A Simulation-Based Analysis of Scheduling Policies for Multimedia Servers *

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

Multimedia-on-demand *(MOD) has grown dramatically in popularity, especially in the domains of education, business, and entertainment. Therefore, the investigation of various alternatives to improve the performance of MOD servers has become a major research focus. The performance of these servers can be enhanced significantly by servicing multiple requests from a common set of resources. The exploited degrees of resource sharing depend greatly on how 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.*

1. Introduction

Recent advances in storage and communication technologies have spurred a strong interest in *multimedia-on-demand* (MOD) systems, which enable customers to watch what they want when they want it and allow them to apply VCR-like operations. *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. Thus, a wide spectrum of techniques has been developed to enhance the performance of VOD servers, including *resource sharing* and *scheduling* [3, 7, 8, 9], *admission control* [10], *disk striping* [15], *data replication* [6], *disk head scheduling* [11], and *data block allocation and rearrangement* [6, 12].

The performance of VOD servers can be significantly improved through resource sharing. The classes of resource sharing strategies include *batching* [3, 5], *patching* [8], *piggy-backing* [7], *broadcasting* [9], and *interval caching* [4]. Batching accumulates requests to same movies and services them 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 divides each movie into multiple segments and broadcasts 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 reduces the request waiting time but increases the overall cost of the server.

The exploited degrees of resource sharing depend greatly on how VOD servers schedule the waiting requests. Batching systems rely entirely on scheduling to boost up their performance. VOD systems that employ other resource sharing techniques also benefit from intelligent scheduling. This paper focuses on VOD servers that employ batching as the primary resource sharing technique.

Scheduling policies for VOD servers include *First Come First Serve* (FCFS) [3, 16], *Maximum Queue Length* (MQL) [3], *Maximum Factored Queue Length* (MFQL) [1], *Minimum Idling Maximum Loss* (IML) [14], and *Minimum Idling Maximum Queue Length* (IMQ) [14]. A VOD server

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

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, whereas MQL selects the longest queue. In contrast, MFQL selects a queue based on both queue lengths and movie access frequencies. Both IML an IMQ improve throughput by exploiting minimum request waiting times. IML selects from the set of eligible queues the queue that will otherwise incur the largest expected loss of requests, whereas IMQ selects the longest queue.

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. Furthermore, previous studies on VOD unfairly discarded a policy, called *Longest Wait First* (LWF) [17], without proper investigation although it was shown to perform very well in other contexts [17, 2]. To simplify the terminology, we refer to it as *FCFS-sum*.

We also propose two scheduling policies, called *Quantized FCFS* (QFCFS) and *Enhanced IML* (IML$^+$), which suite different patterns of customer waiting tolerance. QFCFS combines the benefits of FCFS and MQL/MFQL by scheduling requests based on both waiting times and queue lengths. 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 that meets the MFQL selection criterion.

We compare the performance of various policies through extensive simulation. We analyze three objectives in simulation: the overall customer reneging (defection or turnaway) probability, the average customer waiting time, and unfairness (against unpopular movies). We also 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. Moreover, we examine the impacts of customer waiting tolerance and server capacity (or server load).

The main results can be summarized as follows. (1) 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. (2) 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. (3) When the tolerance follows an exponential distribution, MFQL and MQL yield the highest throughput. (4) 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. (5) The distinct advantages of FCFS are simplicity, fairness, and ability to prevent starvations. (6) 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 policies in Section 2 and preliminarily analyze various policies in Section 3. We present QFCFS and IML$^+$ in Section 4. In Section 5, we discuss the simulation platform, workload characteristics, and the main results. Finally, we draw conclusions in the last section.

2. Scheduling Objectives and Policies

A VOD server maintains a waiting queue for every movie and selects an appropriate queue for service whenever it has an available *channel*. A channel is a set of resources needed to deliver a multimedia stream. The number of channels is referred to as *server capacity*.

2.1. Objectives

Scheduling policies are guided by one or more of the following primary objectives. (1) Minimize the overall customer reneging 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 translates to server throughput. 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. 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. 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 common scheduling policies.

- *First Come First Serve* (FCFS) [3] - It selects the queue with the oldest request.
- *FCFS-n* [3] - It 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) [3] - It selects the longest queue.
- *Maximum Factored Queue Length* (MFQL) [1] - It 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.
- *Group-Guaranteed Server Capacity* (GGSC) [16] - 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 [14] - They aggressively pursue batching by deliberately delaying requests.
- *Minimum Idling* Schemes [14] - These schemes, which include *IML* and *IMQ*, pursue batching without minimum wait requirements and will be discussed in Subsection 4.2.

3. Preliminary Analysis

FCFS is the fairest and the easiest to implement. In contrast with most other policies, FCFS is also believed to provide time of service guarantees. In [16], it is stated that FCFS can provide time of service guarantees, which are either precise or later than the actual times of service. We have shown in [13], however, that FCFS may violate these guarantees. 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 [16], so we will not consider it further in this paper. Similarly, we will not analyze FCFS-n because [16] 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 [14]. 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 [3, 14, 1]. For example, [3] shows that MQL achieves better throughput than FCFS, whereas [1] and [14] indicate the opposite. 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 [3], 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 $\geq$). As shown in Subsection 5.2, MQL-u and MQL-f vary considerably in terms of the achieved throughput, average request waiting time, and unfairness. We managed to reproduce the results in [3, 14, 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 [17]. This policy is essentially the same as MQL-f, but it was not considered for VOD servers.

3.2. FCFS Variants

A variation of FCFS, called *Longest Wait First* (LWF), was investigated in the context of videotex systems [17] 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 [3] 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. We refer to this policy as *FCFS-sum* in the subsequent analysis.

4. Proposed Policies

4.1. 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. 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.

4.2. Enhanced IML (IML$^+$)

Minimum idling schemes (IML and IMQ) were shown to be very effective in exploiting minimum waiting times [14]. 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. 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 differ in the criterion used to select a queue in $\mathcal{H}$. IML selects the queue with the largest expected loss, which is the number of requests that will exceed the minimum waiting time by the next scheduling 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 them. 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. 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.

5. Performance Evaluation

We have developed a simulator for VOD servers and have validated it 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 simulations. The results are collected using a steady state analysis with 95% confidence interval. We present here only a subset of the results because of space limitation. Additional results can be found in [13].

5.1. 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. 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. 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 [5]. For MFQL, we assume that the server knows exactly the access frequency of each movie. 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 of 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 [14].

5.2. Result Presentation and Analysis

5.2.1. Comparing FCFS and MQL Schemes

Let us now study the effectiveness of various FCFS and MQL schemes. Figure 1 compares the performance of MQL-f and MQL-u in terms of reneging percent, average request waiting time, and unfairness. We assume here that the waiting tolerance follows a normal distribution. The results in the case of an exponential distribution are not shown because they exhibit a very similar behavior. This figure 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.

Figures 2 and 3 compare the performance of FCFS, FCFS-sum, MQL-f, MFQL, and RAND in terms of the three metrics with exponential and Gaussian patterns of waiting tolerance of customers, respectively. We use RAND, which selects randomly a waiting queue for service, to demonstrate the effectiveness of the other policies. These 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 [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.

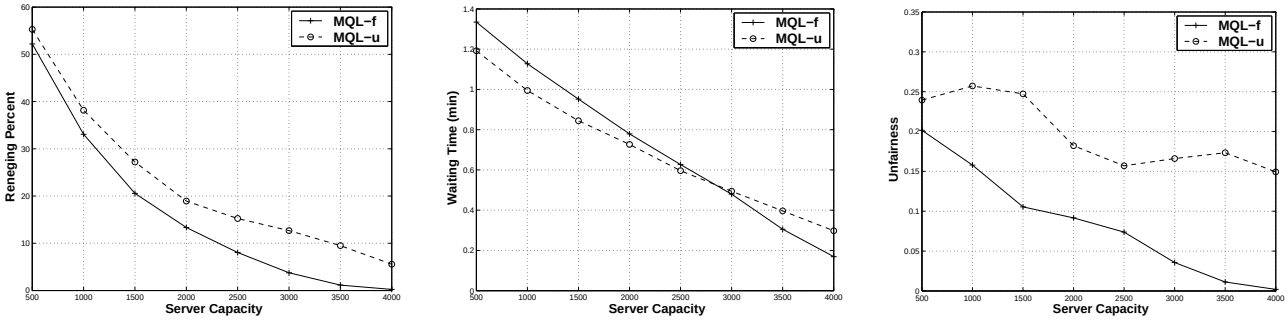
5.2.2. Providing Time of Service Guarantees

We now discuss the performance advantages of providing time of service guarantees. We refer to FCFS that may provide time of service guarantees as FCFSg. We have compared the performance of FCFSg with FCFS, FCFS-sum, and MFQL using similar patterns of waiting tolerance as those in [16]. The figures are not shown because of space limitations, but they can be found in [13]. Assuming that FCFSg is able to provide true time of service guarantees and is the only policy that can influence the waiting tolerance, it can achieve the best throughput and generally the best fairness. It, however, performs the worst in terms of the mean waiting time. Besides, it may violate its time of service guarantees. Furthermore, it is not the only policy that can motivate customers to wait. All other scheduling policies can also motivate customers to wait to various degrees by providing them with expected starting times of service.

5.2.3. Effectiveness of QFCFS

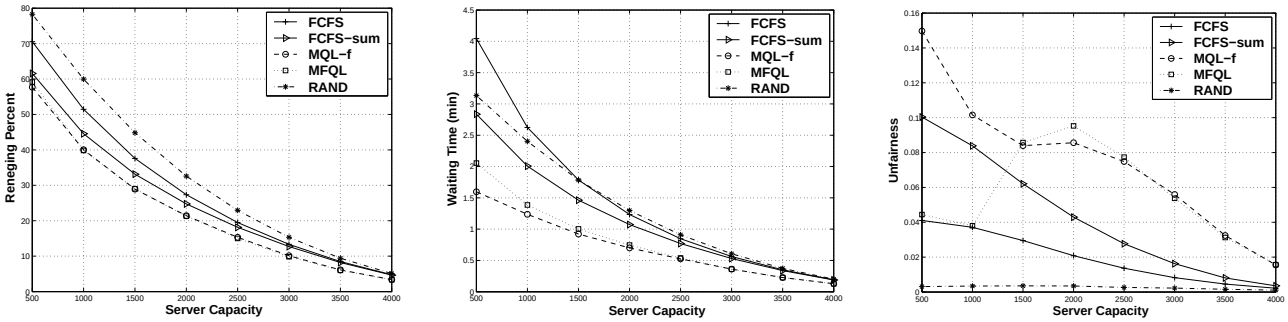
Let us now analyze the performance of the proposed QFCFS policy. We have not observed any considerable difference among the different implementations of QFCFS in the overall performance. We thus limit the analysis to the simplest implementation, which conducts the quantization by truncation and uses 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 4 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, FCFS 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 both throughput and waiting times) degrades. So, Q should be chosen as a tradeoff.



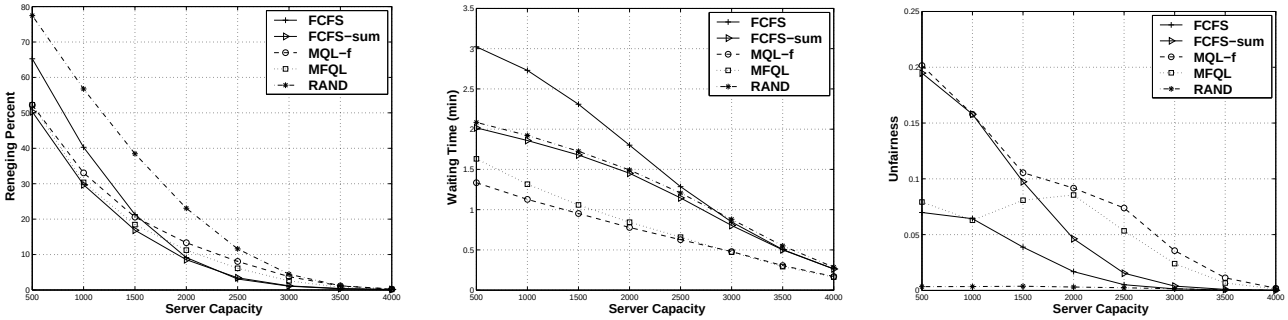
(a) In Reneging Percent (b) In Waiting Time (c) In Unfairness

Figure 1: Comparing Variants of MQL (Normal Distribution)



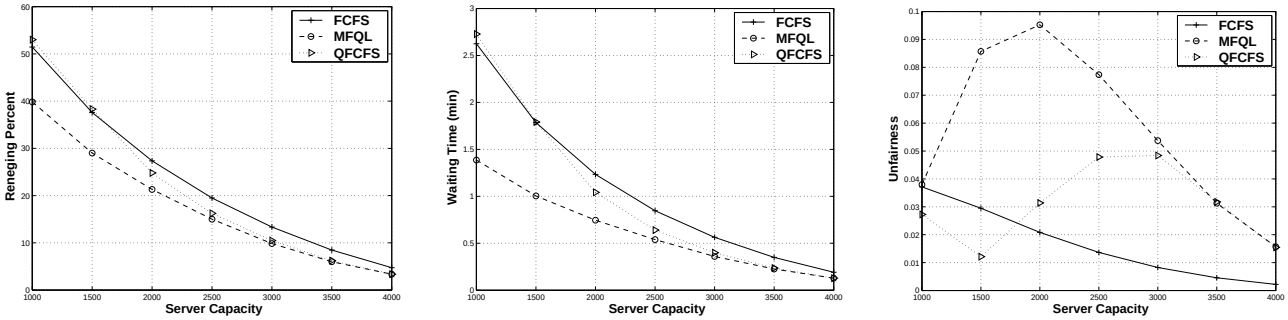
(a) In Reneging Percent (b) In Waiting Time (c) In Unfairness

Figure 2: Comparing FCFS, FCFS-sum, MQL-f, MFQL, and RAND (Exponential Distribution)



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

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



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

Figure 4: Effectiveness of QFCFS (Exponential Distribution, $Q = 120$ sec)

(Simulation results about the impact of Q on performance can be found in [13].) 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.

5.2.4. 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 in [14], we set the threshold T in the case of a c+exponential distribution to the minimum waiting time. In the case of a normal distribution, we define H as the ratio of the threshold T to the average request waiting time and use a value of 0.5 because it leads to the best overall performance [13]. The results with a normal distribution are not shown because they exhibit similar behaviors as those with a c+exponential distribution.

Figure 5 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.) 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.

Figure 6 shows the effectiveness of the proposed IML^+ policy in comparison with IML when the waiting tolerance follows a c+exponential distribution. Note that IML$^+$ achieves up to 21% better throughput and 18% less waiting times than IML. Each of IML$^+$ and IML is fairer than the other in certain regions of the curve.

Interestingly, IML$^+$ also performs almost identically to FCFS-sum in terms of throughput and waiting times when the tolerance follows a normal distribution (where there is no fixed minimum waiting time). FCFS-sum$^+$ is only a little more biased against cold movies. (The figures can be found in [13]). They also have comparable implementation complexities.

6. Conclusions

We have conducted an in-depth investigation of scheduling policies for MOD servers. We have shown that the discrepancy in [3, 14, 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) [17] 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. 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). 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.

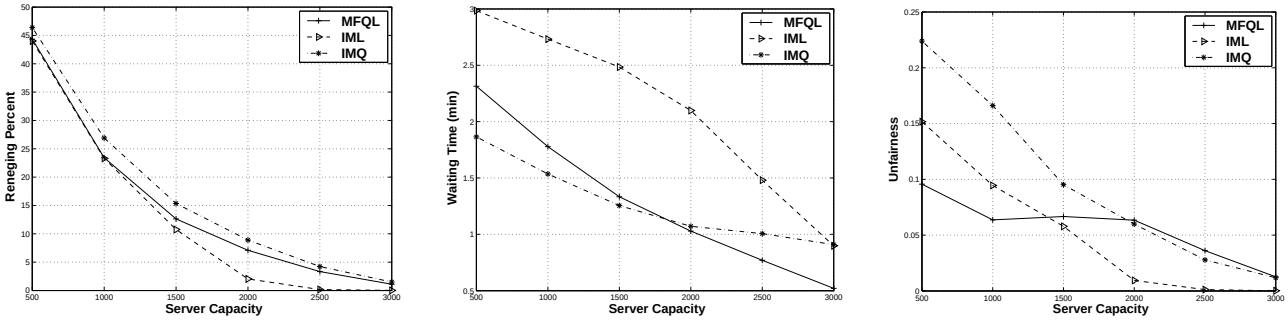
The main results can be summarized as follows. (1) MQL-f is not only fairer than MQL-u, but it also yields higher throughput, especially for high server capacities. (2) MQL-f performs nearly as well as MFQL in terms of throughput, waiting times, and unfairness, even when assuming that MFQL has perfect knowledge of movie access frequencies. MQL-f is also simpler. Thus, contrary to [1], MQL (MQL-f in particular) may be preferred over MFQL. (3) 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. (4) 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. (5) 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

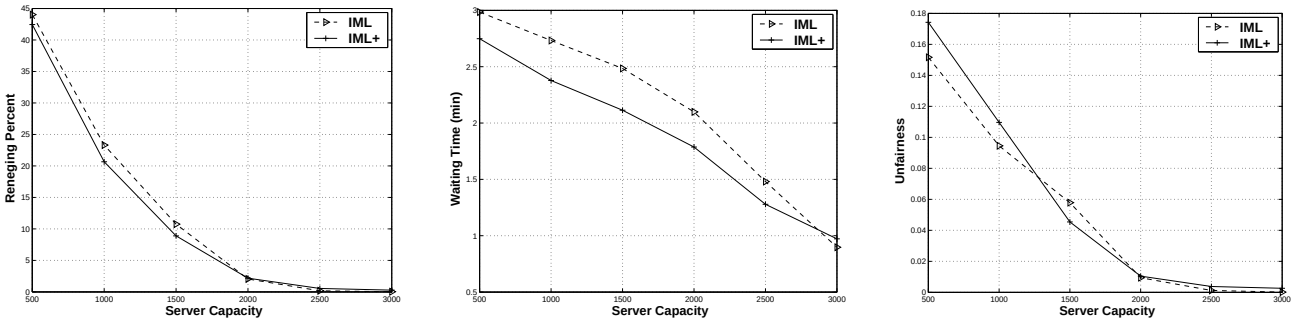
[1] C. C. Aggarwal, J. L. Wolf, and P. S. Yu. The Maximum Factor Queue Length Batching Scheme for Video-on-Demand Systems. *IEEE Trans. on Computers*, 50(2): 97-110, February 2001.

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



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

Figure 5: Comparing IML, IMQ, and MFQL (C+Exponential Distribution)



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

Figure 6: Effectiveness of IML$^+$ (C+Exponential Distribution)

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] N. J. Sarhan and C. R. Das. A Detailed Study of Request Scheduling in Multimedia Systems. Technical Report CSE-03-002, Computer Science and Engineering, The Pennsylvania State University, January 2003.

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

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

[16] 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.

[17] 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.