G-SCAN: A Novel Real-Time Disk Scheduling Using Grouping and Branch-and-Bound Strategy

Taeseok Kim[1], Eungkyu Song[1], Yunji Koh[1], Youjip Won[2,*], and Kern Koh[1]

[1] School of Computer Science and Engineering, Seoul National University,
56-1, Shillim-Dong, Kwanak-Ku, Seoul, 151-742, Korea
{tskim, eungkyu, erdbeere2, kernkoh}@oslab.snu.ac.kr
[2] Division of Electrical and Computer Engineering, Hanyang University,
17, Hangdang-Dong, Seongdong-Ku, Seoul, 133-791, Korea
yjwon@ece.hanyang.ac.kr

Abstract. For mixed-media workload, disk scheduling strategies have to guarantee the QoS (Quality of Service) of requests with timing constraints while optimizing the disk utilization. In this paper, we present a novel real-time disk scheduling algorithm which optimizes the seek time overhead of all requests while meeting the deadlines of requests with timing constraints. Our algorithm first arranges the requests in the queue by SCAN order and clusters the several adjacent requests into group. And then it finds a feasible schedule which meets the different QoS of each requests using branch-and-bound strategy. Through trace-driven simulation, we show that our algorithm outperforms other algorithms in terms of response time, throughput, and QoS guarantee for real-time requests.

1 Introduction

As multimedia streaming services with timing constraints are rapidly becoming prevalent, the traditional time-sharing operating systems are required to be redesigned for these services. In the data storage and retrieval domain, the next generation file systems should deal with the mixed requests with different QoS. For example, it should efficiently handle the real-time requests such as audio/video playback and in the mean time should be able to handle the best-effort requests which only require the delivery or storage of data. Disk scheduling activity for this mixed workload environment is also a challenging problem.

During the last years, there have been many studies on handling mixed-media workload. Based on traditional period-based strategy, Won et al.[1] and Shenoy et al.[2], etc. proposed mechanisms which allocate a certain fraction of each period to each class of requests. Specially, Shenoy et al. presented the Cello disk scheduling framework[2] using two-level disk scheduling architecture: a class-independent

* Work of this author is supported by grant No. R08-2003-000-11104-0 from the Basic Research Program of the Korea Science & Engineering Foundation.

M. Gavrilova et al. (Eds.): ICCSA 2006, LNCS 3980, pp. 1062 – 1071, 2006.

scheduler and a set of class-specific schedulers. There also have been many approaches that try to integrate EDF (Earliest Deadline Firtst)[3] which focuses on deadline with SCAN which minimizes the seek overhead. SCAN-EDF[4], Mokbel et al.[5], Kamel et al.[6], and SSEDO/SSEDV (Shortest Seek and Earliest Deadline by Ordering/Value)[7] are such examples.

In essence, disk scheduling problem is to find an optimal schedule of requests among all the possible schedules. For example, when there are *N* requests in request queue, the number of all possible combinations of *N* requests is *N factorial*. Unfortunately, finding an optimal schedule from this huge solution space is infeasible because of excessive spatial and temporal overhead. Huang et al.[8] presented a new approach called MS-EDF (Minimizing Seek time Earliest Deadline First) which effectively reduces the huge space to a feasible extent for multimedia server system through branch-and-bound strategy. Though they have showed the superior performance of their work, MS-EDF has some limitations for mixed-media workload environment: it handles requests in a batch manner, it considers only real-time requests, etc.

In this paper, we present a novel real-time disk scheduling algorithm called G-SCAN (Grouping-SCAN) for mixed-media workload environment. Our scheduling algorithm resolves the aforementioned problems of MS-EDF by employing on-line mechanism and several techniques. G-SCAN first arranges the requests in the queue by SCAN order and then clusters the adjacent requests without timing constraints into group. Next, G-SCAN reduces the huge solution space to a reasonable extent using several heuristics and brand-and-bound strategy. We demonstrate that G-SCAN is suitable for mixed-media workload since: (i)it is based on on-line request handling mechanism, (ii)it meets the deadline of real-time requests, (iii)it minimizes the seek time costs, and (iv)it has low enough overhead to be implemented.

The remainder of this paper is organized as follows. First, we present the problem definition and assumptions in section 2. And then we illustrate our G-SCAN algorithm in section 3. In section 4, we validate our algorithm through extensive simulation. Finally, we make concluding remarks in section 5.

2 Problem Definition and Assumptions

Our goal is to design a disk scheduling algorithm which is able to meet the deadline of requests with timing constraints and minimize the disk seek time overhead as much as possible. In addition to that, our algorithm should be feasible to be implemented, i.e. it should have justifiable assumptions and reasonable algorithm cost.

To this end, we first classify all requests into two classes: real-time requests and best-effort requests. Real-time requests have their own deadline value and they could be periodic or aperiodic. On the other hand, best-effort requests have no specific deadline, they could be regarded that they have infinite deadline value. We also assume that all requests are independent, i.e., a request does not synchronize or communicate with other requests and all requests are nonpreemptive, i.e. a request could not be preempted by other requests during it is serviced in disk.

3 G-SCAN: An On-Line Real-Time Disk Scheduling Algorithm for Mixed Media Workload Environment

3.1 Motivation

Optimal disk scheduling algorithm is to find a schedule of requests which has optimal cost. For example, given that *N* requests in the queue, optimal algorithm should search an optimal request schedule among *N factorial* possible schedules. Searching in this huge solution space is very intricate and it is known as an NP hard problem[9]. In addition to that, in mixed-media workload environment, scheduling algorithm should be able to satisfy the QoS requirements of each request class. For example, soft real-time applications such as audio/video playback require meeting the deadline of each I/O request, while best-effort applications such as database search, file copy, text editing, etc. require low response time or high throughput.

To solve this problem, we employ the branch-and-bound strategy. The branch-and-bound strategy is one of the most efficient solutions to solve the combinatorial problem. In essential, there are a few optimal solutions in searching space, the branch-and-bound strategy tries to find an optimal solution by cutting down the unnecessary space. We cut down the huge solution space using the abovementioned requirements in mixed-media workload environment. For example, we can remove a schedule which has any deadline missed request or a schedule which has too high total seek costs from all possible schedules. Fig. 1 illustrates an example. In this example, we assume that schedule S_a has request R_3 which could not meet its own deadline and S_b has too high total seek costs to be optimal requests schedule. Hence, these schedules and the descendants of those which include S_a or S_b could be removed from the solution space. Level in fig. 1 denotes the number of request in the queue. When level is 2, solution space is 2(= 2 *factorial*), when level is 3, solution space is 6(= 3 *factorial*), and so on.

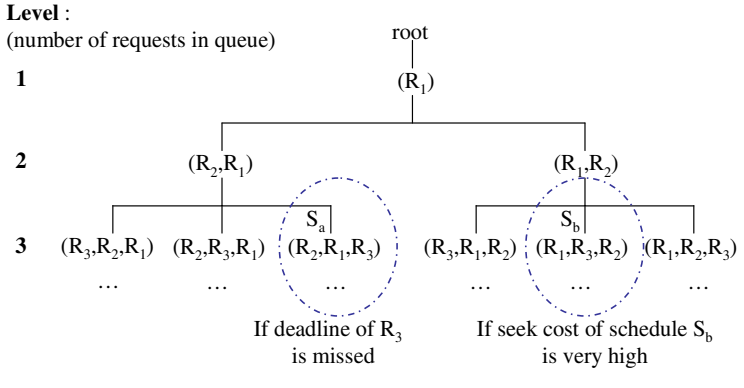


Fig. 1. An example for space reduction. In this example, we assume that R_3 in S_a could not meet its deadline and S_b has very high seek cost.

3.2 Details of G-SCAN (Grouping-SCAN)

In this section, we illustrate the details of our on-line real-time disk scheduling algorithm called G-SCAN. On-line here means that it not only has low overhead, but also should be able to decide a schedule whenever requests arrive at queue or requests leaves from queue. Basically, our algorithm expands the existing schedules whenever the event such as the arrival or the departure of requests is occurred. And then, it cuts down the solution space with criteria which require meeting the deadline of real-time requests, minimizing the seek costs, and so on. Let N and M be the number of requests in the queue and the possible schedules. M is bounded by N *factorial*, obviously. However, we can reduce this huge solution space to a reasonable extent using two basic strategies: the grouping of adjacent best-effort requests and the branch-and-bound scheme by several heuristics.

To reduce the solution space, we first arrange the requests in the queue by SCAN order. Note that the arrangement by SCAN order is just preliminary procedure for grouping. And then we try to cluster the adjacent best-effort requests one another into group. Since best-effort requests have no deadline, it is justifiable to schedule the best-effort requests in a group by SCAN order. This grouping reduces the huge solution space significantly by removing the unnecessary combination of adjacent best-effort requests.

Fig.2 illustrates an example for grouping of best-effort requests. We assume that there are 11 requests in the queue by SCAN order, and thus the solution space is 11 *factorial*(fig.2(a)). Fig.2(b) shows the state after grouping the adjacent best-effort requests. G-SCAN basically clusters all the best-effort requests between two real-time requests into a group. However, if the seek distance between any two best-effort requests is too long, they are not put together into single group. It is because any group which has too long seek time may decrease the possibility of finding the best schedule. In fig.2(b), there are two groups between R_4 and R_{11} because the distance between R_6 and R_7 is too long. We put any two best-effort requests which has the seek distance below threshold τ together into a group. If $\tau$is large, the number of all possible schedules is decreased and thus the solution space becomes smaller, but the possibility of finding the best schedule is also decreased. On the other hand, if $\tau$is small, the number of all possible schedules is increased and thus the solution space becomes larger, but the possibility of finding the best schedule is also increased.

Note that there are two requests order within each group: ascending sector number order and descending sector number order. Hence, in fig.2(b), the solution space after grouping is (8*6 *factorial*) instead of 6 *factorial*. Let N_r, N_b, and N_g be the number of real-time requests, the number of best-effort requests, and the number of groups, respectively. The number of schedules before grouping of adjacent best-effort requests is ((N_r + N_b) *factorial*), while the number of schedules after grouping becomes ((N_r + N_g) *factorial*$*2^{N_g}$). Usually, N_g is much smaller than N_b.

When new request arrives at queue, G-SCAN tries grouping by abovementioned method. If the newly arrived request is best-effort one, it may be merged with the existing group, it may bridge a gap between two groups, or it will create the new group. On the other hand, if the new request is real-time one, it may split the existing group or it is inserted by SCAN order without special action.

Next, with the real-time requests and the grouped best-effort requests, G-SCAN tries to reduce the solution space using the branch-and-bound strategy. To this end, we employ the following heuristics. First two heuristics are adopted in order during new request arrival phase while last heuristic is adopted during request service phase. Since the solution space for scheduling is still huge, it should be reduced as much as possible using following heuristics.

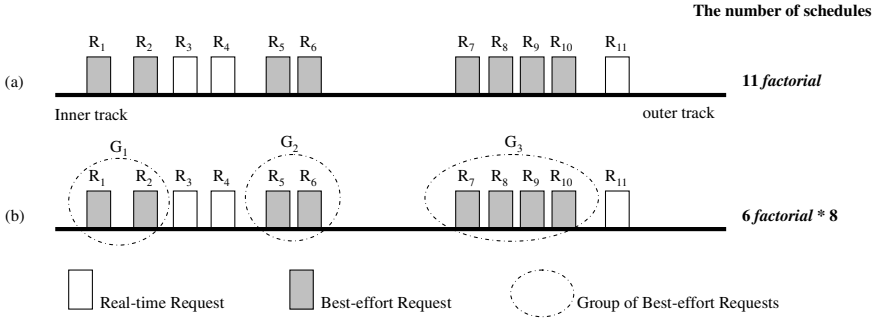


Fig. 2. An example for grouping of best-effort requests located closely each other. In (a), the number of all possible combinations of 11 requests is (11 *factorial*). On the other hand, in (b), the number of all possible combinations is (6 *factorial* * 8).

Heuristic 1. Removal of schedule which has any deadline missed request: A schedule with any request which cannot meet its own deadline could be removed.

Since one of our objectives is to meet the deadlines of real-time requests, this heuristic is trivial. If a schedule has any requests which cannot meet the deadline, the new schedule inherited from that schedule by the arrival of new request will also have any requests which cannot meet the deadline. Assume that there is a schedule with the request order (…, R_i, …), where $1 \leq i \leq N$, cannot meet the deadline of R_i. When a new request R_k arrives at queue, our scheduler will try to expand the existing schedules by inserting R_k. Since R_i cannot meet the deadline, for the expanded schedules (…, R_k, …, R_i , …) and (…, R_i , …, R_k, …), R_i still misses the deadline. The number of schedules removed by this heuristic is denoted by K_d.

Heuristic 2. Removal of schedules which have excessive total seek time costs: If the seek cost difference between any schedule $S_i(N)$ and optimal schedule at level N is greater than the disk full sweep cost, $S_i(N)$ could be removed.

Supposed that $S_{opt}(N)$ is the optimal schedule when there are N requests in the queue. And let $C_{opt}(N)$ and C_{sweep} be the seek time cost for $S_{opt}(N)$, the cost for full sweep of disk head, respectively. This heuristic removes a schedule $S_i(N)$ satisfying following inequality.

$$C_i(N) - C_{opt}(N) > C_{sweep} \tag{1}$$

In (1), $C_i(N)$ is the seek time cost for schedule $S_i(N)$. Supposed that optimal schedule after arrival of new request is $S_{opt}(N+1)$ and the seek cost for $S_{opt}(N+1)$ is denoted by $C_{opt}(N+1)$. And then the new schedule $S_i(N+1)$ inherited from $S_i(N)$ which satisfies

inequality (1) cannot have the cost less than that of $S_{opt}(N+1)$. Hence, $S_i(N)$ satisfying (1) could be removed. The number of schedules removed by this heuristic is denoted by K_s.

It is possible that all schedules are removed by above heuristics. For example, when the I/O subsystem is overloaded, all possible schedules may not satisfy the above heuristics. To resolve this phenomenon, even if all possible schedules cannot satisfy the heuristics, G-SCAN does not remove all of them. In other words, G-SCAN leaves one schedule which has minimal seek overhead and this guarantees the completeness of our algorithm.

Heuristic 3. Removal of unselected schedules: When the first request R_i in optimal schedule leaves from queue to be serviced, a schedule which does not begin with R_i could be removed.

This heuristic is trivial. When disk is ready for servicing a request, G-SCAN chooses an optimal schedule at that level. Once optimal schedule is chosen, the first request R_i of that schedule is removed from queue and all schedules which do not begin with R_i are also removed.

Note that our algorithm might not give an optimal schedule in the true sense of the definition. Essentially, optimal algorithm requires knowledge about the request sequence that will arrive at queue in the future. Our goal is just to design an algorithm which could obtain a schedule close to optimum with reasonable cost. The outline of the proposed algorithm is as follows:

When a new request R_i arrives at the queue

1. insert the new request R_i according to the SCAN order.
2. try to group the adjacent best-effort requests.
   - if R_i is a real-time request, it may split the existing group or it is inserted by SCAN order without special action.
   - if R_i is a best-effort request, it may be merged with the existing group, it may bridge a gap between two groups, or it will create the new group.
3. cut down the unnecessary solution space with the branch-and-bound strategy.
   - create a new schedule S_i including R_i.
   - remove S_i which has any deadline missed real-time request.
   - remove S_i which has excessive total seek overhead.

When any request should leave from the queue to be serviced

1. choose the schedule which has minimal seek overhead while meeting the deadlines of all real-time requests.
2. service a request R_i at head position of S_i and remove schedules which do not begin with R_i.

Even if our algorithm reduces the solution space sufficiently, since it essentially deals with a significant amount of schedules, it should be carefully designed. Hence, we employ min-heap data structure in which the key value is the total seek time costs of each schedule.

4 Experimental Evaluation

4.1 Experimental Methodology

In this section, we present the performance analysis of our G-SCAN algorithm. Our goal in this section is to demonstrate that our algorithm outperforms other algorithm and has feasible overhead to be implemented. To this end, we compare our algorithm with other several algorithms such as C-SCAN, EDF, SCAN-EDF, and Kamel's[6] in terms of seek cost, response time, throughput, and deadline miss rate. And then we show the running cost of our algorithm is not excessive. To simulate the various workloads, we arrange four cases in the simulation. They are listed in table 1. Workload 1, workload 2, and workload 3 emulate the mixed-media workload environment, while workload 4 emulates homogeneous workload of multimedia request type. Note that these workloads are somewhat heavy. The reason that we used the heavy workloads is to show that the overhead of our G-SCAN is not very high in heavy load environment.

4.2 Performance Evaluation

Before investigating the performance of our algorithm, we first analyze the effect of grouping. Fig. 3 illustrates the average number of groups after grouping with the average number of requests in the queue as a function of threshold τ. Note that the number of groups illustrated in Fig. 3 includes the real-time requests as well as the grouped best-effort requests. As can be seen, the solution space, i.e. all possible combinations of requests is significantly reduced after grouping. When τ is 100 tracks, the number of requests in the queue is about 16 and thus the size of solution space is 16 *factorial*. After grouping of best-effort requests, the number of groups is about 8, and thus the solution space becomes 8 *factorial* $*2^N$. In the worst case, N is 8. As τ is larger, the number of groups becomes smaller.

Table 1. The characteristics of workloads used

Work load	Type of tasks	# of tasks	Bandwidth (if periodic task) / Inter-arrival time (if random/sequential task)	IO size (KB)	File size (MB)
1	periodic	1	12 Mbps	64	100
	random	1	20 ms	4-128	200
2	periodic	1	12 Mbps	64	100
	sequential	1	20 ms	64	200
3	periodic	1	10 Mbps	64	100
	random	1	45 ms	4-128	100
	sequential	1	20 ms	64	100
4	periodic	6	4-5 Mbps	64	50-200

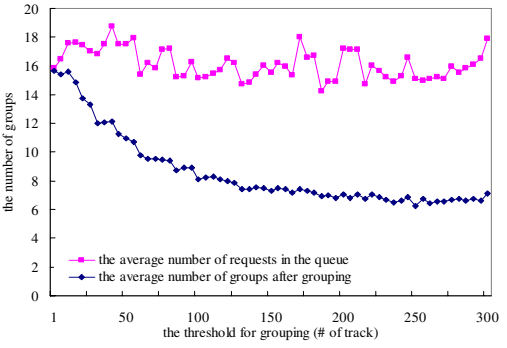
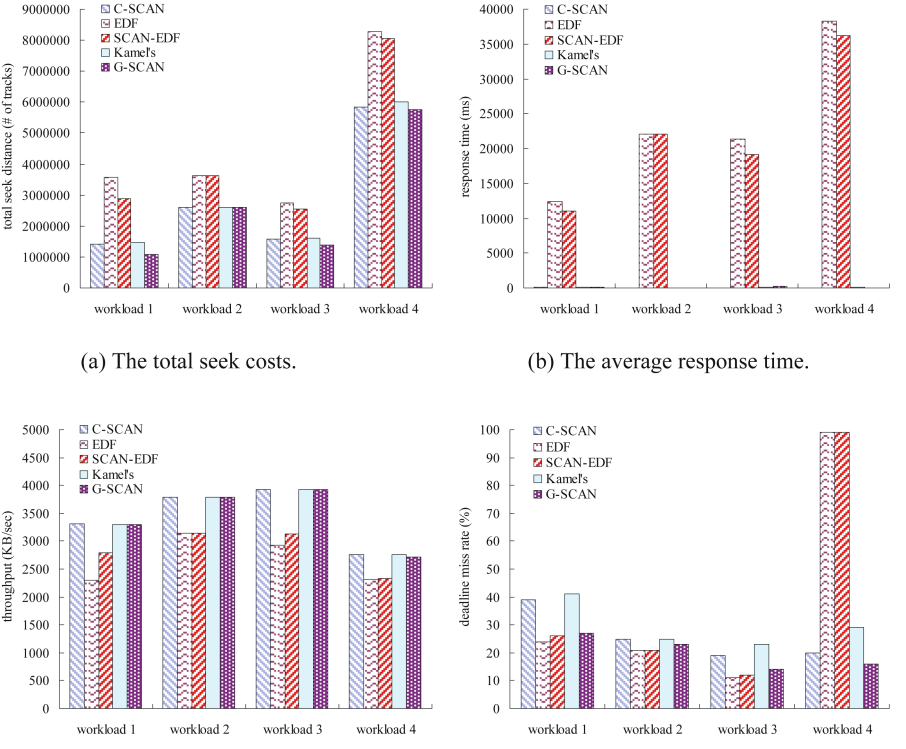


Fig. 3. The number of groups after grouping of adjacent best-effort requests

Next, we present the performance results for various workloads to assess the effectiveness of our scheme. Note that the performance of G-SCAN is measured when τ is 20. First, we investigated the disk head movement behavior of proposed algorithms. As shown in fig. 4(a), C-SCAN, Kamel's, and G-SCAN exhibit relatively low seek overhead. As illustrated, our G-SCAN is a little better than C-SCAN in experiments with workload 1, 3, and 4. It is because C-SCAN handles the requests in one direction and thus it includes unnecessary disk full sweep. Fig. 4(b) and fig. 4(c) show the average response time and the throughput of different algorithms, respectively. For both response time and throughput metrics, our algorithm exhibits superior performance with C-SCAN and Kamel's, too. EDF and SCAN-EDF show the excessive response time in all cases. It is because workloads used in our experiments are heavy. Fig. 4(d) compares different algorithms in terms of deadline miss rate. As expected, EDF and SCAN-EDF shows very low deadline miss rate in experiments with workload 1, 2, and 3. However, we observed that deadline-based scheduling algorithms such as EDF, SCAN-EDF are much worse than C-SCAN in case the workload consists of homogeneous real-time requests (workload 4). All the requests in workload 4 have their own deadlines and thus the deadline-based algorithms which focus only on the deadline value of each request cause too many seek overhead. As can be seen from the fig. 4(d), our G-SCAN algorithm shows relatively low deadline miss rate. When summarizing four performance results, our G-SCAN meets the deadlines of real-time requests as EDF and exhibits so good performance as C-SCAN in terms of disk head movement optimization.

Finally, we show that the overhead of G-SCAN is not excessive. To this end, we measured the number of schedules actually expanded in our experiments. In fig. 5, we compare the total number of schedules in the worst case with the number of schedules actually expanded in experiment with workload 1. Note that the Y axis in fig. 5 is plotted on a log scale. The results show that our G-SCAN generates only a fraction of schedules to obtain the best schedule. In experiment with workload 1, the average number of schedules actually expanded is 1005. And the average of K_d and K_s in abovementioned heuristics are 469, 414, respectively. The overhead in experiments with the other three workloads were negligible. The average number of schedules actually expanded with workload 2, workload 3, and workload 4 are 2, 50, and 4, respectively.



(a) The total seek costs.

(b) The average response time.

(c) The throughput.

(d) The deadline miss rate.

Fig. 4. Performance comparison of C-SCAN, EDF, SCAN-EDF, Kamel's, and G-SCAN

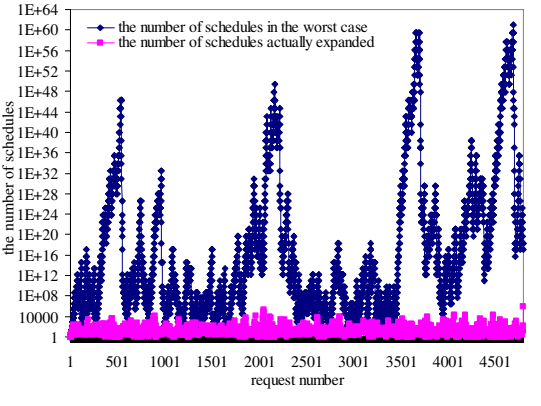


Fig. 5. The number of schedules generated in G-SCAN

5 Conclusions

In this paper, we propose a novel real-time disk scheduling algorithm in support of requests with different QoS requirements. We first recognized the optimal scheduling problem in mixed-media workload as an NP hard problem which finds an optimal solution from very huge space. And then we significantly reduce the huge space to a feasible level by grouping technique and the branch-and-bound strategy. In addition to that, we illustrated that our algorithm is based on on-line method. Through extensive simulation, we demonstrated that our algorithm outperforms other scheduling algorithms in terms of throughput, response time, and deadline miss rate. We also demonstrated that our algorithm has reasonable overhead to be implemented. We anticipate that our algorithm could be equipped with web server with audio/video objects, multimedia servers, ordinary workstation, and home appliances such as set-top box, PVR(Personalized Video Recorder).

References

1. Y. J. Won and Y. S. Ryu: Handling sporadic tasks in multimedia file system. Proc. 8th ACM International Conf. on Multimedia (2000) 462-464.
2. P. Shenoy: Cello: a disk scheduling framework for next generation operating system. Real Time Systems Journal (2002)
3. Liu, C., L, Layland, J., W: Scheduling algorithms for multiprogramming in hard-real-time environment. Journal of the ACM, Vol. 20(1) (1973) 47-61
4. Narasimha Reddy, A., Wyllie, J.: Disk scheduling in a multimedia I/O system. Proc. 1st ACM MM'93, Anaheim, CA, USA (1993) 225-233
5. M.F. Mokbel, W.G. Aref, K. El-Bassyouni, I. Kamel: Scalable Multimedia Disk Scheduling. Proc. 20th IEEE Intl. Conf. on Data Eng. (ICDE), Boston (2004) 498-509
6. I. Kamel, T. Niranjan, and S. Ghandeharizedah: A Novel Deadline Driven Disk Scheduling Algorithm for Multi-Priority Multimedia Objects. Proc. 16th Int'l Conf. Data Eng. (2000) 349-361
7. Chen, S., Stankovic, J., A, Lurose, J.,F, Towsley, D.: Performance evaluation of two new disk scheduling algorithms for real-time systems. Journal of Real-Time Systems, Vol.3 (1991) 307-336
8. Huang Yin-Fu and Jiing-Maw Huang: Disk Scheduling on Multimedia Storage Servers. IEEE Transactions on Computers, Vol. 53(1) (2004) 77-82
9. M.Andrews, M.A.Bender, and L.Zhang: New algorithms for the disk scheduling problem. 37th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press (1996) 550-559