Provision of Time of Service Guarantees in Video-on-Demand Servers

Nabil J. Sarhan and Chita R Das
Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA, 16802
E-mail: {sarhan,das}@cse.psu.edu

Abstract

Recent advances in storage and communication technologies have spurred a strong interest in *Video-on-Demand* (VOD) services. Providing the customers of VOD servers with time of service guarantees offers two major advantages. First, it makes VOD services more attractive by improving customer-perceived quality of service (QoS). Second, it improves throughput through the enhanced resource sharing attained by motivating the customers to wait. In this paper, we propose a new class of scheduling policies, called *Next Schedule Time First* (NSTF), which provides customers with schedule times and performs scheduling based on these schedule times. NSTF guarantees that customers will be serviced no later than scheduled and ensures that the schedule times are very accurate estimates of the actual times of service. We present alternative implementations of NSTF and show through simulation that NSTF works as expected and delivers outstanding performance benefits.

0.1 Introduction

Multimedia information has become an integral and an essential part of the World Wide Web (WWW). Multimedia data differ significantly from textual and numeric data in two main ways. First, they require high storage capacity and high transfer rates. Second, they consist of media quanta, which convey meaningful information only when presented continuously in time. Multimedia networking applications can be classified into three main classes: *Video-on-Demand* (VOD), *Live Streaming*, and *Interactive Real-time* (such as Internet telephony and video conferencing).

The application of interest in this paper is VOD. By contrast with broadcast-based systems such as cable TV, VOD servers enable customers to watch the videos they want at the times of their choosing and allow them to apply VCR-like operations. Besides its use for entertainment, VOD has been of great importance in education and distant learning in particular.

The number of customers that can be serviced concurrently by a VOD server is highly constrained by the stringent 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* [5, 10, 25, 1, 11, 12, 19, 20], *admission control* [13, 26], *disk striping* [23, 3], *data replication* [8, 3, 4], *disk head scheduling* [17, 14], and *data block allocation and rearrangement* [16, 8, 18].

The performance of VOD servers can be significantly improved by servicing multiple requests from a common set of resources. The main classes of resource sharing strategies for VOD servers include *batching* [5, 7, 25, 1, 19], *patching* [11, 22], *piggy-backing* [10], *broadcasting* [12, 15, 2], and *interval caching* [6, 20, 21]. Batching off-loads the storage subsystem and uses efficiently server bandwidth and network resources by accumulating the requests to the same videos and servicing them together by utilizing the multicast facility. Patching expands the multicast tree dynamically to include new requests, thereby reducing the request waiting time and improving resource sharing, but it requires additional bandwidth and buffer space at the client. Piggy-backing offers similar advantages to patching, 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 video into multiple segments and broadcast each segment periodically on dedicated server channels. The improved resource sharing and the fixed waiting times for the playbacks of popular videos come at the expense of requiring relatively very high bandwidth and buffer space at the client. Interval caching caches intervals between successive streams in the main memory of the server. 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 exploited degrees of resource sharing depend greatly on how VOD servers schedule the waiting requests. Through intelligent scheduling, a server can support more concurrent customers, can reduce their waiting times for service, and/or can meet some other objectives. 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 videos are broadcasted when a broadcasting technique is used.) This paper focuses on VOD servers that employ batching as the primary resource sharing technique and assumes that the multicast facility is deployed. Multicast is already employed or can be easily employed in most enterprise and local area networks (LANs), and it has incrementally been deployed over the Internet. In fact, a ubiquitous wide-scale deployment of native (non-tunneled) multicast across the Internet is becoming a reality [9]. 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], *Maximum Queue Length* (MQL) [5], and *Maximum Factored Queue Length* (MFQL) [1]. To facilitate scheduling, a VOD server maintains a waiting queue for every video and services all the requests in a selected queue together using only one stream. FCFS selects the queue with the oldest request, whereas MQL selects the longest queue, and MFQL selects the queue with the largest *factored length*. The factored length of a queue is equal to its length divided by the square root of the access frequency of its corresponding video.

Providing time of service guarantees through scheduling can enhance customer-perceived QoS and can influence customers to wait, thereby increasing server throughput. Unlike most other policies, 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 longer than the actual times of service. In contrast, we show that

FCFS may violate these guarantees because not all customers continue to wait for services. We also show that FCFS is incapable of producing accurate time of service guarantees. Specifically, the average deviation of the actual times of service from the time of service guarantees ranges from 20 seconds to more than 4.5 minutes!

We propose a new class of scheduling policies, called *Next Schedule Time First* (NSTF), which eliminates the shortcomings of FCFS. NSTF provides customers with schedule times, and it guarantees that they will be serviced no later than and accurately at their schedule times. The basic idea of the NSTF is to assign schedule times to incoming requests and to perform scheduling based on these schedule times rather than the arrival times. In the absence of VCR-like operations, a VOD server knows exactly when resources will become available for servicing new requests because each running stream requires a fixed playback time. Hence, when a new request calls for the playback of a video with no waiting requests, NSTF assigns the request a new schedule time that is equal to the closest unassigned completion time of a running stream. If the new request, however, is for a video that has already at least one waiting request, then NSTF assigns it the same schedule time assigned to the other waiting request(s) because all requests for a video can be serviced together using only one stream. Applying VCR-like operations, which are typically supported by using contingency channels [7], leads to early completions and thus servicing some requests earlier than scheduled.

When all customers waiting for the playback of a video defect (i.e., cancel their requests), their schedule time become available and can be used by other customers. This leads to two variants of NSTF: *NSTFn* and *NSTFo*. NSTFn assigns freed schedule times to incoming requests, whereas NSTFo assigns them to existing requests with waiting time guarantees beyond a certain threshold, and thus likely to defect. We show that NSTFn and NSTFo are special cases of a policy, called *generalized NSTF* (GNSTF). We also present three variants of NSTFo: *NSTFo-FCFS*, *NSTFo-MQL*, and *NSTFo-MFQL*, which differ in the selection criterion of existing requests that will be assigned better schedule times. These variants select requests on a FCFS, a MQL, or a MFQL basis, respectively. NSTFo-MQL and NSTFo-MFQL combine the advantages of FCFS and MQL/MFQL.

We study the effectiveness of the proposed policies through extensive simulation. We consider five performance metrics: the overall customer reneging (defection or turn-away) percentage, the average request waiting time, the number of violations of time of service guarantees, the average deviation from the time of service guarantees, and unfairness. The reneging percentage is the most important metric because it translates to the number of customers serviced concurrently and to server throughput. Unfairness measures the bias against unpopular videos. All other performance metrics signify QoS. We also study the impacts of customer waiting tolerance and server capacity (or server load) on the results.

The main results can be summarized as follows. (1) The proposed NSTF policies meet their time of service guarantees and provide accurate schedule times. In particular, the average deviations of the actual times of service from the schedule times produced by NSTFn and NSTFo are within 6 seconds and 0.2 second, respectively. (2) NSTFo-MQL is the clear winner among the four variants of NSTF because of its superiority in both throughput and waiting times. (3) NSTFo-MQL achieves higher throughput and, in certain situations, shorter request waiting times than FCFS that may provide limited time of service guarantees. (4) NSTFo-MQL generally outperforms MQL and MFQL (both of which cannot provide time of service guarantees) in terms of throughput, especially for high server capacities, but MQL and MFQL achieve shorter waiting times. (5) NSTFo-MQL is fairer than MQL and MFQL for high server capacities because schedule times are assigned on a FCFS basis.

The rest of the paper is organized as follows. In Section 0.2, we discuss the main scheduling objectives and policies and explain why FCFS may violate its time of service guarantees. We then present NSTF and its variants in Section 0.3. In Section 0.4, we discuss the simulation platform, the workload characteristics, and the main simulation results. Finally, we draw conclusions in the last section.

0.2 Scheduling Objectives and Policies

We discuss here the major scheduling objectives and policies and explain why FCFS may violate its time of service guarantees.

0.2.1 Objectives

A VOD server maintains a waiting queue for every video, routes incoming requests to their corresponding queues, and applies a scheduling policy to select an appropriate queue whenever it has an available *channel*. A channel is a set of resources (network bandwidth, I/O bandwidth, etc.) needed to deliver a multimedia stream. All requests in the selected queue can be serviced together using only one channel. The number of channels is referred to as *server capacity*.

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

1. Minimize the overall customer reneging probability.
2. Minimize the average request waiting time.
3. Provide time of service guarantees.
4. Minimize unfairness.
5. Eliminate starvation.
6. Minimize the implementation complexity.

The reneging probability is the probability that a new customer leaves the server without being serviced because of a waiting time exceeding the user's tolerance. It is the most important metric because it translates to the number of customers that can be serviced concurrently and 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, third, and fifth objectives are indicators of customer-perceived quality of service (QoS). By providing time of service guarantees, a VOD server can also encourage customers to wait, thereby increasing server throughput. It is also usually desirable that VOD servers treat equally the requests for all videos. Unfairness measures the bias of a policy against cold (i.e., unpopular) videos and can be obtained by the following equation: $unfairness = \sqrt{\sum_{i=1}^{N_v}(r_i - \bar{r})^2/(N_v - 1)}$, where r_i is the reneging probability for the waiting queue i, $\bar{r}$ is the mean reneging probability across all waiting queues, and N_v is the number of waiting queues (and number of videos as well). Finally, minimizing the implementation complexity is a secondary issue in VOD servers because the CPU and memory are not performance bottlenecks.

0.2.2 Existing Policies

The following is a description of the common 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 videos on dedicated channels and schedules the requests for the other videos on a FCFS basis. When no request is waiting for the playback of any one of the n most common videos, the server uses the corresponding dedicated channel for the playback of one of the other videos.
- *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 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 video. 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.

FCFS is the fairest and the easiest to implement. MQL and MFQL reduce the average request waiting time but tend to be biased against cold videos, which have relatively few waiting requests. Unlike MQL, MFQL requires periodic computations of access frequencies. FCFS can prevent starvation, 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 in certain situations than FCFS. A detailed investigation of scheduling policies can be found in [19].

0.2.3 Time of Service Guarantees Through FCFS

In this subsection, we show that FCFS may violate its time of service guarantees. We have also observed violations of time of service guarantees during simulations that use the same model of waiting tolerance used in [25]. Let us discuss first how a server may provide time of service guarantees and let us assume, just for now, 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. Let us now discuss the impact of VCR-like operations. Applying a pause, a fast forward, or a fast rewind 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. 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. Thus, 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 $t4$, $R1$ defects (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 actual time of service for the request(s) granted this guarantee), the server will violate the time of service guarantee of $R3$!

0.3 Next Schedule Time First (NSTF)

We propose a new class of scheduling policies, called *Next Schedule Time First* (NSTF), which eliminates the shortcomings of FCFS. In particular, NSTF assigns schedule times to incoming requests, and it guarantees that they will be serviced no later than scheduled. In addition, it ensures that these schedule times are very close to the actual times of service. NSTF, therefore, improves both QoS and server throughput. Improving throughput is attained by influencing the waiting tolerance of customers. In the absence of any time of service guarantees, customers are more likely to defect because of the uncertainty of when they will start to receive services. Another desirable feature of NSTF is the ability to prevent starvation (as FCFS).

NSTF selects for service the queue with the closest schedule time. The schedule times are assigned as follows. When a new request for a video with no waiting requests arrives, NSTF assigns that request a new schedule time. This schedule time is equal to the closest un-assigned completion time of a running stream. By contrast, when a new request joins a waiting queue that has at least one request, NSTF assigns the new request the same schedule time assigned to the other request(s) because all requests in a queue can be serviced together. Note that a schedule time estimates the time when the server starts to deliver a stream and not the time when the presentation actually starts. Thus, it does not include the network latency or the buffering time at the client for smoothing out the delay jitter.

As discussed in the previous section, VCR-like operations can lead only to servicing requests earlier than scheduled and thus will not result in any violations of time of service guarantees.

Next, we discuss variants of NSTF and their implementations. We also show for the sake of comparison an implementation of FCFS that provides customers with time of service guarantees (which may be violated as discussed in Subsection 2.3). We refer to FCFS that may provide such guarantees as *FCFSg*, and to FCFS that provides no guarantees simply as *FCFS*.

0.3.1 Variants of NSTF

When all waiting requests for a video are cancelled, their schedule time becomes available and can be used by other requests. This leads to two variants of NSTF: *NSTFn* and *NSTFo*. NSTFn assigns the freed schedule times to incoming requests, whereas NSTFo assigns them to existing requests with waiting time guarantees beyond a certain threshold, and thus likely to defect without being assigned better schedule times. Hence, requests that are assigned schedule times that require them to wait beyond a certain threshold should be notified that they may be serviced earlier. This notification may influence them to wait. All other requests, however, should be given hard time of service guarantees. NSTFn is simpler than NSTFo, but it is more biased against old requests.

Let us now discuss how NSTFo works. NSTFo assigns each freed schedule time to an appropriate waiting queue that meets the following three conditions.

1. It is nonempty.
2. Its assigned schedule time is worse than the freed schedule time.
3. Based on its assigned schedule time, the expected waiting time for each request in it is beyond a certain threshold. This threshold is set to 5 minutes in this paper because of the studied waiting tolerance models (Subsection 4.2).

If no candidate is found, NSTFo grants the freed schedule time to a new request. In contrast, if more than one queue meet these conditions, it selects the most appropriate one. We thus present three variants of NSTFo: *NSTFo-FCFS*, *NSTFo-MQL*, and *NSTFo-MFQL*. These variants differ in the selection criterion of existing requests that will be assigned better schedule times. NSTFo-FCFS selects the queue with the longest waiting time, whereas *NSTFo-MQL* selects the longest queue, and *NSTFo-MFQL* selects the queue with the largest factored length. NSTFo-MQL and NSTFo-MFQL combine the benefits of FCFS and MQL/MFQL by assigning schedule times on a FCFS basis and re-assigning freed schedule times on a MQL/MFQL basis.

NSTFn and NSTFo can be generalized in a single policy that we refer to as *Generalized Next Schedule Time First* (GNSTF). Let us reconsider the second condition for a queue that is to be assigned a better schedule time, which is that the current schedule time of that queue is worse than the freed schedule time. If the difference between the two schedule times, however, is too small, the freed schedule time may effectively get wasted for not being granted to a well-deserving queue. In GNSTF, we change the condition to that the schedule time of a queue is greater than the freed schedule time by a certain threshold, T. Note that GNSTF becomes NSTFo if T is zero and becomes NSTFn if T is sufficiently large.

0.3.2 Implementations of FCFSg and NSTF

We now present the implementations of FCFSg and NSTF in C++-like syntax. For simplicity, we assume that the number of channels is greater than or equal to the number of videos, which is typically the case. A VOD server maintains a waiting queue (WQ) for each video and one running queue (RQ) for all videos. All new requests are routed to the corresponding WQs. When a server channel becomes available, all requests in a selected WQ can be inserted as one aggregate request in RQ because all of them can be serviced using only one stream. 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. We refer to the current system time as $currTime$. $WQi.guarantee$ 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.

Figure 1 shows an implementation of FCFSg. 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.

```
Event: Initialization
    RQIndex = -1;
Event: Arrival of request R to waiting queue 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 with the time of service guarantee WQi.guarantee;
    }
Event: Cancellation of request R from waiting queue 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++;
    }
```

Figure 1: An Implementation of FCFSg

Figure 2 shows an implementation of NSTF (with all its variants). The server here needs to maintain a free pool ($freePool$) of freed schedule times. This pool can be implemented using a priority queue, where schedule times are placed in an ascending order. When a request for a video with no waiting requests arrives, the server first tries to assign it a schedule time from the top of $freePool$. $WQi.schedule$ denotes the schedule time assigned to the waiting queue i. If $freePool$ does not contain any live schedule times (i.e. schedule times that are further than the current time), then the server assigns it a new schedule time that corresponds to the next un-assigned completion time.

0.4 Performance Evaluation

We analyze the effectiveness of the proposed policies through simulation. We start our discussion with the simulation environment and workload characteristic, and then we present the main results.

0.4.1 Simulation Platform

We have developed a simulator for a VOD server that supports various scheduling policies. The simulated server starts with a state close to the steady state of the common case to accelerate the simulation. In that state, the server delivers its full capacity of running streams, whose remaining times for completion are uniformly distributed between zero and the normal video length. We have validated many of these results against those generated by simulating an initially unloaded server. The simulation stops after a steady state analysis with 95% confidence interval is guaranteed.

0.4.2 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 videos follow a Zipf-like distribution. With this distribution, the probability of choosing the n^{th} most popular of M videos is $C/n^{1-\theta}$ with a parameter θ and

```
Event: Initialization
  RQIndex = -1;
Event: Arrival of request R to waiting queue 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()){
          while (!freePool.empty()){ // remove the expired schedule times from the free pool
             if (freePool.top() < currTime){
                RQIndex++;
                freePool.pop();
             } // end of if
          } // end of while
          if (!freePool.empty()){ // assign the closest schedule time in the free pool to WQi
             WQi.schedule = freePool.top();
             freePool.pop();
          } // end of if
          else{ // assign the next un-assigned schedule time to WQi
             WQi.schedule = RQ[RQIndex].compTime;
             RQIndex--;
          }//end of else
       } // end of if
       Inform R with the schedule time WQi.schedule;
    } // end of else
Event: Cancellation of request R from waiting queue WQi
    if (R is the only request in WQi){
       if (policy == NSTFn)
          freePool.add(WQi.schedule); // add the freed schedule time to the free pool
       else if (there is queue WQj that meets the 3 conditions and the selection criterion of FCFS, MQL, or MFQL){
          FreePool.add(WQj.schedule); // add the freed schedule time to the free pool
          WQj.schedule = WQi.schedule;
       } // end of else if
       else
          FreePool.add(WQi.schedule); // add the freed schedule time to the free pool
    } // end of if

Event: Completion of a running stream
    if (at least one request is waiting){
       Select for service the queue with the closest schedule time;
       RQIndex++;
    }
```

Figure 2: An Implementation of NSTF

a normalized constant C. The parameter θ controls the skewness of video 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 that $\theta = 0.271$.

We characterize the waiting tolerance of customers by two models. In *Model A*, customers who receive time of service guarantees will wait for service if their waiting times will be less than or equal to five minutes; the waiting times of all other customers follow an exponential distribution with a mean of 5 minutes. *Model B* is used in [25] and is the same as Model A except that a truncated normal distribution with a mean of 5 minutes and a standard deviation of 1.67 minutes is used in place of the exponential distribution.

Truncation excludes the waiting times that are negative or greater than 12 minutes. In both these models, we assume that the customers who are expected (according to their time of service guarantees) to wait longer than 12 minutes will defect immediately. Although they differ significantly, both normal and exponential distributions were used in previous studies.

We study a VOD server with 120 videos, each of which is 120-minute long. We examine the server at different loads by fixing the request arrival rate at 40 requests per minute and varying the number of channels (server capacity) generally from 500 to 1750. VCR-like operations can be supported using contingency channels [7]. Thus, the relative performance of various scheduling policies in terms of reneging probability, waiting times, and unfairness does not depend on these operations as long as the fraction of server channels used for these operations is kept the same (which is typically the case). Therefore, we will not consider VCR-like operations in this simulation study in favor of keeping the analysis focused. The results in terms reneging probability, waiting times, and unfairness can be generalized by assuming that the number of server channels excludes the contingency channels. The accuracy of schedule times, however, is likely to change because VCR-like operations lead to early completions and thus to servicing requests earlier than scheduled. The change increases with the frequency of these operations.

0.4.3 Result Presentation and Analysis

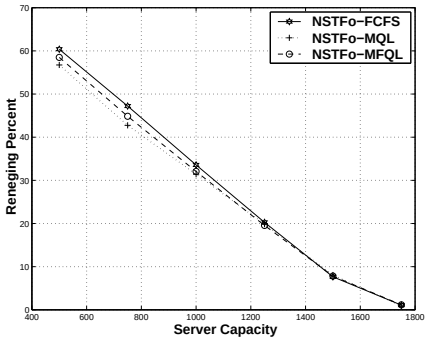
Let us now compare the performance of various NSTF policies with each other, with FCFSg, and with policies that do not provide time of service guarantees: FCFS, MQL, and MFQL. We consider five performance metrics: the number of violations of time of service guarantees, the average deviation from these guarantees, the overall customer reneging percent, the average request waiting time, and unfairness.

Table 1 compares FCFSg, NSTFn, and the three variants of NSTFo in terms of the number of violations of time of service guarantees and the average deviation of these guarantees from the actual times of service. (The schedule times given by NSTF act as time of service guarantees.) The results are shown for the two models of customer waiting tolerance. The numbers of violations are collected per 1000 requests, and the arrows show the variations as the server capacity increases from 500 to 1500. For example, with FCFSg under Model A, an average of 11.6 out of 1000 time of service guarantees are violated when the server capacity is 500, compared with 0.03 violations when the capacity is 1500. The number of violations decreases with the server capacity as expected. The results demonstrate that FCFSg may violate its time of service guarantees, but these violations happen very occasionally (especially for high server capacities) because FCFSg tends to overestimate significantly these guarantees. In fact, overestimating these guarantees is the most critical problem of FCFSg. With FCFSg, the actual times of service differ from the time of service guarantees by 20 seconds to more than 4.5 minutes on the average! The inaccuracy of time service guarantees leads to uncertainty of when customers will start to receive services. Customers would greatly appreciate receiving accurate schedule times so that they could plan accordingly. Moreover, customers are more likely to defect if their waiting times are overestimated. In contrast with FCFSg, NSTF produces accurate schedule times and services requests no later than scheduled. The average deviations of the actual times of service from the schedule times given by NSTFn and NSTFo are within 6 seconds and 0.2 second, respectively.

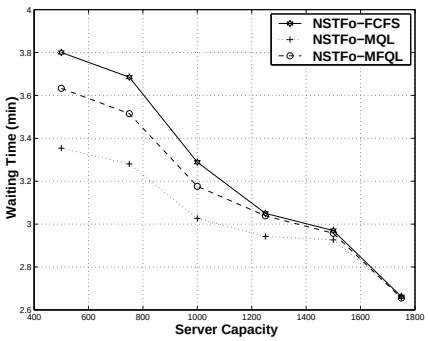
Table 1: Violations of Time of Service Guarantees and Average Deviations from these Guarantees

	Model A		Model B	
Policy	**Violations per 1000 Requests**	**Deviation (sec)**	**Violations per 1000 Requests**	**Deviation (sec)**
FCFSg	$11.6 \rightarrow 0.03$	$226 \rightarrow 19.6$	$4.6 \rightarrow 0.09$	$272.9 \rightarrow 26.2$
NSTFn	0	$3.8 \rightarrow 0.29$	0	$6.03 \rightarrow 1.01$
NSTFo-FCFS	0	$0.12 \rightarrow 0.05$	0	$0.14 \rightarrow 0.16$
NSTFo-MQL	0	$0 \rightarrow 0.04$	0	$0 \rightarrow 0.07$
NSTFo-MFQL	0	$0 \rightarrow 0.05$	0	$0 \rightarrow 0.05$

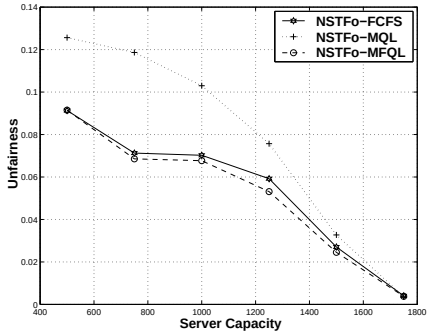
We now discuss the performance of the three variants of NSTFo in terms of throughput, waiting times, and unfairness, and then we use only the best performer among them in the subsequent analysis. Figures 3 and 4 compare the performance of these variants under Model A and Model B of the waiting tolerance, respectively. The results indicate that NSTFo-MQL performs the best in terms of both throughput and waiting times, followed by NSTFo-MFQL. Despite that NSTFo-MQL is not the fairest, it stands out as the clear winner because fairness is far less important than throughput and waiting times.



(a) In Reneging Percent



(b) In Waiting Time



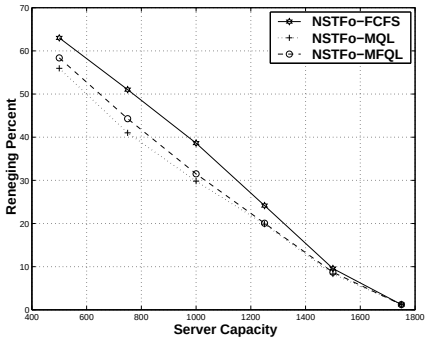
(c) In Unfairness

Figure 3: Comparison among NSTFo Variants (Model A)

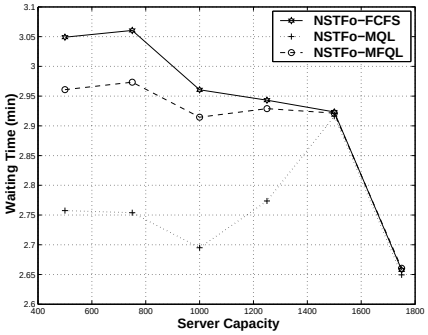
Figures 5 and 6 compare the performance of FCFSg, NSTFn, and NSTFo-MQL. The results demonstrate that NSTFo-MQL achieves better throughput and waiting times than NSTFn under both tolerance models, but NSTFn is fairer. Among the three policies, NSTFo-MQL delivers the highest throughput under both tolerance models. We expect the NSTF policies to perform even better in real systems because customers are more likely to defect with FCFSg, which tends to overestimate the waiting times. Unfortunately, the increased likelihood of defection with FCFSg is not captured very accurately by the tolerance models. In both models, the waiting tolerance of customers does not depend on the assigned time of service guarantees if the waiting times (according to these guarantees) may be greater than 5 minutes. This suggests that we are not entirely fair to the proposed policies. NSTFo-MQL also generally achieves the shortest waiting times when the tolerance follows Model B. In the case of Model A, however, FCFSg generally achieves the shortest waiting times, and NSTFo-MQL performs a little worse. Under both tolerance models, FCFSg is the fairest.

Figures 7 and 8 compare the performance of NSTFo-MQL with policies that do not provide time of service guarantees: FCFS, MQL, and MFQL. Note that NSTFo-MQL leads to the highest throughput under Model A. It also achieves the highest throughput under Model B but only when the reneging percent is less than 20, which is the most likely operating region as much larger reneging percents would be unacceptable. As expected, MQL and MFQL perform the best in terms of waiting times, and FCFS is the fairest.

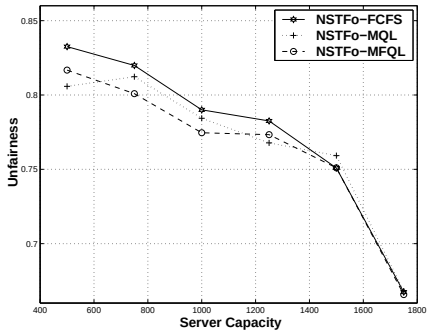
Next, we study the impact of the threshold T in GNSTF on performance. We consider here the implementation of GNSTF that uses MQL for re-assigning schedule times because of its performance benefits. Figure 9 shows the effect of T on throughput and waiting times when the tolerance follows Model A. The results for Model B are not shown because they exhibit a very similar behavior. The results for unfairness are not shown either because T has a little impact on it. Note that GNSTF becomes NSTFo when T approaches



(a) In Reneging Percent



(b) In Waiting Time



(c) In Unfairness

Figure 4: Comparison among NSTFo Variants (Model B)

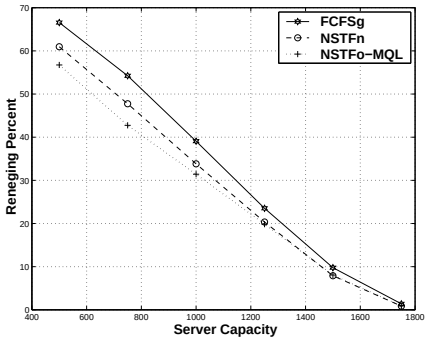
zero and becomes NSTFn when T is greater than a sufficiently large value. The results demonstrate that the performance improves slightly as T increases up to a certain value, then it starts to degrade more significantly. The impact of T is more dramatic for lower server capacities (not shown in the figure). Generally, a value of about 2 minutes achieves the best overall performance.

0.5 Conclusions

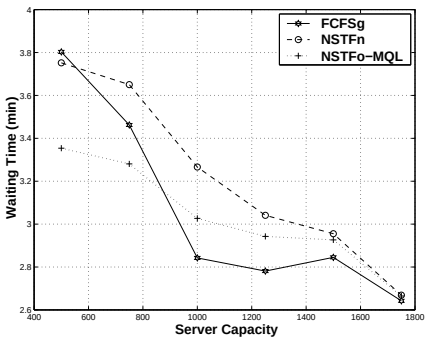
Video-on-Demand (VOD) services have grown dramatically in popularity. Therefore, the investigation of various alternatives to improve the performance and the QoS of VOD servers has become a major research focus. Scheduling is one such avenue and is the theme of this paper.

In this paper, we have proposed a new class of scheduling policies for VOD servers, called *Next Schedule Time First* (NSTF), which provides customers with hard time of service guarantees and with very accurate schedule times. We have presented two variants of NSTF: *NSTFn* and *NSTFo*. NSTFn assigns freed schedule times to incoming requests, whereas NSTFo assigns them to existing requests. We have shown that NSTFn and NSTFo are special cases of a policy, called *generalized NSTF* (GNSTF). By appropriately adjusting a threshold value, T, GNSTF can become either NSTFn or NSTFo. We have presented three variants of NSTFo: *NSTFo-FCFS*, *NSTFo-MQL*, and *NSTFo-MFQL*, which differ in the selection criterion of existing requests that will be assigned better schedule times.

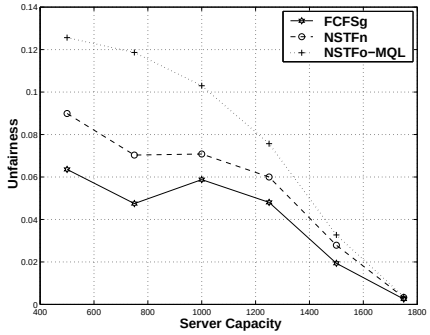
We have demonstrated the effectiveness of NSTF through simulation. We have considered five performance metrics: the number of violations of time of service guarantees, the average deviation of the actual times of service from the time of service guarantees, the overall customer reneging percentage, the average



(a) In Reneging Percent



(b) In Waiting Time



(c) In Unfairness

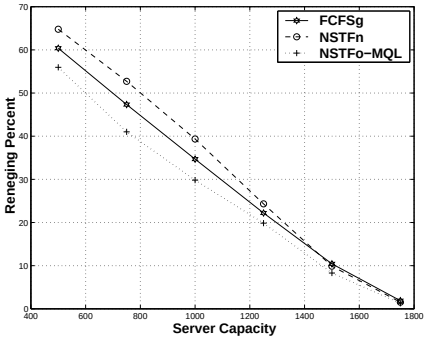
Figure 5: Comparison among FCFSg, NSTFn, and NSTFo-MQL (Model A)

request waiting time, and unfairness. We have studied the impacts of customer waiting tolerance and server capacity (or server load) on the results. We have also studied the effect of the threshold T on performance.

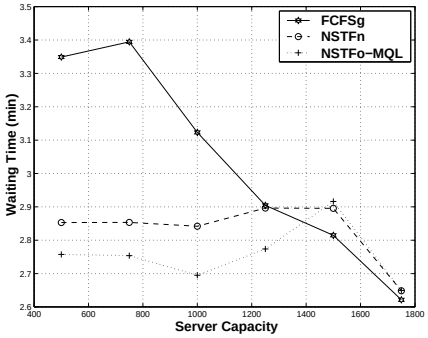
The main simulation results can be summarized as follows.

- NSTF always meets the time of service guarantees and produces very accurate schedule times. The average deviations of the actual times of service from the schedule times are within 0.2 second (when any implementation of NSTFo is used) and 6 seconds (when NSTFn is used). In contrast, FCFS may violate its time of service guarantees, and these guarantees differ from the actual times of service by 20 seconds to more than 4.5 minutes on the average!
- NSTFo-MQL is the clear winner among the variants of NSTF when all performance metrics are considered.
- NSTFo-MQL achieves higher throughput and, in certain situations, shorter waiting times than FCFS that may provide limited time of service guarantees.
- By motivating customers to wait, NSTFo-MQL outperforms MQL and MFQL (both of which cannot provide time of service guarantees) in terms of throughput for one of the models of waiting tolerance. For the other model, NSTFo-MQL also achieves the highest throughput but only within the most likely operating region of the server. NSTFo-MQL is also fairer than MQL and MFQL for high server capacities because schedule times are assigned on a FCFS basis. As expected, NSTFo-MQL leads to longer waiting times than MQL and MFQL.

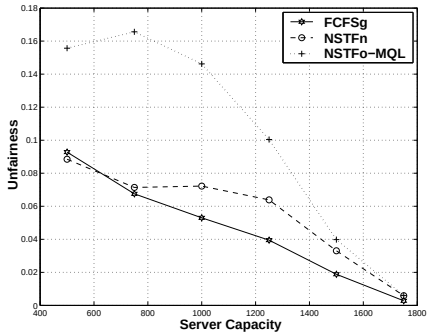
NSTF, therefore, not only can provide hard time of service guarantees and very accurate schedule times, but also can deliver outstanding performance benefits.



(a) In Reneging Percent

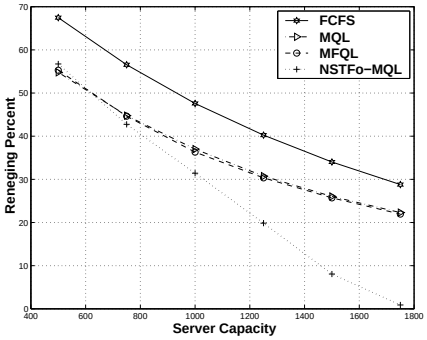


(b) In Waiting Time

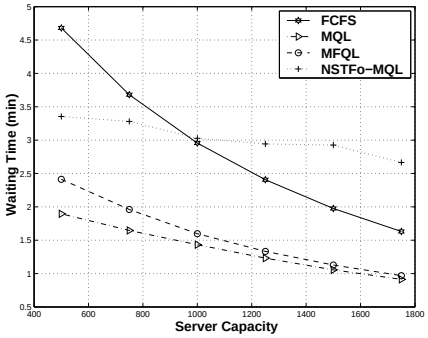


(c) In Unfairness

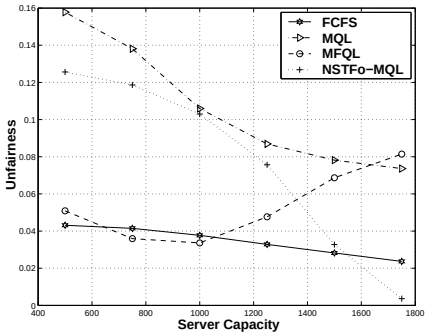
Figure 6: Comparison among FCFSg, NSTFn, and NSTFo-MQL (Model B)



(a) In Reneging Percent

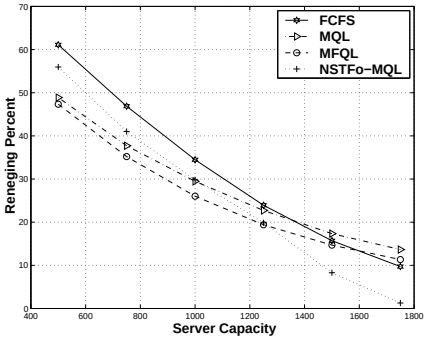


(b) In Waiting Time

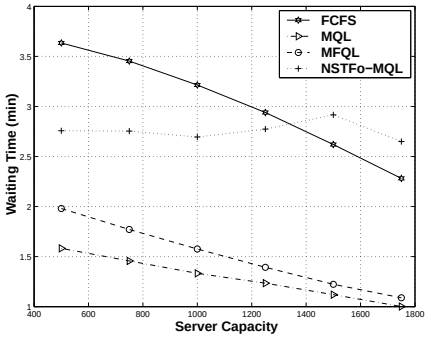


(c) In Unfairness

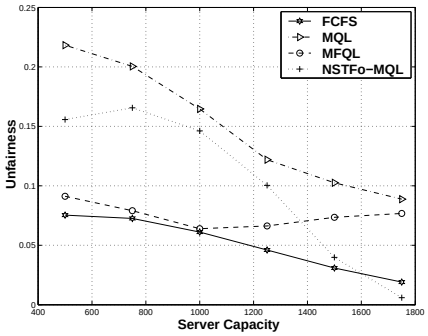
Figure 7: Comparison of NSTF0-MQL with FCFS, MQL, and MFQL (Model A)



(a) In Reneging Percent

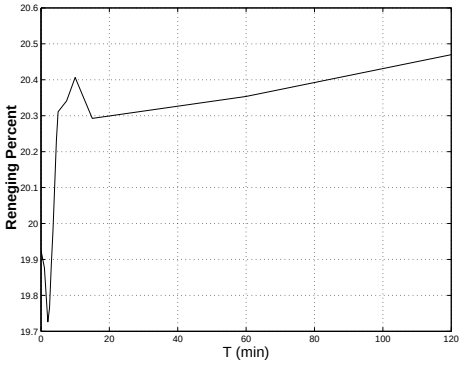


(b) In Waiting Time

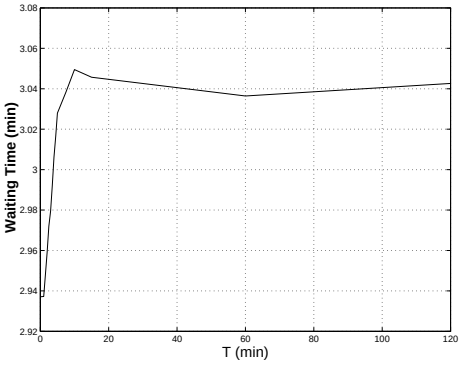


(c) In Unfairness

Figure 8: Comparison of NSTF0-MQL with FCFS, MQL, and MFQL (Model B)



(a) On Reneging Percent



(b) On Waiting Time

Figure 9: Effect of the Threshold T (Model A, 1250 Channels)

Bibliography

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

[3] 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.

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

[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, and R. Tewari. Buffering and Caching in Large-Scale Video servers. *In Digest of Papers. IEEE Int'l Computer Conf.*, pages 217-225, March 1995.

[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] 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.

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

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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 3-7, 2001.

[19] N. J. Sarhan and C. R. Das. A Simulation-Based Analysis of Scheduling Policies for Multimedia Servers. In *Proc. of the 36th Annual Simulation Symp.*, pages 183-190, March 30 - April 2, 2003.

[20] N. J. Sarhan and C. R. Das. An Integrated Resource Sharing Policy for Multimedia Storage Servers Based on Network-Attached Disks. In *Proc. of the 23rd Int'l Conf. on Distributed Computing Systems* (ICDCS), pages 136-143, May 19 - 22, 2003.

[21] N. J. Sarhan and C. R. Das. Caching and Scheduling in NAD-Based Multimedia Servers. To Appear in *IEEE Transactions on Parallel and Distributed Systems*, 2004.

[22] 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.

[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.