

# The Design, Analysis, and Implementation of a Media Server Architecture

---

**James G. Hanko**

Sun Microsystems Laboratories

---

## Abstract

*The primary goal of a media server is to reliably deliver continuous data streams to as many clients as feasible given the available hardware resources. This paper describes the design and implementation of the Sun Media Server, and analyzes the techniques used to meet this goal.*

## 1 Background

---

There is growing commercial interest in building media servers out of components originally developed for computer systems. *Media* in this context means continuous streams of time-critical data, such as digital audio and video, and a *stream* denotes the active process of reading data from the disks and delivering it to a client over a network. An effective media server can today be constructed from commodity computer systems and disk arrays to serve many clients over a high-bandwidth digital network.

A server for continuous media streams primarily acts as an engine for reading media data from disk storage and supplying it to a network connection at the proper delivery rate. In order to be successful, a server must be able to reliably supply as many clients as feasible given the hardware resources. It must also be able to accurately predict whether a particular load of clients is sustainable without disruption to the media streams.

In building a system to serve media clients from a set of disks, the parameters that constrain the number of active users are:

- the number of active streams supported by each disk
- the distribution of stream data over the disks

The number of streams supported by each disk are further constrained by:

- the amount of data transferred in each disk read
- the disk bandwidth during transfers
- the seek time between transfers

It must also be recognized that the response of a modern disk drive (as seen by a system) is non-deterministic. This is due to processing within the disk subsystem such as thermal recalibration, disk retries, error correction using error correcting codes, disk block error remapping, etc. These effects are not under direct system control and typically do not affect the data transferred, but instead affect the time period over which the transfers occur. A successful server design must be able to absorb a substantial amount of variability in disk response without disrupting the media streams.

The general problem of managing shared computing resources for time-critical applications in the face of non-determinism has been studied, and effective approaches have been developed [2], [10]. These methods generally rely on algorithms to predict or detect resource overload and mechanisms to bring the system back into equilibrium. However, in the case of a dedicated media server, aspects of the problem allow simpler methods to be used. Specifically, there is only one time-critical task that the system must perform (i.e., delivering media streams)

and the computational load in doing so is small. Therefore, the only resources that must be carefully managed are disk and network bandwidth. In addition, the load imposed on the system by a stream can be accurately predicted from the characteristics of the media data (e.g., bit rate).

This paper describes the Sun Media Server software design, analyzes the design trade-offs, and describes the implementation. The primary emphasis is on the disk scheduling and the admission control subsystems, which are collectively called the Sun Media Scheduler.

## 2 Assumptions

In general, there is little locality between disk access locations for different streams on a media server. This is because each data file is so large (multiple GBytes each), and it is played out over a long time (e.g., two hours). Therefore, the probability of two clients accessing the same data within a short time period is very small.

The average case disk seek times quoted by disk manufacturers are computed by dividing the sum of the time taken for all possible seeks by the number of possible seeks. The average distance across all possible seeks is  $1/3$  of the number of cylinders, although the time for a seek of this distance does not necessarily correspond to the quoted average seek time [3]. The worst case seek time comes from a seek across the full disk.

Disk seeks have four major components:

- 1) acceleration
- 2) constant velocity motion
- 3) deceleration
- 4) head settle time

With suitably long seeks (as in media servers) acceleration, deceleration, and head settle time can be thought of as a fixed overhead<sup>1</sup>. Because of this, the following relationship holds:

$$S(X_1) + S(X_2) + \dots + S(X_n) \leq n \cdot S\left(\frac{1}{n} \cdot \sum_{i=1}^n |X_i|\right)$$

1. Very short seeks would take less time than this because full acceleration is not required.

where  $S(x)$  is the time to seek a fraction  $x$  of the disk, and  $X_1 \dots X_n$  represent a series of disk seek fractions. That is, the worst case seek time for a group of  $n$  seeks covering some total distance  $d$  occurs when each individual seek is of equal length, i.e.,  $d/n$ .

A track near the outer edge of a disk is longer (i.e., has a larger circumference) than one near the inner edge, so there is potentially a greater storage capacity in the outer tracks. Because of this, modern disks are divided into several regions, with the regions at the outer edge of the disk containing more sectors than those at the inner edge. Therefore, with constant rotational velocity, the bandwidth obtainable from the outer regions is larger than from the inner ones. The worst case bandwidth occurs in the innermost region.

Finally, disk overhead is minimized by transfers of larger amounts of data at a time. This is true because larger transfers mean that a higher proportion of the time is spent transferring data, and a lower proportion is spent doing seeks. Therefore, a server using longer transfers will be able to support more clients, all else being equal.

## 3 Approach

On modern disks, the observed variances between worst case and average case for both seek time and disk transfer bandwidth described above can approach 2-to-1. Therefore, any scheme to maximize the number of supported streams must ensure that, over some small interval of time, the average case bandwidth and seek times are always achieved. If the worst case bandwidth or seek times are allowed to persist over an unbounded period, then the number of supported streams must be constrained by these worst case numbers. In other words, unless active measures are taken to limit worst case behavior, it can persist for an arbitrary amount of time. Then any assumption of better than worst case response will result in the disruption of some or all streams. If, however, bandwidth and seek times can be guaranteed to achieve average case numbers during bounded (short) intervals,

these parameters can be used to calculate the number of supported streams (which would be larger than with the worst case numbers).

Even if the stream admission policy used by a server is stochastic instead of deterministic, guaranteeing average case performance can be used to increase the number of supported streams. This is because the guarantee will minimize the performance variance and, thereby, increase the confidence factor for supporting larger numbers of streams.

The design of the Sun Media Scheduler can guarantee approximately average case bandwidth and seek times over a bounded (short) interval. This is achieved through the disk layout and the stream admission policy.

### 3.1 Disk Layout

The layout of the data on the disks is critical for server performance. In the Sun Media Server, the concept is to divide each disk into  $Z$  zones containing equal numbers of disk blocks (in the current implementation,  $Z$  is set to two). Each zone is a contiguous group of disk blocks and roughly corresponds to one or more of the above-described regions created by the disk manufacturer. For example, if  $Z=2$ , then one zone would contain that half of the disk blocks in the faster regions of the disk, and the other would contain the half of the disk blocks that are in the slower regions.

Each *title* (movie, video clip, etc.) is striped across  $D$  disks and  $Z$  zones. Data for a title is organized into groups of contiguous disk blocks called *extents*. The extents containing the data for the titles are laid out such that extent  $i$  of each title would be placed<sup>2</sup> in zone:

$$\left(i + \frac{i}{D}\right) \text{ modulo } Z$$

2. Note this description is for convenience only. Excessive fragmentation would occur on the last disk because all movies start on the same disk and end at  $(l \text{ mod } D)$ , where  $l$  is a title's length in extents. So, on average only  $1$  of  $D$  movies would place their last extent on disk  $(D-1)$ . A similar problem would occur if all movies started in the same disk zone. To avoid this, starting extents for different movies are distributed over the set of disks and zones.

of disk:

$$i \text{ modulo } D$$

This scheme generally alternates zones on adjacent disks, with a possible repeat of zones in the transition from the last to first disk. For example, with two zones and five disks the layout  $\langle \text{disk}, \text{zone} \rangle$  would be:

$$\langle 0,0 \rangle \langle 1,1 \rangle \langle 2,0 \rangle \langle 3,1 \rangle \langle 4,0 \rangle \langle 0,1 \rangle \langle 1,0 \rangle \dots$$

The server utilizes the concept of a *scheduling interval*, a configuration-specific period of time in which data for all clients is read from the disks. The media data for each title is saved on the disk so that the data needed for a stream in each scheduling interval is stored in exactly one extent. This means that titles of different bit rates (e.g., with different compression methods or parameters) will have different sized extents, but that all the extents will correspond to the same play time interval on their respective titles.

The server is designed using the so-called *push* model in which the pacing of the data to the client is completely under the control of the server. That is, there is no rate-control feedback loop between client and server, and it is the responsibility of the client to follow the server's rate. There are two major reasons for this decision. First, the server is designed to support broadcast and near-video-on-demand (a multicast service), where one stream goes to many clients, as well as a video-on-demand model where each stream only goes to one client (i.e., point-to-point). In a broadcast or multicast environment, there is no meaningful way for the server to combine rate feedback from multiple clients. Since it is necessary for clients to function in this environment, creating a separate mechanism for point-to-point communication would just introduce unnecessary complexity.

A second reason for using the push model is to maximize the capacity of the server, while minimizing resource usage. If different streams proceeded at different rates, there would be no way to prevent them from drifting to a point where they would all be accessing the same disk at the same time. Clearly one disk could not simultaneously supply as many clients as all

disks together. Therefore in order to deal with this, it would be necessary to buffer a full cycle of data from each disk for each stream. That would require an increase in buffer capacity by more than an order of magnitude. In addition, since all clients could request data faster than the nominal rate of the stream, the server capacity would have to be derated by the worst-case (or at least the expected-case) variance in rates.

By using the push model, the Sun Media Server can ensure that no stream drift occurs. The scheduler uses the concept of *slots* (described below) to organize streams into groups that share access to the disks. When two streams are placed together in a slot, the use of the push model ensures that they will never move out of phase with each other, and they will never need to access a disk that is being used by another slot. This minimizes the buffer requirements because a *just-in-time* strategy can be used. In addition, it allows the stream admission algorithm to predict disk loading into the indefinite future both precisely and easily.

### 3.2 Stream Admission and Scheduling

The Sun Media Scheduler organizes sets of streams into *slots* in order to manage the disk bandwidth and seek times. Let  $B$  represent the total bandwidth that can be accommodated by one disk in the server, and  $Z$  be the number of zones used. Streams can be added to a slot only so long as all the streams in the slot use a total bandwidth less than or equal to  $B$ . The number of slots is equal to the number of disks in the system. Streams in one slot always access the same disk in each time period. Within a slot, streams are divided into  $Z$  groups, where all members of a group always access the same disk zone. The total bandwidth used by streams in each group is kept approximately equal. In other words, each zone of every disk is accessed every time period by a group of streams using bandwidth of approximately:

$$\frac{B}{Z}$$

Streams in each slot proceed together from disk to disk in lock-step. When the slot makes the transition from one disk to another, the disk zone accessible to each stream changes in accordance

with the layout policy. This maintains the balance between zone accesses, while allowing every stream to access all areas of the disks.

Within each slot, the read requests for a disk are ordered by disk block. The scheduler seeks to the end of the ordered set of requests nearest the current head position, then proceeds through the set to the nearest subsequent position. This is, effectively, the well-known *elevator* or *Scan* algorithm performed on each slot. Such a configuration of groups using the Scan algorithm has been called a *group sweeping scheme* [1].

At any given time, only one slot may read data from a given disk. Each disk is assigned to one slot in every time period. All of the streams in the slot read their required data from that disk and then send it out over the network in the next time period. This *just-in-time* approach minimizes the buffer occupancy (and therefore total memory requirements) compared to traditional scheduling schemes [1], [8].

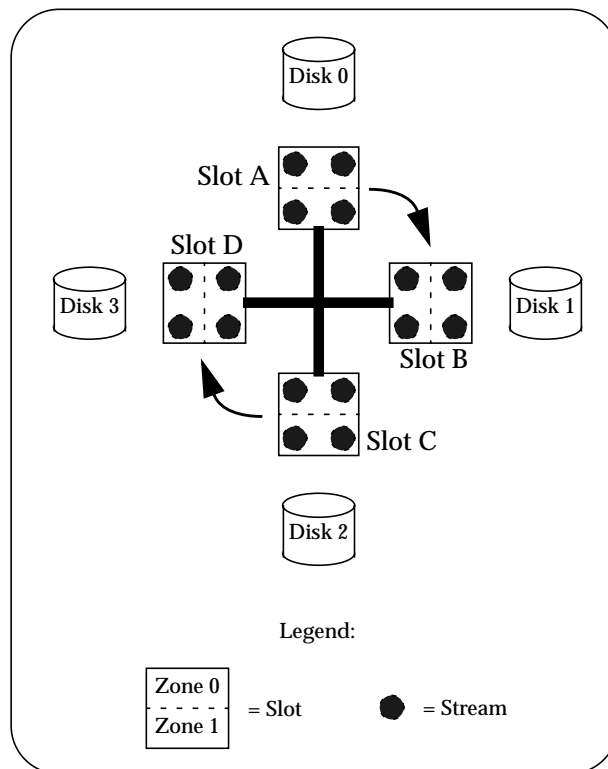


FIGURE 1. Scheduler Operation

The operation of the Sun Media Scheduler can be visualized as a carousel in which each slot is

briefly attached to a disk in order to access information, and then moves on. Figure 1 shows how a system with four disks and four slots would operate. In the position shown, Slot A is attached to Disk 0. While it is attached, it will read data for all four streams in slot A from both zones of the disk (two streams are using each zone). Similarly, Slot B is reading from Disk 1, etc. When the time period is up, Slot A will advance to Disk 1, Slot B to Disk 2, etc., and the data read in the first time period will be sent to the clients via the network. When Slot A moves to Disk 1, the zones that the individual streams in Slot A access on the disk adjust to match the media layout policy; this transition is the same for all titles.

Slot	Stream	Disk, Zone access by time period						
		t0	t1	t2	t3	t4	t5	t6
0	0	0,0	1,1	2,0	0,1	1,0	2,1	0,0
	1	0,0	1,1	2,0	0,1	1,0	2,1	0,0
	2	0,1	1,0	2,1	0,0	1,1	2,0	0,1
	3	0,1	1,0	2,1	0,0	1,1	2,0	0,1
1	4	1,0	2,1	0,0	1,1	2,0	0,1	1,0
	5	1,0	2,1	0,0	1,1	2,0	0,1	1,0
	6	1,1	2,0	0,1	1,0	2,1	0,0	1,1
	7	1,1	2,0	0,1	1,0	2,1	0,0	1,1
2	8	2,1	0,0	1,1	2,0	0,1	1,0	2,1
	9	2,1	0,0	1,1	2,0	0,1	1,0	2,1
	10	2,0	0,1	1,0	2,1	0,0	1,1	2,0
	11	2,0	0,1	1,0	2,1	0,0	1,1	2,0

Table 1: Slot Accesses to Disks and Zones

An alternative view of the Sun Media Scheduler's operation is shown in Table 1. In this example, there are three disks, two zones, and twelve streams shown over seven time periods (scheduling intervals). As can be seen from the table, the disks are accessed in all zones every time period. This activity is balanced among the zones to achieve the average case disk bandwidth. Note that every stream has access to every zone of all the disks, but at differing times. The access pattern of period t6 is a repeat of period t0. In general, the access patterns in the

Sun Media Scheduler repeat after  $disks \cdot zones$  scheduling intervals.

The scheduler is configured to use the Sun Data Pump [5], [9] to deliver the data streams to the network interface and to regulate their rates. In order to avoid the problem of having separate clocks in the scheduler and in the Pump, potentially moving at differing rates, the scheduler uses a feedback mechanism from the Data Pump to control the advance of the scheduler time periods. This means that the scheduler's interval is derived from timing information provided by the pump, ensuring that they advance at identical rates over the long-term.

When a new stream is requested, the admission policy attempts to place it in the slot that will next be assigned the disk containing the first block of the stream. However, if that slot is full, the admission policy scans backwards for a slot that will soon have access to the proper disk and zone and which has sufficient bandwidth for the stream.

Once a new stream has been admitted, the scheduler ensures that it begins in phase with the other streams in the system. This is accomplished by delaying the sending of its first block of data until the last block of data for the previous time period has been sent for a majority of the active streams. This minimizes the spread between the streams in the server and also minimizes buffer usage and eliminates start-up jerkiness for new streams.

The bandwidth allocation for each slot is based on the amount of I/O that each disk can perform in one scheduling period, de-rated by the amount of time lost to seeks.

### 3.3 Work-Ahead

In order to deal with the inherent non-determinism of disk subsystems, a feature called *work-ahead* has been designed into the Sun Media Scheduler. When the disk accesses for one slot in one time period have been completed, the disk is made available to begin work early for the next slot to access the disk. Work-ahead may occur on a disk independent of the other disks' activities.

The admission control subsystem ensures that the full system load does not equal or exceed the total capacity of the disk subsystem. Because of this, there is a long-term rate mismatch such that, over time, the disk subsystem can always supply more data than needed. The work-ahead feature exploits this mismatch, so that in steady state the system is approximately one full time period ahead of where it needs to be to supply the clients. That is, it builds up *slack time* in the disk schedule.

Because of the work-ahead strategy, the admission subsystem can deliberately create short-term overloads, for example by putting too many streams in one slot, as long as it maintains the long term load below the disk capacity. This can be used to minimize the time a new client must wait to get the stream data. In addition, the slack can be exploited to maximize the number of clients overall.

For example, if the disk capacity is such that a disk can serve 4.5 clients each time period, the admission algorithm can alternate slots with four streams and slots with five streams. Although the slots with five streams cannot read the disk in one time period, each pair of slots completes within two time periods. Because of the slack built by work-ahead, the clients see no disruption in service.

Work-ahead also gives the server a level of immunity from the non-determinism of the disk subsystem. Typical disruptions such as sector retries, bad block forwarding, etc. will delay a disk transfer from 20 to 100 milliseconds. Delays introduced by thermal recalibration typically range from 50 to 300 milliseconds<sup>3</sup>. By setting the scheduling interval of the server to 1/2 second (500 milliseconds), the server is capable of absorbing substantial disruptions without affecting the client streams by consuming the built-up slack. As long as the disruptions do not persist (as they do in a failing disk), server will quickly rebuild its slack and suffer no service disruption.

3. Disks specifically designed for media servers have thermal recalibration times near the low end of the range.

The work-ahead feature has been constrained in the Sun Media Scheduler to allow accesses only to the next disk in the sequence, because each additional step of work-ahead can increase the start-up time for new streams. Once a slot has finished using a disk and has passed it on, it is no longer possible for a new stream in that slot to access the disk. Therefore, the stream must be admitted to a slot later in the cycle and suffer an increased start-up delay. In addition, each step of work ahead creates a requirement for more memory buffers to hold the accessed data until it is needed.

Table 2 shows a timeline of work-ahead (W), read (R), and send (S) operations for one slot in a hypothetical system with twelve data disks, D0-D11 over 15 time periods t0-t14. The table shows that the slot sends the data that it read in time t1 from disk D0 out to the network in time period t2. Each slot will normally contain several active media streams. The order of reads for streams in the slot is determined by the elevator algorithm.

Disk	Time period															
	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14	
D0	W	R	S											W	R	S
D1			W	R	S										W	R
D2				W	R	S										W
D3					W	R	S									
D4						W	R	S								
D5							W	R	S							
D6								W	R	S						
D7									W	R	S					
D8										W	R	S				
D9											W	R	S			
D10	S											W	R	S		
D11	R	S												W	R	S

Legend:

- W - Work-ahead read
- R - Read disk
- S - Send over network

Table 2: Single Slot Timeline In Normal Operation

Table 2 represents the activity of only one slot in the scheduler. Other slots in the system follow a similar pattern, although each subsequent slot has its time line shifted one position to the right of its immediate predecessor. For example if Table 2 represents slot 0, then slot 1 will be assigned disk D0 in time period t2. Note that work-ahead is permitted only if the slot assigned the disk for reading has completed all I/O it requires from the disk. Note also that a given slot may be simultaneously accessing one disk as its assigned disk and another for work-ahead.

### 3.4 Buffering

The simplest method of buffer management would be to have two buffers per stream. One would play out while the other is filling from the disk. For work-ahead to be effective, a third buffer would be needed per stream.

A better but more complex method requires approximately one buffer per stream plus one buffer per disk at a minimum. In this case, the buffer is made up of a number of *chunks* that are individually managed and that are kept in a common pool until needed. At the beginning of every period, each active stream in the system has a full buffer of data (read in the previous period), and the pool contains enough free chunks so that the first data needed in that time period from each disk can be read. As the time period progresses, each stream will free up the chunks containing data that have been sent, and these are used for subsequent disk transfers<sup>4</sup>. In a worst-case disk access scenario, more chunks will be ready in time for subsequent disk transfers. In the case where the disk performance is better than expected, the disk process can wait until enough chunks are available. Again, for work-ahead to operate effectively, additional chunks, up to the equivalent of one additional buffer, should be available per stream.

The Sun Media Scheduler uses the second method to handle buffer fragmentation due to varying bit rates, and to provide additional flexibility to cover disk timing transients. It allocates enough buffer chunks to have

4. Extra chunks are required if the number of chunks per buffer is fewer than the number of streams per slot.

approximately 2.75 full buffers per stream, which creates approximately one full time period of slack time. Each chunk is 128 KBytes.

## 4 Disk Failure Recovery

A commercially viable media server must have a high reliability factor. It would be unacceptable to most users if the system were unavailable often or for extended periods. In addition, any reduction in capacity (e.g., due to “graceful degradation” in a failure case) will have the effect of causing total failure to some subset of clients. This section looks at the disk error recovery requirements and outlines the strategy for achieving them in the Sun Media Scheduler.

### 4.1 MTBF Calculation

Current generation disks have MTBF of 500000 or more hours. In operational mode (after infant mortality and before end-of-life) most components follow an exponential error distribution. If this is assumed for disks, we have:

$$Prob(X = x) = \begin{cases} \lambda \cdot e^{(-\lambda \cdot x)} & x > 0 \\ 0 & x \leq 0 \end{cases}$$

and:

$$Prob(X \leq x) = \begin{cases} 1 - e^{(-\lambda \cdot x)} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

and the mean (i.e., MTBF) is  $1/\lambda$  [4].

Therefore, a MTBF of 500000 makes  $\lambda = 1/500000$ .

The probability that a single disk will survive one full time period  $x$  is:

$$1 - (1 - e^{(-\lambda \cdot x)}) = e^{(-\lambda \cdot x)}$$

and the probability that all of  $n$  independent disks survive in time period  $x$  is:

$$(e^{(-\lambda \cdot x)})^n = e^{(n \cdot (-\lambda \cdot x))} = e^{(-n \cdot \lambda) \cdot x}$$

This corresponds to a second exponential distribution with MTBF of  $(n \cdot \lambda)$ .

So, for example, the MTBF of 60 disks is  $500000/60 = 8333$  hours or 347 days.

Note that the probability of 60 disks surviving one hour is the same as the probability of one disk surviving 60 hours. This is because the exponential distribution has no “memory”.

If it is assumed that repair takes less than eight hours, the chance of a second failure during that time is less than one tenth of one percent. Therefore, single point failure tolerance is sufficient for the class of media server addressed by this design.

## 4.2 Recovery Method

In order for a media server to be able to continue delivering data in the face of the failure of any single disk in the system, some redundant information must be available from which to reconstruct the data on the failed disk. Clearly, full mirroring, where all data is replicated on a redundant set of disks, can accomplish this with no additional buffer memory and with minimal impact on the media scheduler. However, mirroring doubles the cost of the disk subsystem, which is already a major component of the server’s cost.

The RAID technique of building a *parity* disk with the XOR (exclusive-OR) of all the data on the other disks can be used to reconstruct any failed disk. It does, however, require additional storage and can require up to twice as many disk accesses (i.e., cut usable bandwidth in half). In order to reconstruct a block from a failed disk, one must first read data from the parity disk and all other data disks involved in the parity set. Then either this information is retained until it is needed (requiring more memory), or it is re-read when needed (increasing disk accesses).

The worst case extra buffering requirement occurs when the failed disk is the first one accessed in each cycle. Then buffering for the entire cycle must be available because every disk must be read before the resulting data can be reconstructed and sent. Therefore the buffering requirement is  $(streams \cdot (disks + 1))$ . The extra one is needed to read in the next parity information while playing out the reconstructed data.

Using parity across all disks is not, in general, feasible because of the explosion of memory requirements and the processing power necessary to perform parity reconstruction over a large data set. However, if the total number of disks is divided into separate *clusters*, and the goal is to recover from a single disk failure in any cluster, the buffering requirements are reduced substantially.

For example, assume 30 disks and 150 streams using 256KB buffers, then 1.2GB of memory would be required ( $150 \cdot 31 \cdot 256KB$ ) if the parity data encompassed all disks. However, if five clusters of six disks each were separately managed using the techniques outlined earlier, then the additional buffering for dealing with a single cluster failure is 55MB (i.e.,  $30 \cdot 7 \cdot 256KB$ ). This is because streams only need additional buffers during the time they pass through the failed cluster. Afterwards, the extra buffers are available for the next streams as they pass through. In this case, there are at most 30 (i.e.,  $150/5$ ) streams in the failed cluster at any time.

In addition to the increased storage capacity, there needs to be additional processor power to perform the XOR operation. It is interesting to note that every byte in every stream in the failed cluster will be XOR’ed once. If we assume 4Mbit/sec streams, then  $(30 \cdot 4094304)/8$  bytes must be processed. This is under 16 MBytes/second and is easily performed by a single processor. Reducing the size of the cluster would make this even easier.

An important assumption of these calculations is that the parity data is kept on a per-stream basis instead of a per-disk-block basis. The bandwidth management scheme being used scatters the disk blocks of a stream around each disk (in zones). By structuring the parity on a per-stream basis, only data that is already needed for the stream is combined with the parity information to reconstruct the lost data. In essence, data from the non-failed disks in the parity set will be used both to reconstruct the failed data and as stream data on its own. In order to avoid worst-case bandwidth on parity reads, the parity data must be zoned similarly to the stream data. A simple way to do this is to layout the redundant blocks

in the same zone as the blocks of one of the data disks (e.g., the first disk of the cluster).

These calculations further assume that the failed disk is replaced and the data reconstructed off-line or with available bandwidth I/O. For example, the missing data could be read from tapes or optical juke-box.

Upon the failure of a disk, there would be a single “glitch” in all the affected streams while the system transitions from just-in-time buffer management to the pre-reading required for recovery. After that, the streams would play out at full rate. The media server is able to recover from any single disk failure without diminishing the active stream capacity. That is, the system is able to support as many users with one disk failure as with none. However, the server is not able to operate with two or more disk failures.

In the Sun Media Server, data for each title is striped across all the disks. However, the error recovery information is grouped into *redundancy sets* (the clusters described above). For example, if there were five disks in each redundancy set, four would be used as data and one used for parity on the four (i.e., 4 data + 1 parity). So, if there were 30 data disks, they would be organized with disks 0-3 containing data and disk 4 containing the parity for 0-3. The media data would continue on disks 5-8 with parity information for 5-8 on disk 9, and so on.

This is similar to, but subtly different from, traditional RAID techniques. Some RAID levels allow greater performance to be obtained during non-failure mode. However, since the server must be able to maintain the same number of streams during failure, it can't use the additional bandwidth available with those RAID schemes. In addition, intermixing media data and parity information on a disk would make the scheduler's slotting scheme much more problematic. Finally, the organization of parity data by title or file, even when the data blocks are spread around the disks, is a significant departure from traditional RAID techniques.

### 4.3 Scheduling

When a disk fails, its place in the schedule is effectively taken by the parity disk for its redundancy set. In addition, the parity information is organized by media stream using the same layout constraints as the media data. Therefore, the parity block read effectively is substituted for the stream's data block on the failed disk. As a result, the disk bandwidth requirements remain the same after the failure; only a substitution has occurred.

For each stream, recovering the data that was contained on the missing disk requires that the data from all remaining disks in the redundancy set be XOR'ed together with the parity data. Because the failed disk could be the first one in the redundancy set, all the other disks must be read first and the XOR performed before the data can be sent. This requires that these disks be read earlier than they would have been without failure<sup>5</sup>. Therefore, the scheduling during a failure is somewhat different than during normal operation. In addition, the number of buffers available in the system must be larger to hold the buffers participating in the XORs. These issues are addressed in a later section.

### 4.4 Media Reconstruction

It is assumed that the scheduler will not be responsible for reconstructing lost data and placing it back onto disk. External software will take care of rebuilding the data. There are several reasons for this:

- There may not be another disk available to recover onto until one is manually added.
- The scheduler won't, in general, know when all streams have been regenerated. In particular, titles that are not sent to any clients would never be reconstructed.
- It unnecessarily complicates the scheduler.

Instead, system management software is expected to use the Available-Bandwidth I/O facility (see Section 5 on page 11) to reconstruct the lost data without perturbing the active media

---

5. In non-failure mode, the scheduler reads disks in a just-in-time manner to minimize buffer requirements.

streams. A set of system management tools has been built for the Sun Media Server to perform these tasks [9].

#### 4.5 Example

Assume 12 data disks D0-D11, and three parity disks P0, P1, and P2 providing parity for disks D0-D3, D4-D7, and D8-D11, respectively. In normal mode, this would operate the same as in the example in Table 2, with the parity disks idle.

##### 4.5.1 Steady State With One Failed Disk

Now, assume a failure on disk D5, with its parity data on disk P1. First we look at the steady state, long after the failure occurred (shown in Table 2). Later, we'll examine the transient conditions that occur when the disk fails.

As a result of the failure, the server will substitute reads to disk P1 for reads to disk D5. In order to recover the data in time, we read disks D4-D7 (with P1 replacing D5) at the same time as disks D0-D3; D0-D3 are used for the normal just-in-time delivery and D4-D7 are used for recovery. Note that D4-D7 must be all in memory before the XOR can be done. We use a full time slot to do the XOR because there will be several streams in the slot and this gives sufficient time to process them all before the data is needed. Note that, at this point, there is no longer a 1-to-1 relationship between slots and disks. However, each disk is only performing work for one slot per time period. A slot will sometimes be using two disks per period and then immediately afterward the slot will use no disks for the same amount of time. In essence, we "borrow" the disk ahead in the train because the slot is not using it; when a slot that borrowed the disk gets to that spot it "lends" its disk to a slot behind in the train. So, it all works out as shown in Table 2.

Note that during time period t5 the recovered data for disk D5 is built (in place) in the parity buffer that had been read from disk P1. During time periods t6 through t8 the stream sends the recovered data along with the other buffers that were read previously, and requires no additional disk I/O. Beginning at time period t9 it begins normal operation on the next redundancy set. As illustrated in Table 2, the diagonal line of sends

(one per time period) is maintained. This ensures that there are no interruptions in the delivery of the data streams.

Disk	Time period														
	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14
D0	W	R	S										W	R	S
D1		W	R	S										W	R
D2			W	R	S										W
D3				W	R	S									
D4	W	R				⊕	S						W	R	S
D5								S'							
D6			W	R		⊕		S							W
D7				W	R	⊕			S						
D8								W	R	S					
D9									W	R	S				
D10	S									W	R	S			
D11	R	S									W	R	S		
P0															
P1		W	R			⊕		↑						W	R
P3															

**Legend:**

- W - Work-ahead read
- R - Read disk
- S - Send over network
- ⊕ - XOR for D5 data recovery
- S' - Send recovered data (from ↑)

Table 3: Timeline In Disk Failure Recovery Operation

##### 4.5.2 Failure Transients

There are several transient conditions that occur when a disk fails. Streams reading from the redundancy set immediately before the one containing the failure (disks D0-D4 in the example) at the time of the failure will drop all data from the failed set (disks D5-D7) for one pass through the set (because they didn't do a read-ahead but must lend their disks). However, from then on they will proceed without interruption. In the example, these streams

would lose 2 seconds of video (i.e., four disks with 1/2 second of video each).

Streams using the disks in the redundancy set containing the failed disk at the time the disk fails will drop the data from the failed disk only. Because the subsequent group of streams will drop all the data from the set containing the failure (see above), the streams in the failure set can continue to use the non-failed disks in the just-in-time manner. They will only lose the data from the failed disk, and only if they hadn't read it before the failure. In the example, these streams would lose at most 1/2 second of video.

All other streams will be unaffected by the failure.

Finally, the admission control algorithm must be modified to avoid placing new streams in slots associated with the redundancy set containing the failed disk or the one immediately before it. In this way, newly admitted streams will be unaffected by the failure.

#### 4.5.3 Buffer Requirements

Only the disks involved in error recovery use extra buffers. When the stream is outside the redundancy set containing the failure, they use just-in-time scheduling. The minimum buffer space required is:

$$(streams + data\_disks) + (streams / redundancy\_sets \cdot disks\_in\_redundancy\_set)$$

In other words, using the Sun Media Server's disk layout and scheduling scheme gives a buffer requirement in normal operation of  $(streams + data\_disks)$ . The extra buffering needed to handle failure recovery is calculated as  $(streams / redundancy\_sets \cdot disks\_in\_redundancy\_set)$ .

For example, assume 150 streams, 30 disks, 256KB buffers, and five disks in each redundancy set (4 + 1 parity). Then the minimum total buffer space for a system with cluster redundancy is  $((150 + 24) + (150/6 \cdot 5)) \cdot 256K = 79$  MBytes. However, this amount of buffering would not be sufficient to permit the work-ahead feature to operate.

Reducing the cluster size would improve the memory requirements. However, it would require more disks to support a given configuration, or would reduce the maximum number of titles that a given group of disks could hold.

## 5 Available-Bandwidth I/O

---

The scheduler supports commands to allow applications to read and write the managed disks within the constraints of the media schedule. These operations are performed as soon as possible without perturbing the active media streams. As a result, the scheduler will use any excess capacity of the system to perform the reads and writes as quickly as possible.

With this facility, external software can perform media reconstruction after disk failures in the most efficient manner for all titles, and then notify the scheduler that the recovery is complete. Note that this may often be accomplished more quickly than if the scheduler itself were doing it because all of the excess bandwidth can be applied at once. The same mechanism can also be used to add new titles to the system during normal operation. To ensure timely completion of these operations, it is possible to create a desired amount of excess bandwidth in the system by limiting stream admissions.

The available-bandwidth I/O mechanism works by associating a list of outstanding requests with each disk. When I/O is completed for a slot, any disk bandwidth available to the slot that is currently unneeded by it will be used to service requests on the list. After this has completed, the normal work-ahead processing begins for the disk.

## 6 Analysis

---

### 6.1 Seek Time

Given a set of N ordered seeks and an algorithm that would seek to the nearest end of the set then proceed through the set to the nearest subsequent position (an elevator algorithm), the worst case

total seek distance for the set would be (as a fraction of the disk):

$$\text{total seek distance} \leq \frac{1}{2} + (N-1) \cdot \frac{1}{N-1} \leq 1.5$$

**case 1:**  $N = 1$ ; The worst case seek is the whole disk.

total seek  $\leq 1$

**case 2:**  $N > 1$ ; The starting disk position is in the middle of the group. Then one seek of at most one half of the disk is used to get to the nearest end of the group from the current head position. After that, there are  $(N-1)$  seeks of average  $\leq 1/(N-1)$  of the disk. That is, the worst case occurs when the starting position is in the middle of the disk and the set spans the disk.

total seek  $\leq 1/2 + (N-1) \cdot 1/(N-1) = 1.5$

**case 3:**  $N > 1$ ; The starting disk position is outside the group. Then there are  $N$  seeks of no more than  $1/N$  of the disk average.

total seek  $\leq N \cdot (1/N) = 1$

Therefore, if  $seek(x)$  is the time to seek fraction  $x$  of the disk, then for the set of  $N$  accesses:

$$\text{total seek time} \leq N \cdot seek\left(\frac{1.5}{N}\right)$$

$$\text{average seek time} \leq seek\left(\frac{1.5}{N}\right)$$

If the number of streams in each slot,  $N$ , is greater than or equal to five ( $N \geq 5$ ), then over each slot the average seek time is guaranteed to be less than the time to seek across the disk's average seek distance. That is,  $seek(1.5/N) < seek(1/3)$ .

Finally, in steady state operation of a media server which uses a two-way elevator algorithm as described above, it can be seen that, on average, each time period will experience no more than one full seek of each disk. Let  $L(i)$  be the lowest block number needed from the disk in time period  $t_i$ , and  $H(i)$  be the highest block number needed from the disk in time period  $t_i$ . Then, for any sequence of time periods  $t_n, t_{n+1}, t_{n+2}, \dots$ , and assuming (without loss of generality) that at the beginning of  $t_n$  the disk head is closer to  $L(n)$  than  $H(n)$ , that the disk will seek no

further than to the extreme points shown by the following pattern:

$$\begin{aligned} &L(n) \\ &max(H(n), H(n+1)) \\ &min(L(n+1), L(n+2)) \\ &max(H(n+2), H(n+3)) \\ &\dots \end{aligned}$$

So, in steady state operation, each disk will perform, on average, no more than one full seek each time period. This means that the average seek time will be less than  $seek(1/N)$ , while the instantaneous worst-case is bounded at  $(1.5/N)$ . Therefore, a server using this method and serving a moderate load of clients will achieve the average case seek of the disks.

## 6.2 Rotational Latency

The worst-case rotational latency for a disk is the time it takes to make one revolution of the disk. Modern high-performance disks rotate at 7200 RPM, so the worst-case latency is approximately 8.3 milliseconds. The average-case latency (assuming random accesses) is one half the worst-case latency, or approximately 4.2 milliseconds. Because the locations accessed in a media server are not correlated (they represent independent streams), average-case latencies can be expected and the worst-case is unlikely to persist across several accesses. Because of this and the fact that the difference between worst-case and average-case rotational latencies are small, it is convenient to treat the average rotational latency as another component of the seek overhead in server capacity calculations (e.g., see Drawing 1 on page 14).

## 6.3 Bandwidth

Each slot contains streams using roughly equal bandwidth from each zone. For example, if all streams use the same bandwidth  $b$ , each slot will contain approximately  $N = B/(b \cdot D)$  streams. For any given slot, the number of streams in the worst case zone is  $\lceil N/Z \rceil$ , and there are at least  $\lfloor N/Z \rfloor$  in the best zone. (If there are more than two zones, the others will also contain at least  $\lfloor N/Z \rfloor$  streams.) So given the worst case bandwidth for transfers from each zone, the worst case bandwidth for each slot can be calculated. This is better than the disk's worst

case and, with typical modern disks, approaches the average case for the disk.

In addition, because the layout of the disks alternates between zones on each data file, each stream will have traversed each zone in every  $Z$  accesses. Therefore, each stream will achieve approximately average case disk bandwidth internally in every  $Z$  time periods (except, perhaps from the last to first disk; however, even this effect cancels over a longer interval). As a result, the server achieves an essentially complete balance between the zones every  $Z$  time periods, assuring the average-case disk bandwidth overall.

Similarly, the admission algorithm ensures, over each set of adjacent slots, that the bandwidth utilization is balanced by zone. Such a set of slots are called a *slot group*. Across the set of slots in each slot group, the bandwidth used by streams in any zone will not exceed the sum of the bandwidths of the slots divided by the number of zones. Then, as the slot group passes over each disk any imbalance in zone usage for the disk is cancelled within the number of time periods equal to the size of the largest slot group. Therefore, each disk will achieve approximately average case bandwidth internally across each slot group.

#### 6.4 Admission

A stream can be assigned to any slot; however it must not transfer until the first data block it needs is on the disk assigned to its slot and the block is in the proper zone. Therefore the worst case time to obtain the first block of data is  $D \cdot Z \cdot t$ , where  $t$  is one scheduling period.

The start-up delay that a stream will experience if admitted to a given slot can be calculated given the current disk assigned to the slot and knowledge of the transitions from disk-to-disk and zone-to-zone. Using this calculation method, the admission control subsystem allows each admission to be constrained by a maximum start-up delay. If the stream cannot be admitted into a slot which can provide the first block within the delay constraint, the admission can be rejected. This can be used to allow the client to find an alternate server that will supply the data in time.

Finally, although the admission process has been described as *first come first served* (FCFS), there are other reasonable policies. FCFS is appropriate where all clients are of approximately equal importance, as when the server is used to deliver movies into a large number of homes. However, if importance values can be assigned to clients, the admission algorithm can use them to maximize the overall value delivered. For example, an appropriate policy could be to preempt a running stream of a lower importance in order to admit a later one with a higher importance.

## 7 Implementation

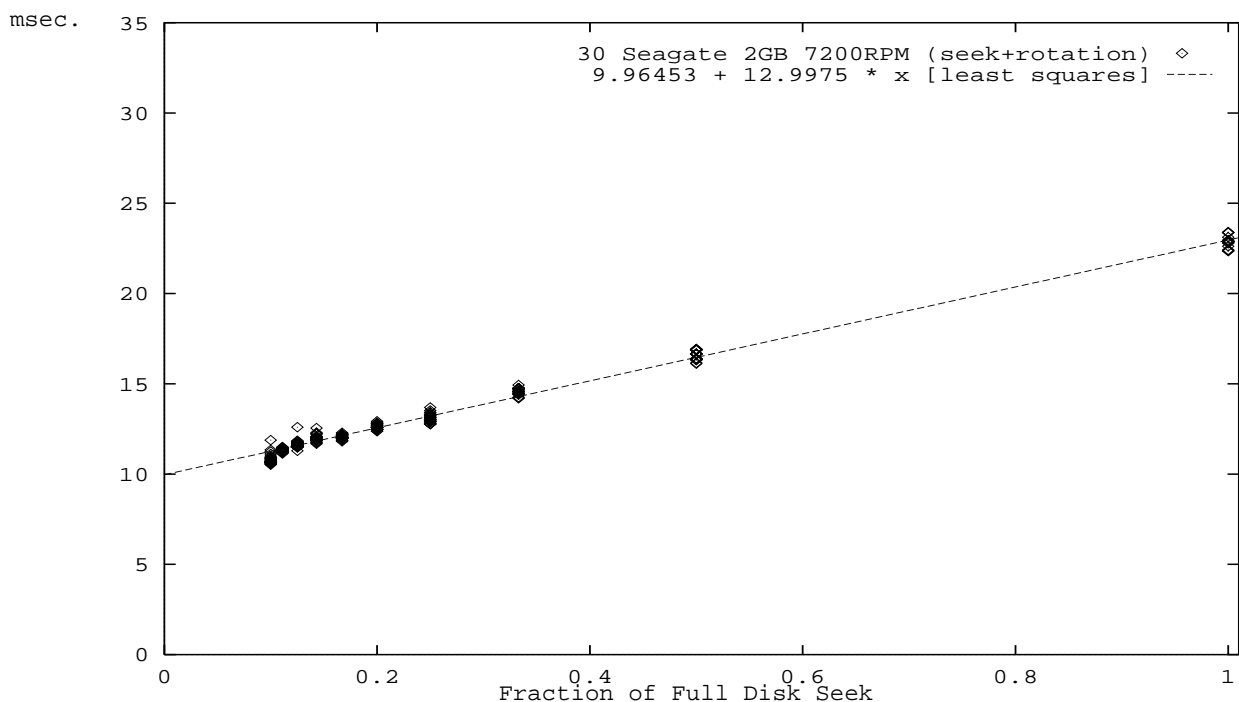
---

The Sun Media Scheduler has been implemented as a Solaris device driver which interacts with the Data Pump (also a device driver). The system uses Solaris' real-time scheduling class to ensure that disk scheduling decisions are made in a timely manner. The scheduling and admission subsystems work from a configuration specification provided by a user-mode configuration utility. The actual parameters are derived from the Media File System [9] specification, augmented by a configuration file in the form of a Tcl [7] script. The configuration parameters include:

- the scheduling interval
- the number of data and parity disks
- disk device identification (major and minor device numbers)
- the access order of the data disks
- the zone transition order
- seek time information for the disks
- the worst-case bandwidth of each zone
- total network bandwidth
- the amount of buffering to use
- site-imposed limits on the number of streams or total bandwidth

Because of this facility, the software can be configured for any size system without changing the scheduling and admission module.

Although the design of the Sun Media Scheduler allows stream delivery guarantees to be made, it



**DRAWING 1.** Measured Seek and Rotational Latency Times

is not necessary to operate a server in this mode. The disk bandwidth values used in stream admission are conservative bounds based on the average of the worst case bandwidth in each zone. However, in practice the actual observed bandwidth will be essentially equal to the disks' overall average. In addition, the system assumes one worst-case seek per scheduling interval on each disk, even though the actual seek will nearly always be less than this. So, if absolute server capacity is very important and if clients can accept occasional times where data delivery is a little late, then the server can be configured to admit streams beyond the guaranteed capacity. This is accomplished in the Sun Media Scheduler by setting an *overbooking* configuration parameter to an amount appropriate to the installation.

The first product to use the Sun Media Server software is the SMS-1000. This server is based on the SPARCcenter 1000 computer and a modified SPARCstorage Array. The SMS-1000 is rated to deliver a sustained 400 Mbits/second, serving individual streams of 1-8 Mbits/second. It can survive the failure of any single disk while maintaining the full delivery rate.

The SPARCcenter 1000 in the SMS-1000 is configured to have two processors, six SCSI controllers, four Sun SAHI ATM interface cards (155 Mbit/second OC-3), and at least 128 MBytes of memory. Approximately 100 MBytes of memory is used as buffers by the Sun Media Scheduler.

The SPARCstorage Array contains up to 30 disks, each running at 7200 RPM and holding 2 GBytes of data. It has been modified to remove the FibreChannel controller, and instead bring six differential SCSI chains directly to the SCSI controllers in the SPARCcenter 1000. This was necessary because the FibreChannel controller could not supply the full bandwidth of the disks.

The disks are organized into six redundancy sets containing four data disks and one parity disk. Each redundancy set occupies one SCSI chain. This ensures that the load on each SCSI chain remains balanced even in the event of a disk failure.

Drawing 1 shows a plot of the combined seek time and average rotational latency (in milliseconds) measured over a large number of trials with 30 identical disks available in the SMS-

1000. As can be observed from the drawing, the assumption of a fixed overhead and a distance-related seek time is valid for these disks.

The transfer bandwidth for these disks ranged from 4.9 MBytes/second at the outer rim to 4.5 MBytes/second near the inner edge. The system can be configured with two zones with worst-case bandwidth of 4.7 MBytes/second and 4.5 MBytes/second. Because of the roughly even response of these disks, the bandwidth management scheme provides a limited gain.

By contrast, the standard Sun 1 GByte 5400 RPM disks range from 4.2 MBytes/second at the outer rim to 2.4 MBytes/second at the inner edge. A server configured with these disks would use two zones of 3.5 MBytes/second and 2.4 MBytes/second. Here the bandwidth difference between the zones is 45%.

In the SMS-1000, the set of disks on each SCSI chain overloads the SCSI controller in simultaneous accesses. That is, when all 24 data disks are active at the same time, the effective bandwidth drops due to SCSI contention. The result is that each disk can provide approximately 3.0 MBytes/second of bandwidth.

The SMS-1000 is usually configured to use 1/2 second scheduling intervals. Typical titles are recorded at 4 Mbits/second. Therefore, the buffer for each such stream in a time interval is approximately 250 KBytes. As a result, the 7200 RPM disks can support approximately 5.2 of the 4 Mbit/second streams each time period using simultaneous accesses. With 4 Mbit streams, each disk spends approximately 86% of its time transferring data, and approximately 14% of its time seeking to the next location. If higher bandwidth titles are used, a larger percentage of the time is spent in transfers.

In the SPARCcenter 1000, processors access memory across the high-bandwidth multiprocessor system bus (the XDBus™) through direct-mapped caches. This design, combined with the higher latencies associated with this type of bus, can lead to *cache thrashing* on operations like parity reconstruction. In order to control the system bandwidth required for disk failure recovery, an optimized XOR

calculation facility was developed. This feature uses cache blocking to ensure that the memory bandwidth is used most effectively. One (64 byte) cache block for each buffer participating in the error recovery is read in turn into the processor and accumulated into the calculation. When this is done, the resulting cache block is written to the buffer for the recovered data. The use of this facility ensures that the absolute minimum system bus bandwidth is consumed.

In normal mode, the Sun Media Scheduler and Data Pump consume approximately 15-25% of one 60 MHz SuperSPARC processor when serving a full load of 400 Mbits/second (with debugging checks turned on). Disk failure recovery (XOR) processing further consumes 40-50% of one processor at full load. Therefore, a fully loaded SMS-1000 will not fully occupy one processor even in disk failure recovery mode.

The basic structure of the SMS-1000 is shown in Figure 2. The ATM network is used to connect to set-top boxes, desktop workstations, or a combination of both. The network provides point-to-point connections or creates the spanning trees necessary to support broadcast and multicast transmissions. Future versions of the Sun Media Server will support other delivery mechanisms, including Fast Ethernet.

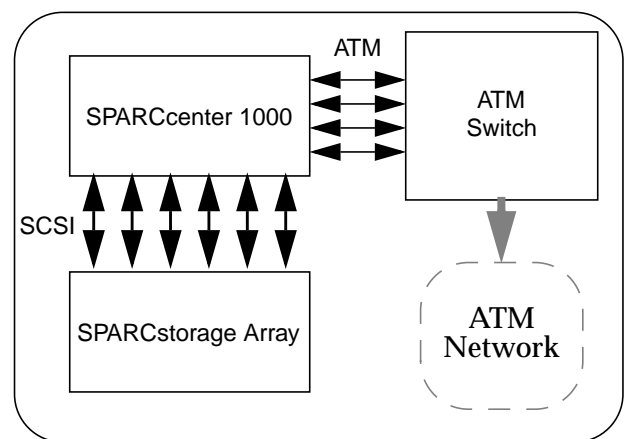


FIGURE 2. SMS-1000

The SMS-1000 has demonstrated sustained, reliable delivery of over 400 Mbits/second in both normal operation and during disk failure recovery mode. Therefore, if 4 Mbits/second

MPEG-2 streams are used, the server can easily support up to 100 users indefinitely.

Additional applications of the Sun Media Server software are planned based on the SPARCstation 20 platform with 1GByte disks for small-scale systems, and the SPARCcenter 2000 for large-scale systems. The SPARCcenter 2000 version can be configured with up to 60 disks, with each disk holding 9 GBytes of data and delivering approximately 4.6 MBytes/second.

## 8 Simulator

---

A discrete event simulator was developed for the Sun Media Scheduler as an aid for the debugging and analysis of the scheduling and admission control processes. The simulator framework provides accurate behavioral models of the disks, the network, the Data Pump, and the kernel synchronization services that the scheduler uses.

The design of the simulator allows approximately 90% of the actual Sun Media Scheduler code to be run in the simulation environment. This is accomplished by imitating the programming interface of the various subsystems used by the scheduler.

Facilities have been included in the framework to allow creation of scenarios involving the admission and shutdown of multiple clients, transient disk slowdowns, and disk failures and restorations. The simulator produces a log file based on trace-points that can be used to verify the correctness of the run. These trace-points can also be monitored in a live Sun Media Server using the Solaris kernel trace facility (vtrace).

The simulator has been especially useful in debugging the Sun Media Scheduler because the kernel debugging environment is rather unforgiving. Some errors in kernel code can cause a machine to seize up, with no additional information available. In contrast, the same code running on the simulator can be monitored with the standard user-mode source level debuggers.

The simulator is also useful for characterizing the capacity of hypothetical configurations. By giving the disk characteristics and the bandwidth

of the titles for a system, an accurate estimate of the number of supported streams can be derived. In addition, it can be used to determine the amount of buffer memory that should be included in a system to provide the desired level of protection from transient disk errors.

Finally, the simulator has been useful in ensuring the reliability of the Sun Media Scheduler software. Because the simulator runs at approximately ten times real time, it was possible to log over one year of run time using randomly-generated scenarios of various client loads, disk problems, etc. before the alpha shipment of the SMS-1000 software.

## 9 Conclusions

---

The design presented in this paper and used in the Sun Media Server can guarantee approximately average case bandwidth and seek times from a set of disks and deliver data at that rate to a set of clients. It is able to do so in the face of substantial non-determinism in the response of the disks themselves and even in the event of complete failure of any single disk. It achieves this with only a moderate investment in both memory and processor resources.

## 10 References

---

- [1] J. Gemmell et al., "Multimedia Storage Servers: A Tutorial," *IEEE Computer*, May 1995, pp. 40-49.
- [2] J. Hanko et al., "Workstation Support for Time-Critical Applications", *Proceedings of the Second International Workshop on Network and Operating Systems Support for Digital Audio and Video*, November, 1991.
- [3] J. Hennessy and D. Patterson, *Computer Architecture A Quantitative Approach*, Morgan Kaufman Publishers, 1990, pp. 557.
- [4] R. Khazanie *Basic Probability Theory and Applications*, Good-year Publishing, 1976.
- [5] K. Mandal, "Data Pump for Video On Demand Applications," Sun internal document, 1994.
- [6] J. D. Northcutt et al., "A High Resolution Workstation," *Signal Processing: Image Communications*, Elsevier, August 1992, pp. 445-455.
- [7] J. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley Publishing Co., 1994
- [8] F. Tobaji et al., "Streaming RAID™ - A Disk Array Management system For Video Files", Starlight Networks Inc., 1993.

- [9] J. Voll, J. Hanko. and K. Mandal, "Video-on-Demand Server Software", *Proceedings of the First SunSoft Technical Conference*, April 1995, pp. 179-192.
- [10] G. Wall et al., "Bus Bandwidth Management in a High Resolution Video Workstation", *Proceedings of the Third International Workshop on Network and Operating Systems Support for Digital Audio and Video*, November, 1992.

## 11 Glossary

---

**admission policy** - An algorithm for deciding whether to admit a new stream into the server and, if so, how it is to be managed.

**chunk** - An area of memory for holding media data that is separately allocated and freed.

**client** - The receiver of a stream of media data.

**Data Pump** - A kernel module that provides optimized zero-copy access to disk and network devices, as well as rate-regulated network delivery.

**disk zone** - An area of a disk with a particular transfer bandwidth.

**extent** - A contiguous group of disk blocks holding stream data for one scheduling interval.

**media stream** - A continuous stream of time-critical data.

**parity** - A value used in error correction or detection that is based on the bit-wise exclusive-or of data bits.

**redundancy set** - A set of disks for which redundant information is kept that enables data recovery after a disk failure.

**rotational latency** - The delay from the time a disk's read/write head arrives on the requested track until the desired location rotates into position under the head and can be accessed.

**scheduling interval** - A site-selected fundamental time period in which a disk read is completed for all streams.

**seek** - The process of moving a disk's read/write head to the track containing a specific location

**slot** - A group of streams that access the same disk(s) at any point in time.

**slot group** - A group of slots over which bandwidth used from all zones is balanced.

**stream** - The active process of reading data from the disks and delivering it to a client over a network.

**thermal calibration** - A process performed by some disk drives to compensate for apparent disk geometry changes caused by thermal effects.

**work-ahead** - A mechanism that allows the media server to begin work early for a subsequent scheduling interval when work for the previous one completes.

## 12 Acknowledgments

---

The zoned bandwidth management and disk layout techniques evolved out of a discussion with Steve Kleiman. The disk error recovery method similarly evolved from discussions with Jerry Wall. The Data Pump was designed and implemented by Kallol Mandal, who also assisted in hooking the Sun Media Scheduler into the device driver framework. Jim Voll developed the Media File System (MFS) tools to create, delete, rebuild, and manipulate titles, as well as the libraries for accessing the server facilities. Chris Moeller and Mike DeMoney created a video-on-demand system with menus and full VCR control using the SMS-1000 and Open TV™ set-top boxes. Kenneth Wong developed an external daemon program to monitor the disks and provide an early indication of disk problems.