A Real-Time Disk Scheduler for Multimedia Integrated Server Considering the Disk Internal Scheduler

Kyung Ho Kim, Joo Young Hwang, Seung Ho Lim, Joon Woo Cho and Kyu Ho Park
Computer Engineering Research Lab., EECS
Korea Advanced Institute of Science and Technology
E-mail: {kyhkim, jyhwang, shlim, jwc, kpark}@core.kaist.ac.kr

Abstract

The SCSI or the fibre channel (FC) disk drive, called SCSI-FC disk drive has its own internal queue and internal scheduler which maximize its own throughput. The request service order scheduled by the block device drive in Linux is changed by the disk internal scheduler (DIS) and we call this reordering. The conventional real-time disk schedulers are based on Linux block device driver, and the deadlines of the real-time requests are not guaranteed by the reordering of the DIS. This paper proposes a real-time disk scheduling architecture on Linux which solves the reordering problem. It consists of the global scheduler and the local scheduler. The global scheduler selects the real-time requests, predicts their disk I/O times and determines the order based on the spatial locality. The local scheduler is the DIS controlled by the SCSI-FC device driver. The device driver notifies the DIS that the real-time requests with the earlier deadlines must be served before those with the later deadlines. The local scheduler feeds back the measured disk service time to the global scheduler when a requested disk service completes. We have implemented the proposed multimedia disk scheduler on Linux, and the experimental results show that the deadline misses of periodic requests can be reduced up to 19 times (94.7%) by the proposed method.

1. Introduction

Multimedia data may be represented in various forms such as text, image, video and audio data. The data types are classified into 3 classes based on their presentation characteristics as follows. Periodic service type includes all the continuous streams. The applications that use the continuous streams issue the periodic requests which should be served in the deadline. Interactive service type requires fast service in order to present a response to the user to preserve its interactive nature. The other data are included into the aperiodic service type which requires the best effort services.

It is known that an integrated file server which serves all the types of data from the same disks shows better scalability than a dedicated file server which serves the type of data from the separated disks[7]. The disk scheduler schedules the requests of the mixed type load guaranteeing the requirements of each type of data and maintaining the high disk throughput. The scheduling of the mixed load can be divided into two methods. Priority based scheduling method schedules the requests of high priority first, but the disk throughput of this method decreases for the multi-level priority[4][10][3]. The round based scheduling method reduces the number of the priority levels by synchronizing the deadlines of the periodic requests to the specific times and optimizes the disk head movements[6][2][5][1]. In [14][15], the round-based scheduling methods are implemented on Linux based system on the block device driver layer.

The modern SCSI-FC disk drive has its internal queues and internal scheduler (DIS) which schedules the queued requests by the methods optimized for the general file access such as Shortest Seek Time First(SSTF), Shortest Positioning Time First (SPTF), Shortest Total Access Time(STAT), and Aged Shortest Access Time First(ASATF) method[12][13][11]. Furthermore, the Linux device drivers for SCSI-FC host interface cards (e.g., ADAPTEC, LSILOGIC, QLOGIC and IPHASE host interface cards) support the DIS. Since the DIS reschedules the requests queued in the disk drive, the requests scheduled by the real-time disk scheduler on Linux are reordered in the DIS and this breaks the real-time guaranteed services of the periodic requests. The reordering problem forces to consider the DIS when the real-time disk scheduler is designed.

One of the methods to solve the reordering problem is
to disable the DIS, but this decreases the disk throughput as shown experimentally in Section 2. Another method is to schedule the requests by the DIS, but it has the drawback that the scheduler should be dependent on the DIS. Also, it is hard to model the disk internal scheduling method accurately.

In this paper, we present a method to solve the reordering problem based on the round-based scheduler. It guarantees the order of the periodic requests such that the earlier deadlines are served first in the disk. The proposed scheduler is composed of the global scheduler and the local scheduler. The global scheduler implemented on the block device driver layer is round-based scheduling, which selects the periodic requests. The interactive and aperiodic requests are served in a round. The local scheduler is the DIS controlled by the modified SCSI device driver. It uses the DIS to maximize the disk throughput for the queued requests and notifies the DIS that the periodic requests with earlier deadlines must be served before those of the later deadlines. A SCSI-FC message, *Ordered Tag* is used to notify the service order by SCSI device driver. The local scheduler feeds back the real disk service time to the global scheduler when a disk service of a request completes in the disk. The global scheduler adjusts the predicted I/O times and reschedules the requests.

We have implemented the real-time disk scheduler, and the experimental results show that deadline misses of periodic requests can be reduced up to 19 times (94.7%) by the proposed method.

The road map of this paper is as follows. In Section 2, we describe the SCSI block I/O operations on Linux and the request reordering by the SCSI-FC DIS. In Section 3, the reordering problem of the periodic requests is described and we propose the real-time load scheduler architecture that solves the reordering problem. The implementation and experiments are described in Section 4. And the conclusion follows in Section 5.

### 2. Behavior of SCSI Block Layer on Linux

In Linux, the requests to access the SCSI-FC disk should be managed by block device driver layer based on SCAN method and SCSI-FC device driver layer. The SCSI-FC disk drive has the internal queue and scheduler. It happens that the order of the requests scheduled by Linux block device driver can be reordered in the DIS.

Figure 1 represents the structure for SCSI-FC disk block I/O in Linux 2.2.14. A process issues a SCSI-FC block request to the block device driver layer and sleeps until the request is serviced. The block device driver enqueues the requests to the queue in block device driver layer, and calls `do_sd_request()`, which dequeues the requests in the queue of block device driver layer and enqueues to the SCSI-FC disk internal queue until the queue of block device driver layer is empty or the SCSI-FC disk internal queue is full. The disk drive schedules and serves the requests in the disk internal queue. After a request is served from the disk, an interrupt occurs and the interrupt handler wakes up the process and calls `do_sd_request`.

The SCSI-FC disks may adopt various methods for scheduling methods such as SSTF, SPTF, STAT, ASATF, vendor specific methods or etc. In general, the methods are optimized to maximize the disk throughput of the queued requests and the DIS can be disabled or enabled[12][13][11]. To see the DIS effect, we modified the scheduling method in the block device driver layer and the FC device driver for QLA 2100 of QLOGIC, `qlogifc.c`[8][9]. The performance of the FIFO, SCAN and SSTF scheduling methods of the block device driver when the DIS is enabled or not are measured from ST318304FC disk by experiments. For the load, 100 processes generated...
the requests to random data on disks.

Figure 2 and 3 show the response times of the requests for the same loads as the results of the various scheduling methods. The x-axis represents the requested times and the y-axis represents the response times. Figure 2 shows the response times of SCAN, SSTF and FIFO methods of the block device driver when the DIS is enabled. Figure 3 shows the response times of SCAN, SSTF and FIFO methods when the DIS is disabled. The scheduling methods with the enabled DIS show higher throughput and lower response times than those with the disabled DIS. For the enabled case, SSTF and SCAN show much higher throughput than FIFO because SSTF and SCAN enqueue the requests that have spatial locality to the disk internal queues, and SSTF shows slightly better throughput than SCAN. The experiments show that the order of the requests served in disk are different from the order scheduled by block device driver when the DIS is enabled.

3. Real-Time Disk Scheduler Architecture for Multimedia Integrated Server

The integrated file server in this paper has the admission controller for the continuous streams such as video and audio data as shown in Figure 5. Initialization requests for opening a new continuous stream have to pass the admission controller first. Only the limited number of concurrent streams can be sustained. The requests for the admitted streams proceed periodically. The period is called a round. The periodic requests issued in this round should be served within the next round. The periodic requests for the admitted streams, the interactive requests and the aperiodic requests are sent to the real-time disk scheduler. The periodic requests are issued with a round period while the other types of the requests are aperiodic. The periodic requests which should be served in a round are issued before the round starts as shown in Figure 4.

The issued requests are managed by the separated queues based on the request type. There are three separated queues, the periodic request queue, the interactive request queue and the aperiodic request queue as in Figure 5. The scheduler determines the service order of the requests for the purpose of the guaranteed deadlines of the periodic requests, the minimum latencies of the interactive requests and maximum throughput of the aperiodic requests. The reordering problem described in Section 2 forces to consider the DIS when the real-time disk scheduler is designed. This paper proposes a real-time disk scheduler architecture which solves the reordering problem. It consists of the global scheduler and the local scheduler. The global scheduler which is implemented on the block device driver layer determines the requests served in a round. It globally schedules to reduce their seek times. It schedules the periodic requests to random data on disks.

Figure 3. Performance comparisons of the SCAN, SSTF and FIFO scheduling method when the SCSI-FC DIS is disabled.

Figure 4. Request characteristics

Figure 5. Architecture of Real-Time Disk Scheduler based on Linux.
requests first to be served in the round and schedules the interactive requests and the periodic requests. To check the guarantee of the requirements, it maintains the predicted I/O times of the scheduled requests. It enqueues the scheduled request to the disk internal queue as soon as possible. The local scheduler is the DIS controlled by the modified SCSI device driver. It uses the DIS to maximize the disk throughput for the queued requests and notifies the DIS of the order of the just requests whose order should be guaranteed. A SCSI-FC messages, Ordered Tag is used to notify the service order by SCSI device driver. After a request is served in the disk, the local scheduler feeds back the measured disk service time to the global schedulers. The global scheduler which received the feed-back adjusts the predicted I/O times and reschedules the requests.

The globally scheduled request is managed in a single queue. The local scheduler consists of the SCSI-FC device driver and the disk internal queue. The SCSI-FC device driver dequeues the request from the global schedule queue and enqueues it to the disk internal queue.

### 3.1. The Global Scheduler

The global scheduler is called when a request is issued to the separated request queue or a request service in the disk is completed, i.e., when the block I/O interface is completed or the interrupt handler is completed. Since the periodic requests has the real-time characteristics, it schedules the periodic requests first. Also, it schedules the interactive requests and periodic requests. To guarantee the deadline of the periodic requests, it predicts the I/O times and stops the schedules if the I/O of the requests are predicted not to guarantee to the their requirements.

It first checks the queue of the periodic requests. If the queue is not empty and the deadline of the request in the queue is the end of the current round, it dequeues from the interactive request queue, schedules it by adopting SSTF from the global schedule queue and enqueues it to the global schedule queue. The global scheduler maintains a variable, $T_{pred}$ which is the predicted I/O times of the scheduled requests of the current round. Since the request is inserted to the global schedule queue, it updates the $T_{pred}$ by calculating the increased I/O times. If the updated $T_{pred}$ is longer than the $T_{remaining}$ which is defined as the current time to the end of the current round, the deadline of the scheduled periodic requests cannot be guaranteed. Therefore, to guarantee the deadlines the global scheduler tries to remove the periodic requests with the next round deadline, the aperiodic requests or the interactive requests from the scheduled queue. In our implementation, since the interactive requests has higher priority than the aperiodic request due to its interactive characteristic, the aperiodic requests are removed first and then the interactive requests are removed. The removed requests return to the head of their separated queues.

If the periodic requests with the deadline of the current round does not exist in the periodic request queue, the scheduler schedules the interactive requests. It dequeues from the head of the interactive request queue, schedules it by adopting SSTF from the global schedule queue and enqueues it to the global schedule queue. It updates $T_{pred}$ and checks the deadline guarantee by comparing $T_{pred}$ and $T_{remaining}$. If $T_{pred}$ is longer than $T_{remaining}$, it tries to remove the aperiodic requests to decrease $T_{pred}$.

After it schedules the requests of the interactive request queue, it tries to schedule the aperiodic requests. If selects a request from the aperiodic request queue and predicts $T_{pred}$. If $T_{pred}$ is longer than $T_{remaining}$, it cancels the schedules. Else, it dequeues from the head of the interactive request queue, schedules it by adopting SSTF from the global schedule queue and enqueues it to the global schedule queue. Finally, it checks the periodic requests with the deadline of the next round and tries to schedule them as $T_{pred}$ is less than $T_{remaining}$.

The predicted I/O time and the real disk service time of a request is different. The global scheduler reflects the real disk service times from the SCSI-FC block device driver to accurately model the predicted I/O times. If the global scheduler is called by the complete of a request service, the disk service time of the request returns to the global scheduler. The global scheduler compares the predicted I/O time of the request and the real disk service time and adjusts the $T_{pred}$ to reschedule the requests.

Algorithm 1 describes the accurate procedure of the global schedulers, where $GSQ$, $PRQ$, $IRQ$, and $ARQ$ are Global Schedule Queue, Periodic Request Queue, Interactive Request queue and Aperiodic Request Queue, respectively. At first it checks the complete of a request service and adjusts the $T_{pred}$. Then it starts to schedule the periodic requests, the interactive requests and the periodic requests until the $T_{pred}$ is not longer than $T_{remaining}$.

### 3.2. The Local Scheduler

The requests scheduled by the global scheduler are sent to the disk drive via the SCSI-FC device driver. The local scheduler is the DIS controlled by the SCSI-FC device driver. It schedules and serves the most requests in the disk internal queue by the disk’s own scheduling method, but it makes the order of the periodic requests be guaranteed. And it measures the service times of the requests and feeds back to the global scheduler.

The reordering by the DIS makes the deadline of the periodic request not be guaranteed. Suppose two periodic requests whose one has faster deadline than the other. The global scheduler enqueues the request with faster deadline
Algorithm 1  The Global Scheduler

$T_{remaining}$ is the time from now to the end of the current round.

$T_{pred}$ is the predicted I/O times of the scheduled requests.

if a request service in the disk is completed then
    Update $T_{pred}$ with the feed-back.
end if

while $T_{remaining} > T_{pred}$ do
    Find a periodic request with the current round of deadline, $p_w$, from PRQ.
    if $p_w$ exists then
        Schedule and enqueue $p_w$ to GSQ and update $T_{pred}$.
        while $T_{remaining} < T_{pred}$ do
            Find a periodic request with the next round of the deadline, $p$, from GSQ.
            Find an aperiodic request, $a$, from GSQ.
            if $p$ exists then
                Remove $p$ from GSQ to PRQ and update $T_{pred}$.
            else if $a$ exists then
                Remove $a$ from GSQ to ARQ and update $T_{pred}$.
            else if $i$ exists then
                Remove $i$ from GSQ to IRQ and update $T_{pred}$.
            end if
        end while
    else
        Find an interactive request, $i_w$ from IRQ.
        if $i_w$ exists then
            Schedule and enqueue $i_w$ to GSQ and update $T_{pred}$.
            while $T_{remaining} < T_{pred}$ do
                Find a periodic request with the next round of the deadline, $p$, from GSQ.
                Find an aperiodic request, $a$, from GSQ.
                if $p$ exists then
                    Remove $p$ from GSQ to PRQ and update $T_{pred}$.
                else if $a$ exists then
                    Remove $a$ from GSQ to ARQ and update $T_{pred}$.
                end if
            end while
        else
            Find an aperiodic request, $a_w$ from ARQ.
            if $a_w$ exists then
                Schedule and enqueue $a_w$ to GSQ and update $T_{pred}$.
                while $T_{remaining} < T_{pred}$ do
                    Find a periodic request with the next round of the deadline, $p$, from GSQ.
                    if $p$ exists then
                        Remove $p$ from GSQ to PRQ and update $T_{pred}$.
                    end if
                end while
            else
                Find a periodic request with the next round of deadline from PRQ.
                If it exists, schedule and enqueue it. Update $T_{pred}$.
            end if
        end if
    end if
end while

Figure 6. Conventional Method : Over time from the deadline of the periodic requests of the conventional real-time scheduler that does not use Ordered Tag Control

to the disk earlier than the request with later deadline. However, the request with later deadline is served earlier than that of the faster deadline if the former is more optimal than the latter by the throughput oriented disk internal scheduling method, which is not predicted by the global scheduler. If the latter lives on the disk queue lasting the deadline, the real-time characteristics are not guaranteed although the re-ordering gives better disk throughput. To prevent the re-ordering problem of the periodic requests, the SCSI-FC device driver in our implementation notifies the DIS of the order of the periodic requests. The implementation for the notification is used by Ordered Tag SCSI-FC standard message[11][13][12]. If the request is queued in the disk with Ordered Tag, the disk drive executes the requests in the order received with respect to other requests queued with Ordered Tag. All other requests queued prior to the request with an request with Ordered Tag are scheduled and served before that request with Ordered Tag is served. And all the requests queued after a request with Ordered Tag, are scheduled and served after that request with Ordered Tag is served. The SCSI-FC device driver of the local scheduler enqueues the last periodic requests, which is scheduled by the global scheduler in a round, to the disk with Ordered Tag. Then all the requests before the ordered tagged request are served before the latter periodic requests.

4. Implementation and Experiments

To implement the proposed scheduler that solves the re-ordering problem of the DIS, we modified the Linux 2.2.14. The struct request is extended to distinguish the request types and to add the deadline field. The conventional sin-
Figure 7. Proposed Method : Over time from the deadline of the periodic requests of the proposed real-time scheduler that uses Ordered Tag Control

gle queue in Linux is extended to 3 independent queues, the periodic request queue, the interactive request queue and the aperiodic request queue. For the global scheduler, we modified the conventional SCAN scheduler part of the add_request(). The SCSI-FC host adapter card is QLA 2100 of QLOGIC[9]. For the local scheduler, we modified the isp2x00_queuecommand in qlogicfc.c files[8]. The disk is ST318304FC disk of Seagate. We set the round as 1 second. The periodic application uses the continuous stream which has about 0.5Mbits/s display rates. For the load of the interactive requests and the aperiodic requests, the applications generate the requests with Poisson distribution.

The first experiment is to see the reordering problem and to see the results by the proposed method. We fixed the number of the concurrent streams to 20, the inter arrival rate of the interactive requests to 0.1 and the inter arrival rate of the aperiodic requests to 0.01. Figure 6 and 7 show the periodic requests that does not satisfy the deadline. The x-axis represents the requested time of the periodic requests and the y-axis represents the over times from the deadlines of the periodic requests. Figure 6 is for the results of the conventional methods that does not use the Ordered Tag control and Figure 7 is for the results of the proposed method that uses the Ordered Tag control. The conventional one shows that the rates of the periodic requests that does not guarantee the deadline is about 4.1% while the proposed one shows about 0.22%. Figure 8 represents the rates of the periodic requests that does not guarantee the deadlines with respect to the number of the concurrent streams. As the load of the periodic requests is light, the rate is small. As the load of the periodic requests increase, the rates of the conventional method increase rapidly, but the rates of the proposed method is within 0.4%. The maximum improvements by the proposed method is about 19 times.

The second experiment is to see the throughput effected by the proposed methods. We measured the disk throughput of the aperiodic requests of 0.1, 0.01 and 0.001 of the inter arrival rates. The number of the continuous streams is 20 and the inter arrival rates of the interactive requests is 0.1. Figure 9 shows the measured disk throughput. For the proposed method, The disk throughput of the periodic requests saturates at the inter arrival rates of the periodic requests of 0.01. But the throughput of the conventional method slightly increases. The reason is that the reordering makes the aperiodic requests be served instead of the periodic requests. The final experiment shows the effect of the separated queue and the prioritized schedule of the interactive requests. Figure 10 represents the average response time of
the interactive requests with respect to the load of the aperiodic requests. We fixed the number of the concurrent streams to 10 and the inter arrival rate of the inter arrival requests to 0.01. By the separated queue and the prioritized schedule of the interactive requests, the average response times are not increases as the load of the aperiodic requests increases.

5. Conclusions

We have designed a multimedia data disk scheduler resided in the SCSI-FC device driver considering the reordering problem due to the DIS. It consists of the global scheduler and the local scheduler. The global scheduler selects the periodic requests, the interactive requests and the aperiodic requests in turn and determines the service order by their special locality. The local scheduler is the DIS controlled by the SCSI-FC device driver. The device driver notifies the DIS that the periodic requests with the earlier deadlines must be served before those with the later ones. The notification message is Ordered Tag provided by the SCSI-FC disk itself. The local scheduler measures the real disk service time when a request service in the disk completes and feeds it back to the global scheduler. The global scheduler reschedules the requests accurately.

We have implemented the real-time disk scheduler, and the experimental results show that deadline misses of periodic requests can be reduced up to 19 times (94.7%) by the proposed method.

Acknowledgement

This work has been supported by National Research Laboratory Project of the Korea Ministry of Science and Technology.

References