [5], *Maximum Factored Queue Length* (MFQL) [1], *Minimum Idling Maximum Loss* (IML) [22], and *Minimum Idling Maximum Queue Length* (IMQ) [22]. 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, while IMQ selects the longest queue. *Longest Wait First* (LWF) [27, 8] is another common scheduling policy, but it was not studied in the context of VOD servers. It selects the queue with the largest sum of request waiting times. To simplify the terminology, we refer to LWF as *FCFS-sum*.

This paper 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, [5] shows that MQL achieves better throughput than FCFS, whereas [1] and [22] indicate the opposite. These discrepancies are not due to any specific assumptions in these studies and are resolved in this paper. Moreover, there is a misconception with regard to the ability of FCFS to provide time of service guarantees. In [25], it is stated that FCFS can provide time of service guarantees, which are either precise or later the actual times of service. In contrast, we show 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 [27, 8, 2].

We also propose 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. 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 finite and typically a small number, 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.

We compare the performance of various policies through extensive simulation. We primarily study FCFS, MQL, MFQL, FCFS-sum, QFCFS, IMQ, IML, and IML$^+$. We analyze three objectives in simulation: 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, while the second signifies the quality of service (QoS) and comes next in importance. Unfairness measures the bias against unpopular movies. We also examine the impacts of customer waiting tolerance and server capacity (or server load) on the effectiveness of each policy. Moreover, we compare 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. Hence, contrary to [1], 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 paper is organized as follows. We discuss the main scheduling objectives and the common scheduling policies in Section 2. In Section 3, we analyze the existing scheduling policies and discuss the applicability of other scheduling policies that were not investigated in the context of VOD systems. Then, we address the issue of providing time of service guarantees in Section 4. We present QFCFS and IML$^+$ in Section 5. In Section 6, we discuss the simulation platform and the workload characteristics and present the main simulation results. Finally, we draw conclusions in the last section.

2 Scheduling Objectives and Policies

A VOD server maintains a waiting queue for every movie, routes incoming requests to their corresponding queues, selects an appropriate queue for service 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 is referred to as *server capacity*. We next discuss the primary scheduling objectives and the common scheduling policies.

2.1 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 as explained in Subsection 3.2.

2.2 Scheduling Policies

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

- *First Come First Serve* (FCFS) [5] - This policy selects the queue with the oldest request.
- *FCFS-n* [5] - With this policy, the server 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, the server uses the corresponding dedicated channel for the playback of one of the other movies.
- *Maximum Queue Length* (MQL) [5] - This policy selects the longest queue.
- *Maximum Factored Queue Length* (MFQL) [1] - 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) [25] - 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 [22] - 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: *BMQ* and *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 [22] - These schemes, which include *IML* and *IMQ*, pursue batching without minimum wait requirements and will be discussed in Subsection 5.2.

3 Preliminary Analysis

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 [25], so we will not consider it further in this paper. Similarly, we will not analyze FCFS-n because [25] 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 [22]. Hence, we will not consider maximum batching schemes further. Next, we discuss two variants of MQL and two variants of FCFS.

3.1 MQL Variants

There is a large discrepancy in the relative performance of MQL with respect to FCFS (and MFQL) in [5, 22, 1]. We have identified that the discrepancy is caused by different possible implementations of MQL. Note that the length of a queue is a finite and typically a small number. Thus, there is a good probability that multiple queues have the same length. The definition of MQL [5], however,

does not specify the selection criterion among the longest queues. We 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 6.2, MQL-u and MQL-f vary considerably in terms of the achieved throughput and average request waiting time. We were able to reproduce the results in [5, 22, 1] 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 [8, 27]. This policy is essentially the same as MQL-f, but it was not considered in previous studies of VOD servers.

3.2 FCFS Variants

A variation of FCFS, called *Longest Wait First* (LWF), was investigated in the context of videotex systems [8, 27] and web servers with data broadcasting [2]. 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 [5] 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 of any sort in VOD servers. To simplify the terminology, we refer to this policy as *FCFS-sum* in the subsequent analysis.

4 Providing Time of Service Guarantees

FCFS is believed to provide time of service guarantees. In [25], it is stated that FCFS can provide time of service guarantees, which are either precise or later than the actual times of service. We show, however, that FCFS may violate these guarantees. A server that provides time of service guarantees can be implemented in several ways. Thus, we base our argument on the concept rather than any specific implementation.

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 [7]. Early completions lead to servicing some requests earlier than their time of service guarantees.

The following example explains why with FCFS, the server may violate time of service guarantees. We assume in this example that $t1 < t2 < t3 < t4 < t5 < t6$ and that $i \neq j$. We 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. 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, 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 $R3$! We have also observed violations of time of service guarantees during simulations that use the same reneging behavior used in [25].

5 Proposed Policies

5.1 Quantized FCFS (QFCFS)

We note that 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. We also note that 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. We, therefore, propose a generalized policy called *Quantized FCFS* (QFCFS) that 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. The first variant chooses the longest queue, while the second 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. 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.

5.2 Enhanced IML (IML$^+$)

Minimum idling schemes were shown to be very effective in exploiting minimum waiting times [22]. Minimum waiting times can be estimated by the server either predictively or conservatively. With minimum idling schemes, the waiting queues are partitioned into two sets: a hot set ($\mathcal{H}$) and a cold set ($\mathcal{C}$). A waiting queue belongs to $\mathcal{H}$ if it corresponds to a popular movie, it has more than one request, or it has only one request and that request has been waiting longer than a pre-specified threshold, T. 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. Minimum idling 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. These 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.

IML and IMQ are two minimum idling schemes. They differ in the criterion used to select a queue in $\mathcal{H}$. IML selects the queue with the largest expected loss. The loss of a queue is defined as its number of 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.

We note that the expected loss of a queue is a finite and typically a small number. 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. In contrast, we propose a policy called *Enhanced IML* (IML^+) that exploits these situations by selecting the queue with the largest factored length among the set of all eligible queues[2]. By combining the benefits of IML and IMQ, IML$^+$ can 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.

6 Performance Evaluation

In this section, we analyze the performance of FCFS, FCFS-sum, MQL-u, MQL-f, MFQL, IML, IMQ, QFCFS, and IML$^+$ through extensive simulation. We have developed a simulator for VOD servers that apply various scheduling policies. We have validated the simulator by reproducing several graphs in previous studies. The server is initialized to 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. We have validated many of these results against those generated by simulating an initially unloaded server. The simulation stops when a steady state analysis with 95% confidence interval is guaranteed. We first start our discussion with the workload characteristic and then present the main results.

[2]We have found that in these situations, choosing the longest queue is not as effective as choosing the queue with the largest factored length.

6.1 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. Hence, the inter-arrival time is exponentially distributed with a mean $T = 1/\lambda$. We also assume, as in previous works, that the accesses to movies follow a Zipf-like distribution. 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 $\theta = 1$. In accordance with prior studies, we assume $\theta = 0.271$. We study a VOD server with 120 movies, each of which is 120-minute long, and we 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, we assume that batching is the primary resource sharing technique, and we do not consider any VCR-like operations. These operations can be supported by allocating contingency channels [7]. For MFQL, we assume that the server knows exactly the access frequency of each movie. This may lead to overestimating its performance. The performance of minimum idling schemes is not sensitive to the value of the threshold τ because they also classify movies as hot or cold dynamically. We fix τ at 20.

As in previous studies, we 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.

We 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 we refer to as *C+Exponential*, customers exhibit a minimum waiting time of 3 minutes, followed by an exponential time with $\mu = 3$ minutes. These distributions were used in [22].

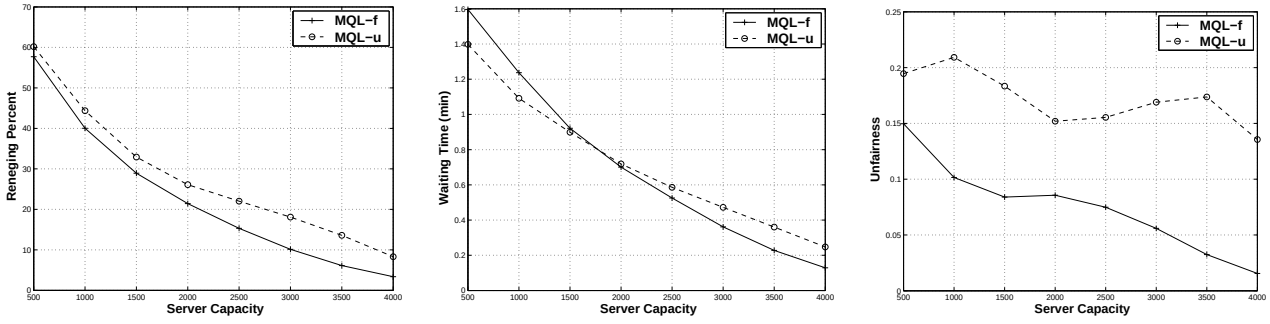
To study the effectiveness of providing time of service guarantees, we characterize the waiting tolerance by the following distribution. In this distribution, which we refer to as *NormalG*, customers who receive time of service guarantee will wait for service if their waiting times will be less than four 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 [25].

6.2 Result Presentation and Analysis

We now compare the various scheduling policies and then summarize the results.

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

Let us first analyze the performance of MQL-f and MQL-u. Figures 1 and 2 compare the performance of these two policies 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 shows that MQL-f is not only significantly fairer than MQL-u, but it also yields higher throughput. MQL-u, however, reduces waiting times better for large 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.

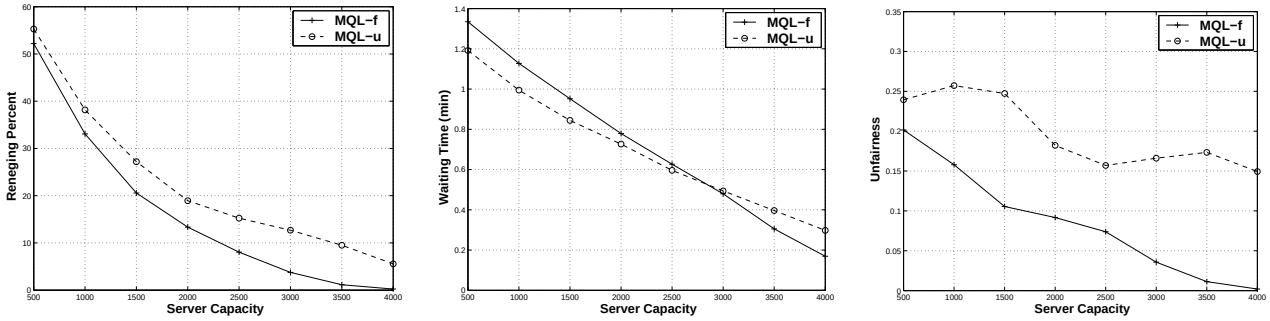


(a) In Reneging Percent (b) In Waiting Time (c) In Unfairness

Figure 1: Comparing Variants of MQL (Exponential Distribution)

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

Let us now analyze the performance of FCFS, FCFS-sum, MQL-f, and MFQL. Figures 3 and 4 compare the performance of these policies in terms of the three performance metrics when the waiting tolerance follows an exponential and normal distribution, respectively. The figures also show the performance of RAND, a policy that randomly selects a waiting queue for service, in order 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



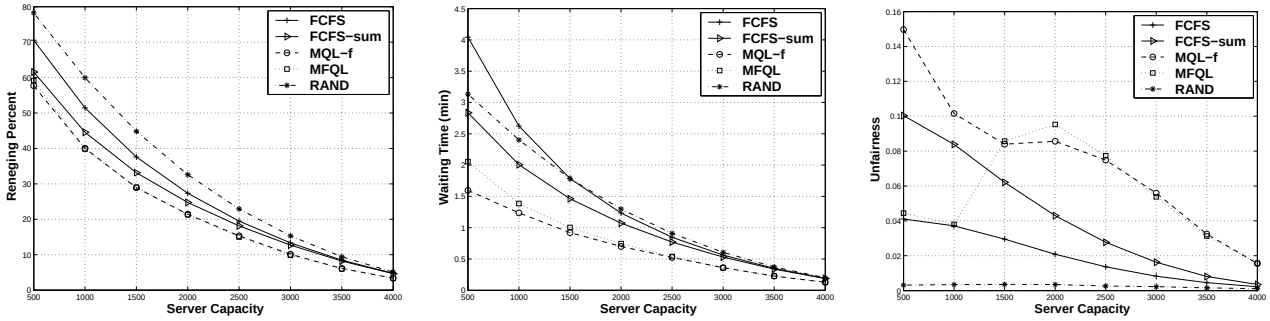
(a) In Reneging Percent (b) In Waiting Time (c) In Unfairness

Figure 2: Comparing Variants of MQL (Normal Distribution)

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 [1], MQL (MQL-f in particular) 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.

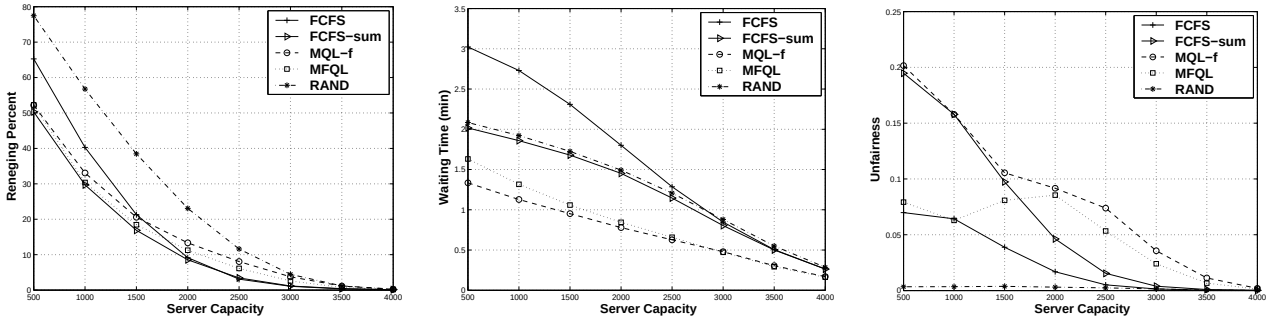
6.2.3 Providing Time of Service Guarantees

Let us now discuss the performance advantages of providing customers with time of service guarantees. We refer to FCFS that may provide time of service guarantees as FCFSg. Figure 5 shows how well FCFSg performs in comparison with FCFS, FCFS-sum, and MFQL when the waiting tolerance follows a NormalG distribution (discussed in Subsection 6.1). 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.



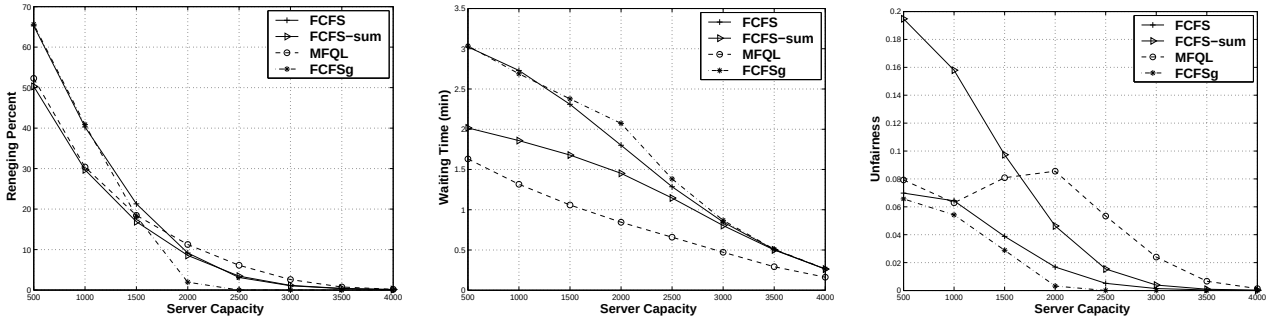
(a) In Reneging Percent (b) In Waiting Time (c) In Unfairness

Figure 3: Comparing FCFS, FCFS-sum, MQL-f, MFQL, and RAND (Exponential Distribution)



(a) In Reneging Percent (b) In Waiting Time (c) In Unfairness

Figure 4: Comparing FCFS, FCFS-sum, MQL-f, MFQL, and RAND (Normal Distribution)



(a) In Reneging Percent (b) In Waiting Time (c) In Unfairness

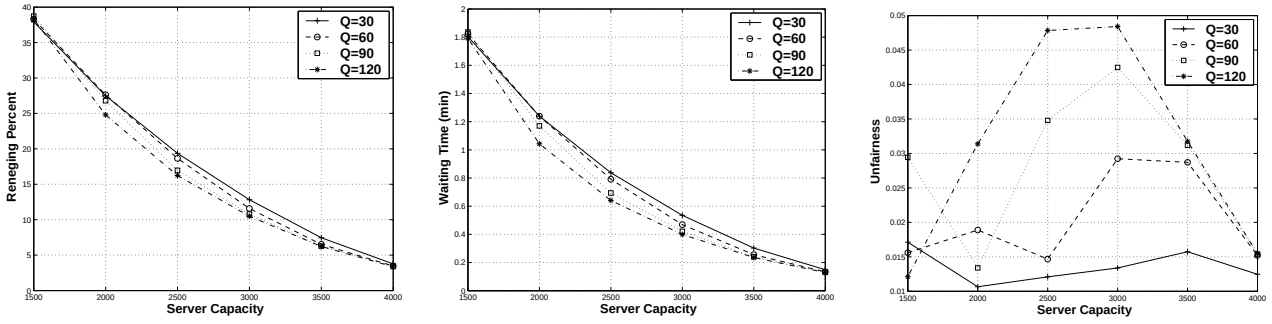
Figure 5: Effectiveness of FCFSg (NormalG Distribution)

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 NormalG 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.

6.2.4 Effectiveness of QFCFS

Let us now examine the performance of the proposed QFCFS policy. We have not observed any considerable difference among the different implementations of QFCFS in the overall performance. We thus limit the analysis to the simplest implementation, which conducts the quantization by truncation and uses MQL for the selection criterion among the queues with the highest quantized waiting time. We 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.

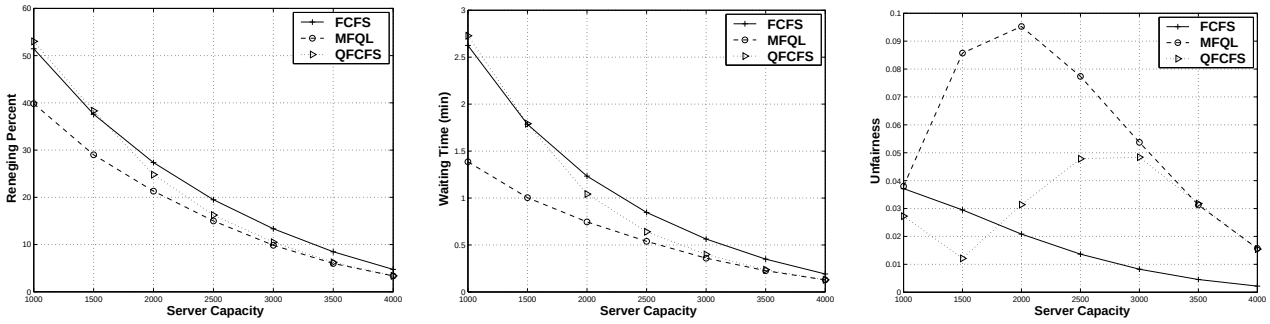
Figure 6 demonstrates the impact of the quantization interval (Q) on performance. As expected, both throughput and waiting times improve with larger values of Q, but unfairness increases.



(a) In Reneging Percent (b) In Waiting Time (c) In Unfairness

Figure 6: Effect of Quantization Level (Exponential Distribution)

Figure 7 compares the performance of 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 the 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. We 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.



(a) In Reneging Percent (b) In Waiting Time (c) In Unfairness

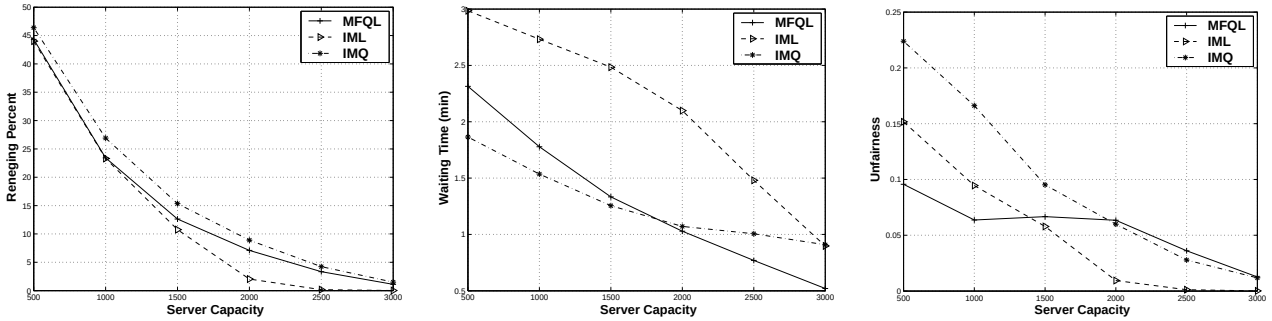
Figure 7: Effectiveness of QFCFS (Exponential Distribution, $Q = 120$ sec)

6.2.5 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 Subsection 6.1, we characterize the waiting tolerance by two distributions: normal and c+exponential. As in [22], we set the threshold T in the case of a c+exponential distribution 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. We define H as the ratio of the threshold T to the average request waiting time (which is mean of the normal distribution, μ).

Figure 8 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.) We do not show the results in the case of a normal distribution 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.

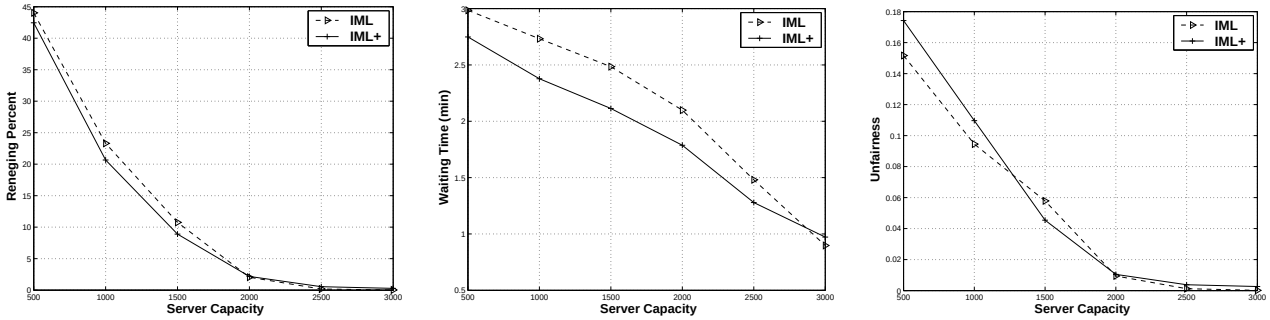


(a) In Reneging Percent (b) In Waiting Time (c) In Unfairness

Figure 8: Comparing IML, IMQ, and MFQL (C+Exponential Distribution)

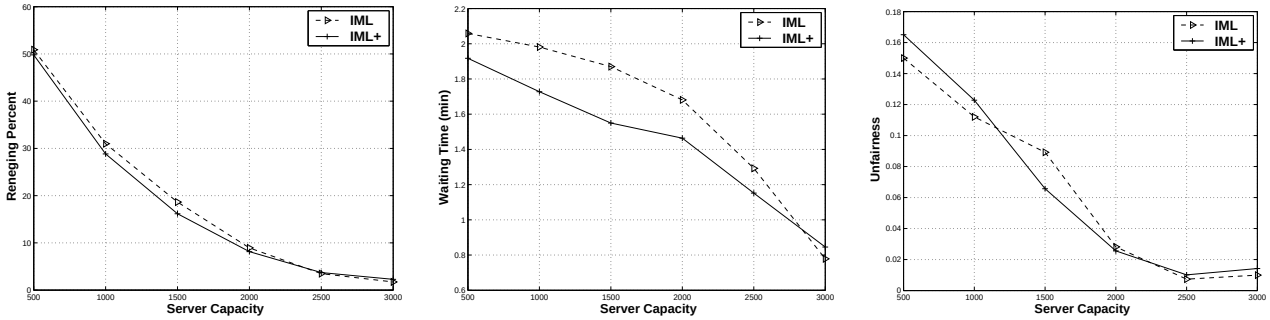
Figures 9 and 10 show the effectiveness of the proposed IML^+ policy in comparison with IML when the waiting tolerance follows c+exponential and normal distributions, respectively. The results show that IML$^+$ achieves up to 21% better throughput than IML when the reneging percent is greater than 5 and the waiting tolerance follows a c+exponential distribution, and it achieves 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 reneging percent is greater than 5 and the waiting tolerance follows a c+exponential distribution, and it reduces the average waiting time 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.

Figure 11 illustrates the impact of H (and thus the threshold T) 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 we 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.



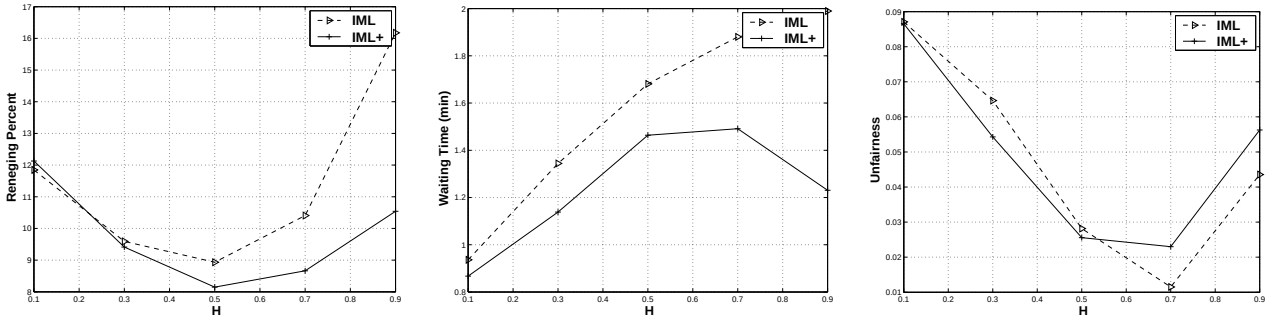
(a) In Reneging Percent (b) In Waiting Time (c) In Unfairness

Figure 9: Effectiveness of IML$^+$ (C+Exponential Distribution)



(a) In Reneging Percent (b) In Waiting Time (c) In Unfairness

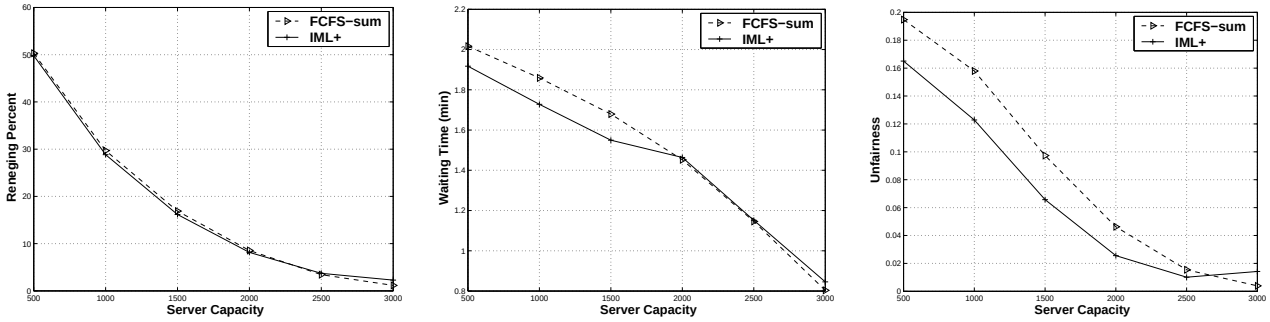
Figure 10: Effectiveness of IML$^+$ (Normal Distribution, $H = 0.5$)



(a) In Reneging Percent (b) In Waiting Time (c) In Unfairness

Figure 11: Effect of Threshold (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 12.



(a) In Reneging Percent (b) In Waiting Time (c) In Unfairness

Figure 12: Comparing FCFS-sum and IML$^+$ (Normal Distribution, $H = 0.5$)

6.2.6 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 1 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 [1], 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 2 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

Table 1: 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

to each other in terms of various metrics. They also have comparable implementation complexities. IML$^+$ is only a little fairer than FCFS-sum.

Table 2: 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 3 compares the best performers when the tolerance follows a c+exponential distribution. In this case, IML$^+$ is the clear winner (Figures 8 and 9).

Table 3: 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.

7 Conclusions

The investigation 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 paper.

We have conducted an in-depth investigation of scheduling policies, including FCFS, MQL, MFQL, IML, and IMQ. In contrast with [25], we have shown that FCFS may violate its time of service guarantees. We have also shown that the discrepancy in [5, 22, 1] with regard to the relative performance of MQL to FCFS is caused by alternative implementations of MQL. We 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, we have studied the effectiveness of *Longest Wait First* (LWF or FCFS-sum) [8, 27] in VOD servers.

We 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.

We have evaluated the effectiveness of various scheduling policies through extensive simulation. We 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, while the second comes next in importance. Moreover, we 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 [1], 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).

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, February 2001.

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

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

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

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

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

[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, November 1995.

[8] H. D. Dykeman, M. H. Ammar, and J. W. Wong. Scheduling Algorithms for Videotex Systems under Broadcast Delivery. *In Proc. of IEEE Int'l Conf. on Communication*, pages 1847-1851, June 1986.

[9] R. Flynn, and W. Tetzlaff. Disk Striping and Block Replication Algorithms for Video File Servers. *In Proc. of the Int'l Conf. on Multimedia Computing and Systems*, pages 590-597, June 1996.

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

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

[12] 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, September 1998.

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

[14] S. Jamin, S. Shenkar, L. Zhang, and D. D. Clark. An admission Control Algorithm for Predictive Real-Time Service. *In Proc. of the Int'l Workshop on Network and Operating System Support for Digital Audio and Video*, pages 349-356, November 1992.

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

[16] J.-F. Pâris, S. W. Carter, and D. D. E. Long. Efficient Broadcasting Protocols for Video on Demand. *In Proc. of the Int'l Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 127-132, July 1998.

[17] J.-F. Pâris. A Fixed-Delay Broadcasting Protocol for Video-on-Demand. *In Proc. of the Int'l Conf. on Computer Communications and Networks*, pages 418-423, October 2001.

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

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

[20] N. J. Sarhan and C. R. Das. Adaptive Block Rearrangement Algorithms for Video-On-Demand Servers. *In Proc. of Int'l Conf. on Parallel Processing*, pages 452-459, September 2001.

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

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

[23] 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.

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

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

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

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