Feedback-based Dynamic Proportion Allocation for Disk I/O *

Dan Revel, Dylan McNamee, Calton Pu, David Steere, and Jonathan Walpole
{*revel,dylan,calton,dcs,walpole*}*@cse.ogi.edu*
Department of Computer Science and Engineering
Oregon Graduate Institute of Science and Technology
20000 NW Walker Road, PO Box 91000
Portland, OR 97291-1000

December 7, 1998

Abstract

In this paper we propose to use feedback control to automatically allocate disk bandwidth in order to match the rate of disk I/O to the real-rate [13] needs of applications. We describe a model for adaptive resource management based on measuring the relative progress of stages in a producer-consumer pipeline. We show how to use prefetching to transform a passive disk into an active data producer whose progress can be controlled via feedback. Our progress-based framework allows the integrated control of multiple resources. The resulting system automatically adapts to varying application rates as well as to varying device latencies.

1 Introduction

Real-rate applications [13] have specific disk I/O rate and throughput requirements that are driven by real-world demands. Examples of applications with real-rate disk I/O requirements include multimedia applications, real-time databases, and Internet servers. Real-rate applications suffer from uneven I/O throughput on conventional systems.

To secure predictable disk I/O rates, programmers have turned to systems that provide proportional reservations of disk bandwidth. However, reservations have a practical limitation: they depend on applications, and ultimately application programmers, to assign disk bandwidth allocations. The problem is that it is difficult to determine the correct allocation. The fact that an application's resource requirements may vary over time further complicates the matter. Our success with using feedback to dynamically control proportional CPU allocations without reservations [13] suggests a solution: using feedback to dynamically control disk bandwidth allocations.

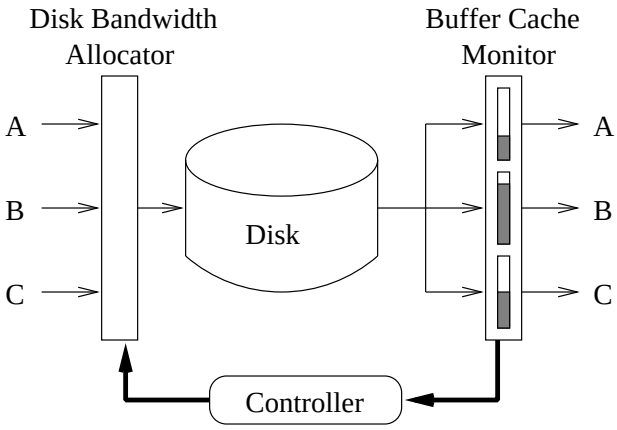
In this paper, we describe an automatic controller that dynamically adjusts proportional disk bandwidth allocations. The goal of our controller is to match disk I/O rates to the real-rate needs of applications. Rather than relying on reservations, our controller monitors each application's rate of progress and controls disk I/O rates by adjusting disk bandwidth allocations. Given a suitable metric of progress our feedback-based approach allows the system to automatically adjust to variations in both the application rate and the performance of the underlying disk subsystem.

The rest of this paper is organized as follows: Section 2 explains the model we use for adaptive resource management. Section 3 describes our feedback-based disk bandwidth allocator. Section 4 describes related work on resource management. Finally, section 5 discusses the current status of our research and presents some concluding remarks.

*This project was supported in part by DARPA contracts/grants N66001-97-C-8522, N66001-97-C-8523, and F19628-95-C-0193, and by Tektronix, Inc. and Intel Corporation.

2 Adaptive Resource Management

In general, data processing applications consist of a series of data producers and consumers. For example, a multimedia player consists of a movie on disk, the device driver that fetches the movie into the buffer cache, the user process that reads and decodes the movie, and finally a display device. Our overall goal is to ensure that data flows smoothly through this pipeline of producers and consumers. The way we propose to achieve this goal is to measure the relative progress of each pipeline stage and to adjust resource allocations to affect their relative rates. Real-rate applications include an externally clocked element in the pipeline that provides the rate that other stages adjust to match. In a data-flow application the relative progress between a producer and consumer can be measured by the fill level of a bounded buffer between them. We have written a CPU scheduler [13] that allocates CPU to different pipeline stages using a feedback controller that monitors fill levels and adjusts CPU allocations to meet the real-rate needs of the application. Our goal in this paper is to describe a feedback controller that ensures smooth flows of real-rate data from a disk device. To achieve this, the components we need are a feedback controller, a progress measure and an allocation mechanism.



In this figure, the disk bandwidth allocator is multiplexing three real-rate streams (A, B, and C) based on allocations set by the controller. The buffer cache is monitored to track the progress of each stream. Stream A is below its target fill level, the controller will increase its proportion. Stream B is above its target, its proportion will be decreased. Stream C is on target, its proportion will be kept constant.

Figure 1: A closed-loop feedback controller

	Fixed allocation	Dynamic allocation
Demand-driven	Real-time	Best-effort
Prefetched	Reserved	Real-rate

Table 1: Taxonomy of I/O streams

3 Disk Bandwidth Allocation

Figure 1 shows the high-level architecture of our disk bandwidth allocator. Our architecture contains three key components: the disk bandwidth allocator, the monitor, and the controller. The disk bandwidth allocator dispatches I/O requests in order to ensure that each client stream receives its assigned proportion of disk bandwidth during its period. The buffer cache monitor tracks the progress of each I/O request stream and measures the rate at which client applications consume the resulting data streams. The controller uses information from the buffer cache monitor to make 'automatic' adjustments of disk bandwidth allocations.

Our disk bandwidth allocator works by interposing between applications and the disk scheduler. We allocate bandwidth by controlling the rate at which streams can issue requests. This approach will work with any underlying disk scheduler, including SCAN or EDF. The following subsections discuss the remaining two components: the adaptive controller and the progress monitor.

3.1 Adaptive Controller

The job of the controller is to allocate disk bandwidth to ensure that each stream makes reasonable progress. Table 1 shows the four classes of I/O streams distinguished by our controller: real-time, best-effort, reserved, and real-rate. For these classes we describe the controller's goals and the policies it uses to achieve these goals:

Real-time and reserved streams can be modularly

supported by a separate I/O controller. All our system has to do is to respect these streams' allocation requirements. The main goal for these streams is to perform admission control to ensure that there are sufficient resources. If there are insufficient resources admission control rejects these streams. The goal for best-effort streams is to maximize their throughput given the available resources. Our controller's policy for thses streams is to evenly share the available bandwidth.

Real-rate streams are distinguished from the other classes by two characteristics. First, they have externally imposed rate requirements. Second, they export to the controller a progress metric which the controller uses to determine bandwidth allocation. The goal of the controller is to allocate bandwidth to streams in order to meet their real-rate requirements. When the system reaches overload the controller automatically scales back the allocated bandwidth using weighted fair share based on each stream's progress.

3.2 Enabling Feedback Control of Real-Rate Streams Using Prefetching

The goal of our controller is to match the rate at which devices produce data to the real-rate needs of consuming applications. Demand-driven I/O is a problem for the controller because the producer and the consumer are synchronous, therefore no relative control is possible. Prefetching separates the production of real-rate streams from their consumption, and it introduces a prefetch buffer which can be monitored to determine relative progress. In effect, prefetching transforms passive demand-driven devices into active producers that can be controlled by feedback.

While prefetching may be inferred, as is common for sequentially accessed files, we allow the use of informed interfaces that enable applications to communicate complex access patterns to the prefetcher. The informed interface we have implemented is called synthetic files [7]. In addition to expressing complex access patterns, synthetic files can describe how to adapt the bandwidth requirements of a data stream. For example, a synthetic file describing a video stream can provide a control that adapts the frame rate by specifying which frames to drop. In overload situations for real-rate applications, we can use a separate feedback controller to scale the synthetic file's bandwidth in order to fit the real-rate application's requirements to within the available resources.

The goal of the controller is to match the rate at which the prefetcher produces data to that of the real-rate consumer. In order to meet this goal, the prefetcher needs to remain asynchronous from the consumer. This means that it has to adjust its prefetch buffer size according to the application's rate and the device latency to avoid buffer misses. We do this according to the following equation, adapted from TIP [11]:

$$BufferSize = 2 * DeviceLatency * ObservedRate$$

Our feedback controller adjusts the rate of the prefetcher in order to keep the prefetch buffer half full. The resulting system automatically adapts to varying application rates as well as to varying device latencies.

4 Related Work

Both Anderson [1] and Jones [4] use resource planners to coordinate the reservation and scheduling of a set of resources on behalf of applications. These planners are reservation-based. They rely on applications to specify their resource requirements and in cases of overload must re-negotiate the set of reserved resources. Instead of using reservations, our system uses a metric of progress, such as buffer fill levels, exposed by applications, to discover their resource requirements and dynamically adjust resource allocations to match the application's real-rate needs.

Rather than using resource reservations, the Odyssey system [10] seeks to support agile application-aware adaptation by monitoring the amount of available resource (e.g. network bandwidth) and then allocating the available resource between competing applications according to their stated resource expectations. If an application's resource expectations cannot be met, an upcall is made to the application requesting a change in fidelity, essentially telling the application to adapt its resource

requirements. In contrast to Odyssey, our resource allocation model is based on an application's measured progress rather than its stated requirements. In addition, synthetic files allow our system to automatically adapt the bandwidth requirements of a data stream to fit within available resources.

Real-time systems can also be used to support real-rate applications. The trade-off is that applications have to use reservations to specify their resource requirements. Examples of real-time file systems include RT-Mach [8] and Cello [12]. The RT-Mach file-system [8] uses a "Just-in-Time" slack-stealing algorithm for disk scheduling that is an EDF-SCAN hybrid and supports bandwidth reservations with variable periods. The Cello file-system [12] has a two-level architecture that uses deadlines and slack-stealing to allocate disk bandwidth between class specific schedulers that implement policies for different classes of applications. We could use either of these systems as an underlying mechanism to provide bandwidth control to our feedback controller. This alleviates the application task of specifying reservations.

Feedback scheduling of system resources has been investigated by a number of researchers [6, 2, 3]. Our system extends this work to a new resource. In addition, the progress-based framework allows us to integrate the control of multiple resources. In our system, the bandwidth allocation is ultimately driven by the real-rate pipeline scheduled by the feedback-controlled CPU allocator.

The use of access pattern information to drive prefetching is not new. Kotz investigated automatic detection and prediction of complex access patterns [5]. Because of the limits of automatic prediction-based approaches, more recent file-system research has proposed informed interfaces. TIP [11], for example, uses "disclosure hints" of upcoming accesses to drive its buffer management policy. Other examples of informed interfaces include dynamic sets [14], and the Galley file system [9]. Synthetic files is a generalization of informed interfaces for file systems prefetching Synthetic files extend informed interfaces by providing a mechanism to adapt the bandwidth requirement of the data stream.

5 Conclusions

Conventional operating systems perform poorly for applications that need real-rate disk I/O. Our feedback-based approach allows the system to automatically adjust to variations in both the real rate of the application and the performance of the underlying disk subsystem.

The thing that enables this feedback-based approach, which is based on monitoring the progress of producers and consumers, is to turn the disk into a variable rate producer using adaptive prefetching.

We are currently constructing a research prototype.

References

[1] David P. Anderson. Metascheduling for continuous media. *ACM Transactions on Computer Systems*, 11(3):226–252, August 1993.

[2] Shanwei Cen. *A Software Feedback Toolkit and its Application In Adaptive Multimedia Systems*. PhD thesis, Oregon Graduate Institute of Science and Technology, October 1996.

[3] Ashvin Goel, David Steere, Calton Pu, and Jonathan Walpole. Adaptive resource management via modular feedback control. *Simultaneously submitted to Hot OS*, 1999.

[4] Michael B. Jones, Paul J. Leach, Richard P. Draves, and Joseph S. Barrera, III. Support for user-centric modular real-time resource management in the rialto operating system. In *Proceedings of NOSSDAV'95*, April 1995.

[5] David Kotz. *Prefetching and Caching Techniques in File Systems for MIMD Multiprocessors*. PhD thesis, Duke University, April 1991. Available as technical report CS-1991-016.

[6] H. Massalin and C. Pu. Fine-grain adaptive scheduling using feedback. *Computing Systems*, 3(1):139–173, Winter 1990. Special Issue on selected papers from the Workshop on Experiences in Building Distributed Systems, Florida, October 1989.

[7] Dylan McNamee, Dan Revel, Calton Pu, David Steere, and Jonathan Walpole. Synthetic Files: Enabling Low-Latency File I/O for QoS-Adaptive Applications. Technical Report CSE-98-012, Dept. of Computer Science and Engineering, Oregon Graduate Institute of Science and Technology, 1998.

[8] Anastasio Molano, Kanaka Juvva, and Raj Rajkumar. Real-Time Filesystems: Guaranteeing Timing Constraints for Disk Accesses in RT-Mach. In *Proceedings of the IEEE Real-Time Systems Symposium*, December 1997.

[9] Nils Nieuwejaar and David Kotz. The Galley parallel file system. *Parallel Computing*, 23(4):447–476, June 1997.

[10] Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker. Agile application-aware adaptation for mobility. In *Proceedings of the 16th ACM Symposium on Operating System Principles*, October 1997.

[11] R. Hugo Patterson, G. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed prefetching and caching. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, December 1995.

[12] Prashant Shenoy and Harrick Vin. Cello: A Disk Scheduling Framework for Next Generation Operating Systems. In *Proceedings of SIGMETRICS'98*, June 1998.

[13] David Steere, Ashvin Goel, Joshua Gruenberg, Dylan McNamee, Calton Pu, and Jonathan Walpole. A Feedback-driven Proportion Allocator for Real-Rate Scheduling. In *Proceedings of the 1999 Symposium on Operating Systems Design and Implementation*. USENIX Association, 1999.

[14] David C. Steere. Exploiting the Non-Determinism and Asynchrony of Set Iterators to Reduce Aggregate File I/O Latency. In *Proceedings of the 16th ACM Symposium on Operating System Principles*, October 1997.