Disk Access Analysis for System Performance Optimization

DANIEL L. MARTENS AND MICHAEL J. KATCHABAW
Department of Computer Science
The University of Western Ontario
London, Ontario
CANADA
{dlmarten,katchab}@csd.uwo.ca http://www.csd.uwo.ca

Abstract: - As the gap between processor and disk performance continues to grow in modern computing systems, so too does the need for improvements in disk performance management. In an effort to remove or reduce the performance bottleneck created by disk accesses, new approaches and algorithms for disk scheduling have been developed in recent years. While providing performance improvements, these approaches each have their own strengths and weaknesses that ultimately limit their applicability and usefulness across a wide variety of system workloads.

This paper introduces a new method of disk access optimization which focuses primarily on dynamic scheduling algorithm selection and algorithm tuning. Disk activity is continuously collected in real-time and cached for later analysis to discover current system load patterns, while a scoring system is used to detect overall trends by system processes. Once analysis is complete, disk scheduling algorithms are automatically selected and/or tuned based on heuristics or criteria to ensure that the disk scheduling algorithm in use is well suited to the current workload of the system. Experimentation to date has been quite positive, demonstrating this approach has great potential for assisting in the optimization of system performance.

Key-Words: - Disk scheduling, disk analysis, disk pattern recognition, system performance optimization

1 Introduction

While processing speed in computing systems has been increasing at a rapid pace for the last two decades, relatively little has been done to improve access to stable and reliable mass storage. On average, processing speeds tend to increase 55% per year, while disk access speeds only increase a mere 7% [5]. Even though cost effective disk space has increased dramatically, access to secondary non-volatile memory is in the order of magnitudes slower than access to faster primary memory. This has created a very large bottleneck in modern computers. It is no longer sufficient to simply store the data; high performance access to it must be provided regardless of size or location to support the rigors of modern computing [2].

New approaches and algorithms for disk scheduling have been developed in recent years in an attempt to eliminate or reduce this performance bottleneck. Most current approaches to disk scheduling implement algorithms that attempt to provide the best-overall performance for every possible workload. This is not possible; however, as operating system workloads are varied, diverse, and change over time, and a single algorithm cannot appropriately service all of them. For example, a scheduler which best serves random pattern disk activity will not provide the same performance under sequential activity. There are simply many fundamental trade-offs in disk scheduling algorithm design that ultimately limit the effectiveness of a single approach on its own [7].

This paper introduces a new approach to disk access optimization to overcome this. Instead of creating algorithms to best service all activity, our approach is to utilize existing algorithms depending on their respective strengths and performance benefits under each different workload scenario. In essence, the disk scheduling algorithm in use is selected and tuned based on the observed workload and performance of the system. If the scheduling algorithm in use cannot appropriately handle the current system workload, it can be quickly swapped out and replaced by one that can. This allows our approach to disk scheduling to effectively exploit the strengths of all disk scheduling algorithms while masking their weaknesses.

At the core of our approach is a real-time analysis engine which detects disk access patterns and imposes decision heuristics to inform the underlying system when to switch algorithms and which available algorithm is best suited for the current workload. The system in turn loads and unloads algorithms on demand as requested. This requires the modularization of algorithm code in a shared library type of approach in which algorithms

must register themselves with the scheduler and be prepared to run at any point in time. The requisite support mechanisms are becoming available in modern operating systems, and are currently available in Linux [1], which was the target platform for our prototyping and proof of concept efforts.

The remainder of this paper is structured as follows. Section 2 presents a brief overview of background and related work in this area. Section 3 describes our I/O Analyzer system (IOAZ), capable of carrying out disk access analyses in real-time and switching and tuning disk scheduling algorithms. This includes architectural and implementation details, along with experimentation used to calibrate disk access pattern detection and algorithm selection. Section 4 presents results from using IOAZ to date. Section 5 concludes this paper with a summary and discussion of future work in this area.

2 Background and Related Work

Several classic disk scheduling algorithms are discussed at length in [10], including First Come, First Serve (FCFS), Shortest Seek Time First (SSTF), SCAN, Cyclic SCAN (C-SCAN), LOOK, and Cyclic LOOK (C-LOOK), each with their own performance characteristics. While being relatively simplistic, these algorithms can still provide solid performance under certain workloads. For example, FCFS performs quite well when disk accesses are dominated by a single process making sequential requests, as discussed later in this paper.

Newer approaches have emerged to provide more stringent and flexible performance controls to disk schedulers. These approaches include Yet another Fair Queuing (YFQ) [3], the Anticipatory Scheduler (AS) [6], and the Deadline Scheduler (DEAD) and Complete Fair Queuing (CFQ) created for Linux by Jens Axboe and discussed in [1], which apply queuing models, deadlines, and advanced heuristics to the problem of disk scheduling. Other approaches have been introduced that take into account physical disk characteristics (such as arm positioning time) into scheduling decisions [4].

The above approaches to disk scheduling each have their own strengths and weaknesses. As a result, there is no clear winner across all possible disk access workloads. Our approach to disk scheduling overcomes this by, in essence, providing a meta-scheduling framework that does not delve into the details of scheduling, but rather focuses on the pattern of activity created by the current workload and the selection of an appropriate disk scheduling algorithm that excels at scheduling that particular type of workload.

3 The I/O Analyzer

In the previous section, we examined the current state of disk scheduling. Most approaches employ algorithms which are best suited for a particular pattern of disk accesses. Some work best for sequential accesses, others for random activity, and others attempt to provide the best overall throughput and provide reasonable performance for both sequential and random accesses.

If these patterns of disk accesses could be detected in real-time, then the system could select the disk scheduling algorithm best suited to this workload, and improve performance accordingly. This is the basis of our I/O Analyzer (IOAZ), discussed in this section.

3.1 Architecture and Design of IOAZ

IOAZ is a collection of dependent modules to collect real-time disk access information, perform analysis of collected data and allow administrative users to view or edit configuration of the entire system. Each module is responsible for a distinct area of the overall IOAZ architecture. This architecture is shown in Figure 1.

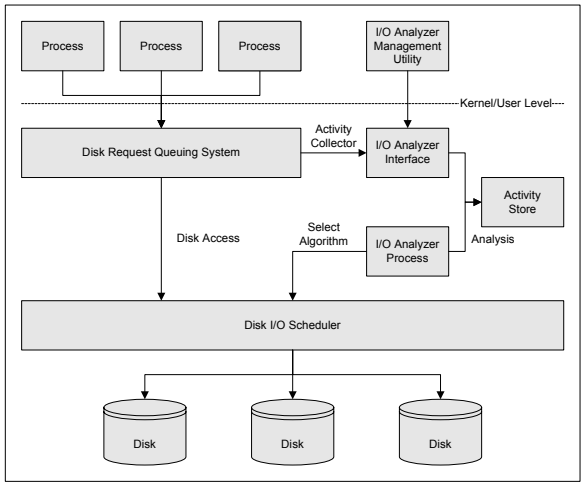

Figure 1. IOAZ Architecture

Without IOAZ present, as system processes submit requests for disk accesses, these requests are submitted to the Disk Request Queuing System to wait for further processing and dispatching. When a new disk request can be serviced, the Disk I/O Scheduler determines the next request to process according to the scheduling algorithm currently in use, fetches it from the Disk Request Queuing System, and then goes to the appropriate disk to satisfy the request. Results then trickle back to the requesting process upon completion.

The presence of IOAZ does not disrupt this flow of activity. IOAZ was designed to easily integrate with the existing disk I/O subsystem of the host operating system without disrupting its operation or consuming great quantities of memory or processing resources. Hooks placed within the Disk Request Queuing System inform the I/O Analyzer Interface of any requests which pass through the layer. IOAZ quickly accounts for the request information in its internal Activity Store and returns control so as not to delay request processing.

Actual analysis of collected data occurs in a passive fashion through a background process, the I/O Analyzer Process, which executes periodically. When it executes, it analyzes data collected during the last period of time from the Activity Store and purges this data from the Store when it is no longer needed. This analysis first identifies the pattern in disk accesses, and then determines the best disk scheduling algorithm to service that pattern of accesses. If the results of this analysis indicate that a new algorithm is required, the I/O Analyzer Process informs the Disk I/O Scheduler of its decision, and the new algorithm is activated.

The last module of IOAZ is the I/O Analyzer Management Utility, responsible for allowing administrative users control over the behavior of IOAZ. This allows manual selection and tuning of disk scheduling algorithms as well as changing of various IOAZ parameters, including how frequently the I/O Analyzer Process executes, and so on.

3.2 Implementation of IOAZ

IOAZ was implemented in C and integrated as kernel modules into version 2.6.11 of the Linux kernel. This version of the Linux kernel supports relatively new abstractions that allow multiple disk scheduling algorithms to co-exist in the kernel to be switched at either boot-time or run-time through a manual process that requires administrative user privileges.

Since the Linux kernel is open source, changes to the kernel could be easily made to integrate IOAZ into its disk I/O subsystem. Collecting data was a simple task, by passing disk request structures from the Disk Request Queuing System to the IOAZ through the I/O Analyzer Interface. Since the Linux kernel already contained mechanisms for switching disk scheduling algorithms at run-time in a manual fashion, adding support for IOAZ controlled switching of algorithms only required modifications to these mechanisms. Since these mechanisms are rarely used in practice, however, many updates were required to fix outstanding or previously unknown problems as part of this process.

3.3 Disk Access Pattern Identification

In order to determine which disk scheduling algorithm is best suited to the current system workload, IOAZ first needs to identify patterns in disk access requests to determine the dominant characteristics of the workload. With this information, IOAZ can make an informed decision as to which algorithm is most appropriate.

While there are a wide variety of potential defining characteristics that could be used in this kind of analysis, in our current work, we focus on the following workload characteristics:

- The number of processes making disk requests. This can be tracked easily because disk requests are tagged according to the process that generated the request, and so IOAZ will have direct access to this information.
- Whether the requests are sequential or random in nature. Sequential access can be detected as an I/O merge activity that pools multiple requests for adjacent disk blocks into a single request for multiple blocks before IOAZ sees the disk request. Observing a large number of merged requests for multiple blocks is highly indicative of sequential activity. Conversely, the absence of this activity indicates a more random workload.
- Whether the requests tend to be read requests or write requests to retrieve or store data respectively. Since requests are tagged with this information, simple counters can collect the necessary information.

Restricting the characteristics initially studied allows us to focus on relatively simple patterns of activity, while the extensibility and flexibility of IOAZ allows us to add support for additional characteristics to support identifying more complex patterns of activity in the future.

After the above analysis, IOAZ has a measure of the number of processes generating activity, the proportion of activity that was sequential in nature as opposed to random, and the proportion of activity that was read requests as opposed to write requests. To simplify pattern matching and algorithm selection, empirically derived thresholds were applied to this data to categorize the workload according to these measurements. Categories were determined by every combination of the following characteristics: number of processes generating the workload (1, 4, 8, or 16 or more), access sequentiality (sequential or random), and access mode (read or write).

To validate our data collection and categorization of identified disk access patterns, IOAZ was fed traces of disk accesses generated by a collection of

workloads with known characteristics. (For example, we used an invocation of the *cat* command to read a file from start to finish to produce a sequential read pattern of activity.) Without fail, IOAZ was able to correctly identify the dominant disk access pattern in all cases. For details on these experiments, the reader is urged to consult [7].

Since IOAZ could now categorize disk workloads in a reliable fashion, the next step was to conduct experimentation to determine which disk scheduling algorithm could best service each workload category.

3.4 Analysis of Disk Scheduling Algorithms

In this section, we analyze the performance of four of the disk scheduling algorithms introduced in Section 2 commonly available in the Linux 2.6.11 kernel: FCFS, AS, DEAD, and CFQ. The purpose of this analysis was to determine which algorithm performed better on which categories of workloads.

To carry out this analysis, experimentation was conducted using the IOZone tool [9] on a test system with the following configuration:

- AMD Athlon 1.2GHZ processor with 266MHZ Front Side Bus and 256KB Cache
- 1GB PC2100 RAM
- 40GB Maxtor IDE Hard Drives, 4MB Cache, 8ms Seek Time
- Linux Kernel 2.6.11

IOZone is an incredibly flexible benchmarking tool, capable of generating disk workloads across all of the workload categories that could be identified by our IOAZ system.

Experimentation consisted of executing standard IOZone tests [9] with 1, 4, 8, and 16 worker processes generating the appropriate workloads, and with each experiment replicated 5 times. Each IOZone test was manually categorized as generating sequential read, sequential write, random read, or random write patterns of access to match the categorizations used by IOAZ.

It was thought originally that each experiment would have a decisive winner, but that was not always the case. Ultimately the success of each algorithm depended on how success was being defined. Did the algorithm maximize the performance of the lowest performing process? Did it maximize the performance of the highest performing process? Or, did it maximize the mean performance across all processes? Since any of these goals ultimately could be desirable, we tracked results for each of these goals separately. This allows administrators to configure IOAZ to base its decisions for switching and tuning disk scheduling algorithms on the given goal for optimizing system performance.

Below, we have summarized the key results of this experimentation, highlighting the best disk scheduling algorithms under the various workload categories currently tracked by IOAZ. Results are given for each of the three performance goals discussed above. Further details of experiments can be found in [7].

# Worker Processes	Sequential Read	Sequential Write	Random Read	Random Write
1	FCFS	FCFS	FCFS	DEAD
4	AS	FCFS	FCFS	FCFS
8	AS	DEAD	FCFS	FCFS
≥ 16	FCFS	FCFS	FCFS	FCFS

Table 1. Best Algorithm for Maximizing Performance of Lowest Performing Process

# Worker Processes	Sequential Read	Sequential Write	Random Read	Random Write
1	FCFS	FCFS	FCFS	DEAD
4	AS	AS	AS	AS
8	AS	AS	AS	AS
≥ 16	AS	AS	DEAD	DEAD

Table 2. Best Algorithm for Maximizing Performance of Highest Performing Process

# Worker Processes	Sequential Read	Sequential Write	Random Read	Random Write
1	FCFS	FCFS	FCFS	DEAD
4	AS	FCFS	AS	AS
8	AS	DEAD	AS	AS
≥ 16	AS	AS	AS	AS

Table 3. Best Algorithm for Maximizing the Mean Performance Across All Processes

As can be seen from Table 1, the FCFS disk scheduling algorithm dominated most workload categories when the goal was to maximize the performance of the lowest performing process. When it came to maximizing the performance of the highest performing process or the mean performance of all processes, as shown in Table 2 and Table 3 respectively, the AS approach tended to be the best suited for most workload categories. It is interesting to note that FCFS was consistently better with only a single worker process generating workload across all goals, except for random write, which fared best with the DEAD approach.

The above experiments were conducted with the IOZone tool imposing a controlled workload with very dominant characteristics that allow for easy categorization of the workload. In practice,

however, this may not be the case, depending on the application mix executing on the system. In such cases, it might become quite difficult to determine the appropriate category for the current workload, making decisions made by IOAZ more difficult, less reliable, and more apt to change over time. Consequently, experiments were also conducted where the sequential versus random and read versus write aspects of disk accesses were ignored, and only the number of worker processes contributing to the workload was tracked.

# Worker Processes	Best Algorithm
1	FCFS
4	AS
8	AS
≥ 16	AS

Table 4. Best Algorithm for Maximizing Performance Tracking Only Number of Processes

Table 4 presents the results of experiments when only the number of worker processes was being tracked, and other workload characteristics were ignored. Interestingly, in this case, the same disk scheduling algorithms worked best for maximizing the performance of the highest performing process, the lowest performing processes, and processes on average.

With experimentation completed, these results were used as the basis for decision matrices to be used by IOAZ in determining which disk scheduling algorithm to use given observations of system workload and an optimization goal. IOAZ can be configured to either use or ignore data on sequential versus random and read versus write patterns in disk accesses, to switch between using decision matrices derived from Tables 1, 2, and 3, or Table 4. Because these decision matrices are logically separate from the decision mechanisms in IOAZ, it is easy to tune the decision making process used by IOAZ by adjusting the appropriate decision matrix if new data suggests a different algorithm should be used in a particular situation. These changes can be put into effect at boot-time or run-time, using the I/O Analyzer Management Utility.

At this point, IOAZ now has everything it needs to function. It can identify patterns in disk accesses, and use these patterns to select an appropriate disk scheduling algorithm capable of best servicing the observed pattern. With this in mind, we can now conduct experiments to examine the performance of IOAZ as a whole in optimizing disk-related system performance.

4 Experimental Results and Experience

To investigate the performance benefits of IOAZ, additional experiments were conducted. These experiments were conducted using the same test system configuration discussed in the previous section, using IOAZ configured to use the decision matrices developed in that section. IOAZ was also configured so that its background process would execute once every second to analyze current disk activity and determine if a new disk scheduling algorithm should be put in place. The results presented below are only a sampling of the results of using IOAZ to date; for complete experimental results, refer to [7] for more details.

4.1 MySQL Experimentation

The MySQL database server [8] provides a standard benchmarking suite to test the performance of a server installation. Because of the heavy disk activity involved with database management systems, this seemed to be an appropriate test of IOAZ performance. The latest version of the MySQL database, version 5.0, was installed on our test system, and the included *sql-bench* benchmark was run to execute all available benchmarking tests, including a mix of read and write tests, and both sequential and random access behaviours. Better performance in this benchmark is shown by a lower time of completion for the benchmark.

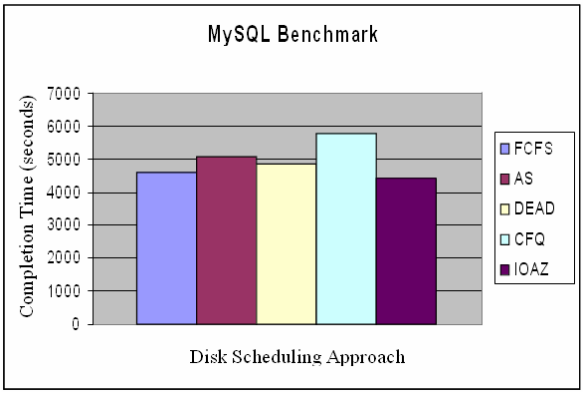

Figure 2. MySQL Benchmark Results

The mean results of 5 repetitions of experiments with the MySQL benchmarking tool are shown in Figure 2. In these experiments, IOAZ provided better performance on average for the MySQL benchmark than all other schedulers tested, with an average test time of 73 minutes and 25 seconds. This is a 4% increase over FCFS and nearly 14% better than AS.

4.2 Disk Zeroing

Additional experimentation was conducted using a test script used for zeroing disks within our department for safe asset disposal. We executed this tool to overwrite a one gigabyte partition entirely with zeroes in a sequential fashion in a single pass, and repeated this experiment 5 times.

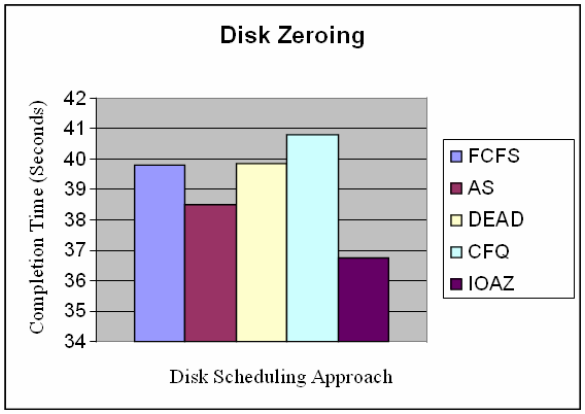

Figure 3. Disk Zeroing Results

As can be seen from the experimental results shown in Figure 3, IOAZ outperforms all other algorithms during this test with an average time savings of 4% to 9%.

4.3 Additional Results

Other tests were carried out using IOZone to produce a variety of extreme workloads. IOAZ consistently outperformed the DEAD and CFQ algorithms, but had difficulty bettering AS in experiments with many processes or FCFS in experiments with a single process [7], because of algorithm switching that occurred when IOAZ concluded one was necessary according to its observations. (This was improved by switching decision matrices to ignore hints of sequential versus random and read versus write accesses in the data, as discussed earlier in the previous section.) Nevertheless, IOAZ appeared to perform better overall as FCFS and AS could not alter their behaviours over time when it was necessary. More testing for further study is currently under way.

5 Concluding Remarks

This paper introduced a new approach to disk scheduling to improve system performance. This approach focuses on using analyses of disk accesses to determine the best disk scheduling algorithm for the current workload, and switching and tuning algorithms as necessary to improve performance. A prototype system, IOAZ, was implemented as a proof of concept, and experimentation with this prototype has been quite positive, yielding interesting results and showing great promise.

There are many possible avenues for continued research in this area worthy of examination. New heuristics need to be developed for the identification of additional and more complex patterns of disk accesses. Further experimentation is necessary to determine the best disk scheduling algorithm to select in a wider variety of workload conditions and hardware configurations. Work is also required to better detect changes in workloads to support faster reaction times without sacrificing stability.

References:

[1] R. Appleton. Disk Scheduling in Linux. *Departmental Talk in the Computer Science Department at Northern Michigan University*. December 2003.

[2] G. Gibson, J. Vitter and J. Wilkes. Report of the Working Group on Storage I/O for Large-Scale Computing. *ACM Computing Surveys, 28 (4)*. December 1996.

[3] P. Goyal, X. Guo, and H. Vin. A Hierarchical CPU Scheduler for Multimedia Operating Systems. *The Second Symposium on Operating Systems Design and Implementation*. Seattle, Washington, October 1996.

[4] L. Huang and T. Chiueh. Experiences in Building a Software-Based SATF Scheduler. *Technical report, State University of New York at Stony Brook*. July 2002.

[5] S. Iyer. The Effect of Deceptive Idleness on Disk Schedulers. *Master's Thesis, Computer Science Department, Rice University*. April 2001.

[6] S. Iyer and P. Druschel. Anticipatory Scheduling: A Disk Scheduling Framework to Overcome Deceptive Idleness in Synchronous I/O. *Proceedings of the 18th ACM Symposium on Operating Systems Principles*. Banff, Canada. October, 2001.

[7] D. Martens. Disk Access Analysis for Optimal Performance. *Master's Thesis, Department of Computer Science, The University of Western Ontario*. September 2005.

[8] MySQL Inc. MySQL 5.0 Reference Manual. *Software Documentation*, 2005.

[9] W. Norcott and D. Capps. IOZone Filesystem Benchmark. *Software User Manual and Documentation*, August 2003.

[10] B. Worthington, G. Ganger, and Y. Patt. Scheduling Algorithms for Modern Disk Drives. *Proceedings of the 1994 ACM Sigmetrics Conference*. Nashville, Tennessee, May 1994.