Available online at www.sciencedirect.com
SCIENCE DIRECT®
The Journal of Systems and Software 76 (2005) 237–250
The Journal of Systems and Software
www.elsevier.com/locate/jss

Multi-disk scheduling for time-constrained requests in RAID-0 devices

Shi-Wu Lo [a,1], Tei-Wei Kuo [a,1], Kam-Yiu Lam [b,*]

[a] Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, ROC
[b] Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong

Received 21 March 2003; received in revised form 23 May 2004; accepted 23 May 2004
Available online 28 July 2004

Abstract

In this paper, we study the scheduling problem of real-time disk requests in multi-disk systems, such as RAID-0. We first propose a multi-disk scheduling algorithm, called Least-Remaining-Request-Size-First (LRSF), to improve soft real-time performance of I/O systems. LRSF may be integrated with different real-time/non-real-time single-disk scheduling algorithms, such as SATF and SSEDV, adopted by the disks in a multi-disk system. We then extend LRSF by considering the serving requests on-the-way (OTW) to the target request to minimize the starvation problem for requests that need to retrieve a large amount of data. The pre-fetching issue in RAID-0 is also studied to further improve the I/O performance. The performance of the proposed algorithm and schemes is investigated and compared with other disk scheduling algorithms through a series of experiments using both randomly generated workload and realistic workload.

Keywords: Real-time disk scheduling; RAID-0; Multi-disk scheduling

1. Introduction

Previous research works in disk scheduling are mainly focused on single-disk scheduling with the objectives to maximize disk throughput and minimize the movement of read/write heads. For example, LOOK (Silberschatz et al., 2001) serves disk requests on the way from one side of a disk to the other side, and then on the way back. The Shortest-Access-Time-First (SATF) (HPL-CSP-91-7, 1991) serves the request with the smallest access time first. Although these disk scheduling algorithms can efficiently improve the disk throughput, they may not be suitable to multi-disk systems, especially for applications where the requests have constraints on their completion times. Example applications are transaction processing in a real-time database system (Ramamritham, 1993). We must point out that the number of applications with soft deadline constraints increases rapidly in recent years due to popularity of Internet applications and distributed multimedia and mobile computing systems (Ulusoy, 1998; Reddy and Wyllie, 1993; Yu et al., 1992). To soft real-time applications, the most important performance goal is to minimize the number of missed deadline requests (Thomas et al., 1996; Ulusoy and Belford, 1993) instead of maximizing the system throughput.

The studies of scheduling algorithms for real-time disk requests have been greatly ignored until recent years. Chen et al. (1991) proposed the *Shortest-Seek-Time-Earliest-Deadline-By-Value* (SSEDV) algorithm in which both deadlines and seek time delays are considered in scheduling disk requests. Reddy and Wyllie (1993) explored the contention in a SCSI bus and

* Corresponding author. Tel.: +852 27889807; fax: +852 27888614.
E-mail addresses: d89015@csie.ntu.edu.tw (S.-W. Lo), ktw@csie.ntu.edu.tw (T.-W. Kuo), cskylam@cityu.edu.hk (K.-Y. Lam).
[1] Fax: +886-23628167.

0164-1212/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2004.05.029

proposed to assign priorities to disk requests using a weighted function of the deadlines of the requests (Reddy and Wyllie, 1993). Bruno et al. (1999) proposed a two-level scheduling mechanism in which each session/process has a queue sorted by the process deadlines. The most urgent request in a queue is selected to be put into the system queue, and then the system serves requests in the system queue following the principles proposed in the LOOK algorithm. Abbott and Garcia-Molina (1990) proposed a SCAN-like deadline-driven algorithm which first picks up a request with the closest deadline, and then serves all of the requests residing at the cylinders between the current cylinder of the read/write head and the cylinder of the request with the closest deadline. Chang et al. (1998) proposed a deadline-monotonic SCAN algorithm which can guarantee the hard deadlines of disk requests if the workload distribution (such as deadlines, disk addresses, etc) of requests is known.

Although researchers have proposed excellent algorithms for single-disk scheduling, little work has been done on multi-disk scheduling, especially for real-time RAID (which stands for the *redundant array of independent (inexpensive) disks*). In particular, Weikum and Zabback (1991) have studied the impacts of stripping size on concurrency and performance of non-real-time applications in RAID. Chang et al. (1999) has studied the synchronization of the disks in real-time RAID scheduling.

The goal of this research is to propose an efficient multi-disk scheduling algorithm for RAID-0 with the objective to minimize the number of deadline-missing requests. An RAID-0 is an RAID with block-stripping. Since disk requests have soft real-time deadlines, a multi-disk scheduling algorithm should maximize the I/O performance, e.g., in terms of throughput or response time, and at the same time to minimize the number of deadline-missing requests. In this paper, we first introduce the system architecture of an RAID-0 for multi-disk scheduling. Then, we propose a real-time multi-disk scheduling algorithm called *Least-Remaining-Request-Size-First* (LRSF), which can integrate with different real-time/non-real-time single-disk scheduling algorithms, such as SATF and SSEDV. We extend LRSF by considering the services of requests on the way to the target request. Finally, we study the performance of the proposed algorithms and schemes by a series of experiments using both randomly generated workload and realistic workload.

The rest of this paper is organized as follows: Section 2 introduces the I/O system architecture of an RAID-0 and the multi-disk scheduling in the system. Section 3 uses an example to illustrate the multi-disk scheduling problem in RAID-0. The proposed real-time multi-disk scheduling algorithm is presented in Section 4. In Section 5, the proposed algorithm is extended by adding the on-the-way mechanism to further improve the disk performance. Section 6 reports the performance results of the proposed algorithm. The conclusions of the paper are in Section 7.

2. The RAID-0 I/O system

In this section, we shall first present the I/O system architecture of an RAID-0 from the view points of drivers and controllers.

2.1. System architecture

2.1.1. The driver model

The operating system provides a standard and uniform I/O interface for applications to access I/O devices. Through the I/O interface, applications can indirectly invoke vendor-supplied drivers to program the corresponding controllers/adaptors. In Fig. 1, we can see that the device drivers are usually divided into two levels: *upper-class module* and *device-specific module*. For example, the specification of *intelligent I/O* (I_2O) (Specification Ver 2.0, 1999) separates the *host-side drivers* into the *OS-Specific Module* (OSM) and the *Device Specialized Module* (DSM). The driver model proposed by Microsoft consists of *Class Drivers* and *Minidrivers*. Usually the OS venders/organizations provide upper-class modules for each category of devices to handle common system tasks. Hardware vendors supply lower-level device modules which include hardware-specific codes to perform lower-level operations. Fig. 1 also shows the connection between the host and the target. The drivers at the host communicate with the hardware controllers at the target via the system buses (for example, the PCI bus).

A real-time embedded operating system executing in a hardware controller (the target) receives I/O requests from the host via the system bus. It may be a simple interpreter responsible for connecting the I/O devices and the host operating system. For this case, I/O operations are initiated by invoking the appropriate handler functions of the drivers at the host. In some cases, it

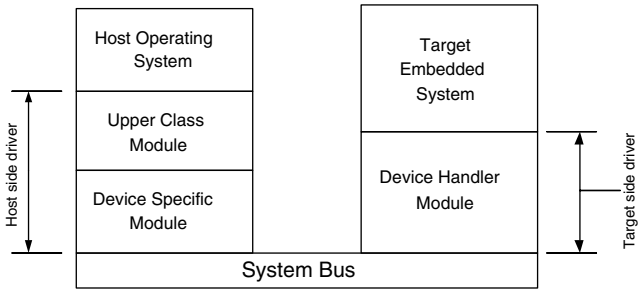


Fig. 1. The driver model.

may be a complex arbitrator of I/O devices and has capabilities to reduce the load of the host processor. In the latter implementation, the target embedded system is like a small general purposes operating system. It has its own device drivers to abstract the details in controlling I/O devices. For example, in an RAID-0, the system may share the CPU workload by migrating some of the RAID-related functions to the hardware controller. The system may provide an inexpensive solution by integrating some of the RAID functions into an upper-class module (for example, the filter driver of Microsoft Windows) to reduce the hardware cost. The proposed disk scheduling algorithm in the paper is suitable to both the implementations.

2.1.2. The hardware controller model

In this section, we will discuss the hardware controller architecture of an RAID-0 by an example product ACARD AEC 6850, which is an RAID adaptor that can support up to 75 disks. As shown in Fig. 2, the I/O system consists of two major components: *host* and *target*. A host can be a PC running a popular operating system such as Microsoft Windows. The host may have other I/O adaptors for other I/O devices (e.g., I/O Adapter A and I/O Adaptor B in the figure). The target is an RAID controller, such as ACARD AEC 6850 in this example. The interface between the RAID-0 controller and the host is a PCI bus. ACARD AEC 6850 has an embedded processor (such as Intel i960), memory, and up to 5 SCSI adaptors. All of the components are connected by the secondary PCI bus as shown in the figure. Each SCSI adaptor can manage up to 15 disks. (Note that IDE disks may adopt a similar hardware architecture.) The memory space of an adaptor maps to the memory addresses of the host by the *Address Translation Unit* so that the host and the target can communicate by DMA.

The RAID are designed to provide a high I/O performance by coordinating multiple disks to operate concurrently. An RAID, such as ACARD AEC 6850, manages a number of disks with data stripping. In particular, we are interested in RAID-0, in which data are stripped in units of blocks. An I/O *request* consists of a set of *jobs*, which may be served by the disks in an RAID concurrently. (I/O requests will be formally defined in Section 2.3.)

We illustrate the operations in an RAID-0 with four disks in Fig. 3. There is a message queue for the entire RAID-0 (named as the RAID Device Queue in the figure) and a message queue for each disk (named as the Disk Queue in the figure). Note that all the queues are inside the RAID. File systems or operating systems are not aware of the internal implementations. Each of the queues is a priority queue, and the message/event priorities are assigned by the real-time embedded operating system which is responsible for transforming an I/O request message in the RAID device queue into job messages. Then, it will insert the job messages into the corresponding disk queues. A job message contains all the necessary information for a disk to retrieve the required data of the job. In general, a thread is associated with each disk queue. If the thread is ready to process its messages, it will invoke the appropriate handler to select the highest priority job from the queue for processing.

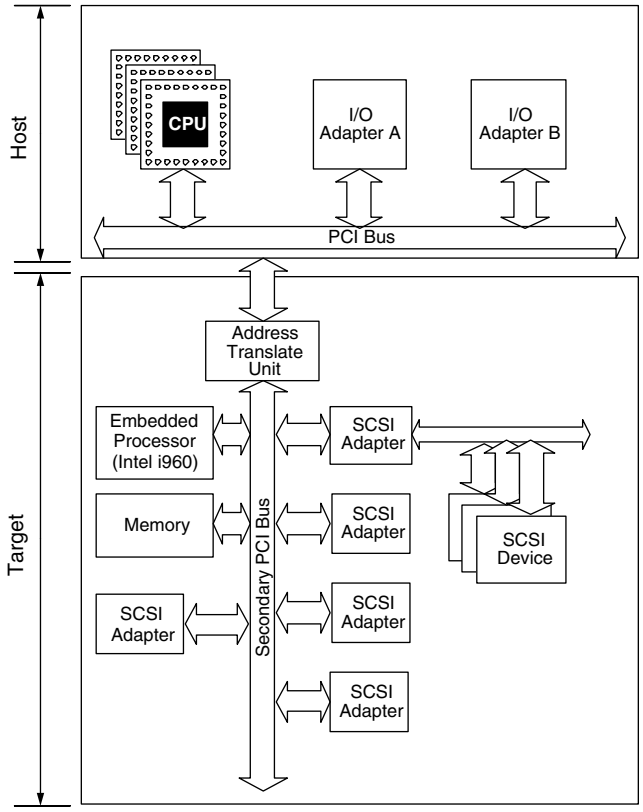


Fig. 2. The hardware model of an RAID.

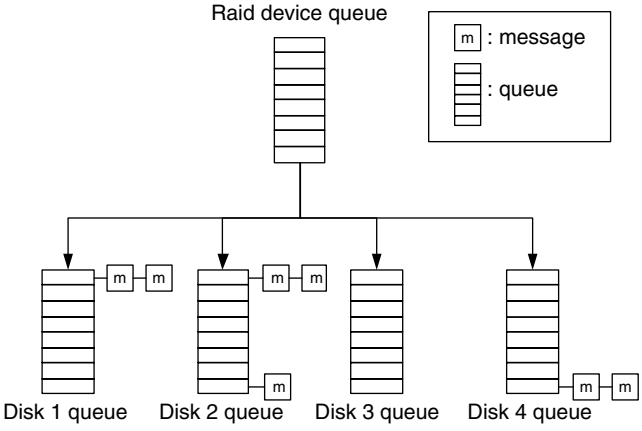


Fig. 3. The RAID device queue and disk queues in an RAID-0.

2.2. Serving a disk request in an RAID-0 device

Each I/O request r_i is modelled 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 of the requested data in bytes, and the deadline of the I/O request, respectively. With block stripping, an RAID-0 controller re-numbers the logical block addresses of the blocks over its disks. Suppose there are n disks managed by the RAID-0 controller, and the *block stripe size* (or *physical block size*) is B. A simple approach for the mapping is to map the jth LBA number of the ith disk to the $\left(n \times \frac{j}{B} + i - 1\right)$th LBA of the RAID-0 for $0 \leqslant j \leqslant$ maximum LBA number of $Disk_i$, and j should be a multiple of B and $1 \leqslant i \leqslant n$. For example, in the LBA re-numbering scheme adopted by the ACARD AEC 6850, the block stripe size is 32 sectors, and each sector is 512 B.

An I/O request $r_i=(arr_i, LBA_i, s_i, d_i)$ is divided into a set of jobs executing in different disks. Thus, an I/O request r_i is re-defined as a tuple $(arr_i, \{j_{dsk_1}, ord_1, \ldots, j_{dsk_n}, ord_n\}, d_i)$, in which each job has a disk number $(dsk_1, \ldots, dsk_n)$ for execution, a scheduling priority $(ord_1, \ldots, ord_n)$ according to the adopted single-disk scheduling algorithm [2] of the disk, a size $size(j_{dsk_k}, ord_k)$ in sectors, and a real LBA number $RLBA(j_{dsk_k}, ord_k)$ as its starting LBA on its assigned disk. Please note that a job in the hth disk with zero size means that the striping algorithm does not map the request to the hth disk.

The completion time of request r_i is the maximum of the completion times of all of its jobs. Therefore, in order to meet the deadline d_i of request r_i, all the jobs belonging to r_i must be completed no later than d_i. In this paper, we shall first propose the concept of request-based multi-disk scheduling, and then present the on-the-way and pre-fetching mechanisms to further improve the performance of an RAID-0.

3. The real-time scheduling problem in multi-disks

Although many efficient real-time and non-real-time disk scheduling algorithms have been proposed for single-disk systems, they may not be suitable to scheduling of real-time requests in multi-disk systems, such as the RAID-0. Fig. 4 shows a schedule of two disks in a multi-disk system using the shortest job first (SJF) algorithm. It is assumed that the jobs arrive at the same time. To simplify the discussion, we ignore the seek time and the rotation delay in serving a job. Let $J_{i,j}$ denote the jth job in the ith disk. The processing times of the jobs are

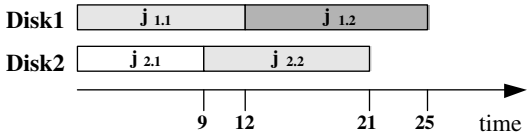


Fig. 4. The shortest-job-first schedule of the multi-disk with two disks.

Table 1
I/O job parameters

Job	Time to process the job (ms)	Request
$j_{1,2}$	13	r_1
$j_{2,1}$	9	r_2
$j_{1,1}$	12	r_3
$j_{2,2}$	12	r_3

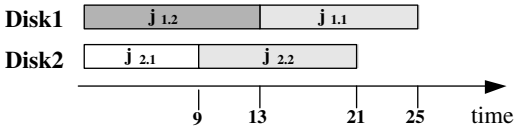


Fig. 5. The schedule after the swapping.

listed in Table 1. The average response time of the three jobs is 18.33 ms, e.g., (25 + 9 + 21)/3. The response time of a request is the maximum response time of its jobs. Referring to this example, the response times of requests r_1, r_2 and r_3 are 25, 9 and 21, respectively. As astute readers may point out, since r_3 already has a lengthy response time, it makes no sense to schedule $j_{1,1}$ before $j_{1,2}$. After swapping the executions of $j_{1,1}$ and $J_{1,2}$, as shown in Fig. 5, the average response time of the three requests is shortened and becomes 15.67 ms, e.g., (13 + 9 + 25)/3. This observation underlies the motivation of this research. Swapping the scheduling of the jobs at some disks may improve the overall system performance in meeting the deadlines of the requests.

Although it is highly important to maximize the performance of the disks in multi-disk scheduling, the consideration of each request as a logical unit for scheduling is of paramount important to maximize the performance of the whole multi-disk system, especially in meeting the deadlines of the requests. Nevertheless, we must emphasize that any multi-disk scheduling algorithms which consider the relationship among the jobs belonging to the same request over multiple disks should not sacrifice the performance of each individual disk too much. A compromise between multi-disk scheduling and single-disk scheduling must be made.

4. Least Request Size First (LRSF) for an RAID-0

In this section, we will propose a new multi-disk scheduling algorithm called *Least Request Size First*

[2] The proposed scheduling framework is an extension of conventional single-disk scheduling algorithms. Its purpose is to adjust the scheduling of the disks in an RAID-0 to improve the response times of the requests in meeting their deadlines.

(LRSF) for job scheduling in an RAID-0 following the framework introduced in Section 2. LRSF can combine with different single-disk scheduling algorithms adopted in the disks of an RAID-0. Disks with/without internal scheduling, such as SCSI and IDE disks, are the target drives of our proposed algorithm. When an RAID-0 controller receives a disk request, it will divide the request into a set of jobs to be served by the disks of the RAID-0 depending on the distribution of the required data of the request in the disks. The major scheduling problem is how to synchronize the scheduling of the multiple disks for serving the jobs belonging to the same request so that the deadline of the disk request can be satisfied, and at the same time the scheduling of the disks is least affected. It is because each disk has its own scheduling algorithm to schedule the jobs in its own disk queue to maximize its performance. The basic principle of LRSF is to assign a higher priority to the jobs of the request with the least remaining jobs in disk scheduling. The purpose is to improve the overall performance of the RAID-0 if the benefit obtained from raising the priorities of the jobs is significantly higher than the degradation in the performance of the disks where the jobs are residing.

The notations and abbreviations used in this paper are listed in Table 2. Consider an RAID-0 with N disks. All the jobs in the disk queues are first sorted according to the single-disk scheduling algorithms adopted in the disks, such as SSTF and EDF. Job $j_{x,i}$ is the ith job in the xth disk. Without loss of generality, the scheduling algorithm adopted in each disk assigns a profit $pf_{x,i}$ to a job $j_{x,i}$ to indicate the benefit (in terms of system throughput) in serving the job first. A higher priority is assigned to a job with a higher profit. For example, in SSTF, the profit of a job is determined based on the inverse of the distance between the r/w heads and the residing cylinder of the job because SSTF assumes that serving the job with the shortest seek time will give the greatest benefit to the system.

In LRSF, we raise the priorities of the jobs belonging to the same request with the least remaining size if the degradation on the disk performance is small. The details of the procedure is shown in Algorithm 1. As shown in the algorithm, for each disk queue, we first identify what the profit of each job in the disk queue is. Let $j_{x,y}$ be a job in the disk queue of the xth disk, and $j_{x,1}$ be the first job in the disk queue. $j_{x,y}$ should be scheduled before $j_{x,1}$ if the performance degradation in the xth disk is small. The amount of degradation can be approximated by comparing the profit functions for $j_{x,1}$ and $j_{x,y}$. If the ratio of their profits is higher than a system pre-defined threshold TH, $j_{x,y}$ will be selected and join a higher priority job set ($\mathbb{T}_1$ in the algorithm). Then, the jobs with the least remaining size in the higher priority job set will be identified and collected into the set $\mathbb{T}_2$. The remaining size $rs_{x,y}$ of a job $j_{x,y}$ is the number of remaining jobs of the corresponding request. Obviously, all the jobs belonging to the same request have the same remaining request size. Then, we select the one having the highest profit for processing.

We shall use an example to illustrate the procedures of LRSF for an RAID-0 in which all the disks adopt SSTF for job scheduling.

Example 1. (LRSF with SSTF:) It is assumed that the RAID-0 controller is connected to three HP97560 SCSI disks with rotation speed of 4002 rpm, and the seek time is modeled by the following formula (HPL-CSP-91-7, 1991; Ruemmler and Wilkes, 1994)

$$\text{seek time} = 3.24 + 0.4 \times \sqrt{\text{seek distance}} \quad (1)$$

where *seek distance* is measured in number of cylinders. The transfer time of a sector, which is equal to 512 B, is approximated to 0.23 ms (i.e., the time to scan a sector). Suppose the r/w heads of the three disks are originally located at cylinder numbers 55, 75, and 65, respectively. The jobs together with their request numbers, cylinder numbers, seek distances, profit(seek time) and disk

Table 2
The summary of the abbreviations

Abbreviation	Meaning
LBA	Logical Block Address—LBA is a value to indicate the logical location of data in an RAID-0. It is used to hide the actual cylinder-head-sector address of the data in the disk
SSTF	Shortest Seek Time First—A disk scheduling algorithm which serves the disk job with the minimum seek time
SATF	Shortest Access Time First—SATF is similar to SSTF but it serves the disk job with the minimum position time which is the time spent on moving the r/w head to the target position
C-LOOK	A disk scheduling algorithm which serves the disk job on its way. At each run, it sweeps the disk from one side to the other side
SSEDV	Shortest Seek Time Earliest Deadline by Value—An heuristic that uses weighted sum of seek time and deadline as the priorities of the jobs
EDF	Earliest Deadline First—A scheduling policy which gives the highest priority to the job with the closest deadline
RRS	Remaining Request Size—The number of remaining jobs of a request
$j_{x,i}$	The ith job in the queue of the xth disk
$pf_{x,i}$	The profit obtained from serving $j_{x,i}$ using the adopted single-disk scheduling algorithm
$j_{x,1}$	The first job in the queue of the xth disk. $j_{x,1}$ has the greatest profit compared to the other jobs in the same queue

numbers are listed in Table 3. Fig. 6a shows the disk queues sorted according to SSTF. The response times of the jobs are listed in Table 4. The average response time of the requests is 9.58 ms.

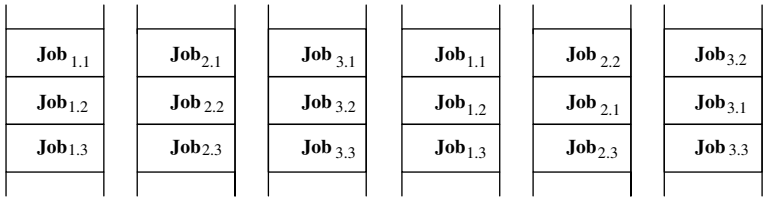
Suppose LRSF is adopted at the RAID-0 controller to synchronize the scheduling of the disks connected to the controller. Let the threshold TH be 0.75. Since $job_{1,1}$ belonging to r_3 has the least remaining size, and it is the 1st job in the queue, $job_{1,1}$ is scheduled for service in the first disk. In the second disk, since $job_{2,2}$ has the least remaining size in the second disk and $\frac{pf_{2,2}}{pf_{2,1}} = \frac{4.04\ \text{ms}}{4.04\ \text{ms}} = 1$ is larger than the threshold 0.75, $job_{2,2}$ is scheduled for service in the second disk although $job_{2,1}$ is the first job in the second disk queue. In the third disk, because $\frac{pf_{3,2}}{pf_{3,1}} = \frac{3.93\ \text{ms}}{4.22\ \text{ms}} = 0.933$ is larger than the threshold 0.75, $job_{3,2}$ is scheduled for service in the third disk. The response times of the jobs using LRSF-SSTF is listed in Table 5. The disk queues sorted according to LRSF-SSTF are shown in Fig. 6b. The average response time of all the requests is 8.79 ms. It is about 8% better than the average response time, compared with the case without LRSF.

Algorithm 1. (Least Request Size First (LRSF))

1: **input:** a sorted job set $\mathbb{J}_x$ of $disk_x$ which contains n_x jobs $j_{x,1},\ldots,j_{x,nx}$. Each job $j_{x,y}$ in $\mathbb{J}_x$ has a profit $pf_{x,y}$ and a remaining size $rs_{x,y}$. TH is a threshold.
2: **output:** a job from $disk_x$
3:
4: /** The following steps collect the jobs which may not degrade the disk throughput too much into the set $\mathbb{T}_1$. Please note that $j_{x,1}$ is the job having the highest profit. */

Table 3
Job parameters

Job	Request number	Cylinder number	Seek distance	Profit (1/seek time)	Disk number
$J_{1,3}$	r_1	39	16	1/4.84	1
$J_{3,3}$	r_1	39	26	1/5.28	3
$J_{2,1}$	r_2	71	4	1/4.04	2
$J_{3,2}$	r_2	71	6	1/4.22	3
$J_{1,1}$	r_3	52	3	1/3.93	1
$J_{1,2}$	r_4	62	7	1/4.30	1
$J_{2,3}$	r_4	62	12	1/4.63	2
$J_{3,1}$	r_4	62	3	1/3.93	3
$J_{2,2}$	r_5	79	4	1/4.04	2



(a) SSTF **(b)** LRSF-SSTF

Fig. 6. SSTF schedules with/without LRSF.

Table 4
Response times of the jobs using SSTF

Jobs in disk 1	Response time (ms)	Jobs in disk 2	Response time (ms)	Jobs in disk 3	Response time (ms)
$J_{1,1}$	3.93	$J_{2,1}$	4.04	$J_{3,1}$	3.93
$J_{1,2}$	9.09	$J_{2,2}$	8.41	$J_{3,2}$	8.37
$J_{1,3}$	13.60	$J_{2,3}$	13.30	$J_{3,3}$	13.88

Table 5
Response times of the jobs using LRSF-SSTF

Jobs in disk 1	Response time (ms)	Jobs in disk 2	Response time (ms)	Jobs in disk 3	Response time (ms)
$J_{1,1}$	3.93	$J_{2,2}$	4.04	$J_{3,2}$	4.22
$J_{1,3}$	8.62	$J_{2,1}$	8.41	$J_{3,1}$	8.66
$J_{1,2}$	13.77	$J_{2,3}$	12.85	$J_{3,3}$	13.82

```
5: for all job j_{x,y} ∈ 𝕁_x do
6:   if pf_{x,y}/pf_{x,1} > TH then
7:     𝕋_1 ← 𝕋_1 ∪ {j_{x,y}}
8: /** 𝕋_2 contains the jobs which belong to
   the request with the least remaining
   size
   */
9: tempSize ← 0
10: for all job j_{x,y} ∈ 𝕋_2 do
11:   if rs_{x,y} > tempSize then
12:     𝕋_2 ← {j_{x,y}}
13:     tempSize ← rs_{x,y}
14:   if rs_{x,y} = tempSize then
15:     𝕋_2 ← 𝕋_2 ∪ {j_{x,y}}
16:
17: /** Select a job with the largest profit
   from the set 𝕋_2 */
18: tempProfit ← −∞
19: forall job j_{x,y} ∈ 𝕋_2 do
20:   if pf_{x,y} > tempProfit then
21:     returnJob ← j_{x,y}
22:     tempProfit ← pf_{x,y}
23: output returnJob
```

5. Extensions

In this section, we will present two methods to extend LRSF by considering the characteristics of an RAID-0. We shall first propose to apply the *on-the-way* (OTW) mechanism (HPL-CSP-91-7, 1991) into LRSF to resolve the starvation problem of large-size I/O requests in LRSF. Since LRSF assigns a higher priority to a smaller-size I/O request, the waiting time of the jobs of a large-size request can be very long. Secondly, we will discuss how to apply pre-fetching mechanisms to further improve the performance of a disk in an RAID-0.

5.1. On-the-way scheduling

One of the major reasons for the good performance of SCAN and LOOK, especially when the workload is heavy, is because they always serve jobs collectively on the way to minimize the movement of the disk heads in searching the required sector of a job. This is also the main reason why many real-time scheduling algorithms, such as EDF, do not perform well when they are applied directly in disk scheduling.

In this section, we apply the piggyback concept (referred to as the on-the-way (OTW) scheme) into LRSF to further improve the performance of an RAID-0 and, at the same time, to resolve the starvation problems of large-size requests. Compared to a seek-time-optimization algorithm (e.g. LOOK, SSTF), the OTW mechanism considers not only the seek distance but also the rotation delay. We use the following example to illustrate the concept of "on-the-way" scheduling.

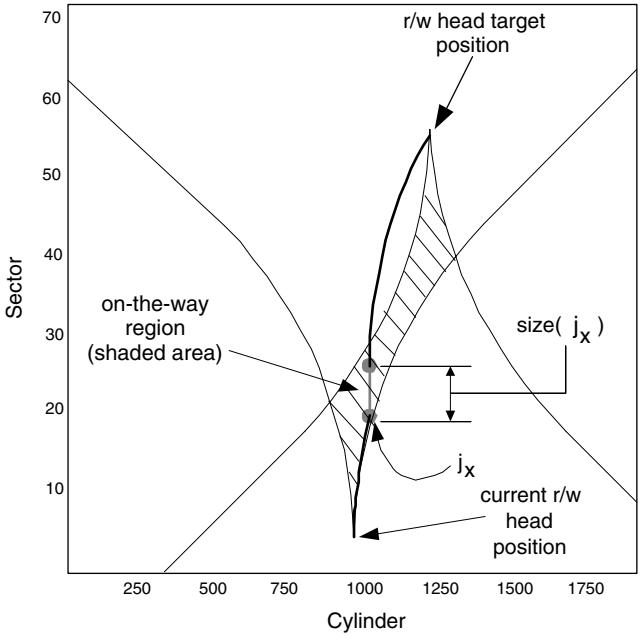


Fig. 7. The on-the-way region.

Let the current r/w heads of a disk stay at the 4th sector of the 975th cylinder as shown in Fig. 7. [3] Since the rotation speed is constant, we can use the distance in sectors to denote the length of time to serve a job. Because the r/w heads accelerate while they are moving across the cylinders towards their target cylinders, the movement curves of the r/w heads in the figure are parabolic. Since the acceleration of r/w heads is independent of the cylinder where they are currently residing, the parabolic curves in Fig. 7 are symmetric. Suppose the disk scheduler decides that the next job to be served is the one at the 55th sector of the 1250th cylinder according to the adopted scheduling algorithm. As shown in Fig. 7, while the r/w heads move from the 975th cylinder to the 1250th cylinder, the r/w heads will travel from the 4th sector to the 40th sector. In order to go to the target position, the r/w heads need to wait until the disk rotates from the 40th sector to the 55th sector. The shaded area shown in Fig. 7 denotes the collection of blocks that can be served on the way from the current r/w heads position to the target position without any extra delay. For example, the r/w heads can serve job j_x. We call the shaded area as the *on-the-way* region of the current head position to the target position.

The on-the-way concept provides an important consideration to improve the performance of a disk in an RAID-0. As the r/w heads move from the current job to the next job (i.e., from the current head position to

[3] A similar figure appears in HPL-CSP-91-7, 1991.

the target position), the jobs which are within the on-the-way region may be served regardless of their remaining request sizes and the job priorities assigned by LRSF and the adopted single-disk scheduling algorithm. Algorithm 2 describes the details of how to apply the OTW into the scheduling of the disks in RAID-0.

Algorithm 2. (On-The-Way)

1: **input:** a job set $\mathbb{J}_x$ of $disk_x$ which contains n_x jobs $j_{x,1},\ldots,j_{x,n_x}$. Each job $j_{x,y}$ in $\mathbb{J}_x$ has a cylinder number $cyl_{x,y}$ and a sector number $sec_{x,y}$ to denote the position of its required data in the disk. A target job j_{tar} in $\mathbb{J}_x$ and the r/w heads position ($cyl_{r/w}$, $sec_{r/w}$) are also given. The sector number and cylinder number of j_{tar} are sec_{tar} and cyl_{tar}, respectively.
2: **output:** a job which can be served without any additional movement of the r/w heads.
3:
4: $D_{seek} \leftarrow |cyl_{\mathrm{r/w}} - cyl_{tar}|$ /** D_{seek} is the seek distance between the target job and the current r/w head position. */
5: $t_{seek} \leftarrow SeekTime(D_{seek})$ /** The seek time of j_{tar}. */
6: $D_{rot} \leftarrow sec_{\mathrm{r/w}} + TimeToRotationDistance(t_{seek}) - sec_{tar}$ /** The rotation distance from the sector under the r/w heads (when the r/w head arrives the target cylinder) to the target sector (sec_{tar}). */
7: $t_{slack} \leftarrow RotationDistanceToTime(D_{rot})$/** The rotation delay might be eliminated. */
8: **for all** $j_{x,y} \in \mathbb{J}_x - \{j_{tar}\}$ **do**
9: $D_{seek'} \leftarrow |cyl_{\mathrm{r/w}} - cyl_{x,y}|$
10: $t_{seek'} \leftarrow SeekTime(D_{seek'})$
11: $D_{\mathrm{rot}'} \leftarrow sec_{\mathrm{r/w}} + TimeToRotationDistance(t_{seek'}) - sec_{x,y}$
12: $t_{\mathrm{trans}'} \leftarrow RotationDistanceToTime(size(j_{x,y}))$
13: $t_{access'} \leftarrow t_{seek'} + t_{rot'} + t_{trans'}$ /** The access time of $j_{x,y}$ is $t_{access'}$ if we serve $j_{x,y}$ before j_{tar} */
14: **if** $t_{access'} \leqslant t_{slack}$ **then**
15: **output** $j_{x,y}$
16: **return** /** exit */

output fail

The application of the on-the-way mechanism into LRSF has three important advantages: (1) The on-the-way mechanism improves the performance of large-size requests because the jobs of large-size requests may be served along the way while the disk is serving a job of a small-size request. (2) The on-the-way mechanism also improves the performance of real-time single-disk scheduling algorithms such as EDF. Note that with the on-the-way mechanism, EDF may serve jobs collectively, similar to LOOK and C-LOOK, without any additional movement of the r/w heads. (3) The on-the-way mechanism may improve the performance of non-real-time requests without sacrificing the performance of real-time jobs.

Please be reminded that although the on-the-way mechanism can partly resolve the starvation problem of large-size requests, some large-size requests may still suffer from the starvation problem if their jobs are always located at the cylinders outside the on-the-way regions of the jobs of small-size requests. The starvation problem could be serious if LRSF is used with a single-disk scheduling algorithm which also has the starvation problem, such as SSTF. A simple but effective way to resolve the starvation problem in such a situation is to move the jobs of the larger-size (or starved) requests forward by raising their priorities if they have been waited for service for a long time, i.e., being larger than a waiting threshold. However, how to raise the priorities of requests is another critical design issue. The performance of LSRF can be sensitive to the priority adjustment method.

5.2. Pre-fetching in RAID-0

An RAID-0 may be equipped with a large amount of memory for pre-fetching or only a small amount of memory merely sufficient for system operations. This section is to explore the pre-fetching mechanisms for both the cases. We must emphasize that pre-fetching in an RAID-0 offers a different kind of performance improvement, compared to the pre-fetching in the operating system. With the knowledge of the RAID-0 configuration and the workloads of the disks, different disks may be initiated in parallel to perform pre-fetching to reduce the service delay of a disk request. There are two ways to pre-fetch disk data into the memory: First, if the disk workload is not very heavy, a smart way of pre-fetching at the RAID controller is to utilize the disks that are idle for pre-fetching. For this case, pre-fetching could be obtained with almost "free" of charge. Of course, the service of the disk cannot be interrupted while pre-fetching is performing. Therefore, if a new request enters the system while the disk is performing a pre-fetching operation, the disk will not be able to serve the request until the pre-fetching is completed. In this situation, the performance of random access will be degraded slightly. Secondly, we can use the on-the-way (OTW) region to serve pre-fetch commands. In this case, the priority given to a pre-fetching job should be lower than those of real-time and non-real-time jobs. Large-size requests and non-real-time requests may compete for the OTW service.

If an RAID-0 is equipped with a large amount of memory for pre-fetching operations, the pre-fetching can be done intuitively: a segment of memory may be allocated as a buffer region. When a request r requests to access s bytes starting from the LBA number lba,

the RAID-0 may issue another request r_{next} to access s bytes starting from the LBA number ($lba+s$). The deadline of r_{next} can be twice of the deadline of r_i. The buffer region can be managed using a common memory management schemes, such as the least-recently used (LRU) or FIFO schemes. When an RAID-0 is only equipped with a small amount of memory for its system operations, pre-fetching is still possible. Pre-fetching can be done by issuing SCSI commands, such as "PRE-FETCH" (the 0x34 SCSI command), so that the disks are given hints to try to cache sectors.

With a powerful processor such as ARM, it is possible to execute a more complicated pre-fetching mechanism inside an RAID-0 device. For example, if a disk is idle, then it pre-fetches the sectors whose LBA numbers are close to the LBA numbers of the existing requests. As a result, in the ideal case, an application may never need to wait for disk operations to retrieve data. It may happen that when an application sequentially reads data in the LBA number order, the disks always finish pre-fetching in time to obtain the data needed by the application.

6. Performance evaluation

6.1. Performance metrics and data sets

The experiments described in this section are: (1) to evaluate the performance of the proposed LRSF multi-disk scheduling algorithm and the on-the-way (OTW) scheme in scheduling RAID-0 requests and (2) to evaluate the effectiveness of the pre-fetching mechanism in improving the performance of the proposed RAID-0 framework when different disk scheduling algorithms are adopted for single-disk scheduling. We have implemented a simulation model following the RAID-0 framework introduced in Section 2 of the paper.

Since we are interested in disk scheduling where the requests have soft real-time deadlines, the primary performance metric used are the miss ratio and the average response time of the requests. Miss ratio is defined as the ratio of the requests that miss their deadlines over the total number of requests generated. The deadline of a request is calculated as $TimeMultiplier \times (RWVTimeout\text{-}Base + (RWVTimeout \times size/64\ \text{K}))$. The generations of the requests follow the Poisson distribution. Each request may request data of a size ranging from 32 to 256 sectors. The block strip size is 32 sectors. In the experiments, eight Maxtor Atlas 10 disks are simulated to connected to an RAID-0 controller. The simulation parameters and their baseline values are summarized in Table 6.

We have performed two sets of experiments. In the first set of experiments, we study the effectiveness of using LSRF and WTO in improving the performance of an RAID-0 when different single-disk scheduling algorithms, such as the earliest deadline first algorithm (EDF), Shortest Access Time First (SATF) (HPL-CSP-91-7, 1991), FIFO, C-LOOK, and the shortest-seek-time-earliest-deadline-by-value (SSEDV), are adopted in the RAID-0 using randomly generated workload based on the parameters of a real disk, Maxtor Atlas 10 K. [4] The chosen single-disk scheduling algorithms cover a range of performance for serving real-time disk requests. It is known that FIFO and EDF may not be good to real-time disk requests while SATF and SSEDV are more efficient. The purpose is to test the performance improvement from LRSF and OTW when they are combined with disk scheduling algorithms with different capabilities. In the second set of experiments, the workload is defined based on real workload to assess the improvement in performance from the pre-fetching mechanism when different single-disk scheduling algorithms are adopted in the RAID-0. Note that disk scheduling algorithms with pre-fetching should be evaluated using real workloads to have a meaningful performance study. Two disk benchmarks are tested. They are "Business" and "High-End" workloads in WinBench98. [5] The "Business" workload is for applications such as databases and graphic playback software such as Adobe PhotoShop, and the "High-End" workload is for applications such as Visual C++.

An important factor on the performance of LRSF is the threshold value TH which determines under which situations the priority of a job in a disk queue may be raised up for processing. From our experiments, we observed that the optimal threshold value TH for LRSF depends on the disk scheduling algorithm adopted by the disks in an RAID-0. Table 7 summarizes the profit function and the threshold value used in LRSF for each simulated scheduling algorithm. The profit function of a scheduling algorithm is defined based on its own scheduling discipline. The threshold values (TH) for different disk scheduling algorithms shown in Table 7 are determined from a series of experiments to optimize the performance of LRSF for the algorithm. It has been found that the optimal TH for an algorithm depends on the system workload. For example Fig. 8 shows the miss ratio when different TH values are used for LRSF in an RAID-0 using C-LOOK for scheduling the disks. The impacts of TH on the other tested disk scheduling algorithms are similar. The five curves in the figure denote five different request inter-arrival times from 0.01 to 0.005 s. It can be seen that the miss ratio of C-LOOK is not very sensitive to the value of TH, and the best value for TH is around 0.5 in C-LOOK. In general, a

[4] Maxtor Corp. http://www.maxtor.com/en/products/scsi/atlas_10k_family/index.htm.

[5] ZDNet Corp. http://www.zdnet.com/etestinglabs/stories/benchmarks/0,8829,2326114,00.html.

Table 6
Simulation parameters

Parameters	Value	Remark
TimeMultiplier	1 ~ 30	
RWVTimeoutBase	0.5	in second
RWVTimeout	0.1	in second
Arrival pattern	mean = 3–12	Poisson distribution
Block stripe size	32 sectors	1 sector = 512 byte
LBA range	0–143,532,240	
Request size	32–256	in sector
Number of disks	8	
Disk model name	Maxtor Atlas 10 K [4] (Bucy and Ganger, 2003)	
Simulation length	600,000	The number of issued requests

Table 7
The value function of each algorithm

Algorithm	Profit function ($pf_{x,y}$)	Threshold (TH)
SATF	the access time of $j_{x,y}$	0.8
EDF	the deadline of $j_{x,y}$	0.6
SSEDV	the weighted sum of the deadline and the seek time of $j_{x,y}$	0.6
C-LOOK	the seek time of $j_{x,y}$	0.5
FIFO	the arrival time of $j_{x,y}$	0.0

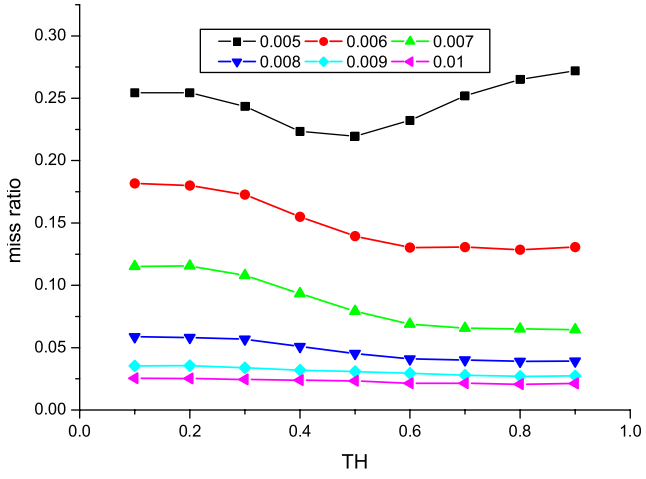


Fig. 8. Values of TH versus miss ratio in C-LOOK.

highly efficient disk scheduling algorithm, such as SATF, usually has a larger threshold to prevent LRSF from carelessly swapping jobs in a disk queue.

6.2. Experiment results

6.2.1. Evaluation on effectiveness of using LSRF and OTW

Fig. 9a and b show the miss ratio and the average response time of requests in FIFO with/without LRSF and OTW, respectively. The inter-arrival time of requests varies from 8 to 12 ms. When the inter-arrival time is smaller than <8 ms, the performance of FIFO is very poor. From the figures, we can see that, in general, FIFO/LRSF improves FIFO by around 30–2% in average response time and by arround 25–4% in miss radio. OTW improves the performance further by around 25–1% and 15–2% in miss ratio and average response time, respectively. Consistent with our expectation that the improvement is higher when the workload is heavier, i.e., smaller inter-arrival time. Urgent requests will have a higher probability of missing their deadlines when the workload is heavy. Raising the priorities of the jobs belonging to an urgent request can improve the system performance.

Fig. 10a and b show the miss ratio and the average response time of requests in EDF with/without LRSF and OTW, respectively. EDF with LRSF and OTW (EDF/LRSF-OTW) and EDF with LRSF (EDF/LRSF) greatly out-perform EDF in both miss ratio and average response time when the workload is heavy. For example, when the inter-arrival time of requests is 0.008 s, the miss ratio of EDF/LRSF-OTW is only about 25% that of EDF only. When the inter-arrival time of requests is large, e.g., 0.01 s, although the improvement is smaller, LRSF and OTW can still improves the average response time of EDF significantly. This is consistent with our expectation since the performance EDF is highly sensitive towards the workload. If workload is heavy, a lot of requests will miss their deadlines due to frequent movement of the r/w heads. With the use of LRSF and OTW, more requests can meet their deadlines by raising the priorities of their jobs in the disk queues.

Fig. 11a and b show the miss ratio and the average response time of requests in C-LOOK with/without LRSF and OTW, respectively. In general, the improvement from LRSF and OTW, especially OTW, is signifi-

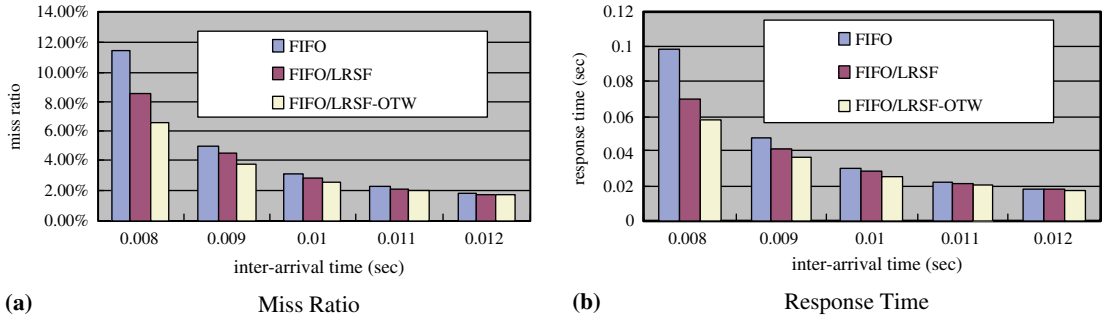


Fig. 9. Miss ratio and average response time of FIFO with/without LRSF and OTW.

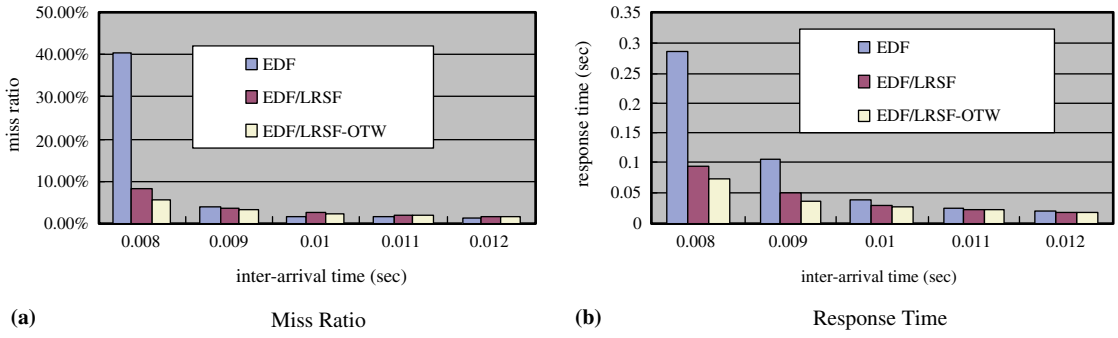


Fig. 10. Miss ratio and average response time of EDF with/without LRSF and OTW.

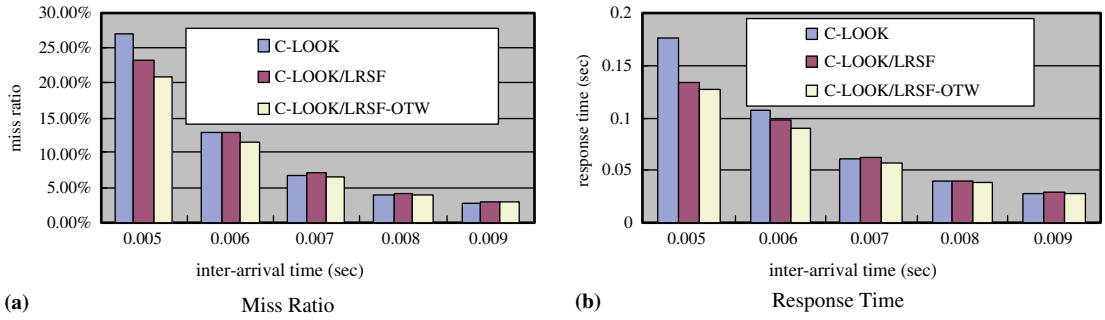


Fig. 11. Miss ratio and average response time of C-LOOK with/without LRSF and OTW.

cant. OTW improves the performance significantly because OTW considers the optimization in rotation delay while C-LOOK only considers seek time. C-LOOK with LRSF and OTW gives the best performance especially when the workload is heavy comparing to C-LOOK only and C-LOOK with LRSF. If the workload is heavy, C-LOOK/LRSF-OTW can improve the average response time and the miss ratio of C-LOOK by around 25% and 15%, respectively.

Fig. 12a and b show the miss ratio and the average response time of requests in SATF with/without LRSF and OTW, respectively. When the inter-arrival time of requests increases from 3 to 7 ms, the performance difference among SATF with LRSF and OTW (SATF/LRSF-OTW), SATF with LRSF (SATF/LRSF), and SATF only (SATF) decreases gradually. It is because SATF is a very efficient disk scheduling algorithm, the improvement from LRSF and OTW is small when the workload is not heavy. Note that SATF was shown to be better than many traditional disk scheduling algorithms such as C-LOOK. In general, SATF/LRSF improves SATF by around 10% in average response time and by around 20% in miss ratio.

Fig. 13a and b show the performance improvement of LRSF and OTW in SSEDV. SSEDV considers the reordering of disk jobs according to the weighted sum of their seek times and deadlines. If the inter-arrival time is increased, the average response time of SSEDV decreases gradually. However, the miss ratio decreases dramatically when the inter-arrival time is decreased from 0.008 to 0.009. This phenomenon shows that SSEDV is a very good real-time disk scheduling

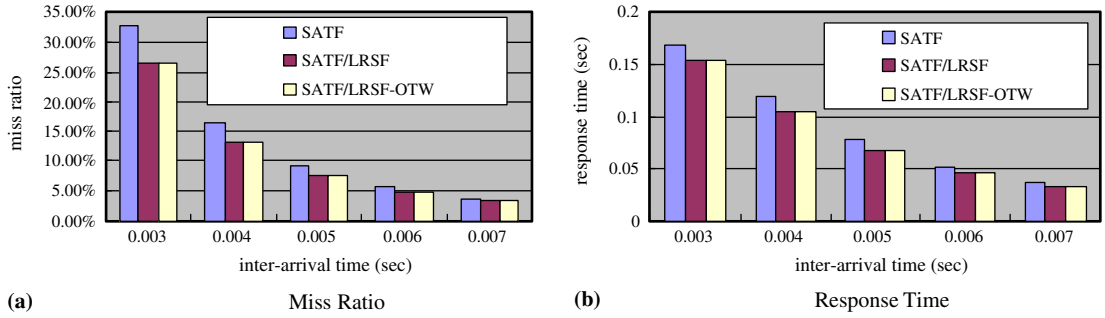


Fig. 12. Miss ratio and average response time of SATF with/without LRSF and OTW.

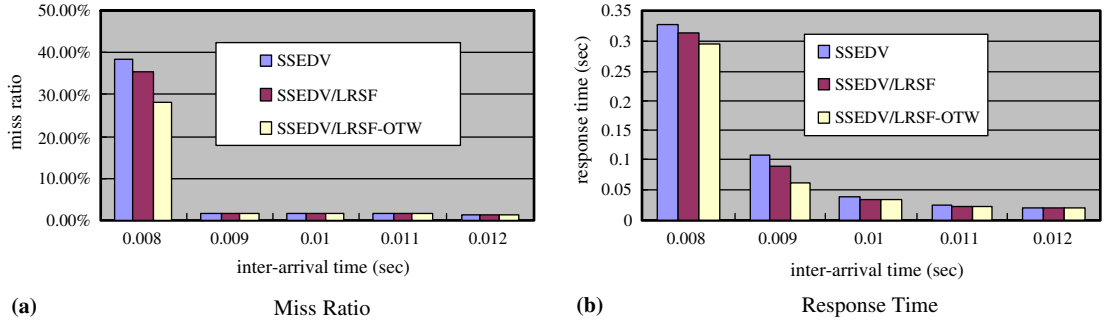


Fig. 13. Miss ratio and average response time of SSEDV with/without LRSF and OTW.

Table 8
The disk utilizations of the proposed algorithms

Algorithms and inter-arrival time	FCFS	EDF	CLOOK	SATF	SSEDV
0.008	0.008	0.006	0.005	0.008	
Disk algorithm only	0.999	0.995	0.172	0.480	0.994
Disk algorithm/LRSF	0.995	0.993	0.907	0.907	0.994
Disk algorithm/LRSF-OTW	0.992	0.992	0.689	0.670	0.753

algorithm if the system has sufficient disk bandwidth. SSEDV/LRSF-OTW still outperforms SSEDV by around 5% if the workload is light, e.g., inter-arrival time larger 9 ms, and the improvement is higher when the workload is heavier, i.e., inter-arrival time = 8 ms.

Table 8 shows the disk utilization of different single-disk scheduling algorithms when LRSF, and LRSF-OTW are applied in the algorithms. In the table, we show the results of the algorithms when the workload is heavy for better comparison. As shown in the table, having LRSF in the scheduling algorithms (C-LOOK, SATF, SSEDV) increases the disk utilization. It is because since the service time of a job is optimized by the adopted single-disk scheduling algorithm, the use of LRSF increases the average service time of the jobs as LRSF may raise the priority of a job waiting in the disk queue for service, instead of choosing the highest priority job in the queue. If the disk serves the highest priority job, the service time should be the smallest. For example, in SATF, the highest priority job in the queue should be the one with the shortest access time. However, some algorithms (EDF, FCFS) do not consider the movement of the r/w heads of a disk. Using LRSF in these algorithms does not significantly increase the access time. Comparing LRSF with LRSF-OTW, we can see that the disk utilization of LRSF-OTW is lower since the problem of higher service time as a result from raising the priority of a job in the disk queue is minimized by the OTW scheme.

6.2.2. *WinBench98-based results for pre-fetching*

While the previous section explores the performance improvement from LRSF and OTW in an RAID-0 with different disk scheduling algorithms, the purpose of this section is to assess the performance improvement from the pre-fetching mechanism using realistic workloads when different disk algorithms are used in the RAID-0. We consider an RAID-0 controller with eight SCSI disks. We simulate a disk drive having 4 MB internal caching space for pre-fetching, and the controller has

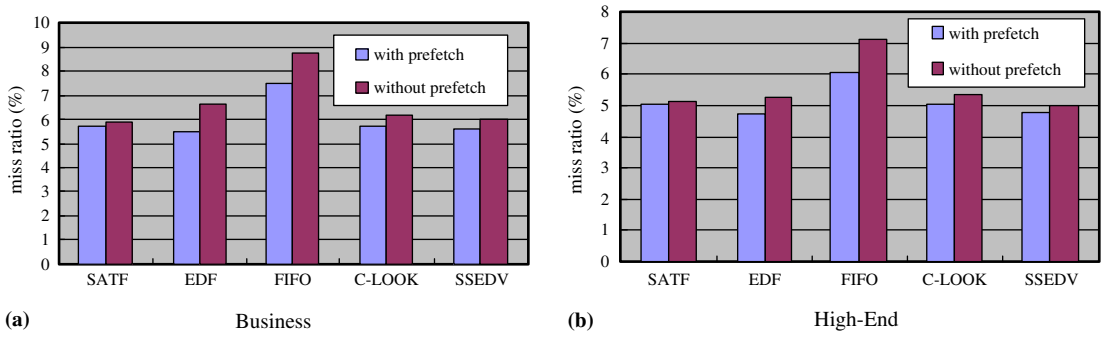


Fig. 14. Miss ratio of different disk scheduling algorithms using the WinBench98 "Business" and "High-End" workloads.

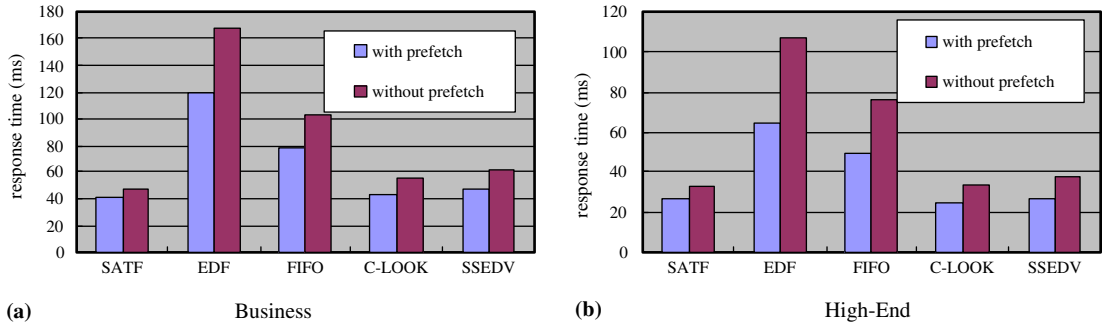


Fig. 15. Average response time of different disk scheduling algorithms using the WinBench98 "Business" and "High-End" workloads.

no caching space. We must emphasize that if the controller has non-zero caching space, the improvement in results will be even higher.

Figs. 14 and 15 show the average response time and miss ratio of different disk scheduling algorithms with/without pre-fetching with the WinBench98 "Business" and "High-End" workloads, respectively. Consistent with the results from the first set of experiments, SATF out-performs other scheduling algorithms in general, especially in average response time. As shown in the figures, the pre-fetching mechanism can significantly improve the less-efficient single-disk scheduling algorithms such as EDF and FIFO, e.g., about 20–30% improvement in response time for EDF and FIFO with both WinBench98 "Business" and "High-End" workloads. Even for SATF, the pre-fetching mechanism improves the response-time (miss-ratio) by around 10%(3%) in the "Business" workload and by around 20%(3%) in the "High-End" workload. As astute readers might point out, the pre-fetching mechanism can be used together with LRSF and OTW, as shown in the previous section. Similar improvement will be achieved.

7. Conclusion

With the advent of high-performance applications and the growing demand for applications with soft real-time constraints, there is an increasing demand of I/O systems which must perform in a soft real-time fashion. Multi-disk systems such as RAID-0 are one of the important choices for the application systems with stringent response-time requirement. In this paper, we study real-time multi-disk scheduling in RAID-0 to improve the I/O performance to minimize the number of deadline violations and mean response time. We propose a request-based real-time multi-disk scheduling algorithm called *Least-Remaining-Request-Size-First* (LRSF), which can be integrated with different real-time/non-real-time single-disk scheduling algorithm, such as SATF and C-LOOK. To minimize the starvation problem to large disk requests, we also have studied how to incorporate the on-the-way mechanism into LRSF. Pre-fetching in RAID-0 is also investigated to further improve the I/O system performance. The performance of the proposed algorithm and mechanisms combined with different single-disk scheduling algorithms, such as C-LOOK and EDF, is studied using randomly generated and realistic workloads.

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

Acknowledgements

The work described in this paper was partially supported by grants from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 1078/00E] and was supported in part by a research grant from National Science Council, Taiwan under Grant NSC 92-2213-E-002-065.

Appendix A. The *TH*-selection algorithm

Algorithm "*TH*-Selection" is used to derive the optimal value for *TH*, given a collection of workloads in a system. The LRSF algorithm is executed for various *TH* values and different workloads with a given step granularity to search for the best value. Suppose that the performance metric is the miss ratio. The best value for *TH* is picked up that results in the minimum miss ratio for the given workloads.

Algorithm 3. (*TH*-Selection)

1: **input:** A set W of workloads. A number *step* ($0 < step < 1$) to denote the granularity for the searching of the best *TH*.
2: **output:** A real number *TH*.
3: $TTH \leftarrow 0$, $minTotalMissRatio \leftarrow \infty$
4: **while** $TTH < 1$ **do**
5: **for all** workload $w_i \in W$ **do**
6: Run the LRSF algorithm with the selected single-disk scheduling algorithm, the workload w_i and the given threshold *TTH* to obtain the miss ratio $miss_i$.
7: $totalMissRatio \leftarrow totalMissRatio + miss_i$
8: **if** $totalMissRatio < minTotalMissRatio$ **then**
9: $minTotalMissRatio \leftarrow totalMissRatio$, $TH \leftarrow TTH$
10: $TTH \leftarrow TTH + step$

output *TH*

References

Abbott, R.K., Garcia-Molina, H., 1990. Scheduling I/O requests with deadlines: a performance evaluation. In: IEEE 11th Real-Time Systems Symposium, December 1990. IEEE Computer Society Press, Lake Buena Vista, Florida, USA, pp. 113–124.

Bruno, J., Brustoloni, J., Gabber, E., Ozden, B., Silberschatz, A., 1999. Disk scheduling with quality of service guarantees. In: IEEE International Conference on Multimedia Computing Systems. Florence, Italy, 7–11 June 1999, pp. 400–405.

Bucy, J.S., Ganger, G.R., et al., (CMU-TR-243-95) 2003. The DiskSim Simulation Environment Version 3.0 Reference Manual. U.Michigan Technical Report.

Chang, R.-I., Shih, W.-K., Chang, R.-C., 1998. Deadline-modification-SCAN with maximum-scannable-groups for multimedia real-time disk scheduling. In: IEEE 19th Real-Time Systems Symposium. Madrid, Spain, 2–4 December 1998, pp. 40–49.

Chang, P., Jin, H., Zhou, X., Chen, Q., Zhang, J., 1999. HUST-RAID: High performance RAID in real-time system. In: IEEE Pacific Rim Conference on Communication, Computers, and Signal Processing. Victoria, B.C., Canada, 23–25 August 1999, pp. 59–62.

Chen, S., Stankovic, J.A., Kurose, J.F., Towsley, D.F., 1991. Performance evaluation of two new disk scheduling algorithms for real-time systems. Journal of Real-Time Systems 3, 307–336.

HPL-CSP-91-7, 1991. Disk Scheduling Algorithms Based on Rotational Position, Hewelett-Packard Company.

Ramamritham, K., 1993. Real-time databases. International Journal of Distributed and Parallel Databases 1, 199–226.

Reddy, A.L.N., Wyllie, J., 1993. Disk scheduling in multimedia I/O system. In: Proceedings of ACM Multimedia. Anaheim, CA, August 1993, pp. 225–234.

Reddy, A.L.N., Wyllie, J.C., 1993. I/O issues in multimedia system. IEEE Transactions on Computers 27, 69–74.

Ruemmler, C., Wilkes, J., 1994. An introduction to disk drive modeling. IEEE Computer 27, 17–29.

Silberschatz, A., Glavin, P.B., Gagne, G., 2001. Operating System Concepts, sixth ed. Addison-Wesley.

Specification Ver 2.0, 1999. Intelligent I/O (I^2O) Architecture Specifications, I^2O SIG™.

Thomas, S., Seshadri, S., Haritsa, J.R., 1996. Integrating standard transactions in real-time database systems. Information Systems 21, 3–28.

Ulusoy, O., 1998. Real-time data management for mobile computing. In: Proceedings of International Workshop on Issues and Applications of Database Technology (IADT'98), Berlin, Germany, July 1998.

Ulusoy, O., Belford, G.G., 1993. Real-time transaction scheduling in database systems. Information Systems 18, 559–580.

Weikum, G., Zabback, P., 1991. Tuning of stripping units in disk-array-based file systems. In: Interoperability in Multidatabase Systems (IMS) '91. Proceedings. pp. 280–287.

Yu, P.S., Chen, M.S., Kandlur, D.D., 1992. Design and analysis of a grouped sweeping scheme for multimedia data, In: 3rd International Workshop on Network and Operating Systems Support for Digital Audio and Video. San Diego, November 1992, pp. 38–49.