Cello: A Disk Scheduling Framework for Next Generation Operating Systems *

Prashant J. Shenoy and Harrick M. Vin

Distributed Multimedia Computing Laboratory
Department of Computer Sciences, University of Texas at Austin
Taylor Hall 2.124, Austin, Texas 78712-1188
E-mail: {shenoy,vin}@cs.utexas.edu, Telephone: (512) 471-9732, Fax: (512) 471-8885
URL: http://www.cs.utexas.edu/users/dmcl

Abstract

In this paper, we present the Cello disk scheduling framework for meeting the diverse service requirements of applications. Cello employs a two-level disk scheduling architecture, consisting of a class-independent scheduler and a set of class-specific schedulers. The two levels of the framework allocate disk bandwidth at two time-scales: the class-independent scheduler governs the coarse-grain allocation of bandwidth to application classes, while the class-specific schedulers control the fine-grain interleaving of requests. The two levels of the architecture separate application-independent mechanisms from application-specific scheduling policies, and thereby facilitate the co-existence of multiple class-specific schedulers. We demonstrate that Cello is suitable for next generation operating systems since: (i) it aligns the service provided with the application requirements, (ii) it protects application classes from one another, (iii) it is work-conserving and can adapt to changes in work-load, (iv) it minimizes the seek time and rotational latency overhead incurred during access, and (v) it is computationally efficient.

1 Introduction

Since the invention of movable head disks, several algorithms have been developed to improve I/O performance through intelligent scheduling of disk accesses. These algorithms can be broadly divided into two classes:

1. Disk scheduling algorithms optimized to service best-effort requests: The simplest of these algorithms is First Come First Served (FCFS), that schedules requests in the order of their arrival. Since the access schedule thus derived is independent of the relative positions of the requested data on disk, FCFS scheduling can incur significant seek time and rotational latency overhead. This limitation has been addressed by several disk scheduling algorithms, such as Shortest Seek Time First (SSTF), SCAN, LOOK, V(R), etc., that schedule requests to minimize seek time [6, 7, 8, 9, 10, 18, 19, 20]; and Shortest Total/Access Time First (STF/SATF), Aged Shortest Access Time First (ASATF), etc., that schedule requests to minimize the total seek time and rotational latency overhead [11, 17].

2. Disk scheduling algorithms optimized to service requests with real-time deadlines: The simplest of these algorithms is Earliest Deadline First (EDF) [14]. EDF schedules requests in the order of their deadlines but ignores the relative positions of requested data on disk in deriving the access schedule. Hence, it can incur significant seek time and rotational latency overhead. This limitation has been addressed by several disk scheduling algorithms, including Priority SCAN (PSCAN), Earliest Deadline SCAN, Feasible Deadline SCAN (FD-SCAN), SCAN-EDF, Shortest Seek Earliest Deadline by Order/Value (SSEDO, SSEDV) [1, 4, 5, 16], etc. These algorithms start from an EDF schedule and reorder requests so as to reduce the seek and rotational latency overhead without violating request deadlines.

Unlike the systems for which these scheduling algorithms were designed, today's general purpose file and operating systems simultaneously support applications with diverse performance requirements [3, 15]. For instance, a typical file server today services requests from interactive best-effort applications (e.g., word processors); real-time applications (e.g., video and audio players); and file transfer applications (e.g., http servers). Interactive applications require the file server to minimize the average response time of requests. Real-time video playback applications require the file server to retrieve successive video frames prior to their playback instants (i.e., deadlines). However, due to the periodic nature of video playback, these applications do not benefit if the frames are retrieved much prior to their deadlines. Finally, file transfer applications require the server to provide high throughput across several requests, but are less concerned about the response times of individual requests.

With the many-fold increase in CPU processing power, network bandwidth, and disk capacity, it is inevitable that general purpose computing environments of the future will support applications of even greater complexity and diversity. We can anticipate that next generation file systems will support applications that process massive amounts of data for visualization and support real-time interactivity. For instance, a repository of satellite imagery might be accessed and processed by programs for feature extraction and real-time visualization; an application for interactive navigation through virtual environments will issue requests for the storage and retrieval of heterogeneous information objects (e.g., imagery, 3-D models, video, etc.) from distributed file servers under real-time constraints. Since most conventional disk scheduling algorithms are optimized for a single performance criterion, they are ineffective at simultaneously supporting applications with such diverse requirements.

*This research was supported in part by an AT&T Foundation Award, IBM Faculty Development Award, Intel, the National Science Foundation (Research Initiation Award CCR-9409666 and CAREER award CCR-9624757), Lucent Bell Laboratories, NASA, Mitsubishi Electric Research Laboratories (MERL), and Sun Microsystems Inc.

Most of the techniques developed to-date for addressing this problem employ simple adaptations of conventional disk scheduling algorithms. To illustrate, consider a mix of real-time and best-effort applications. A real-time disk scheduling algorithm can be adapted to service these applications by modeling the requests generated by best-effort applications as a periodic task with deadlines [13]. This modeling, however, is non-trivial and introduces artificial constraints that reduce the effectiveness of the system. Another common approach for servicing the mix of real-time and best-effort applications is to employ a scheduler that assigns priorities to application classes and services disk requests in the priority order. Unfortunately, such schedulers may violate service requirements of requests and induce starvation [2]. Finally, simply enhancing a conventional scheduler by allocating time-slices to service requests from different application classes may incur substantial seek and rotational latency overhead (for short time-slices) or yield unacceptable response times (for long time-slices) (see Figure 1).

In this paper, we present the *Cello* disk scheduling framework for simultaneously supporting applications with diverse requirements. Cello employs a *two-level disk scheduling architecture*, consisting of a *class-independent* scheduler and a set of *class-specific* schedulers. The two levels of the framework allocate disk bandwidth at two time-scales: the class-independent scheduler governs the *coarse-grain bandwidth allocation* to application classes, while the class-specific schedulers control the *fine-grain interleaving* of requests from the application classes to align the service provided with the application requirements. The two levels of the architecture separate application-independent mechanisms from application-specific scheduling policies, and thereby facilitate the co-existence of multiple class-specific schedulers.

We demonstrate that Cello is suitable for next generation operating systems since: (i) it aligns the service provided with the application needs, (ii) it protects application classes from one another, (iii) it is work-conserving and can adapt to changes in work-load, (iv) it minimizes the seek time and rotational latency overhead incurred during access, and (v) it is computationally efficient.

The rest of the paper is organized as follows: The requirements for a disk scheduling algorithm for next generation operating systems are derived in Section 2. Section 3 describes and analyzes the Cello disk scheduling framework. Section 4 presents the results of our experiments. Finally, Section 5 summarizes our results.

2 Requirements for a Disk Scheduling Algorithm

To determine a suitable disk scheduling algorithm, consider the requirements imposed by applications likely to be simultaneously supported by general purpose file and operating systems of the future:

- *Real-time applications:* These applications require the operating system to provide performance guarantees. Depending on the strictness of the requirements, these applications can be classified as either *hard real-time* or *soft real-time* applications. Whereas hard real-time applications require deterministic guarantees for the response time of each disk request, soft real-time applications require statistical guarantees. Request generation in these applications can either be *periodic* or *aperiodic*, and the applications may consume data immediately following its availability or at predefined instants. For example, video playback is a periodic, soft real-time application, in which the accessed video frames are consumed at predefined instants (determined by the video playback rate and the consumption instants of previous frames). In contrast, applications that support interactive navigation through virtual environments yield real-time requests with low average response time requirements.

  EDF and fixed priority schedulers are suitable for hard real-time applications [14], while scheduling algorithms such as FD-SCAN and SSEDV/SSEDO are are suitable for soft real-time applications [1, 5]. Just-in-time schedulers (which schedule requests just prior to their deadlines) are desirable for real-time applications that initiate data consumption at deadlines (e.g., video playback). Finally, algorithms that schedule requests at the earliest possible instants prior to their deadlines, and thereby minimize the response time while meeting the real-time requirements, are suitable for interactive real-time applications.

- *Best-effort applications*: These applications do not need performance guarantees. They can be further classified as either *interactive* or *throughput-intensive*. Interactive applications require low average response times. Throughput-intensive applications require the file system to sustain high throughput across multiple requests, but are less concerned about the response times of individual requests. For instance, word processors are interactive best-effort applications, while file transfer is a throughput-intensive best-effort application.

  Conventional disk scheduling algorithms such as SCAN, SSTF, SATF, etc. are suitable for these applications.

From this, we conclude that different policies are suitable for scheduling disk requests from different application classes. Hence, to align the service provided with the application needs, a disk scheduling framework should employ different policies for different application classes. Furthermore, such a framework should protect application classes from one another. For example, bursty arrival of best-effort requests should not cause deadline violations for real-time requests; and the arrival of a burst of real-time requests should not starve best-effort requests.

These requirements can be met by partitioning disk bandwidth among the application classes, and then employing an application-specific policy to schedule requests within each partition. However, the granularity of partitioning should be chosen such that (1) the seek time and rotational latency overhead incurred while servicing requests is minimized, and (2) the service provided is aligned to the application requirements. Finally, to efficiently utilize disk bandwidth, the framework must be *work-conserving* (i.e., it should utilize the idle disk bandwidth available to one application class to schedule pending requests from another class); and should adapt to changes in the work-load.

In summary, a disk scheduling algorithm suitable for next generation operating systems: (i) should align the service it provides with the application needs, (ii) should protect application classes from one another, (iii) should be work-conserving and should adapt to changes in work-load, (iv) should minimize the seek time and rotational latency overhead incurred during access, and finally (v) should be computationally efficient. In what follows, we present a disk scheduling framework that meets these requirements.

3 The Cello Disk Scheduling Framework

3.1 Architectural Principles

Cello achieves the above objectives by allocating disk bandwidth to application classes at two time-scales. At the coarse time-scale, it determines the number of requests from each application class to be serviced, and at the fine time-scale, it determines the order for servicing the set of requests from the application classes. Whereas the former enables Cello to protect application classes from one another as well as adapt disk bandwidth allocation with changing work-load, the latter enables it to align the service provided to the application requirements while minimizing the seek time and rotational latency overhead.