Scheduling for Integrated Service in Multimedia Systems

A. L. Narasimha Reddy
Ravi Wijayaratne
Texas A & M University
214 Zachry
College Station, TX 77843-3128
reddy@ee.tamu.edu

Abstract

The recent dramatic advances in computer and communications technology has led to a demand to support multimedia data (digitized video and audio) across data networks. Storage, processor, memory and network subsystems have to be appropriately scheduled to meet the requirements imposed by the continuous media. Proposed techniques for scheduling resources in a multimedia system are discussed. Integrated service for different levels of quality-of-service is studied with an emphasis on the disk subsystem.

1 Introduction

System level support of continuous media has been receiving wide attention. Continuous media impose timing requirements on the retrieval and delivery of data unlike traditional data such as text and images. Timely retrieval and delivery of data requires that the system and network pay attention to notions of time and deadlines. Data retrieval is handled by the I/O system (File system, disk drivers, disks etc.) and the delivery is handled by the network system (network software and the network). Without appropriate CPU scheduling, the requests may experience delays in interrupt processing and scheduling of other resources in the system.

Different levels of service can be provided for continuous media. *Deterministic service* provides guarantees that the required data will be retrieved in time. *Statistical service* provides statistical

guarantees about data retrieval, e.g., 99% of the requested blocks will be retrieved in time. Data streams can be classified as Constant Bit Rate (CBR) or Variable Bit Rate (VBR) depending on whether the stream requests the same amount of data in an interval.

System requirements of a multimedia stream can be characterized by a vector of the CPU load, the disk load and the network load of that stream. Each of the resources, CPU, disk and network have to be appropriately scheduled to meet the performance guarantees of the stream. A video-on-demand stream may require minimal CPU processing while it imposes large load on the disk and the network. A video capture stream may require large CPU power for compressing the video as it is being captured while requiring a little disk bandwidth and no network bandwidth. An uncompressed stream, such as in production quality TV broadcasting, may stress the disk and network bandwidth. Each subsystem has to deal with the allocation and scheduling of available resources to meet the performance requirements of individual streams.

Providing deterministic service at the disk is complicated by the random service time costs involved in disk transfers (because of the random seek and latency overheads). This problem has been addressed effectively by suitable disk scheduling policies [1, 2]. These scheduling policies group a number of requests into rounds or batches and service the requests in a round using a disk seek optimizing policy such as SCAN. Then the service time for the entire round can be bounded to provide guarantees. This strategy works well with CBR streams. However, with VBR streams, the workload changes from round to round and hence such an approach will have to consider the variations in load for providing guarantees.

Providing deterministic guarantees for data delivery has received considerable attention in the networking community [3, 4, 5]. A VBR stream is characterized by its worst-case demand on the network over any period of time. This characterization of the stream's behavior is used to calculate the worst-case delays possible through a switch when multiple streams pass through a switch. Network infrastructure has to be modified so as to enable an application to request and reserve resources [6] for these guarantees to be provided. Protocols have been developed to support best-effort delivery of multimedia data over existing networks [7, 8]. These protocols adapt to the availability of network bandwidth to deliver as high quality data as possible on existing network infrastructure.

The multimedia streams require processor for servicing the interrupts and scheduling of other resources in the system. Without appropriate scheduling, the requests may experience long delays in queues defeating the scheduling in other resources. CPU processing may be required for compressing or decompressing a stream or for network protocol processing for delivery purposes. Strict real-time scheduling of resources or priority-based schemes may not be quite suited for supporting the

various levels of support required by the multimedia data. Lottery scheduling [9] and hierarchical scheduling [10] have recently been proposed as flexible algorithms for scheduling CPU resources for multimedia.

In this chapter, we will look at the disk scheduling problem in detail and give a brief overview of work done in processor and network scheduling. Continuous media applications may require deterministic performance guarantees i.e., guarantee that a requested block will be available within a specified amount of time continuously during the application's execution. A request from an interactive game or a request to change the sequence of frames in a continuous media application may require that the request have low response time i.e., may require a latency guarantee. A regular file request may only require best-effort service but may require that a certain number of requests be served in a given time i.e., may require a throughput guarantee. It may be desirable to provide both deterministic service and statistical service to VBR streams in the same system. Deterministic service may be too expensive on the system's resources. A user may request for statistical service when a request for deterministic service may be denied due to lack of resources. There is a clear need for supporting multiple levels of performance guarantees within the system.

Several interesting questions need to be addressed when multiple levels of QOS need to be supported in the same system: (a) how to allocate and balance resources for the different QOS levels, (b) how to control and limit the usage of resources to allocated levels, (c) how to schedule different requests to meet the desired performance goals, (d) how do system level parameters and design decisions affect the different types of requests?

Section 2 discusses an approach for providing different levels of QOS in a single system. Section 3 discusses a scheduling algorithm for providing integrated service at the disk. Section 4 discusses CPU scheduling algorithms and section 5 looks at approaches for network scheduling. Since it is difficult to discuss all aspects of scheduling in a short chapter, more emphasis is put on disk scheduling and a brief overview of CPU scheduling and network scheduling policies is provided.

2 Multiple levels of QOS

Three different categories of requests are considered here. *Periodic* requests require service at regular intervals of time. Periodic requests model the behavior of video playback where data is retrieved at regular intervals of time. Periodic requests can be either CBR or VBR. *Interactive* requests require quick response from the I/O system. Interactive requests can be used to model

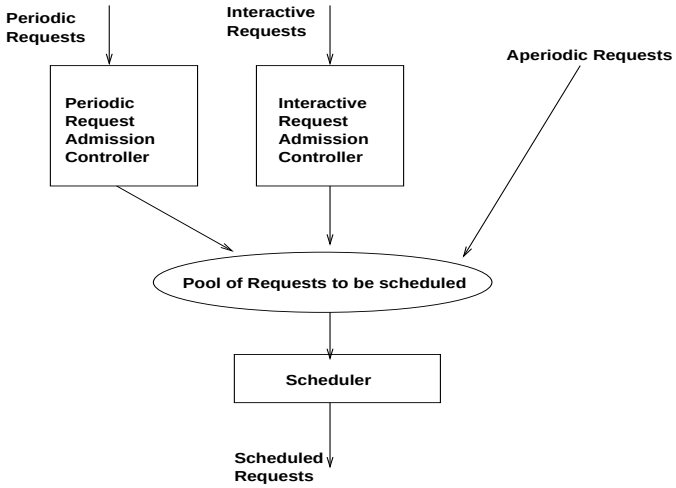

Fig. 1. Supporting multiple QOS levels.

the behavior of change-of-sequence requests in an interactive video playback application or the requests in an interactive video game. These requests arrive at irregular intervals of time. *Aperiodic* requests are regular file requests. The following performance goals are considered: (a) deterministic or statistical guarantees for periodic requests, (b) low response times for interactive requests and (c) guaranteed minimum level of service or bandwidth guarantees for aperiodic requests.

Integrated service is provided by allocating bandwidth to classes of requests and then appropriately scheduling the requests at the disk. Bandwidth allocation and rate controls are used to ensure that a class of applications does not interfere with the performance objectives of other classes of applications. Disk bandwidth is allocated appropriately among the three different types of requests. Periodic requests and interactive requests employ admission controllers to limit the disk utilization of these requests to the allocated level for these requests as shown in Fig. 1. To provide throughput guarantees for aperiodic requests, we limit the allocated bandwidth for periodic and interactive requests ($< 100\%$) through admission control. Aperiodic requests utilize the remaining disk bandwidth. This guarantees that the aperiodic requests do not get starved of service and receive some minimum level of service.

The admission controllers employed for periodic requests and interactive requests depend on the service provided for these requests. In the next section, we discuss how to provide deterministic service for periodic requests. The proposed approach can be modified to implement statistical

guarantees for periodic requests as well. Interactive requests are treated as high-priority aperiodic requests. The scheduler and the admission controller are designed to provide low response times for these requests. A leaky-bucket controller is used for interactive requests. A leaky bucket controller controls the burstiness of interactive requests (by allowing only a specified number of requests in a given window of time) and thus limits the impact it may have on other requests. Similar bandwidth allocation approaches have been adopted in CPU scheduling [10] and network scheduling [3]. However, the scheduling policies employed in each subsystem are different. The scheduling policies exploit the different characteristics of the resources being scheduled.

3 Disk Scheduling

3.1 Deterministic guarantees for VBR streams

Providing deterministic service for VBR streams is complicated by the following factors: (i) the load of a stream on the system varies from one round to the next, (ii) scheduling the first block doesn't guarantee that the following blocks of the stream can be scheduled. To ensure that all the blocks required by a stream can be retrieved, the peak rate of a stream can be computed and enough disk bandwidth can be reserved to satisfy the peak requirements of the stream. However, this approach results in considerable overestimate of required resources and results in supporting fewer streams.

Resource allocation based on the peak demands of the stream will underutilize the disk bandwidth since the peak demand is observed only for short durations compared to the length of the duration of the stream. However, when many streams are served in the system, the peaks do not necessarily overlap with each other and it may be possible to serve more steams than what is allowed by the peak-rate allocation. Can this statistical multiplexing be exploited to increase the deterministic service provided by the system? An approach is presented here that allows the system to exploit statistical multiplexing while providing deterministic service.

Disk service is broken into fixed size time units called rounds or batches. Each round may span 0.25-1 seconds of time ([1]). An application requiring service for a VBR stream supplies the I/O system with a trace of its I/O demand. This data could be based on frame rate i.e., given on a frame to frame basis or could be more closely tied to the I/O system. Specifying the load on a frame basis is more flexible and the application doesn't have to be aware of how the I/O system is organized (block size or round size). If the I/O system's block size is known and the duration

of each round is known, then the trace can be compacted by specifying the I/O load on a round by round basis in terms of the blocks. For example, a frame by frame trace may look like 83,888, 9,960, 10,008, 27,044, ...which indicates the number of bits of data needed to display each frame. If the round size is say 2 frames i.e., 1/12th of a second, and the I/O system uses a block size of 4KB, then the compacted trace would have $\lceil (83888 + 9960)/(4 * 1024 * 8) \rceil = 3$ in the first entry. The second entry would have $\lceil ((10008 + 27044) - (3 * 1024 * 8 - 83888 - 9960))/(4 * 1024 * 8) \rceil = 1$ block. Hence, the equivalent compacted trace for the stream would be 3, 1, ... A 40,000 frame trace of the movie "Silence of the Lambs" (24 frames/second) requires 203,285 bytes on a frame by frame basis compared to a 3,333 byte description of the same movie when compacted with the knowledge of the round size of 0.5 seconds and a block size of 8KB. It is assumed that this information, the *demand trace*, is available to the I/O system in either description.

The I/O system itself keeps track of the worst-case time committed for service in each round at each of its disks in the form of a *load trace*. Before a stream is admitted, its demand trace is combined with the load trace of the appropriate disks to see if the load on any one of the disks exceeds the capacity (committed time greater than the length of the round). The load trace of a system consists of load traces of all the disks over sufficient period of time. This requires the knowledge of the placement of blocks of the requesting stream. This information can be obtained from the storage volume manager.

A stream is admitted if its demand can be accommodated by the system. It is possible that the stream cannot be supported in the round the request arrives. The *stream scheduling* policy will look for a round in which this stream can be scheduled. We will assume that a stream will wait for a maximum amount of time, given by *latency target*, for admittance. (The impact of *stream scheduling* policies has been studied in [11].) Let *load[i][j]* denote the load on disk i in round j. Let the demand of a stream be given by $demand[j]$ indicating the number of blocks to be retrieved by that stream in round j. Then, a stream can be admitted if there exists a k such that $load[i][j+k] + serv\_time(demand[j]) \leq round\ time$, for all j, where i = disk storing data for round j, and k is the startup latency $\leq latency\ target$. If multiple disks may store the data required by a stream in a round, the above check needs to be appropriately modified to verify that these disks can support the retrieval of needed data. The function $serv\_time()$ estimates the worst-case service time required for retrieving a given number of blocks from a disk given the current load of the disk. This function utilizes the current load of the disk (number of requests and blocks) and the load of the arriving request to estimate the worst-case time required to serve the new request along with the already scheduled requests. A similar check can be applied against buffer resources when needed.

Data layout plays a significant role on the performance of disk access. It has been suggested

by many researchers [12, 13, 14] that video data should be striped [15] across the disks for load balancing and to improve throughput available for a single data stream. Data for a VBR stream can be stored in (i) Constant Data Length (CDL) units or (ii) Constant Time Length (CTL) units. In CDL, data is distributed in some fixed size units across the disks, say 64KB blocks. In BCTL (block-constrained CTL, a variant of CTL considered here), data is distributed in some constant time units, say 0.5 seconds of display time rounded to the closest block (4KB). In CDL, a stream may require service from several disks in a round and in BCTL, a stream will require service from at most one disk.

Fig. 2 shows the impact of block size and data layout strategy on the stream throughput. The workload consisted of an equal number of 27-minute streams of a cartoon *Asterix*, two movies *Silence of the lambs* and *Terminator* and a news clip *News* [16]. It is observed that peak rate based allocation leads to significantly less throughput than the proposed approach. The proposed approach achieves 130% -195% more stream throughput than the peak rate allocation. This improvement is primarily achieved by exploiting the statistical multiplexing of different streams. When requests arrive at the same time, flexible starting times (through latency targets) allow the peaks in demand to be spread over time to improve the throughput.

As the block size is increased a stream fetches less number of blocks in a round and hence CDL tends to be more efficient at larger block sizes (due to smaller seek and rotational latency costs). The stream throughput for CDL improves significantly for all the data streams as the block size is increased from 32 KB to 256 KB. The stream throughput drops slowly for BCTL as the block size is increased. This is due to effects of larger quantization of service allocation for a request. For the block sizes considered, the BCTL data layout performs better than the CDL data layout. CDL performs worse than BCTL due to paying a rotational latency penalty for each block compared to each time slice in BCTL.

Fig. 3. shows the impact of statistical guarantees on stream throughput. Instead of requiring that every block of data be retrieved in time, a fraction of the blocks for each stream were allowed to miss deadlines or not be provided service. This fraction is varied among 0.1%, 0.2%, 0.5%, 1.0%, 2.0% and 5.0% at various latency targets. It is observed that as more blocks are allowed to be dropped, it is possible to achieve more throughput compared to deterministic guarantees. Stream throughput can be improved by up to 20% by allowing 5% of the blocks to be dropped. Dropping blocks is more effective at lower latency targets than at higher latency targets. For example, dropping up to 2% of the blocks improves the stream throughput by 14.5% at a latency target of 100 rounds compared to an improvement of 6% at a latency target of 1000 rounds. Stream throughput can also be improved by relaxing the latency targets.

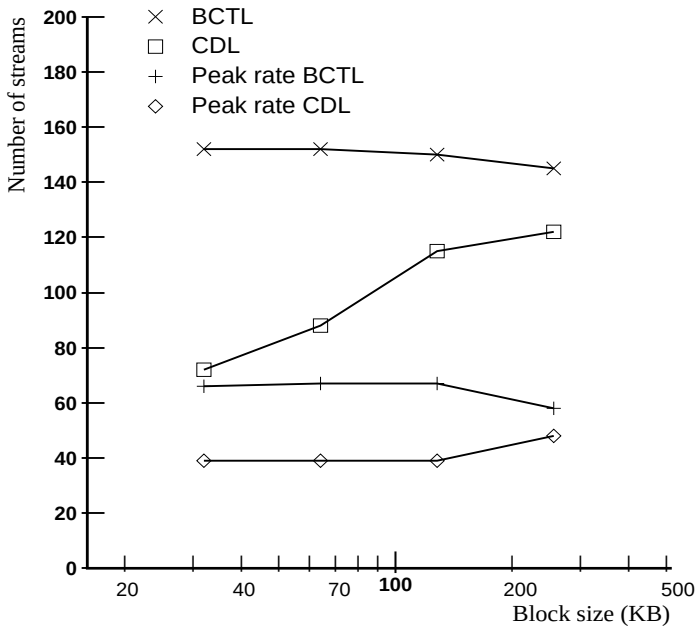

Fig. 2. Impact of data layout and block size on VBR streams.

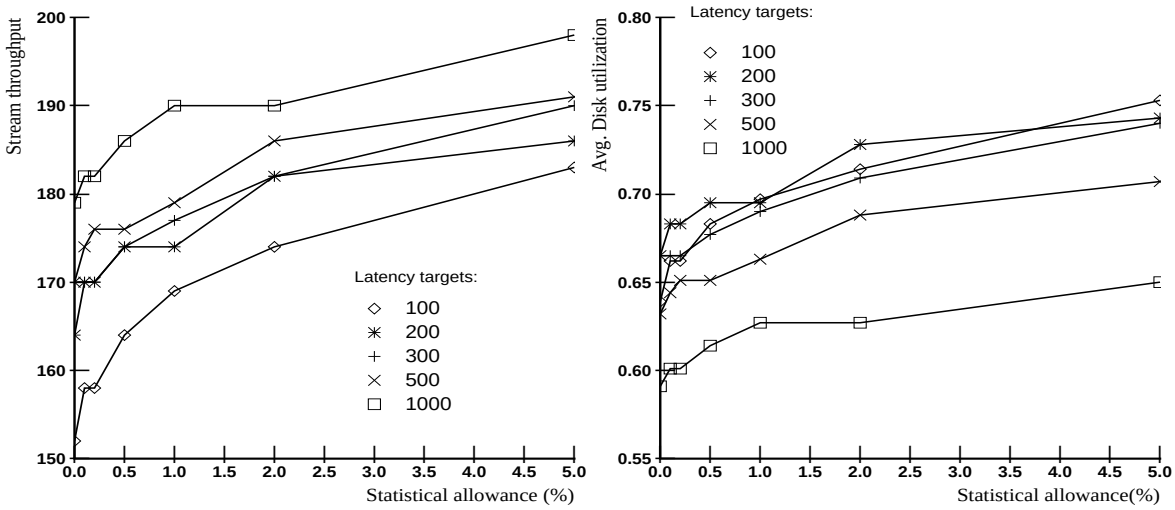

Fig. 3. Effect of statistical allowances.

Fig. 3. also shows the tradeoffs possible between latency targets and the number of blocks allowed to be dropped. At a latency target of 100 rounds, 152 streams can be provided deterministic service. To achieve higher stream throughput, latency targets can be increased or blocks can be dropped. For example, when the latency target is increased to 1000 rounds, 179 streams could be scheduled without dropping any blocks. However, to achieve the same throughput at a latency target of 100 rounds, more than 2% of the blocks have to be dropped. Hence, desired throughput can be achieved either by allowing larger latency targets or by allowing a fraction of the blocks to be denied service.

Fig. 3. also shows the impact on the average disk utilizations as a function of the statistical allowances and latency targets. As higher statistical allowances are made, the disk utilizations are improved as more number of streams are supported. As latency targets are increased from 100 rounds, average disk utilizations first increase and then decrease to lower levels. As latency targets are increased, an arriving stream finds more choices to find a suitable starting spot to utilize the available bandwidth. However, as the latency targets are increased further, the streams are scheduled farther and farther into the future and hence results in decreasing disk utilizations.

Usually considered statistical guarantees of 99% (i.e., dropping 1% of blocks) did not provide

significant improvements in stream throughput compared to deterministic guarantees. In our measurements, the improvements were less than 6% for all the latency targets except 100 which achieved an improvement of 11%. Most of the dropped blocks tended to be dropped in a small range of time. Even though a 30 minute movie drops only a few blocks during its playback, the dropped blocks tended to be clustered over a 2 minute window instead of over the entire movie. This suggests that storing the data during the peak demands of popular movies in memory (not the entire movie) can lead to considerable improvement in performance.

3.2 Latency and bandwidth guarantees

Aperiodic requests are provided bandwidth guarantees by restricting the periodic and interactive requests to certain fraction of the available bandwidth. The admission controller for periodic and interactive requests enforce the bandwidth allocations. Aperiodic requests utilize the remaining I/O bandwidth. If periodic and interactive requests cannot utilize the allocated bandwidths, aperiodic requests are allowed to utilize the available bandwidth to improve the response times for aperiodic requests. In our system, bandwidths are allocated statically to different types of requests.

Latency guarantees are provided by the disk scheduler. The next section provides a description of the scheduler and how it provides low response times for interactive requests. The arrival behavior of interactive requests was modeled by a random exponential arrival process. When requests arrive randomly, a burst of requests can possibly disturb the guarantees provided to the periodic requests. To avoid this possibility, interactive requests are controlled by a leaky-bucket controller which controls the burstiness by allowing only a certain maximum number of interactive requests served in a given time window. For example, when interactive request service is limited to, say, 5 per second, the leaky-bucket controller will ensure that no more than 5 requests are released to the scheduler irrespective of the request arrival behavior. Hence, an interactive request can experience delay in the controller as well as at the scheduler for service. If sufficient bandwidth is allocated for these requests, the waiting time at the controller will be limited to periods when requests arrive in a burst.

3.3 Scheduling for multiple QOS levels

Disks typically employ seek optimization techniques to minimize the seek costs in serving requests. Deadline scheduling and such similar real-time requests provide real-time guarantees but may not

be efficient because of associated higher seek costs. Disk scheduling strategies have been proposed that allow seek optimizations while providing delivery guarantees for continuous media applications [2, 1, 17, 18]. Most of this work considered seek optimizing policies that can provide delivery guarantees required by continuous media in Video-on-demand (VOD) and such similar servers. A few of these studies [1, 18] have considered scheduling policies for multiple levels of service within the same system.

Since the requests do not have strict priorities over each other, priority scheduling is not feasible. Periodic requests have to be given priority over others if they are close to missing deadlines. But, if there is sufficient slack time, interactive requests have higher priority such that they can receive lower latencies. Periodic requests are available at the beginning of the round and interactive and aperiodic requests arrive asynchronously at the disk. If periodic requests are given higher priority and served first, aperiodic and interactive requests will experience long response times at the beginning of a round until periodic requests are served. Moreover, it may be possible to better optimize seeks if all the available requests are considered at once.

It is assumed that the requests are identified by their service type at the scheduler or provided different queues for different request types. The scheduler is designed such that it is independent of the bandwidth allocations. This is done such that the bandwidth allocation parameters or the admission controllers can be changed without modifying the scheduler. To enable this, the aperiodic requests are queued into two separate queues. The first queue hold requests based on the minimum throughput guarantee provided to these requests. The second queue holds any other requests waiting to be served. The scheduler considers the requests from the second queue after periodic requests and interactive requests are served such that these requests can utilize the unused disk bandwidth.

The scheduler merges the periodic requests and aperiodic requests (from queue 1) into a SCAN order at the beginning of a round. These requests are then grouped into a number of subgroups based on their location on the disk surface. The scheduler serves a subgroup of requests at a time and considers serving interactive requests only at the beginning of a subgroup i.e., the disk SCAN order is not disturbed within a subgroup. If an interactive request is to be served, the scheduler groups this request into the closest subgroup and serves that group next. An interactive request hence gets a quick response. Interactive requests are served by a first-come first-serve policy to limit the maximum response time of a single request. Since an interactive request (if waiting for service) is served every subgroup, the response time for an interactive request $\leq (n+2)^*$ (time for serving a subgroup), where n is the number of interactive requests waiting in front of an arriving request. Hence, the response times for interactive requests are determined by the burstiness of the interactive requests and the size of the subgroup. The size of the subgroup can be decreased if

tighter latency guarantees are required. If aperiodic requests are considered only at the beginning of the round, it is likely that requests arriving later will suffer long response times. To avoid this problem, aperiodic requests (of queue 1) are considered and merged into one of the subgroups on their arrival. Admission controllers are designed such that the total time for serving the requests seen by the scheduler in the periodic, interactive and the first aperiodic queue takes less time than a round. Requests in the second aperiodic queue are served when no periodic and interactive requests are waiting for service. The requests in this queue use the unutilized bandwidth of aperiodic and interactive requests.

Fig. 4. shows the average disk utilization when periodic requests are allocated 50% bandwidth, aperiodic and interactive requests are each allocated 25% bandwidth. A scheduling round of 0.5 seconds and a subgroup of 62.5 milliseconds was considered. The number of periodic requests streams was maintained at the maximum that the system can support. Aperiodic request rate is varied while maintaining the interactive request rate at 50 requests/sec (request rates are measured over the entire system of 8 disks). It is observed that the average utilization of the periodic streams stays below 50%. Because of variations in demand over time, more periodic streams could not be admitted. The utilizations of periodic and interactive requests are unaffected by the aperiodic request rate. It is also observed that as the aperiodic request rate is increased, aperiodic requests take up more and more bandwidth and eventually utilize more than the allocated 25% bandwidth. When periodic and interactive requests don't make use of the allocated bandwidth, aperiodic requests make use of any available bandwidth (25% is the minimum available) and hence can achieve more than the allocated 25% utilization of the disk. This shows that disks are not left idle when aperiodic requests are waiting to be served.

Fig. 5. shows the average and maximum response times of aperiodic and interactive requests as a function of the aperiodic request rate. The number of streams is kept at the maximum allowed by the system and the interactive arrival rate is kept at 50 requests/sec. Interactive response times are not considerably affected by the aperiodic arrival rate and the maximum interactive response time stays relatively independent of the aperiodic arrival rate. It is also observed that the interactive requests achieve considerably better response times than aperiodic requests (260ms maximum interactive response time compared to 1600ms for aperiodic requests both at 50 reqs/sec). Both average and maximum response times are better for interactive requests than for aperiodic requests even at lower aperiodic arrival rates. It was observed that the maximum interactive response times are only dependent on the burstiness of arrival of interactive requests and the bandwidth allocated to them. Zero percentage of periodic requests missed deadlines as aperiodic request rate is varied.

Fig. 6. shows the response times of aperiodic requests and interactive requests (both at 25

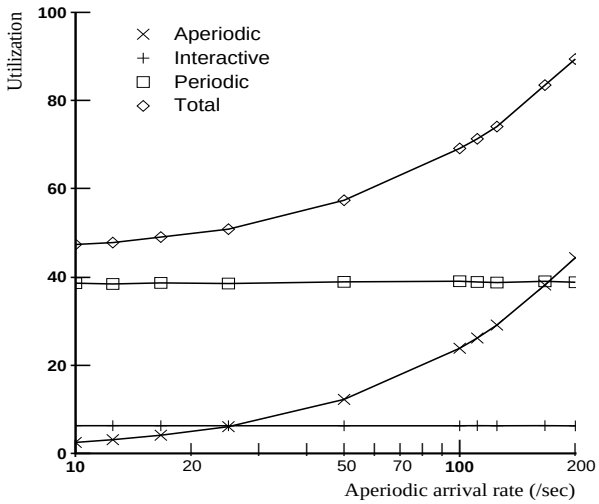

Fig. 4. Average disk utilizations across request categories.

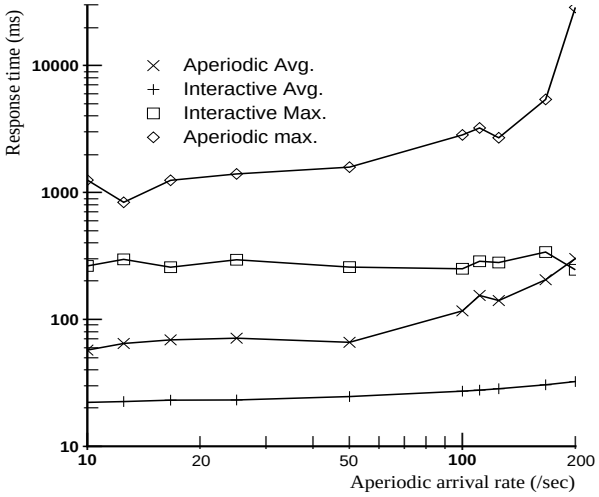

Fig. 5. Impact of aperiodic arrival rate on response times.

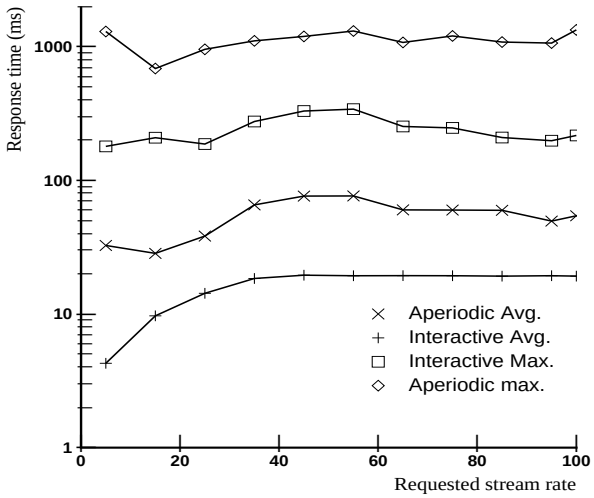

Fig. 6. Impact of requested stream rate on response times.

requests/sec) as the number of requested streams in the system is varied from 5 to 100. With the considered allocation of bandwidths, the system could support a maximum of 33 streams. Hence, even when more number of streams are requested, the system admits only 33 streams. This shows that the periodic request rate is contained to allow aperiodic requests and interactive requests to achieve their performance goals. We observe that the maximum response times of interactive requests are not considerably impacted by the number of requested streams in the system.

Fig. 7. shows the performance of interactive requests when interactive arrival rate is varied. It is observed that the response times of interactive requests stay bounded at the scheduler irrespective of the arrival rate. Beyond the supported rate, the interactive requests are forced to wait longer in the leaky-bucket controller. Hence, beyond the supported rate, the total delay experienced by an interactive request can exceed the desired latency bound. If higher interactive request rate needs to be supported, these requests have to be allocated higher bandwidth so as to limit the delays at the leaky-bucket controller. We observed that the maximum number of streams supported by the system stayed constant (at 33) irrespective of the arrival rate of the interactive requests and zero percentage of these requests missed deadlines as interactive request rate is varied.

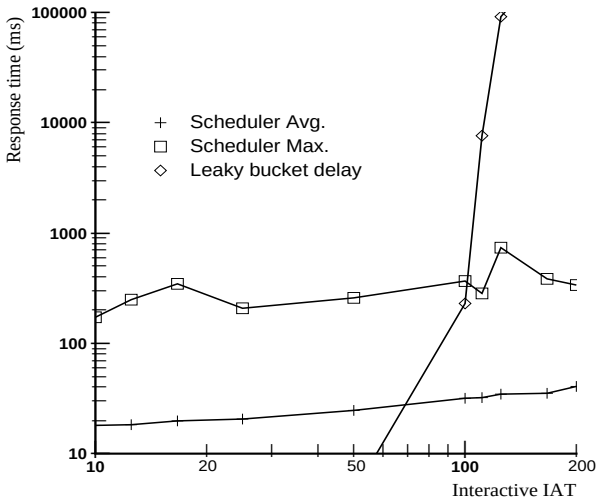

Fig. 7. Impact of interactive arrival rate on response times.

3.4 Other issues in Disk scheduling

Data striping increases the need for protecting against failures since the failure of any disk can render service unavailable for all the streams. Parity protection such as RAID [15] or data duplication can provide data availability guarantees after a failure. But these techniques cannot guarantee availability of the necessary I/O bandwidth for timely delivery of data after a failure. Scheduling data delivery after a component failure is a difficult problem. Much work needs to be done in this area. Overdesigning the system is one of the possible options such that even after a failure the data can be delivered in a timely fashion. Dynamic resource allocation for tolerating failures is another possible option. In certain cases, only statistical guarantees may be provided after a failure.

Providing a VCR-like capability of pause/resume, fast-forward/reverse requires that sufficient resources be available at the server to absorb the variations in bandwidth demands due to such operations [19]. Pause operation may reduce the bandwidth demand at the server and other operations may increase the bandwidth requirements. To support such operations, the server has to determine that these bandwidth variations do not affect the delivery of other scheduled streams. It is also possible to keep the bandwidth requirements at the server constant by serving alternate

copies of video streams that have less quality encoding at the higher frame rates required by the fast-forward and reverse operations [20]. Then the server can serve the alternate copies of video data during fast-forward and reverse operations without altering the schedule of the scheduled streams.

4 CPU scheduling

Most existing operating systems use priority-based schemes for scheduling processes with different performance requirements. Priority based schemes however do not provide a convenient mechanism for dividing bandwidth among multiple classes of applications. To satisfy the conflicting demands of different multimedia applications, it may be necessary to give different priorities for the same process depending on the nature of the competing processes at that time. It has been observed that the assignment of priorities and the dynamic adjustment of priorities are often ad hoc [21]. Fair share schedulers [22] have been proposed for distributing CPU resources fairly over long periods of time (several minutes). Techniques also exist for achieving service rate objectives using decay-usage scheduling [23]. These schemes monitor the CPU usage and dynamically alter the priorities to achieve the service objectives. These techniques work over long periods of time and cannot provide any service guarantees over shorter durations. Microeconomic and market-based approaches for resource allocation have been studied in [24]. In these schemes, processes bid for resources based on their "funds" or allocations and unsuccessful processes are allowed to increase their bids to improve chances of acquiring the resources in the future.

Lottery scheduler [9] assigns the CPU based on a randomized fair algorithm to achieve proportional CPU allocation. Because of the inherent randomness, this algorithm achieves fairness over long periods of time. Hierarchical scheduling algorithm makes a distinction between CPU allocation among different request types and the scheduling of CPU resources among multiple threads of a request type. Hierarchical scheduler uses static resource allocation and the recent CPU usage information to decide the next process to schedule. The threads within a request type or a process can be scheduled differently based on the application requirements, for example, hard real-time requests could use EDF and soft real-time requests could use a fair scheduler.

5 Network scheduling

Traditionally, the network scheduling is done on a best-effort basis i.e., provide as small a delay as possible and as large throughput as possible but without any guarantees on the delivered performance. Best-effort protocols for multimedia delivery follow this approach with an aim to utilizing the existing network resources. These protocols typically adapt delivery to the available bandwidth. The multimedia application may adjust the compression such that lower quality data is received when network congestion reduces available bandwidth. For example, video frame rate or the bit rate in each frame can be adjusted to suit the available network bandwidth. Transport protocols such as TCP adjust the sending rates based on the available bandwidth. However, TCP is designed for reliable delivery of data without regard to the delays experienced by individual packets in a transmission and hence not quite suitable for video delivery. Other popular transport protocol UDP does not adapt to network resources.

Buffering can be used to ameliorate network congestion problems. However, buffering increases the delay between the time of transmission and the time of consumption of delivered data. Large delays may be acceptable when delivering stored media such as prerecorded video and audio. In such cases, buffering delay will result in a larger startup delay at the client. Large delays (due to buffering or network) are however unacceptable in live media transmissions such as in teleconferencing. Packets lost in the network due to congestion or buffer overflow may have to be retransmitted. In compressed video, not all the packets contain the same quality data and hence it is important that some packets be given more priority over others during delivery. It has been shown that prioritized transmission of data can deliver better quality [8]. Protocols can be designed to give priority to certain packets over others at the sender during transmission to achieve better quality data delivery [7]. Feedback can be exchanged between the sender and receiver to adapt the delivery to the current network conditions. The feedback can be in the form of measured delay [7] or in the form of available bandwidth. Feedback can also be exchanged between routers and switches in the network [8]. Adaptation can be in the form of adjusting the frame rate or the bit rate [25] or in the form of delivering only higher quality frames [7] while dropping the lower quality frames.

The other approach to delivery is based on reserving resources in the network based on the application needs [6]. In this connection-oriented approach, each application characterizes its requirements and passes them on to the network at the time of connection establishment. These requirements are processed by the network switches to estimate the possible guarantees that can be provided. When this negotiation process between application requirements and the network guarantees is completed, it is expected that if the application limit its resource usage to its

requirements, the network will be able to meet the promised performance guarantees. To make this possible, the switches in the network have to schedule packets appropriately. Much work has been done in characterizing the application requirements and the bounds that can be provided under various scheduling policies [5, 3, 26, 4, 27]. Statistical and deterministic guarantees are considered in these scheduling policies.

A scheduling policy is termed *non-work-conserving* if the switch can be idle even when there are packets to be switched and *work-conserving* otherwise. Non-work-conserving policies can waste network bandwidth, but simplify network resource control significantly by restricting the output traffic rate at each switch. To ensure that the user limits network resource consumption to the stated demands, the network switches may adopt *rate control* or per-connection queuing. RCSP [3], Stop-and-go [26] are examples of non-work-conserving scheduling policies. Fair queuing [27] and general processor sharing [4] are examples of work-conserving scheduling policies.

6 Summary

Multimedia applications can be provided different levels of service depending on the nature of the application and the availability of resources. An application requires service from the CPU, disks, memory and network to meet the performance goals. Bandwidth allocation and rate controls can be used to ensure that applications do not interfere with a different class of applications in the system. Appropriate scheduling policies then can provide the performance guarantees required by the applications.

In this chapter, we provided a comprehensive view of the disk scheduling problem in providing integrated service. A method has been presented that allows exploitation of statistical multiplexing while providing deterministic service for VBR streams. Through bandwidth allocations and round-based seek optimization scheduling, performance goals have been achieved for different classes of applications. Similarities in CPU scheduling and network scheduling strategies have been pointed out.

References

[1] A. L. N. Reddy and J. Wyllie. I/O issues in a multimedia system. *IEEE Computer*, Mar. 1994.

[2] P. S. Yu, M. S. Chen, and D. D. Kandlur. Grouped sweeping scheduling for dasd-based multimedia storage management. *Multimedia Systems*, 1:99–109, 1993.

[3] H. Zhang and D. Ferrari. Rate-controlled static-priority queueing. *Proc. IEEE INFOCOM*, Apr. 1993.

[4] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control in integrated service networks: The multiple node case. *Proc. IEEE INFOCOMM'93*, pages 521–530, Mar. 1993.

[5] R. Cruz. A Calculus for Network Delay, Part I: Network elements in isolation. *IEEE Trans. on Information Theory, vol.37, no.1*, pages 114–131, Jan. 1991.

[6] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. Rsvp: A new resource reservation protocol. *IEEE Network magazine*, Sept. 1993.

[7] B. Smith. Cyclic-UDP: A priority-driven best-effort protocol. *Tech. Report, Cornell University*, 1996.

[8] H. Kanakia, P. Mishra, and A. Reibman. An adaptive congestion control scheme for real-time packet video transport. *Proc. of ACM SIGCOMM*, Oct. 1993.

[9] C. Waldspurger and W. Weihl. Lottery Scheduling: Flexible proportional-share resource management. *Proc. of Symp. on Operating System Design and Implementation*, Nov. 1994.

[10] P. Goyal, X. Guo, and H. Vin. A hierarchical CPU scheduler for multimedia operating systems. *Proc. of Symp. on Operating System Design and Implementation*, 1996.

[11] A. L. Narasimha Reddy and R. Wijayaratne. On providing deterministic guarantees for vbr streams. *Tech. rep. TAMU-ECE-9701, Texas A&M University*, Apr. 1997.

[12] R. Haskin. The shark continuous-media file server. *Proc. of IEEE COMPCON*, Feb. 1993.

[13] Microsoft. The tiger video server. *Microsoft Press Release*, Apr. 1994.

[14] A. Laursen, J. Olkin, and M. Porter. Oracle media server: providing consumer based interactive access to multimedia data. *Proc. of SIGMOD*, pages 470–477, 1994.

[15] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). *ACM SIGMOD Conference*, June 1988.

[16] O. Rose. Mpeg trace data sets. *ftp-info3.informatik.uni-wuerzburg.de*, 1995.

[17] D. J. Gemmell and S. Christodoulakis. Principles of delay-sensitivie multimedia storage and retrieval. *ACM Trans. on Info. Systems*, pages 51–90, 1992.

[18] C. Martin, P. Narayanan, B. Ozden, R. Rastogi, and A. Silberschatz. The Fellini multimedia storage server. *in Multimedia Information Storage and Management, Ed: S.Chung, Kluwer-Publishers*, 1996.

[19] K. D. Jayanta, J. D. Salehi, J. F. Kurose, and D. Towsley. Providing vcr capabilities in large-scale video server. *Proc. of ACM Multimedia Conf.*, pages 25–32, Oct. 1994.

[20] M. S. Chen, D. Kandlur, and P. S. Yu. Support for fully interactive playout in a disk-array-based video server. *Proc. of ACM Multimedia Conf.*, pages 391–398, Oct. 1994.

[21] H. Deitel. Operating systems. *Addison Wesley*, 1990.

[22] J. Kay and P. Lauder. A fair share scheduler. *Comm. of ACM*, Jan. 1988.

[23] J. Hellerstein. Achieving service rate objectives with decay usage scheduling. *IEEE Trans. on Software Engg.*, Aug. 1993.

[24] D. Ferguson, Y. Yemini, and C. Nikolau. Microeconomic algorithms for load-balancing in distributed computer systems. *Int. Conf. on Distributed Computer Systems*, 1988.

[25] K. Jeffay, D. Stone, T. Talley, and F. Smith. Adaptive, best-effort delivery of digital audio and video across packet-switched networks. *In Proc. Network and Operating System support for Digital Audio and Video*, 1993.

[26] S. Golestani. Duration limited statistical multiplexing of delay-sensitive traffic. *Proc. of INFOCOMM*, pages 12–20, 1991.

[27] A. Demers, S. Keshav, and S.Shenker. Analysis and simulations of a fair queuing algorithm. *Proc. of ACM SIGCOMM*, pages 3–12, 1989.