Storage Access Support for Soft Real-Time Applications

Joel C. Wu Scott A. Brandt

Computer Science Department
University of California, Santa Cruz
{jwu,sbrandt}@cs.ucsc.edu

Abstract

Most research on QoS-aware storage has 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 through external disk schedulers may become infeasible in the future. In this paper, we present a coarse-grained approach to storage bandwidth management that does not rely on external disk schedulers. The goal is to provide better storage access support for storage-bound soft real-time applications. Our approach gives priority to disk requests generated by soft real-time applications by controlling the rate that best-effort disk requests may be dispatched.

1. Introduction

Today's commodity computer systems are expected to support mixed workloads including both best-effort and soft real-time applications. 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 approaches on QoS-aware CPU scheduling have been proposed [16, 11, 4]. However, having a QoS-aware CPU scheduler is not enough. Many soft real-time applications today are also *storage-bound* [8], they have stringent storage bandwidth requirements. The quintessential example of this are the multimedia applications that need to access the disk regularly. If the CPU scheduler is QoS-aware but the disk subsystem is not, the storage may become the bottleneck and dictate the progress of the soft real-time processes.

The need to provide disk access in a timely manner has long been recognized [1, 10]. Most often this issue is addressed by employing 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 additional functionality 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 intelligent storage devices of the future. We present an approach to providing 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. When the disk bandwidth is saturated and in the presence of both soft real-time and best-effort disk requests, this approach can better support storage-bound soft real-time applications by throttling the rate of best-effort disk requests.

2. Motivation

QoS-aware disk schedulers are currently the predominant approach used to provide QoS for data storage. Existing disk schedulers can be classified into three types: 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 knowledge of deadlines or QoS requirements. Both real-time and mixed-workload disk schedulers are QoS-aware. The difference being that real-time disk schedulers assume every I/O request has an associated timing constraint [17] , while mixed-workload disk schedulers are designed to handle heterogeneous disk requests [20, 22, 6].

Disk scheduling is an intrinsically difficult problem. Optimal disk scheduling is NP-complete in general [21] and differs from CPU scheduling due to its stateful and non-preemptable nature. In addition, 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.

Hard drives used to be dumb devices that exported their hardware profile to the system software. Today they are intelligent and autonomous units that encapsulate the internal details. The internal complexity is hidden from the outside and the disk is accessed through standardized interfaces. Modern fine-grained disk schedulers require disk profiling in order to extract the values of the needed parameters [19, 9, 23]. 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 [15]. Issues such as coarse observation, on-board caching, drive internal scheduling, rotational offset, and autonomous internal disk activities complicate external disk scheduling. For example, 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 [12]. These solutions are often *ad hoc* and not ideal.

Although fine-grained external disk scheduling is possible now with disk profiling, it may 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. 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

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 through 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) [5] 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 external best-effort disk scheduler.

We adapt the concept of traffic shaping from networking for disk bandwidth management. Token Bucket Filter (TBF) [7] is a mechanism for traffic shaping. In the networking context in which it was developed, TBF is placed

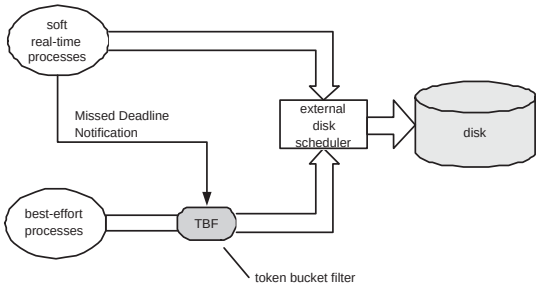


Figure 1: Shaping the best-effort traffic going to the disk

on a data path to impose a prioritization policy on data transmissions. We use token bucket filter to control the bandwidth of data going to and from the disk. We choose token bucket filter because of its simplicity and ease of implementation. Other mechanisms that can control the rate of events can also be used. The disk requests are differentiated to allow the data to be viewed as distinct flows, and 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 [2]. 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.

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 proposed as part of a mechanism that allocates disk bandwidth proportionally by monitoring application's rate of progress [18].

4. Missed Deadline Notification

The TBF mechanism needs a specification of how to shape the BE disk request rate. As with most resource management, 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 use Missed Deadline Notification (MDN) [3], a feedback-based scheme, 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.

5. Storage Bandwidth Management

This section describes the rate-based approach to storage bandwidth management using TBF and MDN. We use TBF to indirectly shape the BE pipe size by controlling the rate of BE requests, and MDN to handle feedbacks from SRT processes in order to control the TBF. This approach requires no *a priori* knowledge about the resource usage requirements.

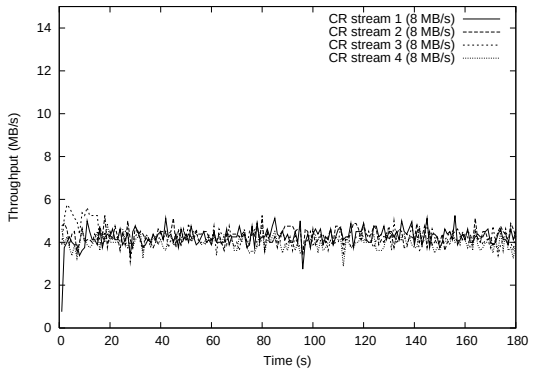
Under normal operating conditions when the disk bandwidth is not overloaded, we do nothing and the system behaves identical to one without our implementation. It is only when the disk bandwidth is overloaded that the effect of our mechanism becomes visible. The concept is simple: when the disk bandwidth 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 1 depicts this mechanism. MDN notifies TBF to shrink the BE pipe to allow the expansion of the SRT pipe. The mechanism tries to find a point at which the SRT pipe receives the right bandwidth it needs while the BE pipe takes up the rest. The following are the parameters associated with the mechanism.

- r: The rate the token bucket is replenished with new tokens. r is dynamically adjustable subject to $r_{min} \leq r \leq r_{max}$.
- r_{min}: The minimum rate of token replenishment. Token rate will not drop below this value. This guarantees that best-effort requests will not starve.
- r_{max}: The maximum rate of token replenishment. Set to an arbitrary high value above the maximum attainable rate.
- c: The creepback rate.
- d: The percentage of reduction on r to be made each time.
- $r_{observed}$: The actual rate observed.

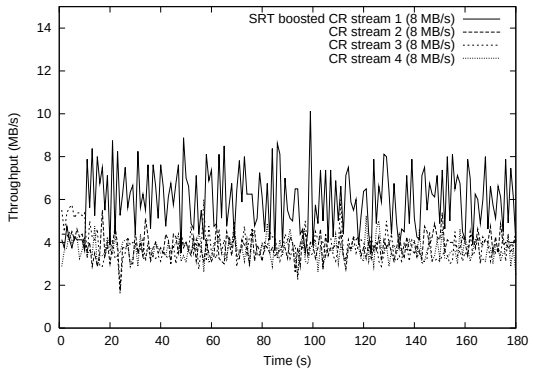
The values of c and d used in testing were obtained by empirical tuning. Initially the token rate is set to an arbitrary high value beyond the maximum attainable rate (r_{max}), in essence placing no rate limitation. When an MDN is received, there are two options. If the current value of r is beyond the maximum attainable rate, such as the case when the first MDN is received, we set r to $r_{observed} \times (1 - d)$. Otherwise, we set r to $r \times (1 - d)$. To prevent starvation of BE requests, the token rate will not drop below a prespecified lower bound r_{min}. To enable dynamic adaptivity, we must also have a way of increasing the BE pipe size. There are two ways this is achieved in our mechanism. Both methods were implemented and tested.

The first method takes an optimistic view in the absence of any *a priori* QoS specification. The BE token rate assumes that any reduction in rate is spurious and there is unused bandwidth. It therefore tries to increase itself automatically any time it needs additional bandwidth. Every time the token bucket is replenished, we increase the token rate r by a small increment c. 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.

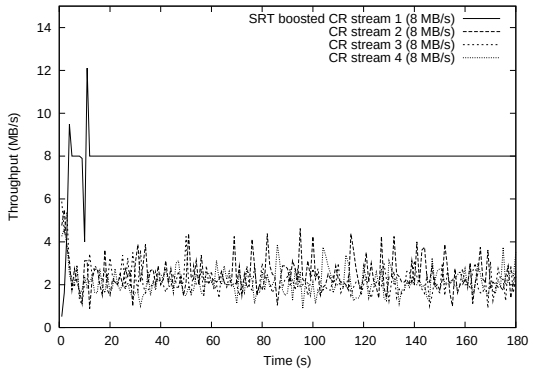
The second approach takes a pessimistic view and assumes that any missed deadline is an indication that the current BE token rate is too high and should be reduced until we are sure it can be increased. Instead of having the BE pipe size constantly trying to expand itself to make full utilization of the bandwidth, we will tell it when to expand. Specifically, 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



(a) No SRT boosting - normal Linux behavior

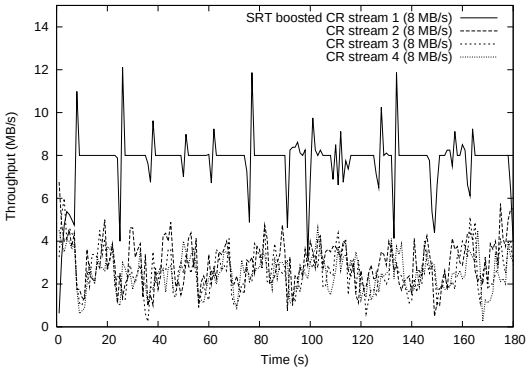


(b) BE token rate: (90 requests/second) - SRT boosted stream receives more bandwidth but still not the desired amount

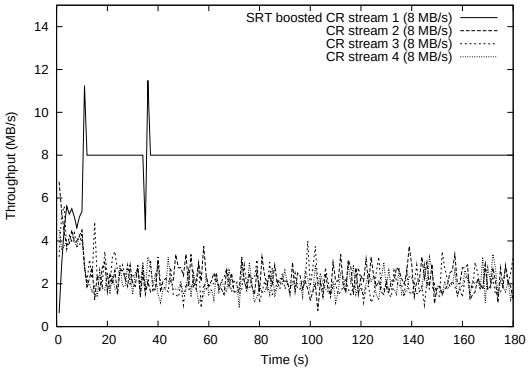


(c) BE token rate: (50 requests/second) - SRT boosted stream receives desired bandwidth

Figure 2: 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) Stream 1 receives SRT boosting - optimistic method.



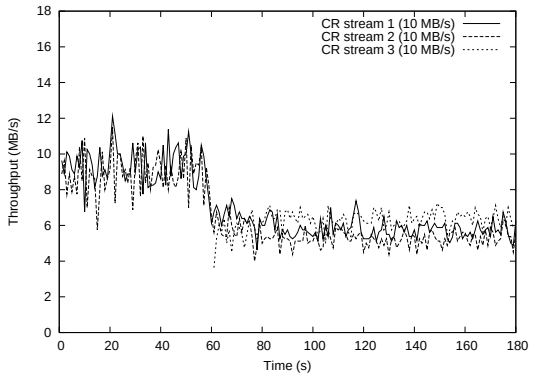
(b) Stream 1 receives SRT boosting - pessimistic method.

Figure 3: Using MDN to control BE token rate.

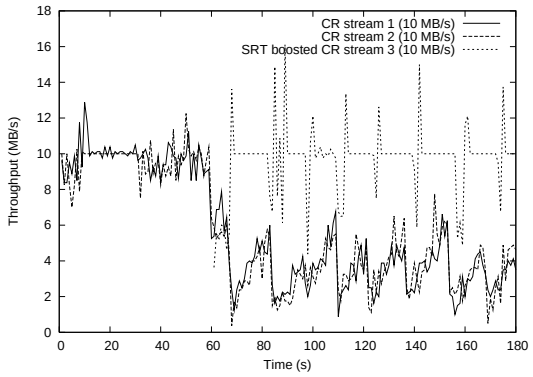
happen when an SRT stream is introduced, when an MDN call is received, and when an SRT stream is done accessing a file. This can be done by a system call or automatically when an SRT process closes a file or exits. In both approaches, our anti-cheat feature [3] can be used to prevent processes from abusing the MDN mechanism.

6. Experimental Results

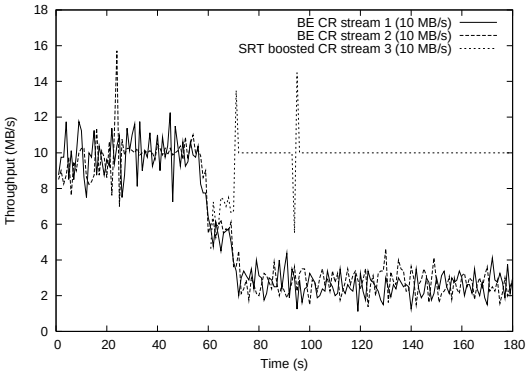
We implemented this approach for storage bandwidth management on Linux 2.6.0. For each request queue associated with a block device, we added a token bucket filter and a wait queue. A BE process must obtain a token before it can issue a request. Processes whose requests can not be issued because of token unavailability will be placed on the token wait queue pending availability of new tokens. A kernel thread is used to replenish the tokens and handle the creepback of token rate. Missed Deadline Notification is implemented as a simple system call. The files being accessed



(a) No SRT boosting - normal Linux behavior



(b) Stream 3 receives SRT boosting - optimistic method



(c) Stream 3 receives SRT boosting - pessimistic method

Figure 4: Three constant-rate streams. Stream 1 and 2 span entire running time, stream 3 starts at time 60.

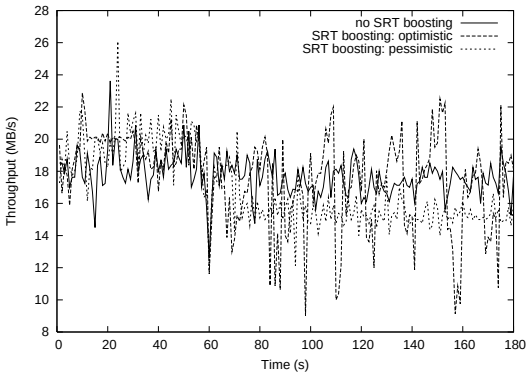


Figure 5: Total throughput relative to BE token rate

by soft real-time processes are tagged and the tags are permeated down the VFS layer so that its resulting disk access request can be distinguished.

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 reads. We developed a test program *sbsrtgen* that models the storage access behavior of video players that follow a regular pattern of behavior.

Testing focused on the performance of constant bit rate (CBR) sequential read. In CBR mode *sbsrtgen* tries to read data at a constant rate. In SRT boosted constant-rate mode, it utilizes the bandwidth management mechanism by declaring itself as SRT and uses Missed Deadline Notification. Although mpeg video 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. Our approach would be advantageous mainly to workloads having continuous traffic flows such as video streams. MDN-based reactive control may be too slow to adapt and less beneficial to SRT processes with short and sporadic traffic patterns. Also, this scheme's effectiveness on VBR stream would be less than on CBR, as the natural fluctuation of stream bandwidth would cause more instability and more difficult for MDN to adapt to. In this section, a stream refers to a flow of data between the disk and a process, whereas a pipe in Section 5 referred to a collection of individual streams of the same type, either SRT or BE.

Figure 2(a) shows the result of four 8 MB/s streams running simultaneously without using our mechanism. The total demand is beyond the maximum bandwidth of the disk. We see that all four streams receive approximately the same bandwidth that is less than the desired rate of 8 MB/s each. The total throughput is 16.8 MB/s. This is the default Linux behavior.

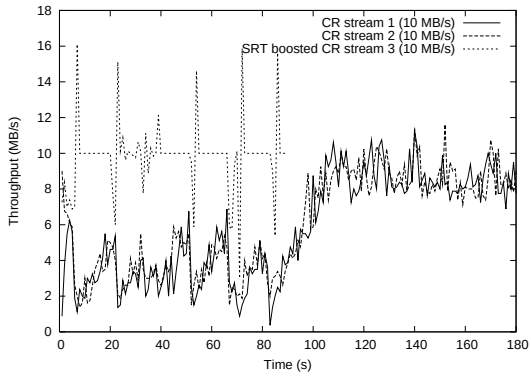
We now show the validity of using TBF to shape disk bandwidth by setting the token rate to fixed values, in the

same way that a reservation-based scheme would work. In this experiment, one of the four 8 MB/s streams declares itself as SRT. Figure 2(b) shows the result when the BE token rate is set to 90 tokens per second (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 decrease the BE token rate further to 50 t/s, and as shown in Figure 2(c), the SRT boosted stream is able to receives its desired 8 MB/s at the expense of the BE streams. This method is highly effective at meeting SRT deadlines when we know the storage QoS requirements of the processes in advance. The 50 t/s token rate results in a 12% loss of overall throughput compared to the default Linux behavior. Interestingly, our 90 t/s filter actually achieved 3% higher utilization than Linux, indicating that in some cases traffic shaping can actually increase overall utilization, perhaps by increasing the overall sequentiality of the requests presented to the disk.

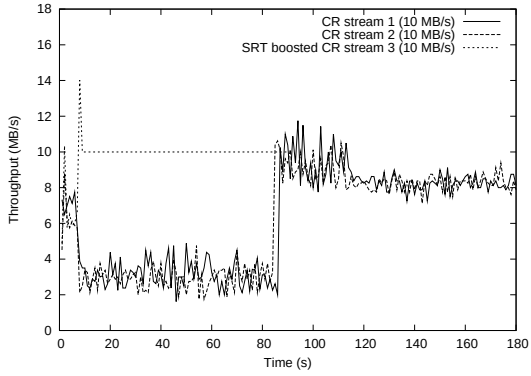
Without *a priori* knowledge of the storage QoS requirements, the static TBF values of the above experiment are relatively useless. We now show the result of using MDN for feedback-based control without such *a priori* information. Figure 3(a) shows the result when one of the streams uses our optimistic method for SRT boosting. The SRT boosted stream is able to receive its required 8 MB/s bandwidth. The periodic spikes in its bandwidth are caused by the MDN calls and the behavior of mpeg players after missing deadlines. The overall throughput is 15.85 MB/s. Figure 3(b) shows the result when the SRT boosted stream uses the pessimistic 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. Overall throughput is 14.48 MB/s. The pessimistic method is more aggressive at boosting SRT stream performance at the expense of about 8.6% drop in total throughput.

The next experiment shows the effect of an SRT boosted stream entering a system that is approaching bandwidth saturation. Each of the three streams consume a bandwidth of 10 MB/s. Two streams started at time zero and span the entire running time while the third stream is introduced at time 60. Figure 4(a) shows the result without using SRT boosting. The introduction of the third stream saturates the bandwidth and brings down the bandwidth of the two other streams already in progress, and the third stream itself is also unable to receive the 10 MB/s desired. All three streams receive approximately the same bandwidth.

Figure 4(b) shows the effect when the third stream utilizes the optimistic method for SRT boosting. Its introduction at time 60 caused the other two streams' bandwidth to drop, but its own bandwidth remains constant at 10 MB/s with the expected periodic spikes. Figure 4(c) shows the effect when the third stream utilizes the pessimistic SRT



(a) Optimistic method: constantly tries to creepback, causing periodic spikes



(b) Pessimistic method: creepback only when bandwidth changes

Figure 6: Comparing the optimistic method and pessimistic method of creepback. The SRT boosted stream terminates at time 90.

boosting method. After two spikes resulting from bandwidth adjustments, it is able to obtain and maintain the desired bandwidth without further spikes. Figure 5 compares the total throughput for no SRT boosting and using SRT boosting with the two methods. As expected, the optimistic method is more unstable. After the introduction of the third stream, with no boosting, the overall throughput is 17.37 MB/s. With SRT boosting, the optimistic method has an overall throughput of 16.63 MB/s, and the pessimistic method has an overall throughput of 15.48 MB/s. The overall throughput of the optimistic method is 7% higher than the pessimistic method. Again, the optimistic method is more aggressive and therefore the total bandwidth utilization is higher. The tradeoff is that it causes more missed

deadlines and more MDNs to be generated, and therefore the periodic spikes in SRT bandwidth.

Figure 6 shows the difference in the creepback of BE bandwidth between the optimistic and the pessimistic approach. The SRT boosted stream terminates at time 90. In the optimistic approach shown in Figure 6(a), the BE streams constantly tries to increase its size and drop when it encroach on the SRT stream's bandwidth. When the SRT boosted stream is terminated, the bandwidth of the BE streams creeps back gradually. In the pessimistic method shown in Figure 6(b), the BE bandwidth does not try to creep back constantly and therefore it does not cause the spikes. After the SRT boosted stream terminates, it is able to regain the newly freed bandwidth immediately. In this test case, the overall throughput up to the point where the SRT boosted stream terminates is 16.81 MB/s for the optimistic method, and 16.47 MB/s for the pessimistic method, a difference of only 2%.

Figure 7 shows the effect when we introduce and remove an SRT boosted stream using the pessimistic method. An 11 MB/s constant-rate stream is running and receiving its desired rate. When a 9 MB/s SRT boosted stream is introduced at time 40, the first stream drops to a little above 4 MB/s while the SRT boosted stream gets the 9 MB/s it wants after some small transient spikes in bandwidth. When the SRT boosted stream terminates at time 140, the bandwidth of the first stream returns to 11 MB/s immediately.

Figure 8 shows a scenario with more than one SRT boosted stream using the pessimistic method. Three 8 MB/s constant-rate streams and one 8 MB/s SRT boosted stream start at time zero. The second SRT boosted stream is introduced at time 80 and desires 6 MB/s. Its introduction caused the bandwidth of the first 3 streams to drop even further as they make room for the SRT boosted stream. The rate of the first SRT boosted stream fluctuates a little when the second SRT boosted stream is introduced as the system adjusts the bandwidth allocation. Both SRT streams receive their desired rates.

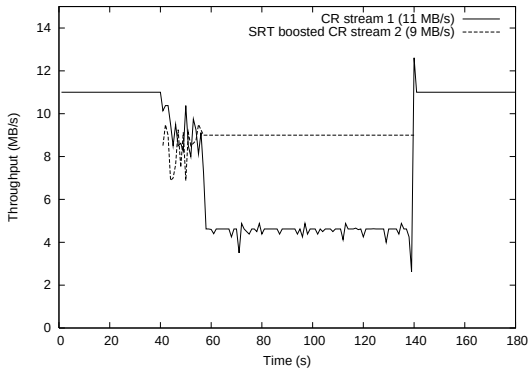


Figure 7: SRT boosted stream using pessimistic method enters at time 40 and terminates at time 140.

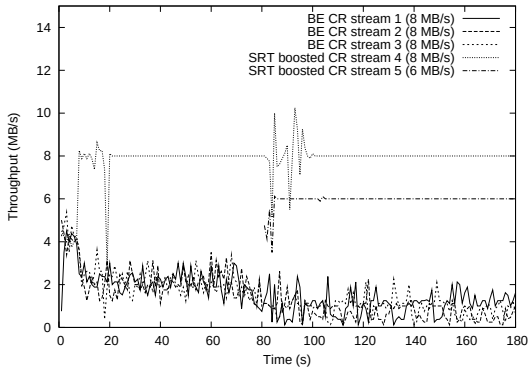


Figure 8: More than one SRT boosted streams using pessimistic method. Second SRT boosted stream starts at time 80.

7. Future Work

We plan to explore this idea further by refining our implementation. 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 another 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. In addition, the bandwidth management mechanism can be further refined to increase overall bandwidth utilization.

Although we are pleased with our current results, the feedback-based control mechanism using MDN is nevertheless an *ad hoc* heuristic-based scheme. Control theory has been applied in feedback-based real-time scheduling [14] and QoS management [13]. In the future we will be exploring more formal control-theoretic approaches to reap the benefits of having a theoretical foundation. This would allow the analysis of the parameters, control delays, and stability issues.

8. Conclusion

Modern disk drives are becoming increasingly intelligent. We believe that an approach to provide QoS for storage without relying on fine-grained external disk scheduling is needed to cope with the intelligent storage devices of the future. We presented a token-based approach that can better

support storage-bound soft real-time applications by controlling the rate non real-time applications can issue disk requests. Our results demonstrated that it is feasible to provide QoS for disk from the coarse-grained perspective of bandwidth management. This approach can be used to form a comprehensive framework for storage bandwidth management incorporating both reservation-based and feedback-based resource allocation schemes.

Acknowledgements

This research was supported in part by Lawrence Livermore National Laboratory, Los Alamos National Laboratory, and Sandia National Laboratory under contract B520714, and by Intel Corporation.

We are also grateful to our sponsors: The National Science Foundation, The USENIX Association, Hewlett Packard Laboratories, IBM Research, Microsoft Research, ONStor, Overland Storage, and Veritas.

References

[1] R. Abbott and H. Garcia-Molina. Scheduling I/O requests with deadlines: A performance evaluation. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS '90)*, pages 113–124, December 1990.

[2] 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)*, pages 46–60, January 2002.

[3] 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.

[4] 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 IEEE Real-Time Systems Symposium (RTSS '03)*, December 2003.

[5] 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 '98)*, pages 307–317, December 1998.

[6] J. Bruno, J. Brustoloni, E. Gabber, B. Ozden, and A. Silberschatz. Disk scheduling with quality of service guarantees. In *IEEE International Conference on Multimedia Computing and Systems*, volume 2, pages 400–405, June 1999.

[7] D. Clark, S. Shenker, and L. Zhang. Supporting realtime applications in an integrated services packet network: Architecture and mechanisms. In *Proceedings of the ACM SIGCOMM*, 1992.

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

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

[10] 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.

[11] 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.

[12] K. Kim, J. Hwang, S. Lim, J. Cho, and K. 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.

[13] B. Li and K. Nahrstedt. A control-based middleware framework for quality of service adaptations. *IEEE Journal on Selected Areas in Communications*, 17(9):1632–1650, 1999.

[14] C. Lu, J. Stankovic, G. Tao, and S. Son. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Systems Journal, Special Issue on Control-theoretical Approaches to Real-Time Computing*, 23(1/2):85–126, July/September 2002.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

[20] P. Shenoy and H. 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] R. Wijayaratne and A. L. Reddy. Integrated QOS management for disk I/O. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, volume 1, pages 487–492, June 1999.

[23] 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.