QoS Support for Intelligent Storage Devices

Joel C. Wu Scott A. Brandt
Computer Science Department
University of California, Santa Cruz
{jwu,sbrandt}@cs.ucsc.edu

Abstract

Most existing research on real-time storage have focused on the use of QoS-aware disk schedulers. However, the increasing intelligence and autonomy of modern disk drives have made fine-grained external disk scheduling difficult. As this trend continues, providing QoS-aware storage by using external disk schedulers may become infeasible in the future. The goal of this paper paper is to present a general framework for QoS-aware storage that does not rely on external disk schedulers.

1. Introduction

Commodity computer systems today are expected to support mixed workloads including traditional and multimedia applications. The principal challenge with designing such system stems from the requirement to provide simultaneous support for both soft real-time and non real-time processes. Commodity operating systems typically employ best-effort resource management that does not provide sufficient support for this mixed-workload scenario. To address this issue, various works on QoS-aware CPU scheduling have appeared in the literature [13, 14, 9, 16, 3]. These schedulers are aware of the different types of processes and can allocate the CPU resource accordingly. However, having QoS-aware CPU scheduler alone is not enough. Most soft real-time processes are also *storage-bound* [6], they have stringent storage bandwidth requirements. These types of applications must be able to access the storage continuously and in a timely manner in order to make satisfactory progress. If the CPU scheduler is QoS-aware but the disk subsystem is not, when the disk bandwidth is overloaded, the storage may become the bottleneck and prevents storage-bound soft real-time processes from making satisfactory progress even if it can receive all the CPU resources it needs.

The need to provide real-time storage access has long been recognized. Significant amount of literature exists in the multimedia field on this subject. Traditionally, this issue has been addressed through four areas [8]: disk scheduling, data placement, admission control, and multimedia file systems. Of these four areas, disk scheduling has received the most attention because of its criticality. Most often real-time storage capability is provided by the use of external QoS-aware disk schedulers. Such schedulers require detailed knowledge about the disk drive internals in order to make accurate prediction of service time. However, as technology advances and computing power becomes cheaper, storage devices are gaining more intelligence and encapsulate the increasingly complex internal details. Future disk drives are expected to continue along this line and may even off-load functionalities from the main CPU, at which point fine-grained external disk scheduling would become infeasible.

We believe that an approach to providing QoS for storage without relying on fine-grained external disk scheduler is necessary in order to handle storage devices of the future. We propose a general framework to provide QoS for storage from the coarse-grained perspective of bandwidth management instead of using fine-grained external disk scheduling. Our approach uses traffic shaping above the external disk scheduler, and can work with either reservation-based or feedback-based control mechanisms.

2. Motivation

QoS-aware disk schedulers are the predominant approach used to provide QoS for data storage. Existing disk schedulers can be classified into three categories: best-effort, real-time, and mixed-workload. For all disk scheduling algorithms, the goal is to balance the two conflicting requirements of response time and overall throughput while meeting the design objectives of the scheduler. Best-effort disk schedulers have no sense of deadlines or QoS requirements but are desirable because of its relative simplicity. An example of modern best-effort disk scheduler is the Anticipatory Scheduler [10] used in the new Linux 2.6 kernel. Real-time disk schedulers assume that each I/O request is associated with a deadline and attempt to order the requests in a way that satisfies these timing constraints. A common theme is to combine the timeliness of EDF with the bandwidth utilization of elevator algorithm [15]. Best-effort disk schedulers and real-time disk schedulers are inadequate for mixed-workload scenario. Mixed-workload schedulers are designed specifically to handle heterogeneous workloads, they typically classify incoming disk requests into different categories that share the total disk bandwidth. For example, Cello [20] is a two-level disk scheduler with a class independent and three class dependent schedulers. The class independent scheduler controls the bandwidth allocation to application classes while the class specific scheduler control the interleaving of requests from different classes.

Disk scheduling is an intrinsically difficult problem. Optimal disk scheduling is NP-complete in general [21]. It differs from CPU scheduling for two main reasons. First, disk scheduling is stateful. All CPU cycles are equal but the same is not true with disk. The time it takes to access a piece of data on disk not only depends on the location of the desired data, but also depends on the current location of the disk head. Second, unlike CPU scheduling, disk operations are not preemptible. A high priority request cannot preempt a low priority one that is already in progress. Even if preemption is possible, the preemption of a lower priority disk request by a higher priority disk request does not guarantee a better result.

Most importantly, providing QoS through disk scheduling requires fine-grained knowledge of disk drive internals. The external disk scheduler needs to be aware of parameters such as seek time, rotational latency, logical to physical mapping, and other hardware-related details in order to make accurate prediction of the service time. Having detailed knowledge about disk drive internals allows the driver to tune performance more aggressively. Otherwise worst case assumptions would have to be made.

Hard drives used to be dumb devices that exported their hardware profile to the system software. For example, the system software on early personal computers must be aware of the drive geometry in order to access it correctly. The system software has complete control over the passive disk drives. The first popular hard drive interface for personal computers was the Seagate ST412/506. Hard drives using the ST412/506 interface don't even have onboard controller. The controller for the drive containing the microprocessor, firmware, and sector buffer are on a separate controller card. The interface to the drive is a pure physical signal interface. Thanks to advancements in hardware and integration, hard drives have continued to gain intelligence throughout the years. Maxtor conceived the ESDI interface in early 80's that moved the data separator onto the drive, but the rest of the control circuitry is still on a separate card. The IDE and SCSI interface integrated the controller onto the drive itself. The interface between system and hard drive became a logical one and can accept high-level commands. While the IDE interface is merely an extension of the computer's internal bus interface, the SCSI interface is much more complicated and feature-rich, and today it has evolved into an encompassing multi-layered architecture.

Hard drives today are intelligent and autonomous units that encapsulate the internal details. The internal complexity is hidden from outside and the disk is accessed through standardized interfaces. In order to obtain the disk drive characteristics necessary for fine-grained external disk scheduling, disk profiling must be used to extract the values of the needed parameters [18, 7, 22]. The probing of the drives is non-trivial and is getting more difficult as drives become more intelligent.

Some of the challenges faced by fine-grained external disk scheduling were identified [12]. Issues such as coarse observation, onboard caching, drive internal scheduling, rotational offset, and autonomous internal disk activities complicate external disk scheduling. For example, the response time observed from outside of the drive is made up of different component times, and it is difficult to determine the individual component delays inside of the black box of the drive. The drive internal scheduler poses a problem because the requests ordered by the external disk scheduler may be reordered by the disk drive itself. Current solutions to this problem include disabling the disk internal scheduler if possible; issuing only one request at a time; or relying on protocol support [11]. These solutions are often ad hoc and not ideal.

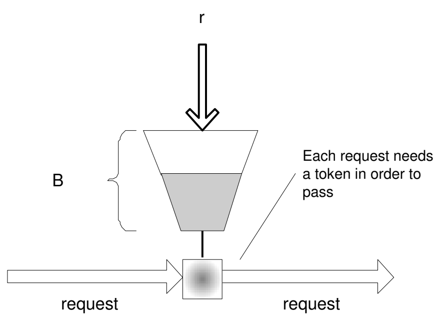


Figure 1: Token Bucket Filter with depth B and rate r

Although fine-grained external disk scheduling is possible now with disk profiling, it is like to become infeasible in the future as disk drives become ever more intelligent. As the trend toward smarter disks continues, in the future it would be very difficult, if not impossible, to maintain fine-grained control over every minute operation that goes on inside of a drive. This would necessitate a different approach to provide QoS for storage, as the effectiveness of external disk schedulers diminshes. Regardless of how intelligent disk drives are, the requirement of retrieving data in a timely manner remains. Therefore, we believe that an alternative approach to providing QoS for storage that does not require intricate knowledge of disk drive internals is needed.

3. Traffic Shaping

For direct-attached storage that are intelligent and autonomous, a viable alternative to providing QoS support for storage-bound soft real-time applications is to take a coarse-grained view. Instead of providing QoS by fine-grained scheduling of the disk, we can achieve QoS by bandwidth management at the layer above the external disk scheduler. Our previous work on Dynamic QoS Level Resource Management (DQM) [4] showed that by adjusting resource usage such that the set of running applications use less than 100% of the available resources, a best-effort scheduler is able to provide reasonable soft real-time performance. Here we are applying this result to disk. This approach can work with any underlying best-effort disk scheduler.

We adapt the concept of traffic shaping from networking for disk bandwidth management. Instead of providing QoS by ensuring that the deadline of each individual disk request is met, we take a coarse-grained approach by ensuring that the bandwidth needs are satisfied. This framework consists of two components: a traffic shaping mechanism that enables us to control the bandwidth, and a policy that decides how to do it. The policy can be either reservation-based or feedback-based.

We use token bucket filter (TBF) [19, 5], a technique for traffic shaping developed for use in networking, to control the bandwidth of data going to and from the disk. The disk requests are differentiated to allow the data to be viewed as distinct flows, and

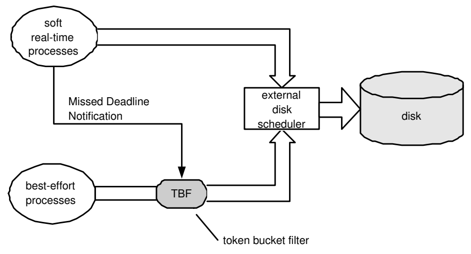


Figure 2: Shaping the best-effort traffic going to the disk

the TBF can then be applied to shape the flows to achieve bandwidth management. The distinction of disk requests is done by associating the disk request with the issuing process. In our current implementation, we make a distinction between disk requests generated by soft real-time (SRT) processes, those with a QoS requirement, and best-effort (BE) processes, those with no QoS requirement. Differentiation is done by checking a flag in the disk request. Before accessing files, a process can make a system call to declare itself as an SRT process. Alternatively, this information can be automatically determined at run-time [1]. Subsequent disk I/O requests generated by SRT processes will be tagged as SRT, and all non-tagged requests will be treated as BE requests. Therefore, we can imagine a disk as having two access pipes, one pipe carries all the SRT data and the other pipe carries all the BE data.

Figure 1 shows the concept of TBF used to shape flow of data to disk. The token bucket receives tokens at the rate of r. The depth of the token bucket B limits the maximum amount of tokens that can be accumulated at any given time, and hence the burst rate. We associate each BE disk access request with a token. A BE request must have a token in order to be issued. If a request is to be issued but there is no token available, the process triggering the request will be blocked until tokens become available. TBF serves as the enforcer that controls the rate BE requests can be issued and therefore shapes the size of the BE data pipe. Since we are only classifying disk requests into two types, we only need to shape the BE data pipe and not the SRT data pipe, unless the SRT traffic begins to starve the BE traffic. The idea of controlling disk bandwidth by controlling the rate of requests is not new, it was mentioned in [17] as part of a mechanism that allocates disk bandwidth proportionally by monitoring application's rate of progress.

4. Feedback-Based control using Missed Deadline Notification

The TBF mechanism needs a specification of how to shape the BE disk request rate. The allocation decision can be either reservation-based or feedback-based. Reservation-based schemes are conceptually simpler but require a-priori knowledge of resource usage requirement, which may be difficult to determine. We plan to implement both reservation-based and feedback-based schemes. This section describes a feedback-based scheme that we have already implemented. We use Missed Deadline Notification (MDN) [2] to determine how to shape the BE disk request rate. MDN is a mechanism for soft real-time processes to notify the operating system that they have missed a deadline, so the operating system can adjust resource allocations accordingly. It allows the operating system to receive feedback at run time on the status of the soft real-time processes without requiring the soft real-time processes to specify their resource usage requirements. Our previous work on MDN focused on CPU scheduling. In this paper we use Missed Deadline Notification to signify that a missed deadline has occurred due to the inability to access data from storage in time. We use MDNs generated by SRT processes as an indication that the disk bandwidth is saturated and the BE pipe needs to be reduced. In an integrated approach, an application can utilize two different MDN calls, one for CPU and one for disk.

The only information a soft real-time application needs to provide to the operating system prior to execution is to declare itself as an SRT process. At run time, feedback is achieved by using MDN, which is implemented as a simple system call with only one argument: the file descriptor for the data file being accessed. No information about deadlines or bandwidth requirements are needed.

Under normal operating conditions when the disk bandwidth is underutilized, we do nothing and the system behaves identical to one without our implementation. It is only when the disk bandwidth is overloaded that the effect of our mechanism becomes visible. The concept is simple: when the disk bandwidth is saturated and SRT processes cannot receive the disk bandwidth they desire, we give SRT requests preferential handling by throttling the rate of competing BE requests. Figure 2 depicts this mechanism. MDN notifies TBF to shrink the BE pipe to allow the expansion of SRT pipe. The mechanism tries to find a point at which SRT pipe receives the right bandwidth it needs while BE pipe takes up the rest.

Initially the token rate is set to a high value beyond the maximum achievable rate, in essence placing no rate limitation. When an MDN is received from SRT process, we reduce the BE rate. We must also have a way of increasing the BE pipe size when the SRT pipe size decreases. There are two ways this is achieved in our mechanism. Both methods were implemented and tested.

In the base approach, every time the token bucket is replenished, we increase the token rate by a small increment. This allows the token rate to creep back up additively over time. The outcome is that when the BE pipe size is capped (less than the maximum achievable rate), it constantly increases its size, eventually pushing on the boundary of SRT pipe until an MDN is generated by SRT process, causing the BE pipe to decrease its size and then repeat the growth again. When the workload is constant and disk bandwidth is overloaded, this leads to a sawtooth pattern for BE pipe size. When the disk bandwidth is not saturated, the BE pipe size eventually grows to beyond the maximum achievable rate, and the effect of our mechanism becomes invisible.

In the extended approach, instead of having the BE pipe size constantly trying to expand itself to make full utilization of the bandwidth, we will tell it when to expand. This requires the SRT processes to make a system call when it finishes accessing a file. The goal of this second system call is to notify the system that the aggregate SRT bandwidth has changed, and adjustment in bandwidth allocation is needed. In this method, the BE token rate will drop whenever an MDN is received, and it will not increase until an SRT process is done with a stream. The constant increasing of BE pipe size that pushes against SRT pipe size periodically is eliminated. The dynamic adjustment of BE pipe size will only happen when SRT stream is introduced, when MDN call is received, and when an SRT stream is done accessing a file. This can of course be done automatically when an SRT process exits. In both approaches, anti-cheat feature [2] can be used to prevent processes from abusing the MDN mechanism.

5. Preliminary Results

We have implemented the traffic shaping mechanism and the feedback-based control using MDN on Linux 2.6.0. Our test system is a 1.5 GHz P4 with 512MB of RAM. The disk is a Seagate ST340810A IDE drive formatted with an ext2 file system. The bandwidth of the disk is about 27.59 MB/s for sequential read. We developed a test program *sbsrtgen* that models the storage access behavior of mpeg players. Testing focused on the performance of constant bit rate (CBR) sequential read. In constant-rate mode *sbsrtgen* tries to read a constant rate of data. In SRT boosted constant-rate mode, it utilizes our bandwidth management mechanism by declaring itself as SRT and implements Missed Deadline Notification. Although mpeg data are of variable bit rate (VBR), we use CBR in *sbsrtgen* because it allows the characterization of performance more clearly by avoiding the fluctuations in bandwidth due to the random nature of VBR. In this section, a stream refers to a flow of data between the disk and a process, whereas a pipe in Section 4 referred to a collection of individual streams of the same type, either SRT or BE.

First we show the validity of using TBF to shape disk bandwidth by setting the token rate to fixed values. Figure 3 shows this result. In this experiment, there are three 8 MB/s constant-rate streams and one 8 MB/s SRT boosted constant-rate stream. In Figure 3(a) where the token rate is set to 100 tokens per second (t/s), we do not see any difference between the disk bandwidth usage by different streams. Figure 3(b) shows the result when we decrease the BE token rate to 90 t/s, the SRT stream is able to receive more bandwidth than the three BE streams, but still not receiving its desired 8 MB/s. We further decrease the BE token rate to 50 t/s, and as shown in Figure 3(c), the SRT boosted stream is able to receives its desired 8 MB/s at the expense of the BE streams.

Now we show the result of using MDN for feedback-based control. In this experiment, we run four 8 MB/s streams simultaneously, which is beyond the maximum bandwidth of the disk. Figure 4(a) shows the result when none of the streams utilize SRT boosting. We see that all four streams receive approximately the same bandwidth that is less than the desired rate of 8 MB/s each. Figure 4(b) shows the result when one of the stream uses our base method for SRT boosting. The SRT boosted stream is able to receive its required 8 MB/s bandwidth. The periodic spikes in its bandwidth is caused by the MDN calls and the behavior of mpeg players after it has missed a deadline. Figure 4(c) shows the result when the SRT boosted stream uses the extended method. We see that the bandwidth of the SRT boosted stream spikes only twice during the adjustments to decrease the BE pipe size, and then it is able to maintain the 8 MB/s rate smoothly. This method is more aggressive at boosting SRT streams at the expense of overall throughput.

6. Future Work

In this paper, we argue that because of the increasing intelligent of disk drives, an approach to provide QoS for storage that does not rely on the external disk scheduler is needed. We proposed a token-based approach that manages disk bandwidth above the external disk scheduler. We have implemented the traffic shaping mechanism and feedback-based control using MDN and showed some preliminary results. We plan to further extend and refine this framework. Work is under way to add the reservation-based mechanism to complement the feedback-based mechanism. We also plan to integrate the feedback-based control with the BEST [1] heuristics for complete automated control. The distinction of disk requests can be refined further to allow the control of individual streams associated with a process. Currently we associate a token with a disk request, which represents data located contiguously on disk but may be of variable size. An alternative method is to associate a token with a fixed size of data. Associating a token with other unit of resource such as time is also possible. It may also be beneficial to introduce more types of disk requests, for example, adding an interactive disk request type in addition to best-effort and soft real-time. The current implementation does not differentiate between read and write, and we plan to refine our approach by making this distinction and handle read and write bandwidth separately. We also plan to perform more realistic testing by using real-world applications to generate the best-effort disk traffics. In addition, the mechanism that controls the request rate can be further refined to increase overall bandwidth utilization. Our eventual goal is to apply this approach to beyond direct-attached storage, and allow it to work across a network such as in an object-based storage system.

References

[1] S. Banachowski and S. Brandt. The BEST scheduler for integrated processing of best-effort and soft real-time processes. In *Proceedings of the SPIE, Multimedia Computing and Networking (MMCN)*, volume 4673, 2002.

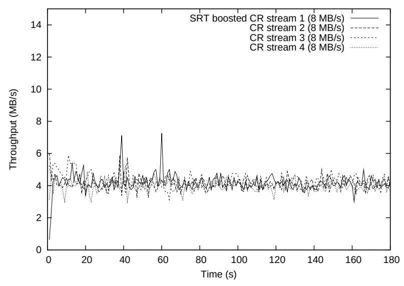
[2] S. Banachowski, J. Wu, and S. Brandt. Missed deadline notification in best-effort schedulers. In *Proceedings of the SPIE, Multimedia Computing and Networking (MMCN)*, January 2004.

[3] S. Brandt, S. Banachowski, C. Lin, and T. Bisson. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. In *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS 2003)*, December 2003.

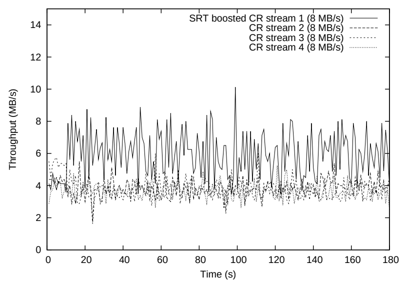
[4] S. Brandt, G. Nutt, T. Berk, and J. Mankovich. A dynamic quality of service middleware agent for mediating application resource usage. In *Proceedings of IEEE Real-Time Systems Symposium (RTSS 1998)*, pages 307–317, December 1998.

[5] P. F. Chimento. Standard token bucket terminology. http://qbone.internet2.edu/bb/, May 2000.

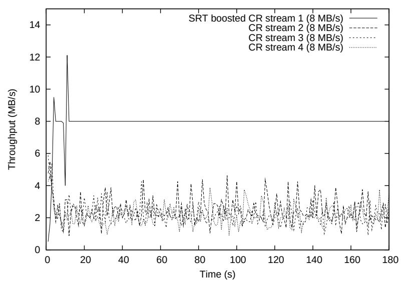
[6] Z. Dimitrijevic and R. Rangaswami. Quality of service support for real-time storage systems. *International IPSI-2003 Conference*, October 2003.



(a) BE request rate: (100 requests/second) - no discernible differences

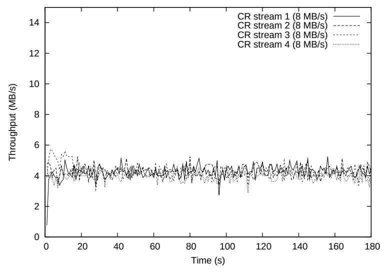


(b) BE request rate: (90 requests/second) - SRT boosted stream receives more bandwidth but still not the desired amount

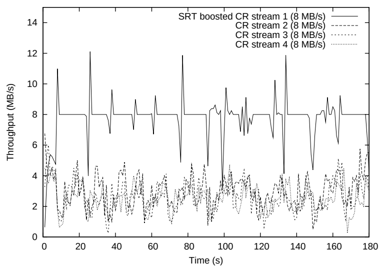


(c) BE request rate: (50 requests/second) - SRT boosted stream receives desired bandwidth

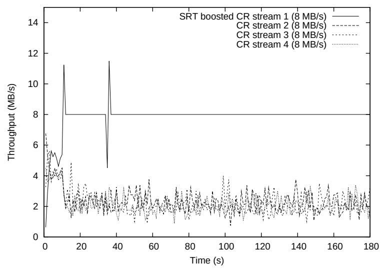
Figure 3: Reserving bandwidth for SRT boosted stream by setting BE token rate to fixed values. By decreasing the BE token rate, we increase the boost to SRT bandwidth.



(a) No SRT boosting - normal Linux behavior



(b) Stream 1 receives SRT boosting - base method. Stream 1 able to receive desired bandwidth with periodic spikes



(c) Stream 1 receives SRT boosting - extended method. Stream 1 able to receive desired bandwidth without periodic spikes

Figure 4: Using MDN for feedback-based control.

[7] Z. Dimitrijevic, R. Rangaswami, and E. Chang. Diskbench: User-level disk feature extraction tool. Technical report, UCSB, November 2001.

[8] J. Gemmell, H. Vin, D. Kandlur, P. Rangan, and L. Rowe. Multimedia storage servers: A tutorial and survey. *IEEE Computer*, 28(5):40–49, 1995.

[9] P. Goyal, X. Guo, and H. M. Vin. A hierarchical CPU scheduler for multimedia operating systems. In *Proceedings of the 2nd Symposium on Operating Systems Desgin and Implementation (OSDI '96)*, pages 107–121, October 1996.

[10] S. Iyer and P. Druschel. Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O. In *Symposium on Operating Systems Principles*, pages 117–130, 2001.

[11] K. H. Kim, J. Y. Hwang, S. H. Lim, J. W. Cho, and K. H. Park. A real-time disk scheduler for multimedia integrated server considering the disk internal scheduler. In *Proceedings of the International Parallel and Distributed Processing Symposium*, pages 124–130, April 2003.

[12] C. Lumb, J. Schindler, and G. Ganger. Freeblock scheduling outside of disk firmware. In *Proceedings of the Conference on File and Storage Technologies (FAST), USENIX*, January 2002.

[13] C. W. Mercer, S. Savage, and H. Tokuda. Processor capacity reserves: Operating system support for multimedia applications. In *International Conference on Multimedia Computing and Systems*, pages 90–99, 1994.

[14] J. Nieh and M. Lam. The design, implementation and evaluation of smart: A scheduler for multimedia applications, October 1997.

[15] A. L. Reddy and J. Wyllie. Disk scheduling in a multimedia I/O system. In *Proceedings of ACM Conference on Multimedia*, pages 225–233. ACM Press, 1993.

[16] J. Regehr and J. A. Stankovic. HLS: A framework for composing soft real-time schedulers, December 2001.

[17] D. Revel, D. McNamee, C. Pu, D. Steere, and J. Walpole. Feedback based dynamic proportion allocation for disk I/O. Technical Report CSE-99-001, Oregon Graduate Institude of Science and Technology, December 1998.

[18] J. Schindler and G. Ganger. Automated disk drive characterization. Technical Report CMU-CS-00-176, Carnegie Mellon University, December 1999.

[19] S. Shenker, C. Partridge, and R. Guerin. Specification of guaranteed quality of service. *RFC 2212*, September 1997.

[20] P. J. Shenoy and H. M. Vin. Cello: A disk scheduling framework for next generation operating systems. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 44–55. ACM Press, 1998.

[21] H. Vin, A. Goyal, and P. Goyal. Algorithms for designing large-scale multimedia servers. *Computer Communications*, 18(3):192–203, March 1995.

[22] B. Worthington, G. Ganger, Y. Patt, and J. Wilkes. On-line extraction of SCSI disk drive parameters. In *Proceedings of the 1995 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 146–156. ACM Press, 1995.