

# Automatic I/O Scheduler Selection for Latency and Bandwidth Optimization

Seetharami Seelam, Jayaraman Suresh Babu, Patricia Teller  
{seelam, jsuresh, pteller}@utep.edu  
Department of Computer Science  
The University of Texas at El Paso  
El Paso, TX 79968

**Abstract**—Disk systems are increasingly accessed by multiple workloads with diverse characteristics and requirements. The Linux 2.6 kernel provides four disk I/O schedulers that cater to varied workloads but only one can be active at any time. As a consequence of this fixed scheduler scheme, potentially, some workloads obtain suboptimal performance. Currently, there is no automatic mechanism that can activate the appropriate scheduler at the appropriate time. We present a novel approach called ADIO that provides latency guarantees while providing fair disk allocation to each I/O-generating process. DASS, a feedback-based controller, sits on top of the deadline and CFQ schedulers and activates one of them based on observed system response to I/O requests, while ensuring latency and bandwidth guarantees. Our prototype implementation in the Linux kernel shows improved I/O resource utilization, while enforcing strict latency bounds.

## I. INTRODUCTION

The Linux 2.6 release provides four disk I/O schedulers: anticipatory, deadline, completely fair queuing (CFQ), and noop along with an option to select one of these four at boot time or runtime. The selection is based on *a priori* knowledge of the workload, file system, and I/O system hardware configuration, among other factors. In the absence of a selection, one of the schedulers is provided as the default.

To provide best possible performance, I/O scheduler selection must exploit the workload characteristics, hardware configuration of the I/O system, file system etc., Although the Linux kernel (releases 2.6.11 and above) provides an interface to dynamically switch among I/O schedulers, diverse workload requirements and their I/O characteristics, and the idiosyncrasies of the hardware and software make such selection a daunting task, at best.

Therefore, automating scheduler selection and doing so dynamically based on workload requirements is an important objective. The question, however, is whether automatic and dynamic scheduler selection is feasible even for two schedulers.

This paper presents preliminary results of our work towards automatic and dynamic scheduler selection based on workload and system needs, and hence, attempts to answer the above-stated question. Note that such a scheduler selection methodology would be progress in the right direction if it

could achieve any performance improvement over the default scheduler. Providing a generalized and complete solution for the entire set of I/O schedulers is out of the scope of this paper, however, we present preliminary results w.r.t selection of one of two schedulers.

In mixed workload and multi-user systems it is critical to provide request response time with a bounded latency and maximize the utilization of the I/O subsystem. The deadline scheduler is designed to provide the bounded request latency, while CFQ is designed to share bandwidth equally among all active I/O applications, thus, increasing the utilization of the I/O system. However, scheduling optimized for bounded latency (as in the deadline scheduler) may result in reduced utilization of I/O resources. On the other hand, fair queuing scheduling (such as CFQ in the Linux kernel or SFQ, etc.) improves I/O resource utilization but may not provide bounded latency. We propose to select one of two schedulers, one that optimizes for bounded latency and one that provides fair queuing scheduling, according to the following two criteria.

1. provide maximum request latency within a certain bound.
2. provide best throughput when the request latency falls below the latency bounds.

Thus, our Automatic and Dynamic I/O scheduler selection algorithm, called ADIO, is optimized for bounded per-request latency as well as disk utilization.

To evaluate this idea, we implemented ADIO in the Linux kernel to select between the deadline and CFQ schedulers. This selection is automatic and dynamic based on feedback from the I/O system and the workload. The implementation was tested on a RAID-0 system. The testing shows that ADIO is able to switch between the schedulers and provide bounded latency and maximal throughput, which cannot be achieved with just any one of the schedulers.

The paper is organized as follows. Section 2 motivates the need for automatic and dynamic I/O scheduler selection, while Section 3 describes our methodology for scheduler selection. Section 4 presents an experimental evaluation of our methodology on a set of synthetic benchmarks and real workloads. Conclusions and future work are presented in Section 5.

## II. THE NEED FOR AUTOMATIC AND DYNAMIC SCHEDULER SELECTION

The Linux 2.6 release provides four disk I/O schedulers to cater to diverse workload requirements in terms of throughput, per-request latency, average request latency, etc. The anticipatory scheduler (AS) is the default for kernels downloaded from kernel.org; it has been shown to provide best global throughput under several situations [6, 7, 11]. However, for kernels distributed by some of the major Linux operating system vendors, such as Novel, SUSE and RedHat, CFQ is the default scheduler as it offers the best global throughput and minimal per-request latency f

period  $t$  (i.e., after every  $W$  seconds), the monitor collects the following information:

- maximum latency for any read request,  $L_{r-\max}(t)$ ,
- maximum latency for any write request,  $L_{w-\max}(t)$ ,
- number of read requests serviced,  $N_r(t)$ , and
- number of write requests serviced,  $N_w(t)$ .

The above statistics are used by the controller (described below) to determine system status for the given time period as well as to decide if the active scheduler should be swapped out to accommodate latency and bandwidth guarantees.

2. Scheduler Selection Controller: After every adaptation time period,  $W$ , the scheduler selection controller ~~makes a decision~~

scheduler cannot maintain fairness in distributing the available I/O resources among all competing processes.

### B. I/O Schedulers: Completely Fair Queuing (CFQ) Scheduler

The CFQ scheduler maintains one queue for every process class making I/O requests. A process class can be created based on process group id, thread group id, user id, or group id. During an enqueue operation, the request is inserted in FIFO order into a queue indexed by its process's class id. During a dequeue operation, a set of requests from each of the non-empty queues is selected, sorted, and placed on a dispatch list, and is later sent to the disk controller. The number of requests fetched from each class is controlled by a tunable parameter, *quantum*.

The CFQ scheduler aims to distribute the available I/O bandwidth equally among all process classes in the system. However, it does not impose any bound on per-request latency. This scheduler is used mainly in database applications that do not require real-time response [see, e.g., 5 and 12]. It also provides better I/O system utilization than does the deadline scheduler [12].

### C. DASS Components

DASS consists of two components: a request statistics collection monitor and an I/O scheduler selection controller. These components determine when the current active scheduler should be swapped to satisfy the deadline requirements or to impose fair bandwidth when request latencies are within the latency bounds. These two components are described further below.

1. Request Statistics Collection Monitor: The monitor records a number of statistics reported by the block layer of each I/O scheduler. Every  $W$  seconds the block layer records these statistics based on I/O arrivals from workloads and I/O completions reported by the underlying storage system. We call  $W$  the *length of the adaptation time period*.  $W$  is a tunable parameter; we set  $W = 1$ . At the end of every adaptation time

cache, running Linux 2.6.11. A total of four processors are used in the study. To eliminate the interference of I/O requests from the operating system (OS), all benchmarks are configured to access drives that are different from the disk hosting the OS. The external device is a RAID-0 with four Maxtor IDE 7.2K RPM 20GB drives. It is configured with an ext3 file system; to remove cache effects, prior to each experiment, the file system is unmounted and remounted.

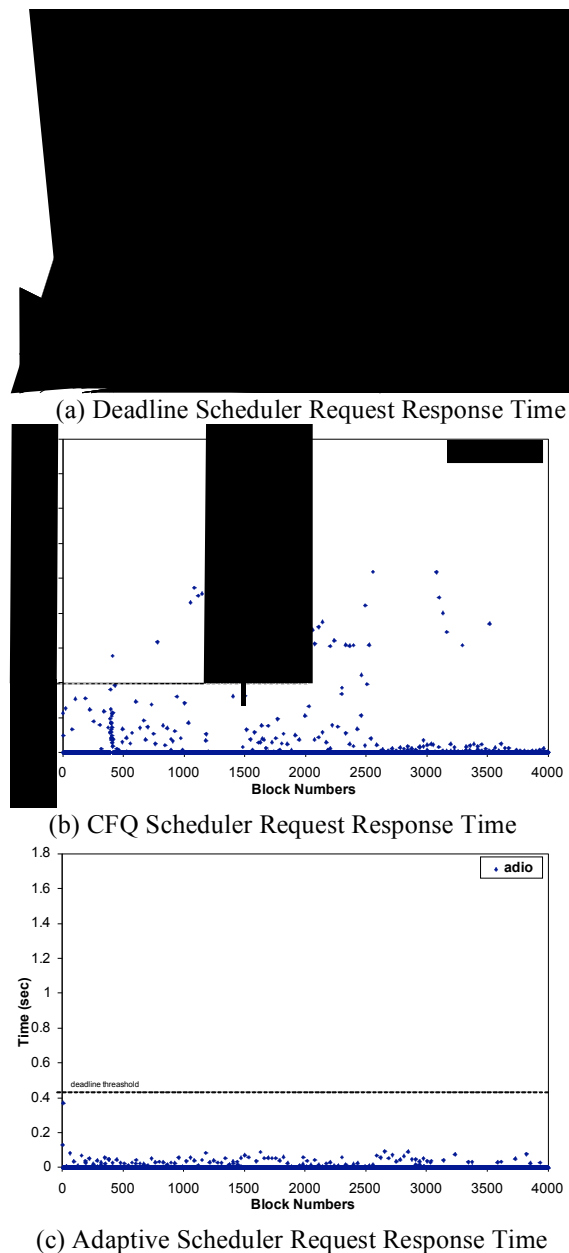


Figure 2: Request Response with Different Schedulers

#### A. Enforcing Latency Bounds with ADIO Scheduler Selection

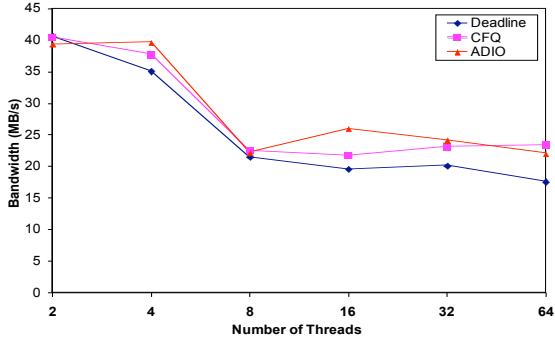
This experiment focuses on guaranteeing latency bounds with the ADIO methodology. Since latency bounds are imposed on a per-request basis by the DASS controller, to

show ADIO’s effectiveness, we compare request response times of ADIO scheduler selection with those of the deadline and CFQ schedulers. For this purpose, we use tiobench, a file system benchmark that tests I/O performance using multiple threads that make simultaneous accesses in a sequential or random fashion. Given the number of blocks to read/write, read/write pattern (random or sequential), the block size, and the number of threads, this benchmark executes the operations and reports the average bandwidth provided by the system, average latency of request responses, and maximum latency of all accessed blocks. In order to determine whether or not ADIO provides a better latency, we augmented tiobench to report the latency of every request. tiobench was configured to read 2 GBs of data in 4KB blocks with 32 concurrent threads, which results in 8192 read requests. Request latency response time is shown in Figures 2 (a), (b), and (c) for the deadline scheduler, CFQ scheduler, and ADIO scheduler selection, respectively. We plot every other request response time to reduce the x-axis size by half. The 400ms deadline bound for each read request is represented by the dotted horizontal line in the figures. Observe that both the deadline and CFQ schedulers have several requests with latencies that exceed the deadline bound. In contrast, ADIO scheduler selection strictly enforces the bound by swapping between the schedulers an average of six times; hence, all requests have response times below the dotted line. We should reiterate that if the workload requirements exceed the I/O system capacity, ADIO scheduler selection will not be able to enforce the latency bounds. After all, these are enforced in ADIO scheduler selection by the underlying deadline scheduler. Now, the interesting question is: What is the impact of ADIO scheduler selection on the bandwidth of the application? This is explored in the next section.

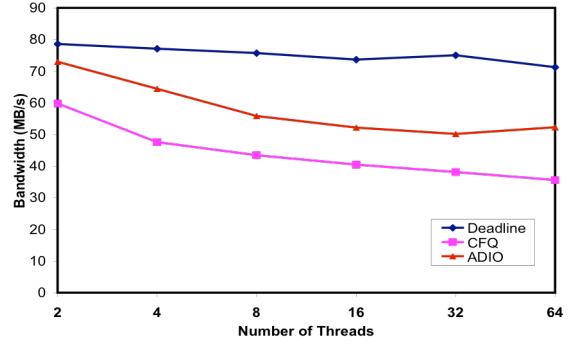
#### B. Bandwidth Optimization with ADIO Scheduler Selection

This experiment compares the disk utilization of the three scheduling methodologies under four different workload profiles. For this experiment, we configure tiobench to access 2GBs of data in 4KB blocks using 2, 4, 8, 16, 32, and 64 concurrent threads that execute sequential reads, random reads, sequential writes, and random writes. The bandwidth reported by each of the scheduling methodologies is shown in Figures 3 (a), (b), (c), and (d), respectively. The general observation is that in all the cases ADIO, deadline, and CFQ have some disk utilization differences. Now to the specifics: for two of the four profiles, i.e., for sequential writes and random writes, which have a four-second deadline bound for each request, ADIO has better disk utilization than the default CFQ scheduler. For the other two profiles, the percentage differences among the schedulers corresponding to the reads are much smaller. CFQ and deadline have better utilization, but as shown in Figure 2 (a) and (b), for streaming reads, although these schedulers achieve higher disk utilization, several requests are not serviced in the latency bounds. In contrast, ADIO, by enforcing strict latency bounds, results in slightly lower disk utilization. ADIO clearly provides the bounded latency and optimizes for the disk utilization.

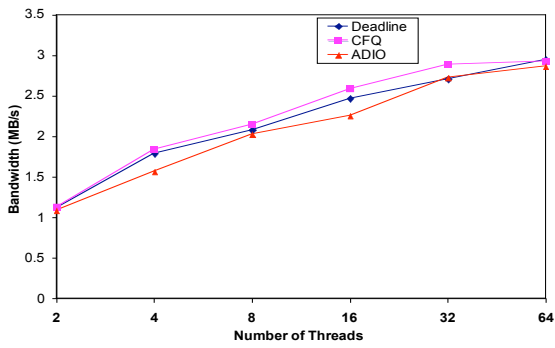




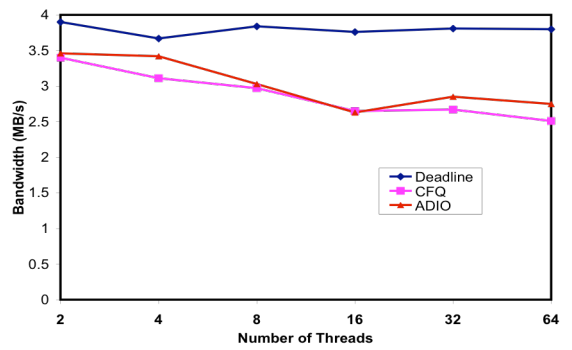
(a) Sequential Reads – Disk Utilization



(c) Sequential Writes – Disk Utilization



(b) Random Reads – Disk Utilization



(d) Random Writes – Disk Utilization

Figure 3: ADIO, Deadline, CFQ Performance Comparison — Different Workloads