Real-Time Disk Scheduling for Block-Stripping I2O RAID

Tei-Wei Kuo, Ji-Shin Rao[†], Victor C. S. Lee[‡], and Jun Wu[†]
Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan 106, ROC
[†]Department of Computer Science and Information Engineering
National Chung Cheng University, Chiayi, Taiwan 621, ROC
[‡]Department of Computer Science, City University of Hong Kong
83 Tat Chee Avenue, Kowloon, Hong Kong

Abstract

The emerging of Intelligent I/O (I2O) architecture provides a standard for high-performance I/O subsystems and introduces intelligence at the hardware level. With an embedded processor, I2O adaptors can offload the major I/O processing workload from the CPU and, at the same time, increase the I/O performance. This paper addresses the essential issue in the design of disk scheduling for I2O RAID-0 devices. We explore on-line real-time multi-disk scheduling for I2O requests and propose highly efficient algorithms to minimize the number of deadline violations and, at the same time, to improve the response times of requests. The proposed methodologies are verified by a series of experiments under realistic and randomly generated workloads.

1 Introduction

Traditional work on disk scheduling has been focused on single disk systems, such as SCAN, Shortest-Seek-Time-First (SSTF), Circular SCAN (C-SCAN), and FIFO [11], where SCAN services disk requests on the way from one side of the disk to the other side and then on the way back, etc. C-SCAN is a variation of SCAN, except that C-SCAN always services disk requests from one side to the other side, and as soon as the r/w head reaches the other side, it immediately returns to the beginning of the disk, without servicing any request on the return trip. SSTF always services the request closest to the current r/w head position. FIFO services requests according to their arrival order. Andrews, et al., [1] showed that the optimization problem for single disk scheduling is an asymmetric traveling salesman problem and provided approximation algorithms.

Recently, researchers start proposing various real-time disk scheduling algorithms based on the request deadlines, e.g., [2, 3, 4, 5, 6, 7, 10]. Chen, et al. [5] proposed a weighted-function-based algorithm called *Shortest-Seek-Time-Earliest-Deadline-By-Value* (SSEDV) by considering request deadlines and their seek times. Reddy and Wyllie [10] explored the contention of SCSI bus, in which priorities are defined as a weighted function of their deadlines. Bruno, et al. [3] proposed a two-level scheduling mechanism, in which each session/process has a deadline-based queue,and jobs are sorted according to their deadlines. The most urgent job in each deadline-based queue is inserted into the system queue, where the system services jobs in the system queue in a SCAN fashion. Hwang and Shin [7] proposed a deadline-driven algorithm, in which requests are partitioned into groups based on their urgency. Urgent requests are always serviced first. Abbott and Garcia-Molina [2] proposed a SCAN-like deadline-driven algorithm. The algorithm first picks up a request with the closest deadline and then services all requests residing at cylinders between the current cylinder and the cylinder of the request with the closest deadline. Chang, et al [4] proposed a deadline-monotonic SCAN algorithm which guarantees hard deadlines of disk access, where the workload distribution (such as deadlines, disk addresses, etc) of disk access is known. Although researchers have proposed various excellent algorithms for single disk scheduling, little work has been done for multiple-disk scheduling, especially for real-time RAID applications. In particular, Weikum and Zabback [12] studied the impacts of stripping size on RAID concurrency and performance, for non-real-time applications. Cheng, et al. [6] proposed to synchronize all disks for real-time RAID scheduling. Sequential access is favored, at the cost of random access.

0-7695-1221-6/01 $10.00 © 2001 IEEE

The goal of this research is to explore real-time disk scheduling for I2O RAID-0 devices with low run-time overheads, where RAID-0 stands for the redundant array of independent disks with a block-stripping scheme. We consider I2O RAID-0 devices which adopt a less-powerful embedded processor for disk scheduling. Since requests on I2O RAID-0, in general, have soft real-time deadlines, a disk scheduling algorithm must not only maximize the I/O performance, e.g., in terms of throughput or response time, but also minimize the number of requests which miss their deadlines. The major contribution of this work is on the exploring of on-line real-time multi-disk scheduling, which is suitable to I2O devices adopting a less-powerful embedded processor. We not only propose an incremental real-time multi-disk scheduling algorithm but also include an approximate version with a linear time complexity for real-time and non-real-time data access. We have shown that our proposed methodologies are both efficient and effective in scheduling I2O RAID-0 requests.

The rest of this paper is organized as follows: Section 2 illustrates the I2O system architecture and its RAID implementation. Section 3 first defines I2O RAID-0 requests and the performance goal. We then propose our incremental multi-disk scheduling algorithm called *Value-Based Real-Time Job-Group Scheduling* algorithm (VRT-JG). An iterative procedure is proposed to reorder jobs of requests in each disk queue with an objective to minimize the number of deadline violations and, at the same time, to optimize the response times of requests. In Section 4, the capability of the proposed algorithm is evaluated by a series of experiments under randomly generated workloads, for which we have some encouraging results. Section 5 is the conclusion.

2 Intelligent Input/Output System Architecture

2.1 Intelligent Input/Output Architecture

The Intelligent I/O (I2O) specifications are proposed by major players in the industry, such as Microsoft, Intel, Hewlett-Packard, 3COM, Compaq, etc, as a standard for the next-generation I/O subsystems. The introducing of the I2O specifications provides intelligence at the hardware level and standardize platforms for all segments of the industry. They specify an architecture that is operating-system-vendor-independent and adapts to existing operating systems, such as Microsoft Windows NT and 2000. The I2O specifications enable the OS vendors to produce a single driver for each class of devices and concentrate on optimizing the OS portion of the driver. With an embedded processor, I2O adaptors can offload the major I/O processing workload from the CPU and, at the same time, increase the I/O performance.

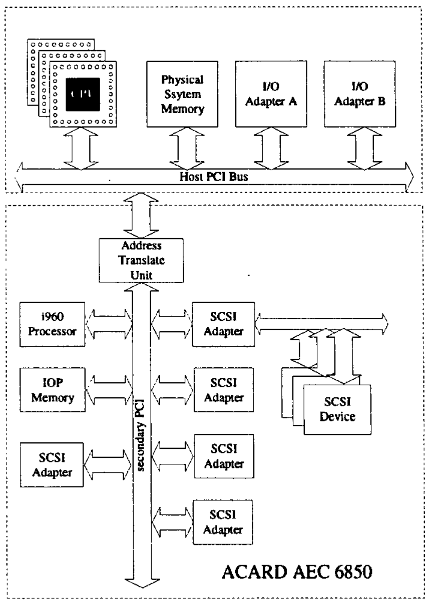

Figure 1: An I2O hardware architecture

We shall illustrate the architecture of the I2O specifications by an example product ACARD AEC 6850, which is to be released to the market by the ACARD Corp. ACARD AEC 6850 is an I2O RAID adaptor, which can manage up to 75 hard disks. Its hardware architecture is as shown in Figure 1. There are two major components separated by a dash line: Host and Target. A host can be any PC running a popular OS such as Windows 2000. The host can have other I/O adaptors for other I/O devices. The target is an I2O adaptor, such as ACARD AEC 6850 in this example. The interface between the I2O adaptor and the host is currently defined as a PCI bus. ACARD AEC 6850 has an embedded processor, such as Intel i960, memory, and up to 5 SCSI adaptors. Each SCSI adaptor may be connected to 15 disks. (Note that IDE disks might be adopted in similar products.) The memory space of an I2O adaptor can be mapped to the memory address domain of the host so that the host and the target can communicate by DMA.

The I2O architecture splits drivers into two parts: OS-Specific Module (OSM) and Device Driver Module (DDM), as shown in Figure 2. OSM is implemented

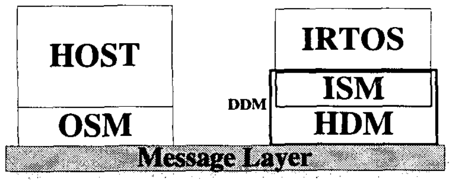

Figure 2: I2O driver modules

at the host side, and DDM is at the target side. OSM provides an interface to the host operating system and is usually supplied by OS vendors. OSM communicates with DDM via messages (on the top of a PCI bus). An I2O real-time operating system (IRTOS) (and its related programs) runs on the I2O adaptor's processor to receive I/O requests from the host via OSM and schedule disk services. All disk operations are initiated by invoking appropriate DDM handler functions. DDM may consist of two parts: Intermediate Service Module (ISM) and Hardware Device Module (HDM). HDM contains hardware-specific code to manage device controllers, and ISM lets hardware vendors add more functionality to plain devices (stacked over HDM), e.g., having real-time disk scheduling or resource management [8].

2.2 Intelligent Input/Output RAID

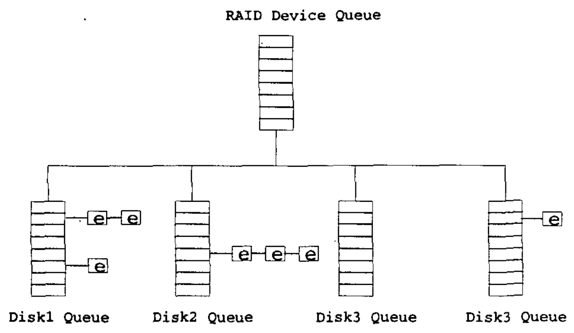

Figure 3: The event flow in an I2O RAID-0 device

I2O devices are designed to fulfill the demand of high-performance I/O, and one of the most important applications is I2O RAID's. An I2O RAID device, such as ACARD AEC 6850, may need to manage a number of disks with data stripping technology. In particular, we are interested in RAID-0, in which data are stripped in units of blocks such that an I/O request may be serviced by several disks simultaneously. For the purpose of this section, an I/O request is tentatively defined as a collection of i jobs (for $i \geq 1$), which may be serviced by different disks.

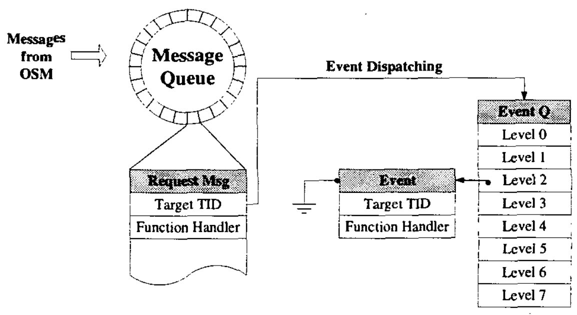

Figure 4: Message dispatching in an I2O RAID-0 device

We shall illustrate the system operation in terms of an I2O RAID-0 device with four disks. According to the I2O specifications, there is an event queue for the entire RAID device and each of its disks, as shown in Figure 3. Each of the queues is a priority queue, where event priorities are determined by applications (via OSM). An IRTOS (and its related programs) is an event-triggered system. When the host issues an I/O request via OSM, the request is transformed into a message and inserted into the corresponding message queue, as shown in Figure 4. The message insertion will trigger the execution of the corresponding system thread to process the message and insert an event into the event queue for the entire RAID device, as shown in Figure 3. The event carries all of the necessary information for the I/O request received via OSM. In general, there is a thread associated with each event queue. The event insertion will trigger the execution of the thread assigned to the RAID device event queue. As a result, the I/O request will be decomposed into a collection of jobs, and an event for each of the jobs will be inserted into the event queue of the corresponding disk. Threads which are assigned to the event queues of the disks will then become ready to process their events and invoke DDM handler functions to initiate I/O operations.

3 Real-Time RAID-0 Scheduling

Traditional disk scheduling focuses on single disk systems. The main performance metrics are response time and throughput for non-real-time disk services and miss ratio and response time for real-time data access. Little work has been done for multiple-disk scheduling, especially for real-time applications. In this paper, we are interested in I2O RAID devices, in which multiple disks are adopted to maximize the I/O bandwidth. Disks with/without internal scheduling, such as SCSI and IDE disks, are potential drives for our target I2O RAID devices. A simple but effective scheduling algorithm for multiple disks must be adopted and run on the embedded processor of the I2O RAID device to meet the hard deadline of each request and, at the same time, to minimize requests' response time.

An important objective of I2O RAID devices is to push down the I/O functionality to a lower level, i.e., the I2O controller level, such that high-performance storage devices can be obtained. Data stripping is a popular technology to distribute data over multiple disks to utilize parallelism to maximize I/O bandwidth. The objective of this work is on real-time RAID-0 disk scheduling, i.e., real-time scheduling of I/O requests with block stripping. Under the I2O specifications, each I/O request has a reasonable deadline, and an I/O request may be up to $4GB$ (the byte count is of 4 bytes in *BsaBlockRead request message*) [8]. In other words, a request can be scattered over several disks. The deadline setting of an I/O request depends on many factors, such as the types of requests, request slack (called TimeMultiplier in the I2O specifications), etc. For example, the deadline of a read (and write) request is defined as

$$TimeMultiplier \times (RWVTimeoutBase + (RWVTimeout \times size/64K)),$$

where $RWVTimeoutBase$ and $RWVTimeout$ are two constants set by OSM during system initialization, and $size$ and $TimeMultiplier$ are the byte count and the slack of the I/O request, respectively. The deadline of a cache flush request for a specified DDM is defined as $TimeMultiplier \times timeout\_Base$, where $timeout\_Base$ is another constant set by OSM during system initialization. The deadlines of I/O requests are, in general, soft deadlines and reasonably large. However, in some implementation, the deadline violation of certain I/O requests may result in a system reset.

3.1 System Model and Overview

Each I2O I/O request can be modeled by four parameters $r_i = (arr_i, LBA_i, s_i, d_i)$, where arr_i, LBA_i, s_i, and d_i are the arrival time, the starting logical block address (LBA), the size in bytes, and the deadline of the I/O request r_i, respectively. With block stripping, an I2O adaptor must re-number the logical block addresses of blocks over its disks, where the logical block address starts with 0. Suppose that there are N disks managed by an I2O adaptor, and the *block stripe size*

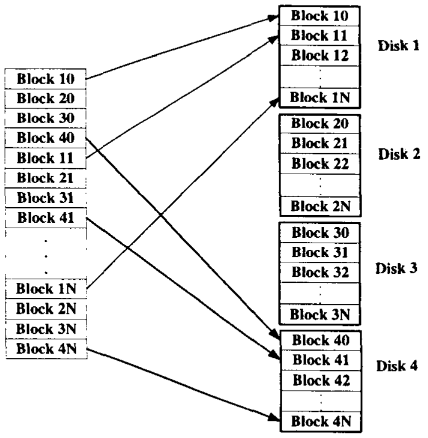

Figure 5: Block stripping when the no. of disks is four.

(or *physical block size*) be B. A common approach is to assign the j_{th} LBA of the i_{th} disk as the k_{th} LBA of the I2O device (for $0 \le j \le Max\_LBA\_Disk_i$ and $1 \le i \le N$), where $k = (N * B * \lfloor (j/B) \rfloor + (i-1) * B + (j\%B))$. An I2O device is defined as an I2O adaptor and its managed disks, and % is a mod operator. For example, a re-numbering scheme of LBA over four disks is shown in Figure 5, where the block stripe size is 32 sectors, and each sector on an ordinary PC disk is of $512B$. The LBA number of a block for an I2O device is called an *I2O LBA number* or LBA number, when there is no ambiguity. The LBA number of a block for a disk (managed by the I2O adaptor) is called a *real LBA number*.

The four parameters of an I2O I/O request $r_i = (arr_i, LBA_i, s_i, d_i)$ can be further abstracted as a collection of jobs executing on different disks (or a single disk if the I/O request is of a small size). That is, an I2O I/O request r_i can be re-defined as a tuple $(arr_i, \{J_{i,1}, \cdots, J_{i,n_i}\}, d_i)$, where each job $J_{i,j}$ has a disk number $dsk_{i,j}$ to execute the job, a size in bytes $s_{i,j}$, and a real LBA number $RLBA_{i,j}$ as its starting LBA on its assigned disk. The completion time of an I2O I/O request r_i is the maximum completion time of all of its jobs. Therefore, in order to meet the deadline of an I2O I/O request r_i, every job $J_{i,j}$ must complete its I/O transfer of $s_{i,j}$ bytes (starting from the real LBA address $RLBA_{i,j}$ on disk $dsk_{i,j}$) no later than the deadline d_i.

In this paper, we are interested in an I2O RAID-0 device (with a model number ACARD AEC 6850), which is built under a joint project with the ACARD Corp. We consider I2O RAID-0 products which might be built with IDE disks (because of cost consideration) or SCSI disks. A disk queue is associated with each disk, and the corresponding disk services the job at the head of the disk queue. A real-time RAID-0 scheduler, which runs on the processor of the I2O RAID-0 device, must schedule jobs over multiple disks in a real-time fashion. The purpose of real-time RAID-0 scheduling is to minimize the miss ratio and the response times of I/O requests. With a less powerful processor, such as Intel i960 or ARM, usually adopted in an I2O device, we shall explore efficient multiple-disk scheduling algorithms, especially used in an incremental way. We will also adopt performance metrics which reflect not only miss ratio and response time of requests but also their byte-counts.

The basic idea of the real-time RAID-0 scheduling algorithm proposed in this paper is to optimize a value function for inserting jobs of each incoming I/O request into proper positions of disk queues such that the miss ratio and response times of all pending requests and the incoming request can be minimized. The incremental algorithm should try to "sink" each job of an incoming request from the end of its assigned disk queue to a proper location in the queue. We will demonstrate the strength of our job-group scheduling approach with other traditional and real-time disk scheduling algorithms under randomly generated workloads.

3.2 Basic Mechanism - A Value-Driven Approach

The value function for the job ordering in disk queues at time t is defined as a value function of the weighted miss ratio and the average weighted response time of the current pending requests:

$$v(JO,t) = \alpha * weighted\_miss\_ratio + (1-\alpha) * avg\_weighted\_resp, \; for\ 0 \leq \alpha \leq 1$$

Let $w(r_i)$ and $resp(r_i)$ be the weight and the response time of a request r_i (e.g., $w(r_i)$ can be defined as the number of sectors accessed by r_i), respectively, and *avg_resp* be the average response time of all completed requests up to the current time t. JO denotes the current job ordering in disk queues, and the response time of a request is the difference of its arrival time and the maximum completion time of the request's jobs. *weighted_miss_ratio* and *avg_weighted_resp* are defined as follows:

$$weighted\_miss\_ratio = \sum_{deadline\text{-}missing\ pending\ requests} w(r_i) / \sum_{all\ pending\ requests} w(r_k)$$

$$avg\_weighted\_resp = \sum_{all\ pending\ requests} (resp(r_i) * w(r_i)) / (avg\_resp * \sum_{all\ pending\ requests} w(r_k))$$

The rationale behind the definitions of *weighted_miss_ratio* and *avg_weighted_resp* is to reflect the sizes of requests in performance evaluation, where a large-byte-count request contributes more on disk bandwidth (as far as disk scheduling is concerned). In this paper, deadline-to-be-missing requests are called tardy requests.

A basic real-time RAID-0 scheduling algorithm called *Value-Based Real-Time Job-Group Scheduling* algorithm (VRT-JG) is defined as follows: Let JO denote the current job ordering when a new request r_{new} arrives at time t, and $JS_i^{critical}$ be the maximum collection of jobs in request r_i which have the worst completion time. Note that the size of $JS_i^{critical}$ is no less than 1. For the rest of this paper, when there are several jobs of r_{new} being assigned on the same disk,we merge those jobs into one job with a large size although the LBA's accessed by the merged job are not consecutive.

```
VRT-JG($JO, r_{new}$)
    $JO'$ = $JO$ with jobs of $r_{new}$ appended at the end
          of every corresponding disk queue;
    $Value = Value" = v(JO', t)$;
    $JO" = JO'$;
    Do
        $Value = Value"$;
        $JO' = JO"$;
        $JS_{new}^{critical}$ be the maximum collection of jobs
              in $r_{new}$ which has the worst
              completion time under the job
              ordering $JO'$;
        Let $JO"$ be the job ordering $JO'$ by
            switching each job $JS_{new,j}^{critical}$ in $JS_{new}^{critical}$
            with the job of another pending request
            which is immediately before $JS_{new,j}^{critical}$
            in the same queue;
        $Value" = v(JO", t)$;
    While ($Value" \leq Value$);
    Return $JO'$;
```

The *VRT-JG* algorithm is a simple greedy algorithm which keep moving jobs of the incoming request r_{new} forward in the queues if there is some improvement on the value function. However, there is one major shortcoming of the *VRT-JG* algorithm. The calculation of the value function is not trivial. Note that the calculation of the average (real) response time of all completed requests up to the current time t (i.e., *avg_resp*) can done incrementally in a constant time. The calculation of the average weighted response time of all pending jobs is of a complexity $O(Q * N)$, where Q and N are the number of disk queues and the number of pending tasks, respectively. The calculation of the weighted miss ratio, given response times of all pending requests, is of a complexity $O(N)$. Thus, the calculation of the value function is $O(Q * N)$. Because the finding of $JS_{new}^{critical}$ is of a complexity $O(Q)$, given the response times of jobs of r_{new} (otherwise, it is of a complexity $O(N * Q)$), the time complexity of the *VRT-JG* algorithm is $O(N * Q * R)$, where R is the number of rounds in switching jobs ($R \leq N$). To reduce the complexity of the disk scheduling algorithm, we have explored the approximation of the value function. Due to space limit, the detail is not included in this paper. We surmise that the VRT-JG algorithm with an approximate value function has a linear time complexity $O(N)$.

4 Performance Evaluation

4.1 Performance Metrics and Data Sets

The experiments described in this section are meant to assess the capability of the VRT-JG algorithm in scheduling I2O RAID-0 requests. We have implemented a simulation model for a I2O RAID-0 device under realistic benchmarks and randomly generated workloads. Due to space limit, the results under benchmarks are not included. We compare the performance of the VRT-JG algorithm, the earliest deadline first algorithm (EDF), the least slack time first (LSF) [9], and the well-known disk scheduling algorithm, such as FIFO, C-LOOK, and SSTF.

The primary performance metric is the weighted ratio of requests that miss deadlines, referred to as the *Weighted Miss Ratio.*

$$Weighted\_Miss\_Ratio = \sum_{deadline\text{-}missing\ requests} w(r_i) / \sum_{all\ requests} w(r_k).$$

Another performance metric is the average weighted response time of requests, referred to as the *AVG_Weighted_Resp.*

$$AVG\_Weighted\_Resp = \sum_{all\ requests} (resp(r_i) * w(r_i)) / (avg\_resp * \sum_{all\ requests} w(r_k)),$$

where *avg_resp* is the average response time of all requests in the system so far. We also adopt the performance metric *throughput*, which is similar to performance metrics used in traditional disk scheduling, referred to as the *Throughput. Throughput* is calculated as the average number of bytes which are accessed in a second. Another primary performance metric is the ratio of requests that miss deadlines, referred to as the *Miss Ratio.* Let num_i and $miss_i$ be the total number of task requests and deadline violations during an experiment, respectively. *Miss Ratio* is calculated as $\frac{miss_i}{num_i}$. The other performance metric is the average response time of requests, referred to as the *AVG_Resp.*

Parameters	Value
Request Deadline Slack "TimeMultiplier"	$(2, 32)$
RWVTimeoutBase	0.5
RWVTimeout	0.5
Arrival Pattern	Possion mean=$(12ms, 30ms)$
Block Stripe Size	32 sectors $512B$ per sector
LBA Range	$(0, 100000)$
Request Size	$(1, 512)$ sector
The Number of Disks	4
Sustained Transfer Rate	$2MB/Sec$
Time Granularity	$1ms$
Simulation Time	$200 seconds$

Table 1: Simulation Parameters.

The randomly generated data sets were generated based on the parameters of real disks and a commercial I2O product ACARD AEC 6850. The deadlines of requests were calculated based on the I2O specifications, where *TimeMultiplier* ranged from 1 to 30. The arrivals of requests followed the Possion distribution with a mean ranging from $3ms$ to $7ms$. Each request may request data of a size ranging from 1 sector to 512 sectors. The block strip size (or physical block size) is 32 sectors. Four HP97560 SCSI disks were adopted, and their sustained transfer rate was $2MB/Sec$. The simulation time was $100,000ms$. The simulation parameters are summarized in Table 1.

4.2 Experimental Results

Figure 6 shows the weighted miss ratio of requests under VRT-JG, EDF, LSF, FIFO, C-LOOK, and SSTF, where VRT-JG(x) meaned VRT-JG with $\alpha = x$. VRT-JG(1) out-performed other algorithms, regardless of the disk workload. It was because VRT-JG(1) tried to optimize the weighted miss ratio. FIFO, LSF, and EDF had the worst weighted miss ratio among all algorithms. It was mainly because they moved disk heads a lot and did not consider each request as a unit in multiple disk scheduling. The rest were between VRT-JG(1) and C-LOOK. When the system was not overloaded, i.e., the inter-arrival interval was more than

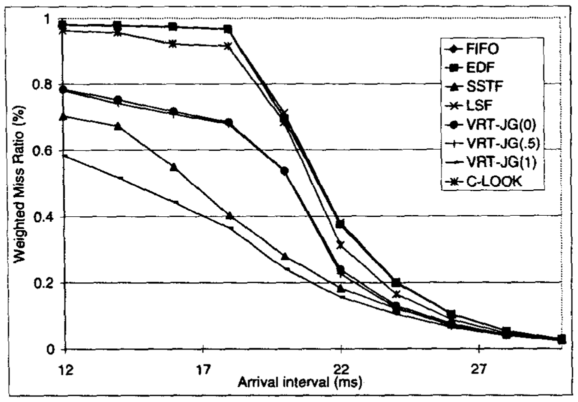

Figure 6: The weighted miss ratio of simulated algorithms

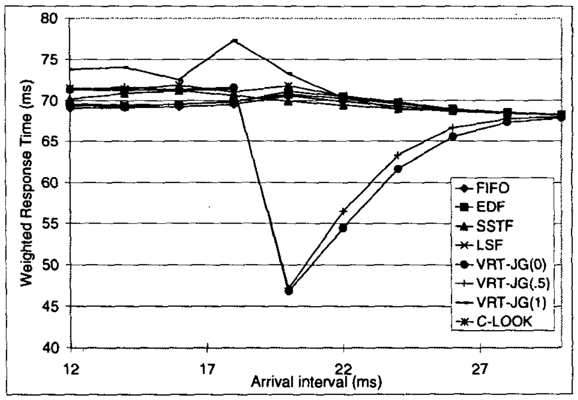

Figure 8: The weighted average response time of simulated algorithms

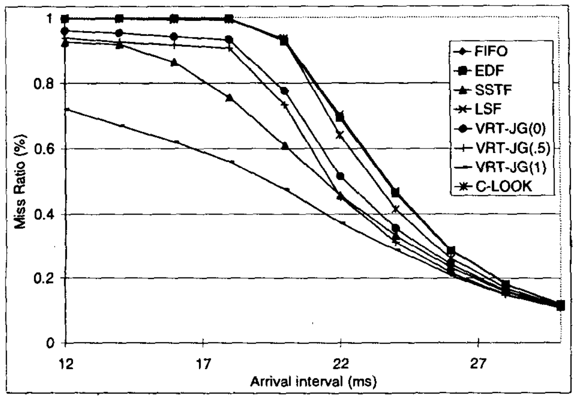

Figure 7: The miss ratio of simulated algorithms

$21ms$, VRT-JG(1) improved the weighted miss ratio by over 15% over SSTF. When the system was overloaded, VRT-JG(1) improved the weighted miss ratio by over 17% over SSTF. When the performance metric was the miss ratio, as shown in Figure 7, FIFO, LSF, EDF, and C-LOOK were the worst, and VRT-JG(1) was the best. In general, the consideration of each request as a scheduling unit did improve the weighted miss ratio. We must emphasize that we did not try to optimize the disk head movement in this paper, as done by C-LOOK and SSTF. With the consideration of head movement, the performance of VRT-JG would be surely improved further.

Figure 8 shows the weighted average response time of requests under VRT-JG, EDF, LSF, FIFO, C-LOOK, and SSTF. It was interesting to see that all algorithms had similar performance in terms of weighted response time. However, when consider the response time of all requests (without having a weight on the request size), as shown in Figure 9, SSTF was the best because of the optimization of disk seek time.

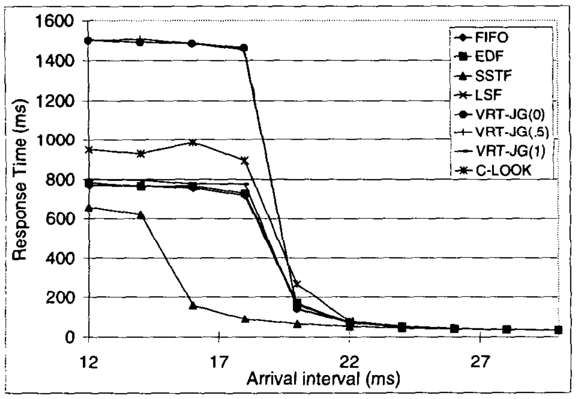

Figure 9: The average response time of simulated algorithms

When the system became overloaded, i.e., the inter-arrival interval was no less than $22ms$, the weighted response time of VRT-JG, except VRT-JG(1), dropped suddenly, as shown in Figure 8. It was because the average response time of all requests in the system suddenly increased, compared to that before the system overload. VRT-JG(1) was not affected because $\alpha = 1$ such that the average response time of all requests was not considered in the value function. Later on, since the average response time of all requests tended to settle down, the weighted response time stablized again. Figure 10 shows the throughput of VRT-JG, EDF, LSF, FIFO, C-LOOK, and SSTF. The performance of SSTF was significantly better than the other, and VRT-JG(1) was the second. However, all algorithms performed similarly when the system was not overloaded.

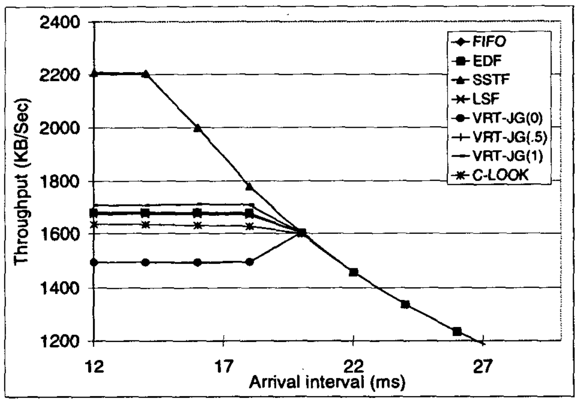

Figure 10: The throughput of simulated algorithms

5 Conclusion

This paper targets an essential performance issue in the design of I2O RAID devices, where RAID is one of the most important implementations for I2O devices. Our goal is to improve the performance of I2O RAID-0 devices and to verify our results under a realistic product ACARD AEC 6850, which is a high-performance I2O RAID-0 adaptor to be released to the market by the ACARD Corp. We explore real-time multi-disk scheduling under I2O RAID-0 to improve the I/O performance and, at the same time, to minimize the number of deadline violations. We illustrate the system architecture of I2O devices and define the performance goal. We then propose our incremental multi-disk scheduling algorithm called *Value-Based Real-Time Job-Group Scheduling* algorithm (VRT-JG) for I2O RAID-0 requests. The capability of the proposed algorithm is evaluated by a series of experiments under randomly generated workloads, for which we have some encouraging results. In particular, VRT-JG(1) outperforms other disk scheduling algorithms in terms of the weighted miss ratio and the miss ratio, regardless of the disk workload.

For the future research, we shall further explore real-time multi-disk scheduling to fit different I2O RAID devices which might adopt embedded processors with different computing power. We shall also explore multi-disk scheduling for other types of I2O RAID devices, such as those for mirroring and parity-based stripping schemes.

References

[1] M. Andrews, M.A. Bender, L. Zhang, "New Algorithms for the Disk Scheduling Problem," *The 37th Annual Symposium on Foundations of Computer Science*, 1996, pp. 550-559.

[2] R.K. Abbott and H. Garcia-Molina, "Scheduling I/O Requests with Deadlines: a Performance Evaluation," *IEEE 11th Real-Time Systems Symposium*, Dec. 1990, pp. 113-124.

[3] J. Bruno, J. Brustoloni, E. Gabber, B. Ozden, and A. Silberschatz, "Disk Scheduling woth Quality of Service Guarantees," *IEEE Int. Conf. on Multimedia Computing and Systems*, 1999, pp.400-405.

[4] R.-I. Chang, W.-K. Shih, and R.-C. Chang, "Deadline-Modification-SCAN with Maximum-Scannable-Groups for Multimedia Real-Time Disk Scheduling," *IEEE 19th Real-Time Systems Symposium*, Dec. 1998, pp. 40-49.

[5] S. Chen, J.A. Stankovic, J.F. Kurose, and D.F. Towsley, "Performance Evaluation of Two New disk scheduling Algorithms for Real-Time Systems," *Journal of Real-Time Systems*, 3(3):307-336, 1991.

[6] P. Chang, H. Jin, X. Zhou, Q. Chen, and J. Zhang, "HUST-RAID: High Performance RAID in Real-Time System," *IEEE Pacific Rim Conf. on Communication, Computers, and signal Processing*, 1999, pp. 59-62.

[7] K. Hwang and H. Shih, "Real-Time Disk Scheduling Based on Urgent Group and Shortest Seek Time First," *The 5th Euromicro Workshop on Real-Time Systems*, 1993, pp. 124-130.

[8] I^2O specifications.

[9] A.K. Mok, "Fundamental Design Problems for the Hard Real-Time Environment," MIT Ph.D. Dissertation, Cambridge, MA, 1983.

[10] A.L. N. Reddy and J.C. Wyllie, "I/O Issues in Multimedia System," *IEEE Transactions on Computers*, March 1994.

[11] A. Silberschatz and P.B. Glavin, "Operating System Concepts", 4th Ed., Addison Wesley, 1994.

[12] G. Weikum and P. Zabback, "Tuning of Stripping Units in Disk-Array-Based File Systems," *Second Int. Workshop on Research Issues on Data Engineering, 1992: Transaction and Query Processing*, 1992, pp. 80-87.