An Evaluation of VBR Disk Admission Algorithms for Continuous Media File Servers

Dwight Makaroff, Gerald Neufeld, and Norman Hutchinson
{makaroff,neufeld,hutchinson}@cs.ubc.ca

Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1Z4
Canada

Abstract

In this paper, we address the problem of choosing a disk admission algorithm for continuous media streams where each stream may have a different bit rate, and more importantly, where the bit rate within a single stream may vary considerably. We evaluate several different Variable Bit Rate (VBR) disk admission control algorithms for continuous media. An algorithm which accepts too few streams under-utilizes the server resources, while an algorithm which accepts too many streams over-utilizes the resources resulting in inadequate service (i.e. missing or delayed data) to the clients. The evaluation process is based on a representative set of video streams encoded in MJPEG. We conclude that one particular algorithm, the VBR simulation algorithm, performs the best among realizable algorithms in terms of system utilization and delivery guarantees and performs close to an optimal algorithm.

Keywords: multimedia, file servers, variable bit rate, admission control

1 INTRODUCTION

The motivation for the design of a specialized file server for continuous media such as video and audio is well established ([1, 5, 15]). Data Access patterns, as well as the services provided to clients by a continuous media file server (CMFS), differ considerably from a conventional distributed file service such as NFS. A continuous media client typically transfers large volumes of sequential data at specific moments in time. As well, the resource requirements of the network and server itself differ considerably. In order to guarantee continuity, the allocation of network bandwidth must be guaranteed, as well as processor cycles, RAM and disk bandwidth. The guarantees provided by servers and networks can be either deterministic or statistical in nature.

Characterizations of the bandwidth of the disk and the resource requirements of the client are necessary in order to provide such guarantees at the server. Most of the prior research in this area has made simplifying assumptions about these requirements [1, 7, 15]. Even in systems that support different media encodings with differing transfer rates, much analysis still assumes that each individual stream has a constant bit rate (CBR) [12, 18]. Compression methods such as MJPEG or MPEG-2 can produce Variable Bit Rate (VBR) streams. Since computers and networks are well-suited to handle bursty traffic, it seems reasonable to design a file service which can explicitly accommodate such variation in resource requirements and thereby increase the number of simultaneous streams supportable [2, 6, 7].

In any continuous media file server, there exists a disk admission control algorithm which determines whether a client's request for a new stream can be supported by the disk. The requirements of the new stream are calculated and then added to the allocated resources for the existing clients. If the required resources are not available, the new client's request is rejected. If the new stream is accepted, then the server provides some guarantee that the available disk resources are sufficient for all the allocated streams. Algorithms for allocating other scarce resources, such as network bandwidth are beyond the scope of this paper and therefore, we will refer to disk admission control algorithms simply as admission control algorithms.

An admission control algorithm can be too conservative and admit too few streams, thereby under-utilizing the server resources, or it can admit too many streams resulting in over-utilization, which manifests itself as delay or loss of data at the client. Although probabilistic methods exist to amortize the cost of this failure [2, 17], this is undesirable in general. This paper evaluates five different VBR disk admission algorithms to determine how well they behave for a typical set of streams. We will show that the *vbrSim* algorithm can efficiently make correct admission decisions and that its performance approaches that of an optimal algorithm.

The remainder of this paper is organized as follows. The next section contains a description of the five different algorithms as well as the model for representing the disk I/O resources. This is followed by the presentation of a "typical" stream set and a description of how the performance tests are run. The results of these tests are presented in Section 4. Related work is then described, followed by conclusions and future work.

2 DISK ADMISSION ALGORITHMS

To define bandwidth measurements, most servers divide time into intervals called slots or rounds, during which sufficient blocks of data are read off the disk and/or transmitted across the network for each active stream to allow continuous playback at the client application. A reasonable length for such a slot is 500 milliseconds, providing for fine level of granularity while attempting to limit the amount of overhead required for the operation of the server. A slot time of several seconds tremendously increases the amount of buffering needed at the server for high bandwidth streams. Smaller slot times increase the relative amount of time the disk spends seeking, since each read operation corresponds to a shorter playback duration.

To understand the relevant differences between the disk admissions algorithms, we need to define what resources are measured. In this study, we measure two resources: the disk read bandwidth, and the number of buffers available. The bandwidth is defined as the number of fixed size (64KByte) blocks the server can read from the disk system into user space in a fixed amount of time (one slot). The guaranteed number of blocks that can be read in a slot is called *minRead*. This number is calculated by running a calibration program that determines the maximum number of blocks that can be read when the blocks are located on the disk under the worst conditions. This value most accurately reflects the actual capacity of the server since it includes all transfer delays (through SCSI bus and I/O bus to memory) as well as server software overhead. Of course, *minRead* will likely be less than what is actually experienced by the server when reading some set of streams. As well, disk layout techniques [3, 4, 6, 15, 16] can be used to help improve the server's performance. Nonetheless, *minRead* is useful in calibrating the lower bound of disk bandwidth performance that the server will ever experience. This lower bound is used to provide hard guarantees of data delivery to the clients. The number of buffers available to the algorithms is determined by the amount of main memory at the server.

Whenever a client makes a request for a portion of a media stream at a particular display rate, a block schedule for the stream is created that contains one entry per slot for the duration of the stream playout. Each entry in the schedule is the number of blocks that must be read for the stream in that slot for continuous client playout. The rate at which this data is needed is called the *playout rate*. For a constant bit rate stream, all the values in the block schedule would be the same (modulo disk block granularity), reflecting the constant rate. The values would vary for VBR streams in a manner dependent on the encoding. For instance, Figure 1 presents an extract of the block schedule from one of our sample streams.

Slot	1	2	3	4	5	6	7	8	9	10	11	12
Blocks	2	3	6	6	6	7	6	6	7	7	6	8

Figure 1: Typical Stream Block Schedule

The block schedule can be characterized by its average, standard deviation and coefficient of variance. The last of these measures can be used for comparing the variation between streams with significantly different average playout rates.

A user can request to have only a portion of a continuous media stream delivered. In fact, we give details in [13] regarding the scope of the flexible user interface. Parameters are provided which enable fast (slow) motion delivery (in forward or reverse) which increases (decreases) the disk bandwidth required. As well, the skipping of user-defined "sequences" may be requested, which will appear to provide fast motion at the same average disk utilization and display rate. All this implies that the bit-rate profile can vary depending on the mode in which the stream is retrieved. The block schedule is created at the time playback is requested and is efficient to calculate. Preliminary results on the performance of the admission algorithm itself are given in Section 4.1.

As blocks are read from the disk, they are stored into available buffers which are then passed to the network for transmission to the clients. The speed at which buffers are filled is dependent on how fast the server reads blocks from the disk (we know it will be at least as fast as *minRead*). The speed at which the buffers are freed depends on how quickly the network can transmit the data to the client. This latter speed is itself dependent on the speed of the network and the number of buffers that the client has allocated to receive the data. In the following algorithms, we assume that the network management system transmits data only as fast as the client can consume the data; that is, at the playout rate.

We now describe the five admission algorithms. In all of these algorithms we use *minRead* as the basis for accepting or rejecting streams. For 4 of the 5 algorithms, this results in a deterministic guarantee that the server will not fall behind in reading.

2.1 Simple Maximum

In this algorithm, we reduce the characterization of each stream to a single number - the maximum number of reads required in any slot. If the sum of this maximum value for the new stream plus the value for the current set of streams is greater than *minRead*, we reject the new stream. Using the block schedule in Figure 1, for example, we would choose 8 as the value for the stream. If the current sum was 20 and *minRead* equaled 26, the new sum of 28 would result in a rejection.

A clear advantage of this algorithm is its simplicity. If the variation in the stream's block schedule is small, then this is a reasonable algorithm. In fact, it has been used in several CBR file systems [7, 15, 16]. Another advantage of this

algorithm is that it produces deterministic guarantees for reading from the disk. Unfortunately, it significantly under-utilizes the resources as block schedule variation increases, rejecting streams which could be delivered, as we will see in the next section.

2.2 Instantaneous Maximum

The next admission control algorithm keeps the sum of all of the currently admitted block schedules in a vector called the *server block schedule*. When a new stream is to be admitted, we add its block schedule to the current *server block schedule*. If any slot in the resulting schedule is greater than *minRead*, the new stream is rejected, otherwise it is accepted. A variation of this algorithm is described in Chang and Zakhor [4]. Consider the server block schedule as shown in Figure 2.

Slot	1	2	3	4	5	6	7	8	9	10	n-1	n
Blocks	15	10	24	23	12	16	18	24	22	25	20	23

Figure 2: Multiple Stream Block Schedule

We again assume that $minRead = 26$ and that we are attempting to admit the stream in Figure 1. The stream would be rejected since the number of blocks that need to be read in slot 3 of the *server block schedule* is 24+6=30.

This algorithm is also deterministic, and is provably better than Simple Maximum since it performs a more fine-grain evaluation of the schedules. It is still rather conservative and also may reject streams it could deliver. This can be corrected by the next algorithm.

2.3 Average

One problem with both algorithms 1 and 2 is that they do not take into account the amount of read ahead possible when we are reading slots that require fewer than *minRead* blocks. If we permit the server to read into buffers as fast as it can, then we can smooth out the peaks in a VBR schedule. In order to make use of this read ahead we can calculate the average blocks per slot for each stream. We can then sum this average (rather than the individual slots) and compare the summed averages to *minRead*. This algorithm will admit more VBR streams than the previous algorithms, because the average bit rate is less than the maximum of each stream and also less than the possible bandwidth peaks in combinations of streams. Unfortunately, it would appear that, intuitively at least, this algorithm does not provide deterministic guarantees that it will not over-allocate the server resources. Interestingly, as will be shown later, there are circumstances in which it may also under-utilize the server resources.

2.4 VBR Simulation (vbrSim)

Our next algorithm solves the problem of being too conservative in admissions, but still gives deterministic guarantees to the clients. This algorithm builds on the Instantaneous Maximum algorithm by making better use of the server block schedule. The algorithm utilizes the fact that many blocks needed in the future for existing streams will have already been read into buffers at the server whenever disk bandwidth is greater than the playout rate. It also assumes the server will read *minRead* blocks in each future slot, thereby smoothing bandwidth peaks in the future by simulating the disk reading ahead of the schedule requirements whenever possible. This rate of reading is only possible when there are buffers in which to store the data which is read early.

As we mentioned before, the server is capable of reading more than *minRead* blocks per slot. While we cannot guarantee this for the future, we can take advantage of read ahead that has already been accomplished in the past. If the server can read blocks faster than it can transmit them to the clients, we can read ahead an arbitrary number of slots.* Blocks that cannot be sent to the clients immediately are buffered. Unfortunately, there is not an infinite supply of buffers in the server, and we have to stop reading ahead once there are no buffers. For purposes of buffer consumption we again make the conservative assumption that the server reads *minRead* blocks per slot. As buffers are transmitted on the network, they are freed and this information is factored into the admission algorithm.

Data is transferred to the client and the corresponding buffers are freed at a rate which depends on the amount of buffer space available at the client and the negotiated bit rate of the network connection. We therefore maintain another schedule called the buffer allocation schedule. This schedule is initially the same as the server schedule. As data is transferred to the client and buffers are freed, however, the values in the buffer allocation schedule are decremented. Note that there is at least a one slot delay in recovering buffers. In other words, buffers containing data to be sent to a client in slot i are not reclaimed until the start of slot $i+1$. For streams which use buffers at a very slow rate, the buffers cannot be reclaimed until the last portion of data is sent across the network, which may be several slots later.

For the server block schedule there is a *current slot* index, identifying the next slot to read. As well, there is a *should be* index, referring to the current time. This index is incremented by one on each clock interval. The difference between *current slot* and *should be* is the number of slots read ahead.

Another aspect of the algorithm is that if the server reads ahead arbitrarily far it may use up all the available buffers, resulting in rejection due to insufficient buffers. In order to avoid a rejection in this situation, buffers are freed by reclaiming them from previous reads. We free those buffers needed furthest in the future, and then add them back into the schedule in their original position, allowing them to be absorbed properly via the vbrSim smoothing. A detailed description of this algorithm including pseudo-code is given in [13].

2.5 Optimal Algorithm

In order to calibrate our algorithms, we compare them to an optimal algorithm that is allowed complete knowledge of the future in making its admission decisions. The algorithm can predict the bandwidth that will be achieved for every slot

*Assuming the server is not 100% utilized.

time in the future and thus can be thought of performing the same readahead simulation as the vbrSim algorithm, but using a different value for *minRead* in each slot, namely the number of blocks actually read. This will always be greater than *minRead* whenever there is sufficient buffer capacity for the set of reads. The set of streams accepted by the optimal algorithm is called the *valid* set.

Of course, it is not possible to realize this optimal algorithm; it is included for comparison purposes only.

3 EXPERIMENTAL DESIGN AND TEST DATA

To evaluate the performance of these five algorithms, we choose a representative set of streams, and then compare the admission decisions made by each algorithm. The following describes the experimental design and the test data used.

3.1 Stream Selection

For experimental data, streams with a reasonable degree of variability were chosen to show the benefits achieved by explicitly considering the VBR nature of the data. The streams were captured at 30 fps at a resolution of 640x480 and compressed at a variable bit rate using MJPEG. We chose several clips of sports highlights which had alternating scenes of interviews and sports action as well as selected minor scenes from motion pictures. Each stream was between 5 and 7 minutes in length, which is sufficient to exercise the limits of all of the algorithms. For admission performance tests, we generated 250 stream scenarios that utilized between 30 and 100 percent of potential disk bandwidth. Client requests were staggered in time so as to not arrive at the server simultaneously in every case. The length of this stagger was chosen to be either 0, 5, or 10 seconds.

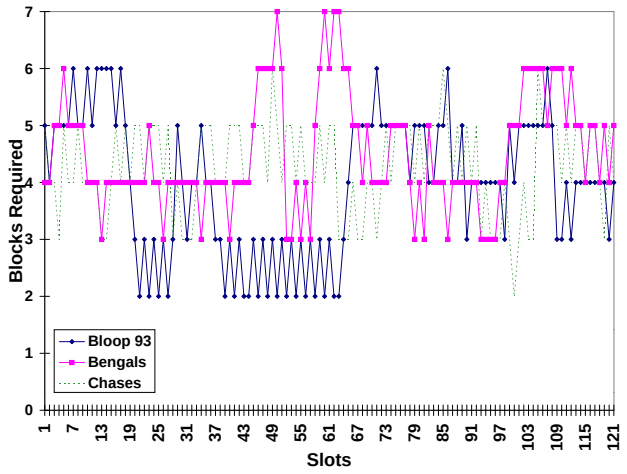

Figure 3: Variable Bit Rate Profiles

The VBR profiles of the beginning portions of three of the streams are shown in Figure 3. The vertical axis measures the number of 64 KByte disk blocks that are required to be read per slot for each stream. It can be clearly shown that most of the streams have significant bandwidth peaks well beyond their average bit-rate. Characterizations which summarize the variable bit rate data requirements of all the streams used in this study are given in Table 1. The units of measurement for Min, Max, Ave, and Std. Dev. are in blocks per slot. With slots of 500 msec, and block sizes of 64 KBytes, 1 block/slot is equivalent to 1 Mbps.

Stream	Frames	Disk Blks	Min	Max	Ave	Std Dev	Coef of Var
bloop 93	10810	3256	2	9	4.52	1.29	.287
bengals	10498	3234	2	7	4.62	0.98	.212
chases	9924	2746	1	9	4.14	1.03	.249
rescue	9299	2100	2	6	3.38	0.71	.209
intro	10560	3342	2	9	4.75	1.35	.285
maproom	10843	3065	1	9	4.24	1.46	.340
coaches	11023	3640	2	8	4.95	1.29	.287
boxing	10775	3721	2	11	5.17	1.30	.250
aretha	12535	3102	2	7	3.71	.68	.184

Table 1: Stream Summary Statistics

3.2 Server Configuration

To run the experiments on the CMFS developed at the University of British Columbia [11], specific hardware configurations were chosen. The server was an IBM RS/6000 Model 250 (66 MHz Power PC 601 at 62.6 SpecInt), running AIX 3.2.5, and had a 2 GByte disk attached via a SCSI-II Fast/Wide adapter. The server and client machines were connected to a local ATM network, using a NewBridge Mainstreet 31650 ATM Switch, with 100 Mbps Taxi network interface cards. Results for stream scenario acceptance decisions are given based on this hardware configuration, due to the availability of network bandwidth to transmit the streams and Asynchronous I/O facilities in the AIX operating system to read blocks off the disk device.

The server has been implemented on several hardware platforms, including Pentium processor based PCs running FreeBSD, connected via a 10 Mbps ethernet network. To examine the execution performance of the admission control algorithm itself, we chose to use this configuration because of its superior CPU speed. This environment did not have the bandwidth to send the continuous media data when the experiments were initially performed.

3.3 Disk Calibration

Three separate methods of calibrating the disk were used. Initially, a program was written that performed disk reads on an otherwise empty system, completely independent of the CMFS. This program simply requested disk blocks in various granularities from evenly spaced locations on the disk and measured the bandwidth achieved, using the raw disk I/O facilities on the AIX operating system, and performing separate seek and read system calls for each read request. In every case, at least 40 blocks per second could be successfully

read off the disk. For 500 msec slots, 20 is a possible value for *minRead*.

The CMFS, however, is designed to utilize the asynchronous I/O facilities of the host operating system if available. When the same test was performed on the same disk device, but making requests for groups of blocks, the disk could retrieve 23 blocks in a slot time in nearly every case. The worst read time was 502 msec. Given that this is a contrived worst case example, we believe that 23 is a more realistic value of *minRead*.

The third method utilized the CMFS to calibrate the disk performance, Simultaneous requests for several CBR streams were submitted to the server to determine the worst case disk performance. The server was capable of supporting 26 streams which were spread out across the entire surface of a single disk and required an average of 1 block per slot, so the level of seek activity was high. Therefore, 26 could also be chosen as the value for *minRead*.

For the purposes of this study, the actual value of *minRead* is not of ultimate importance, because it will vary among different disk configurations. These values provide interesting bounds on disk performance, however, and allow us to investigate the significance of the value of *minRead*. In particular, we wish to examine whether a conservative estimate of *minRead* significantly affects the maximum cumulative bandwidth of stream scenarios accepted by the algorithms.

3.4 Initial Algorithm Evaluation

There are two other performance factors utilized in evaluating the algorithms. The average number of blocks read during a slot is defined to be *averageRead* and varies depending on the location of the data on the disk. On our experimental data, *averageRead* varied between 27 and 33 blocks per slot. The maximum number of blocks which can be read in a slot is defined to be *maxRead*. This value is determined by reading as many contiguous blocks as possible during a slot time where all the blocks are on the outside edge of the disk and was observed to be 40 blocks.

Figure 3 and Table 1 show that the streams are highly variable; the Simple Maximum Algorithm would reject many stream configurations. In particular, the Boxing clip has a maximum bandwidth which is more than twice its average bandwidth. For streams with these characteristics, Simple Maximum is very conservative, accepting less than half of what the Average algorithm accepts.

The Instantaneous Maximum Algorithm is also very conservative, but less so than the Simple Maximum algorithm. Figure 4 depicts an example of a simultaneous arrival pattern for 5 streams. The average bandwidth requirements for this set of streams is 22 blocks per slot, but the peak is at 32. Although the average rate is significantly below *minRead* (= 26), this set of streams will be rejected by both Simple Maximum and Instantaneous Maximum algorithms and accepted by the Average and VBR Simulation algorithms.

Although both Simple Maximum and Instantaneous Maximum are overly conservative, one major advantage of both of these algorithms is that neither of them require any additional buffer space for read-ahead at the server, since all the disk requirements for every slot can be met by the disk reads performed during that slot time. The amount of buffering needed in these algorithms is $2 * minRead$ buffers, as we em-

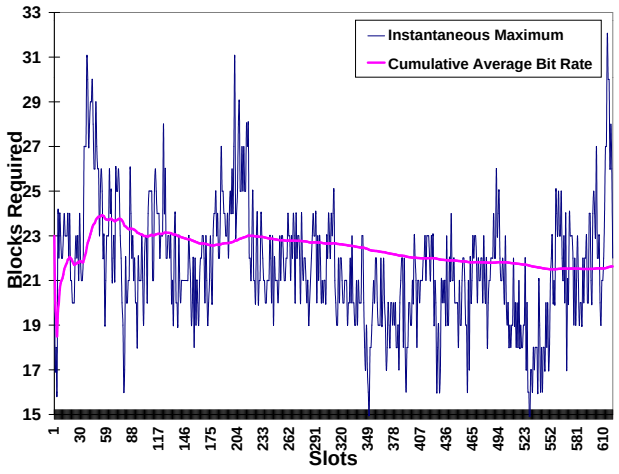

Figure 4: Instantaneous Maxima of 5 streams

ploy a double-buffering technique that reads data into one set of *minRead* buffers while transferring data across the network from the other.

It has already been shown [13] that the VBR Simulation algorithm (hereafter referred to as vbrSim) provides deterministic guarantees of data delivery to the clients. The set of streams in Figure 4 is accepted by vbrSim due to the smoothing effect.

In fact, all scenarios accepted by the Simple Maximum and Instantaneous Maximum algorithms are also accepted by vbrSim because no instantaneous peaks above *minRead* are allowed in the former algorithms and therefore, vbrSim has no peaks to smooth out.

The Average algorithm makes use of server buffer space to read at the average rate of the accepted streams or faster if possible. Buffered data which is read earlier than required is transmitted to the clients during the times that the required disk bandwidth rate is above the average. The Average algorithm, however, is overly aggressive and can fail in two ways: 1) there may be insufficient disk bandwidth at some point to support the requirements of the set of streams; or 2) there is insufficient buffer space.

The first case happens when cumulative average disk bandwidth required exceeds cumulative average disk bandwidth achievable. This typically happens early in the stream. As a result, it is possible for some set of streams, where the variation is great but the average is low, to be accepted but for which it is not possible to properly read the blocks during peak times.

The second case occurs because buffer space is finite. The Average algorithm will not be able to read at the average disk transfer rate once all server buffers are full. The performance will decrease to the rate at which buffers are freed (i.e. playout rate of admitted streams). This translates into a lack of read ahead for future transient peaks in the schedule.

Another typical scenario is depicted in Figure 5. The value

of *averageRead* in this scenario is 27.6. With *minRead* = 26, the Average algorithm accepts the set of streams as the sum of the average of each stream is 25.55. It is obvious from the graph that there is insufficient cumulative average disk performance to support the transfer of the required amount of data for many slots during the playout. The average requirement is initially over 27.6 blocks and only much later in the scenario does it drop below *minRead*.

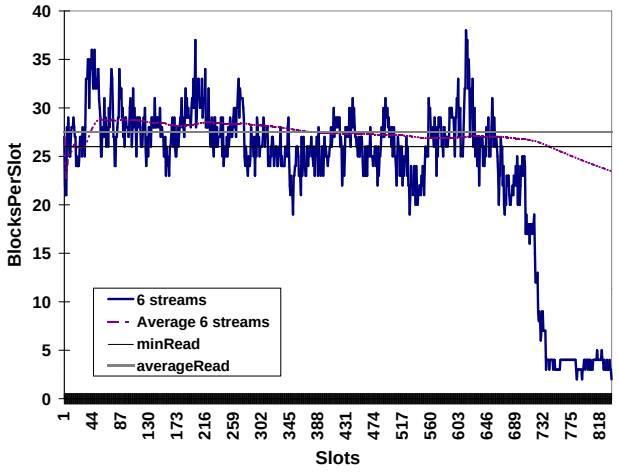

Figure 5: Average Bandwidth Algorithm

It is interesting to note that vbrSim is optimal when *averageRead*, *maxRead*, and *minRead* are all equal. We observe that when there is no variability in disk performance, the vbrSim algorithm knows the future performance of the disk. Any set of streams with a cumulative average bandwidth less than *minRead* will be accepted, provided there is enough buffer space. So, under these conditions, when buffer space limitations result in a rejection of a stream scenario by the vbrSim algorithm, it would also be the case that the optimal algorithm would reject that scenario.

The Simple Maximum and Instantaneous Maximum are also optimal in this case, but the Average Algorithm is not. It will still make incorrect admission decisions if peaks in disk bandwidth or buffer usage are above the capacity of the system.

If there is any variability in disk performance (the common case), vbrSim is less than optimal. In this case, *minRead* will be smaller than *averageRead*. Therefore, the cumulative average rate that can be accepted by vbrSim is less than *averageRead* and thus depends on the value selected for *minRead*. Figure 6 shows the required *minRead* values (x-axis) for scenarios with a particular disk bandwidth (y-axis), from the scenarios generated in Section 3. A particular scenario was matched to the minimum value of *minRead* which could admit that scenario, although it is obvious that any larger value of *minRead* would also accept the scenario. This plot shows that streams are accepted when the cumulative average bandwidth of the scenario approaches *minRead*, and occasionally a small amount more.

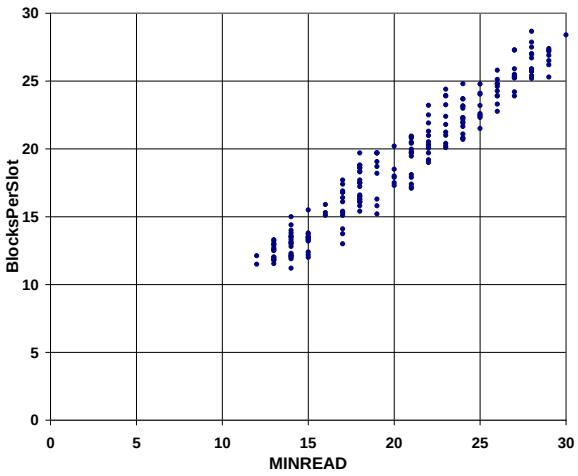

Figure 6: Bandwidth Supportable as a Function of *minRead* using the vbrSim algorithm

3.5 Scenario Characterization

There is an infinite number of possible combinations of stream requests that could be presented to a continuous media file server. They can be characterized in the following way: all possible scenarios, all valid scenarios, and all scenarios accepted by a particular algorithm. This is depicted in Figure 7 for the Average and vbrSim algorithms. The scenarios accepted by the Simple Maximum and Instantaneous Maximum algorithms are contained inside those accepted by vbrSim (as described earlier) and so are not shown. This diagram allows us to examine only the qualitative differences† between the scenarios accepted by the algorithms. For this discussion, we assume *minRead* = 26.

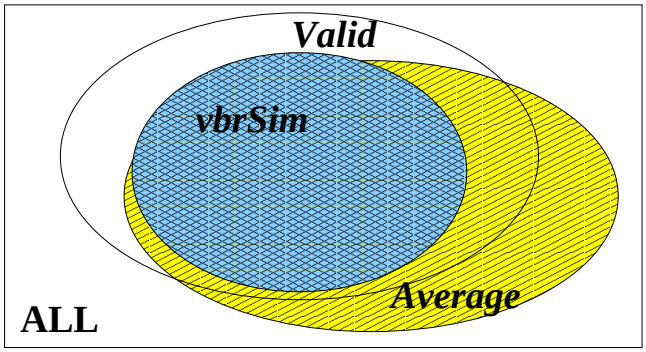

Figure 7: Streams Accepted by Admission Algorithms

There are several cases of inclusion to consider:

†Diagram is not "to scale", i.e. does not represent quantitative differences.

Case 1: $vbrSim \subseteq Valid$. All streams which are accepted by the vbrSim algorithm are valid streams. The vbrSim algorithm will not accept any invalid stream scenarios, since we guarantee that the disk has sufficient bandwidth to read all the required blocks and that there is always enough buffer space in which to read the blocks. This is shown in more detail in [13].

Case 2: $Average - Valid \neq \{\}$. The Average algorithm accepts invalid scenarios. If requests for the 6 streams in Figure 5 arrive simultaneously, the average bandwidth needed is 25.55, calculated by adding up the sums of the individual average bandwidths. This scenario would be accepted by the average algorithm, but the disk cannot support this scenario as too many blocks must be read in the early part of the scenario.

Case 3: $Valid - (Average \cup vbrSim) \neq \{\}$. There are valid scenarios that neither Average nor vbrSim accepts. This occurs if 7 of the selected streams (aretha, bloop93, chases, sintro, maproom, coaches and boxing) are requested with a 5 second stagger between request times. The sum of the average bandwidths is 27.13 blocks per slot. This is higher than *minRead*, so the Average algorithm would reject the scenario. The vbrSim algorithm also rejects this set of streams, but the scenario is valid.

Case 4: $(Valid - Average) \cap vbrSim \neq \{\}$. There are valid scenarios which vbrSim accepts while Average rejects. Consider a 2 stream scenario where Stream A has a constant bit rate of 12 blocks per slot for 50 slots and Stream B arrives 2 seconds later and has a requirement of 20 blocks per slot for 6 slots. The average algorithm rejects this scenario because the average is 32. The vbrSim algorithm accepts this set of streams because it will have achieved readahead of 56 by the time the request for the second stream arrives (assuming sufficient buffer space). The new stream will then read at *minRead* for 4 slots adding to the amount of read-ahead blocks by 6 blocks per slot, because it was only necessary to read 20 blocks in each slot. For the next 2 slots, 32 blocks are needed, so this uses up 6 read-ahead blocks per slot, but there is sufficient of read-ahead achieved in the previous slots to accept these stream. In practice, such a scenario is unlikely, but possible.

Case 5: $(Valid - vbrSim) \cap Average \neq \{\}$. There are valid stream scenarios which Average accepts and which vbrSim rejects. This is observed in a particular experimental scenario where 6 streams (bengals, chases, rescue, intro, boxing, and aretha) are requested simultaneously. The sum of the average bandwidth of each these streams is 25.77 blocks per slot. There is insufficient read-ahead guaranteed by *vbrSim*, so this scenario is rejected, but the server was capable of delivering all the data on time. Thus, the optimal algorithm would have accepted the scenario, due to disk bandwidth above *minRead* during some slot times.

4 RESULTS

This section describes the results of the experiments for the Instantaneous Maximum and vbrSim algorithms. The Simple Maximum algorithm accepts so few streams that the results are not interesting at all. We also do not show the results for the Average algorithm for two reasons: 1) its performance is analytically predictable accepting all streams with cumulative average rate less than *minRead* and 2) it has been shown to provide incorrect admission decisions, either by being both too conservative or too aggressive, with respect to the set of valid streams.

4.1 Admission Algorithm Execution Time

A major advantage of the Simple Maximum and Average algorithms is their execution time. Since stream characterization and is done with a single number, the time to determine admission is a single calculation, merely adding the new requirements to the existing requirements. Both Instantaneous Maximum and vbrSim require a block schedule to be computed and the bandwidth in each slot to be compared with the guarantee. The amount of overhead in these operations is a concern when utilizing them in a real system.

The block schedule for a 2-hour video clip can be computed in approximately 125 msec on a Pentium 200 running FreeBSD. If segments are to be skipped, or fast motion requested, less of the data stream or fewer slots (or both) are required, so in these situations the calculation typically takes less time. The block schedule for the same video clip stored at 30 frames per sequence, and skipping one sequence for every one displayed takes 64 msec to compute. When a stream is stored in smaller sequences, there is more calculation to perform, leading to a longer time for schedule creation and a smaller reduction in execution time when skipping sequences.

The time to examine the block schedule simulating readahead is proportional to the length of the non-empty portion of the server block schedule. For the same 2 hour video clip, the admission control process takes roughly 34 msec on a Pentium 200 running freeBSD. This value is higher than previously reported [13] due to the measurement of the algorithm within the entire CMFS. Earlier measurements were performed on the admission algorithm in total isolation.

4.2 Admission Decisions

As argued in Section 3, the vbrSim algorithm is an optimal algorithm if the disk performance is constant. Since that is not true in practice, we investigate how the performance of the algorithms degrades on a disk system with variable disk transfer rates. In order to isolate disk bandwidth issues from buffer space, we model a server with an unlimited amount of buffer space.

The results for the vbrSim algorithm, the Instantaneous Maximum algorithm and the optimal algorithm are compared using the three specific values for *minRead* obtained by calibration in Section 3.3: 20, 23, and 26. It is most interesting to consider those scenarios having cumulative average bandwidth near *minRead* blocks per slot and which would be accepted by the optimal algorithm.

Figures 8 and 9 show the acceptance rate of stream scenarios for the admission algorithms. Since we only consider scenarios that the disk was able to support, the values for the optimal algorithm would be uniformly at 100%, so it is not shown on the graph. Since each scenario uses different disk resources, we chose to group the scenarios in bands of the cumulative average bandwidth requirements as a percentage of the average disk performance obtained in the execution of that scenario. The observed values of disk performance

vary considerably, depending on the location of the streams on the disk and the amount of contiguous disk reading that was possible. This is most noticeable when a small amount of stagger is introduced. In this case, the disk reads the new stream, which is usually contiguous on disk, for a short amount of time immediately after being admitted until it catches up to the level of read-ahead of the existing streams, increasing the overall performance of the disk. When $minRead = 20$, $minRead/averageRead$ ranges between .60 to .74, showing that disk performance is variable. Similar ranges exist for the other values of $minRead$.

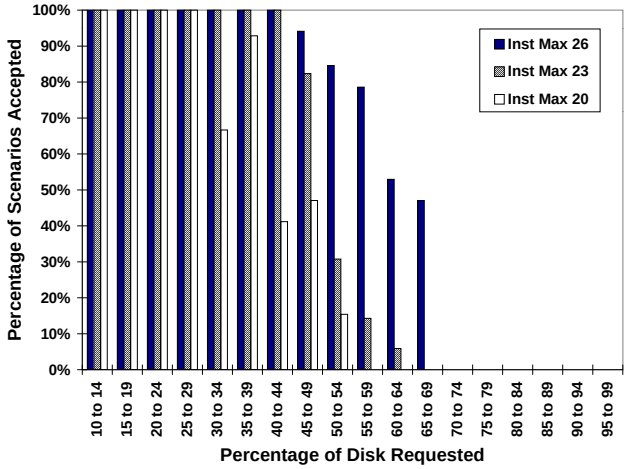

Figure 8: Acceptance Rate - Instantaneous Maximum

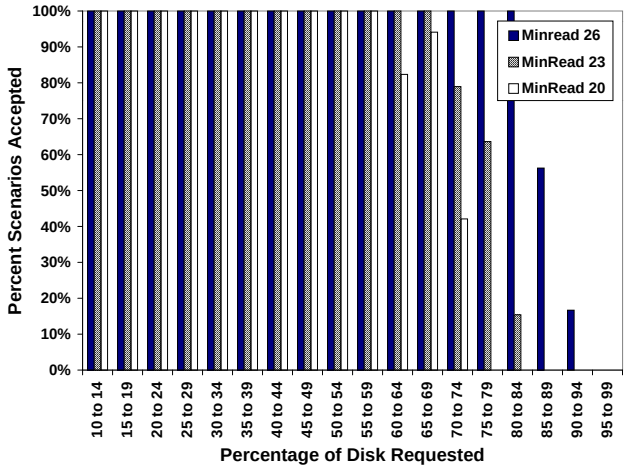

Figure 9: Acceptance Rate - vbrSim

Both figures show that for low levels of disk utilization, all the stream scenarios are accepted. In Figure 8, with $minRead = 20$, no scenarios above 54% disk utilization are accepted by the Instantaneous Maximum algorithm, and some scenarios that request less than 35% of the available disk bandwidth are rejected. When $minRead = 26$, no scenarios are accepted whose bandwidth request is greater than 69% of available disk bandwidth.

The vbrSim algorithm performs much better. Below 60%, the vbrSim algorithms accept all stream scenarios for all selected values of $minRead$. When $minRead = 20$, the acceptance rate starts to decline in the 60 to 64 percent band, which is very close to the percentage that $minRead$ is of $averageRead$. The vbrSim algorithm with $minRead = 20$ accepts some streams whose disk utilization request is over 70%, due to the combination of readahead smoothing in the past (prior to the arrival of some streams) and read-ahead in the future (at $minRead$). Further investigation will attempt to identify which of these components is more significant. When $minRead = 23$, almost all scenario requests below 75% are accepted and a steady drop-off is observed as the requested level of disk utilization is increased. The pattern is similar for $minRead = 26$. The range of $minRead/averageRead$ is .79 to .96, and some stream scenarios are accepted in the 90 to 94% range. This shows that vbrSim does come quite close to accepting streams with cumulative average bandwidth very near the level of $minRead/averageRead$. For some of these scenarios, the cumulative average bandwidth required did indeed exceed the value of $minRead$.

More insight into the vbrSim algorithm can be gained by measuring the bandwidth achieved by the server as it accepts partial stream scenarios. This is shown in Figures 10 and 11. It would be reasonable to assume that the server would be capable of supporting near or slightly more than $minRead$ blocks per slot, regardless of the request size.

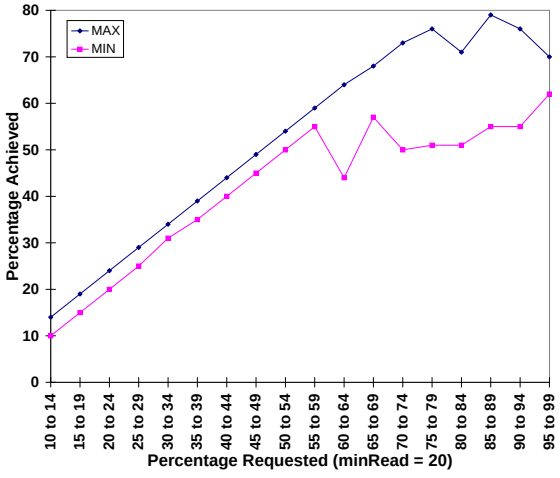

Figure 10: Bandwidth Supported by Disk Requests (vbrSim only)

Each graph contains two lines: one which indicates the largest bandwidth accepted for scenarios within that percentage band of requested disk utilization, and one which indicates the smallest bandwidth that was accepted. This

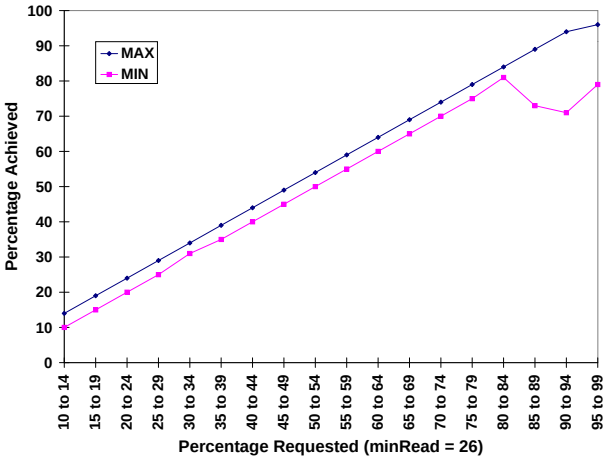

Figure 11: Bandwidth Supported by Disk Requests (vbrSim only)

is when the total request fit into the particular percentage band and only part of the scenario was accepted. Thus, the lines are parallel during the low level of disk utilization where all scenarios are accepted.

When the level of disk performance requested increases, the range between the 2 lines begins to increase. As streams are rejected, the minimum value accepted flattens out. This happens immediately after 59% for $minRead = 20$. For $minRead = 26$, this happens later.

There is also a flattening of the line for the maximum bandwidth supportable. This comes at slightly over 75% for $minRead = 20$. The graph indicates that no matter what level of disk performance is requested, the most that *vbrSim* with $minRead = 20$ will achieve is 79% of the disk bandwidth. This is reasonable, considering $minRead/averageRead$ ranges from 61-74% of disk utilization. For $minRead = 26$, the maximum achievable bandwidth is approximately 94%, again an indication that disk utilization approaches $minRead/averageRead$.

4.3 Buffer Space Requirements

Since a real server will not have an unlimited amount of buffer space, we now consider the impact of limited buffer space. Some of the analysis in this section uses inflated values of *minRead*, so that we can isolate buffer space utilization from the restrictions placed on stream acceptance by the amount of guaranteed bandwidth.

If the rate at which buffers are freed is less than *averageRead*, the value of *averageRead* decreases towards the playout rate of the accepted streams. The server was instrumented to capture the number of buffers utilized to keep the server reads from slowing down as a result of insufficient buffers. In one particular scenario, the first 7 streams of Table 1 were requested with a 3 second stagger in arrival time. This scenario was supported by the server. The cumulative average disk bandwidth achieved was approximately 30.5 blocks per slot. In order to support that rate, this particular scenario utilized 832 buffers (approximately 52 MBytes).

Static analysis of the bandwidth requirements determined that 30 was the smallest value of *minRead* which was able to accept this scenario. Static analysis of the buffer requirements was also performed to determine the minimum number of buffers *required* for the vbrSim algorithm to accept a scenario. This analysis assumed that the disk reads precisely *minRead* blocks in every slot. For $minRead = 30$, only 605 buffers were required to smooth out all the peaks in the block requirements of these 7 streams. This implies that with this amount of buffer space, the disk must throttle itself at some point during the reading process, while waiting for buffers to be returned to the system, but will still never fall behind. Further analysis shows that when $minRead = 31$, 321 buffers are needed when $minRead = 32$, the number required drops to 157 buffers. This is because there are fewer peaks above *minRead* and they are of shorter duration.

Figure 12 shows the number of buffers required to accept several of the stream scenarios with $minRead = 26$. A similar pattern is observed with the other selected values of *minRead*. This seems to suggest a relationship between the number of buffers and the cumulative average bandwidth that can be supported. We can see from this scatter plot that the amount of buffer space appears to grow exponentially as the cumulative average bandwidth of the admitted streams approaches *minRead*. When valid scenarios with larger bandwidth requirements (w.r.t. *minRead*) are submitted to the CMFS, there is a significant increase in buffer requirements. In these scenarios, staggering arrivals allows the cumulative average bandwidth in the accepted streams to be over *minRead* at many points in the schedule. In one particular case (not shown on this graph), when 5 streams are submitted (bloop93, bengals, intro, maproom, and coaches), with 5 seconds between arrivals and with $minRead = 20$, the cumulative average bandwidth is 21.34 blocks per slot and the number of buffers required is 1175, which is 75.2 MBytes, but vbrSim does accept the scenario.

It is easy to construct other scenarios in which large amounts of buffering will be beneficial and increase performance significantly. If streams are relatively small compared with buffer space, entire streams could be buffered in the server. When the stream requests arrive simultaneously, admission would still be largely based on bandwidth. If inter-arrival time was long, relative to stream length, then the ability to read contiguously from the disk during stream start up would result in disk performance much greater than average for those points in time. These considerations introduce extra levels of complexity into the analysis which are beyond the scope of this paper.

For most of the stream scenarios depicted in Figure 12, a reasonable amount of buffer space is required. This presents a design alternative for the administrator of a continuous media file server. Additional buffers may be used to extract the most bandwidth out of a particular disk device, or the number of disk devices may be increased, which also has the effect of increasing bandwidth.

Adding more disks must also be accompanied by more buffer space, or the number of buffers available for each disk decreases. Increasing from n to $n+1$ disks incurs a $1/(n+1)$ decrease in buffer allocation for each disk. As can be seen in

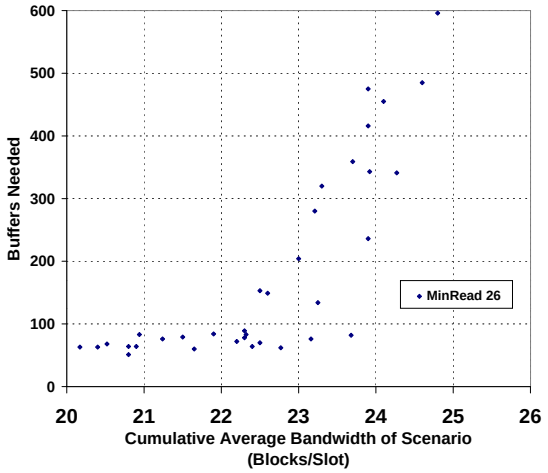

Figure 12: Bandwidth supportable versus Buffer Space

this section, this will reduce the ability of the system to accept scenarios that require average bandwidth close to *minRead*, and may not provide an increase in system throughput. A more detailed quantitative analysis of this tradeoff is planned as part of future work.

5 RELATED WORK

There have been several admission control algorithms introduced in the literature. Much of the work in continuous media servers has focussed on either conservative approaches based on bandwidth peaks or statistical methods which model arrival rates and stream bandwidths with probability distributions and determine a satisfactory level of performance by the disk and network subsystems, either separately or as a system.

The majority of early work in VBR systems simply extended the principles used in CBR streams and followed the pattern of Simple Maximum ([7, 18]). In Vin and Rangan [18], data layout patterns as well as admissions scheduling was done with streams of varying data rates, but the rate within a stream was considered a constant. Lau and Lui [11] perform an admission test that considers the peak rate needed to determine if a stream is admissible, by delaying the starting time and readjusting the disk tasks to minimize other measures of performance. They utilized streams that had bandwidths which varied uniformly between 2.5 and 5 MBytes per second, and performed statistical analysis of arrival rates with their deterministic algorithm. We can see that for streams with a reasonable difference between their average and their peak rates, this provides an unacceptably low level of utilization.

The performance of a video server is measured in Liu et al. [12], wherein differing types of video are compared. In their study, each type of video has a constant bit rate, and the focus is on data placement (i.e. striping) and buffer usage, not on admission control details.

In Kamath et al. [9], the admission control checks to see if the cumulative necessary disk transfer rate required violates the guarantee and that there would be sufficient buffer space to read the additional disk blocks. Their algorithm is similar to Instantaneous Maximum, but their focus is on sharing data between multiple users, thereby achieving a reduction in disk activity when the same stream is requested more than once within short periods of time.

Statistical methods based on the recent past performance are introduced in Jamin et al. [8] that combine client specified parameters of the new stream with the observed measurements of the system to make an admission decision. The new performance measurements are then utilized in future admission control requests.

In Vin et al. [17], a statistical admission control algorithm is presented, which considers not only average bit rates, but the distributions of frame sizes, and probability distributions of the number of disk blocks needed during any particular service round. They acknowledge that the algorithm breaks in certain circumstances referred to as overflow rounds (i.e. over subscribes the disk). In overflow rounds, the system has the complexity of dealing with the loss of data. A greedy disk algorithm attempts to reduce the actual occurrence of overflow rounds, and the system attempts to judiciously distribute the effective frame loss among the subscribed clients. This requires some knowledge of the syntax of the data stream, at least to the point of knowing where display unit (i.e. video frame) boundaries exist and which display units are more important than others (i.e. MPEG I-frames vs. MPEG B-frames).

A more complex version of the Average Algorithm is given for Constant Time Length (CTL) video data retrieval by Chang and Zakhor [3]. Constant Data Length retrieval methods introduce buffering for the purposes of prefetching portions of the stream and incorporating a start-up latency period. In further work [4], they show via simulation that a variation of deterministic admission control admits 20% more users than their statistical method for a small probability of overload.

Knightly et al. [10] perform a comparison of different admission control tests in order to determine trade-offs between deterministic and statistical guarantees. The streams used in their tests are parameterized by a traffic constraint function, known as the empirical envelope. It describes the bandwidth needed at various points during stream transmission, so it is somewhat similar in form and function to the block schedule as presented in this paper, but much less detailed. This characterization is used in admission control. This is then combined with different packet transfer schemes and so does not particularly isolate each subsystem. Zhang et al.[19] have worked on network call admission control methods with smoothed video data taking advantage of client buffering capabilities. This reduces the amount of buffering needed at the server and increases potential network utilization.

Recent work by Dengler et al. [6] and Biersack and Thiesse [2] builds on the work of Knightly et al.[10] and Chang and Zakhor [4], describing admission control methods which provide statistical and deterministic guarantees of disk service for VBR streams. The major focus is data placement strategies and the use of traffic constraint functions is prominent. Constant Time Length (CTL) placement with deterministic guarantees is investigated in [6], while statistical admission control and Constant Data Length is examined in [2].

6 CONCLUSIONS AND FUTURE WORK

In this paper, we have presented five disk admissions control algorithms for variable bit rate continuous media streams. We introduced the VBR Simulation Algorithm (vbrSim) that explicitly considers the detailed variable bit rate profile and accommodates disk read ahead based on the *achieved* read ahead in the past and the *guaranteed worst-case* read ahead performance in the future.

The Simple Maximum, Instantaneous Maximum, and vbrSim algorithms provide deterministic guarantees that the continuous media will be read off the disk for proper delivery to the client, while the Average algorithm provides only statistical guarantees. We showed that the Average algorithm can make incorrect decisions on rejecting and accepting stream scenarios. The performance of the Instantaneous Maximum algorithm was significantly worse than vbrSim for all cases of the representative selection of Variable Bit Rate media stream scenarios.

Comparisons of admission performance showed that the vbrSim algorithm admits a very large percentage of stream scenarios that have disk utilization below *minRead/averageRead*. As the disk bandwidth requested approaches *minRead/averageRead*, fewer scenarios are accepted, but the level of bandwidth supported by the algorithm remains relatively constant. Having an appropriate value for *minRead* significantly affects the disk utilization, and in situations where the worst case disk performance is significantly less than *averageRead*, the vbrSim algorithm degrades proportionally.

Designers of continuous media systems must take into account the negative effect of aggressively incorrect admission decisions on the client population. As well, the amount of buffer space that can be allocated for read-ahead to smooth the peaks in disk data-rate requirements is a factor which must be considered.

We have shown that reasonable amounts of buffer space can enable the *vbrSim* algorithm to work close to its maximum acceptance rate, which is close to optimal. It is also simple and efficient to execute [13]. Utilizing a simpler algorithm will result in conservative resource utilization. The use of statistical algorithms may result in failure to deliver the data to the client. In statistical algorithms, if the disk performance measure is the average disk bandwidth, then the admission decision results in failure to deliver if either the instantaneous maximum over-subscribes the disk or the disk achieves less than this average bandwidth. Utilizing *minRead* as the disk performance estimate provides little performance benefits over *vbrSim*, while eliminating delivery guarantees. Thus, *vbrSim* seems a completely reasonable choice of admission algorithm for continuous media servers.

Further performance analysis is underway to examine the sensitivity of several other factors in the configurations of the streams and the server resources. These factors include: the amount of variability in the streams, the length of the streams, the availability of additional client buffer space (accompanied by sufficient network bandwidth), and the interarrival time of requests at the server. This performance analysis must be extended to consider the network transmission. This has been examined in some detail in [14], and more work is planned for the near future.

ACKNOWLEDGMENTS

The authors would like to acknowledge the efforts of David Finkelstein for his work on the client used in the testing and on his contribution to the ideas used in this paper. As well, the work of Roland Mechler contributed greatly to the underlying network protocol and system level support for the CMFS.

References

[1] David P. Anderson, Yoshitomo Osawa, and Ramesh Govindan. A File System for Continuous Media. *ACM Transactions on Computer Systems*, 10(4):311–337, November 1992.

[2] Ernst W. Biersack and Frederic Thiesse. Statistical Admission Control in Video Servers with Constant Data Length Retrieval of VBR Streams. In *Third International Conference on Multimedia Modeling*, Toulouse, France, November 1996.

[3] Ed Chang and Avideh Zakhor. Admissions Control and Data Placement for VBR Video Servers. In *1st IEEE International Conference on Image Processing*, pages 278–282, Austin, November 1994.

[4] Ed Chang and Avideh Zakhor. Cost Analyses for VBR Video Servers. In *IST/SPIE Multimedia Computing and Networking*, pages 381–397, San Jose, January 1996.

[5] Ming-Syan Chen, Dilip D. Kandlur, and Philip S. Yu. Support for Fully Interactive Playout in a Disk-Array-Based Video Server. In *ACM Multimedia*, pages 391–398, San Francisco, October 1994.

[6] Johannes Dengler, Christoph Bernhardt, and Ernst W. Biersack. Deterministic Admission Control Strategies in Video Servers with Variable Bit Rate Streams. In *Interactive Distributed Multimedia Systems and Services, European Workshop IDMS'96*, Heidelberg, Germany, March 1996.

[7] Jim Gemmell and Stavros Christodoulakis. Principles of Delay-Sensitive Multimedia Storage and Retrieval. *ACM Transactions on Information Systems*, 10(1), 1992.

[8] Sugih Jamin, Scott Shenker, Lixia Zhang, and David D. Clark. An Admission Control Algorithm for Predictive Real-Time Service. In *3rd International Workshop on Network and Operating Systems Support for Digital Audio and Viudeo*, November 1992.

[9] Mohan Kamath, Krithi Ramamritham, and Don Towsley. Continuous Media Sharing in Multimedia Database Systems. Technical Report 94-11, Department of Computer Science, University of Massachussets, Amherst MA 01003, 1994.

[10] Edward W. Knightly, Dallas E. Wrege, Jorg Lieberherr, and Hui Zhang. Fundamental Limits and Tradeoffs of Providing Deterministic Guarantees to VBR Video Traffic. In *ACM SIGMETRICS '95*. ACM, 1995.

[11] S. W. Lau and John C. S. Lui. A Novel Video-On-Demand Storage Architecture for Supporting Constant Frame Rate with Variable Bit Rate Retrieval. In *5th International Workshop on Network and Operating System Support for Digital Audio and Video*, Durham, N.H., 1995.

[12] Jonathan C. L. Liu, Jenwei Hseih, and David H.C. Du. Performance of A Storage System for Supporting Different Video Types and Qualities. *IEEE Journal on Selected Areas in Communications: Special Issue on Distributed Multimedia Systems and Technology*, 14(7):1314–1341, September 1996.

[13] Gerald Neufeld, Dwight Makaroff, and Norman Hutchinson. Design of a Variable Bit Rate Continuous Media File Server for an ATM Network. In *IST/SPIE Multimedia Computing and Networking*, pages 370–380, San Jose, January 1996.

[14] Gerald Neufeld, Dwight Makaroff, and Norman Hutchinson. Server-Based Flow Control in a Continuous Media File System. In *6th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 29–35, Zushi, Japan, April 1996.

[15] P.V. Rangan and H.M. Vin. Designing File Systems for Digital Video and Audio. In *Proceedings 13th Symposium on Operating Systems Principles (SOSP '91), Operating Systems Review*, volume 25, pages 81–94, October 1991.

[16] P.V. Rangan and H.M. Vin. Efficient Storage Techniques for Digital Continuous Multimedia. *IEEE Transactions on Knowledge and Data Engineering Special Issue on Multimedia Information Systems*, August 1993.

[17] Harrick M. Vin, Pawan Goyal, Alok Goyal, and Anshuman Goyal. A Statistical Admission Control Algorithm for Multimedia Servers. In *ACM Multimedia*, pages 33–40, San Francisco, October 1994.

[18] Harrick M. Vin and P. Venkat Rangan. Admission Control Algorithms for Multimedia On-Demand Servers. In *3rd International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 1992.

[19] Zhi-Li Zhang, Jim Kurose, James D. Salehi, and Don Towsley. Smoothing, Statistical Multiplexing and Call Admission Control for Stored Video. *IEEE Journal on Selected Areas in Communications: Special Issue on Real-Time Video Services Multimedia Networks*, 1997.