A HEURISTIC-BASED REAL-TIME DISK SCHEDULING ALGORITHM FOR MIXED-MEDIA WORKLOAD

Taeseok Kim, Junseok Park, Seongyong Ahn, and Kern Koh
School of Computer Science and Engineering, Seoul National University
San 56-1, Shillim-Dong, Kwanak-Ku, Seoul, Korea
{tskim,redo,syahn,kernkoh}@oslab.snu.ac.kr

Youjip Won
Division of Electrical and Computer Engineering, Hanyang University
17, Hangdang-Dong, Seongdong-Ku, Seoul, Korea
yjwon@ece.hanyang.ac.kr

Hyokyung Bahn
Department of Computer Science and Engineering, Ewha Womans University
11-1, Daehyun-Dong, Seodaemun-Ku, Seoul, Korea
bahn@ewha.ac.kr

ABSTRACT
For mixed-media workload, disk scheduling strategies should meet the deadlines of requests with timing constraints while optimizing the disk utilization. To design a scheduling algorithm in mixed-media workload environment is very intricate because finding an optimal schedule is NP-hard as well as different QoS (Quality of Service) of each request should be guaranteed. In this paper, we present a novel real-time disk scheduling algorithm which meets the QoS of each requests. Our algorithm employs several heuristics using the requirements of each request and thus could reduce the huge solution space to a feasible extent. 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.

KEY WORDS
disk scheduling, real-time, heuristics, QoS

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 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], EDL[5], and Kamel et al.[6] 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.[7] 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. 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 on-line real-time disk scheduling algorithm called HB-EDF (Heuristic Based Earliest Deadline First) for mixed-media workload environment. Our scheduling algorithm resolves the aforementioned problems of MS-EDF by employing on-line mechanism and several heuristics. Our heuristics exploits the characteristics and requirements of each request. Through these heuristics, we could reduce the huge solution space to a reasonable extent and thus acquire the satisfiable request schedule which guarantees the QoS of each request class with moderate cost. We demonstrate that HB-EDF 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 the system architecture in section 2. And then we illustrate our heuristic-based algorithm in section 3. In section 4, we validate our algorithm through extensive simulation. Finally, we make concluding remarks in section 5.

2. System Model

2.1 Problem Definitions and Assumptions

Our goal is to design a disk scheduling algorithm which is able to meet the deadline constraints of real-time requests 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 in our model. 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 and nonpreemptive.

2.2 Conditions for Jitter-free Playback

To service a given I/O request satisfying its QoS requirements, the I/O subsystem should not be overloaded. To this end, the appropriate amount of disk bandwidth should be allocated to each request class, i.e. real-time requests, best-effort requests, etc. Shenoy et al. and Joel et al. proposed efficient frameworks which allocate the disk bandwidth to each request class in [2], [8], respectively. In this paper, we focus only on our disk scheduling algorithm called HB-EDF. Hence, we just present a hint for design of deterministic admission control module for real-time environment in this section.

For jitter-free playback of audio/video, a deterministic admission control module should admit a new real-time streaming request only if it satisfies following two conditions. The first condition is that the number of data blocks retrieved during time T should be greater than the amount of data blocks needed for playback for same period of time. This condition could be denoted by (1).

$$T \cdot r_i \; < \; n_i \cdot b \qquad (1)$$

In (1), n_i , b, and r_i are the number of data blocks read during T, size of I/O unit, and playback rate of stream i, respectively. This condition should hold for each stream, i = 1,…, m. The second condition is that it should take less than time T to retrieve the data blocks for all streams.

$$T \geq \{\sum_{i=1}^{m} \frac{n_i \cdot b}{B_{\max}}\} + O(m) \qquad (2)$$

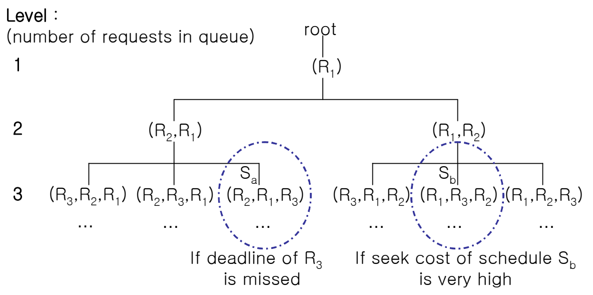
In (2), m and B_{max} are the number of streams, maximum transfer rate of the disk, respectively. And $O(m)$ is the disk movement overhead such as seek and rotation latency in reading the data blocks for m streams. When a multimedia player opens an audio/video file, admission control module is required to compute above two equations and determines whether to accept or reject the I/O requests.

3. A Real-time Disk Scheduling Scheme

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 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 instinctively an NP hard problem.

In addition to that, in mixed-media workload environment, scheduling algorithm should be able to satisfy QoS of each request class. Soft real-time applications such as audio/video playback require meeting the deadline of each I/O request, while best-effort applications such as file copy, text editing, etc. require low response time or high throughput. However, these requirements in mixed workload environment make the abovementioned scheduling problem feasible rather than intricate. Actually, owing to the requirements of each requests class, it is not necessary to search all N *factorial* schedules. For example, we can remove a schedule which has any deadline missed request or a schedule which has too high seek cost 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 seek cost 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 queue. When level is 2, solution space is 2(= 2 *factorial*), when level is 3, solution space is 6(= 3 *factorial*), and so on.



Level :
(number of requests in queue)
root
1
(R1)
2
(R2,R1)
(R1,R2)
Sa
Sb
3
(R3,R2,R1)
(R2,R3,R1)
(R2,R1,R3)
(R3,R1,R2)
(R1,R3,R2)
(R1,R2,R3)
If deadline of R3 is missed
If seek cost of schedule Sb is very high

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 HB-EDF(Heuristic Based Earliest Deadline First): An On-line Real-time Disk Scheduling Algorithm

In this section, we illustrate an on-line real-time disk scheduling algorithm called HB-EDF(Heuristics Based-Earliest Deadline First). On-line here means that it not only has low overhead to be feasible, 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's assume that there are N requests in queue and M possible schedules. Obviously, in the worst case, M is equivalent to N *factorial*. However, we can reduce this huge search space to a reasonable extent using several heuristics. Before discussing the heuristics, we first describe the states and the state transitions. Fig. 2 shows the state definition and the event for state transition when the current state is (N, M). The state transition is defined as follows. If current state is (N, M) and a new request arrives at queue, the state is changed to (($N+1$, $(N+1)*M–K$). $(N+1)$ in the first term of new state is trivial because a new request have arrived. Since a new request could be inserted into $(N+1)$ possible location for each schedule, all the possible new schedules are computed by multiplying existing schedules M by ($N+1$). In this expression, K means the number of schedules removed by several heuristics to be explained later. Hence, the second term of next state becomes $(N+1)*M–K$. On the other hand, if current state is (N, M) and a requests leaves from queue to be serviced, the state is changed to (N-1, M-D). D in the second term also means the number of schedules removed by heuristics.

Now, we explain four heuristics. First three heuristics are adopted in order during new request arrival phase while last heuristic is adopted during request service phase. Since all search space for scheduling is very huge, it should be reduced as many as possible using following heuristics.

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 goal is to meet the deadlines of real-time

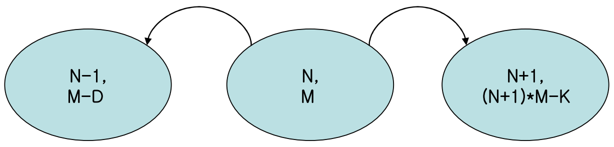


Fig. 2 The definition of state and state transition. Whenever a new request arrives at queue or a request leaves from queue to be serviced, the transition is occurred.

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 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.

$$C_i(N) - C_{opt}(N) > C_{sweep} \qquad (3)$$

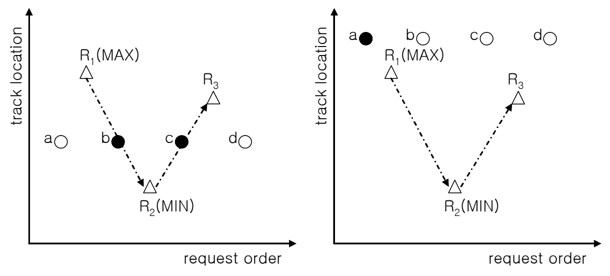
In (3), $C_i(N)$ is the seek time cost for schedule $S_i(N)$. Supposed that optimal schedule after arrival of new request at queue is $S_{opt}(N+1)$ and the seek cost for $S_{opt}(N+1)$ is denoted by $C_{opt}(N+1)$. And then the new schedule $S_i(N+1)$ inherited from $S_i(N)$ which satisfies inequality (3) cannot have the cost less than that of $S_{opt}(N+1)$. Hence, $S_i(N)$ satisfying (3) could be removed. The number of schedules removed by this heuristic is denoted by K_s.

Heuristic 3. Removal of schedules which have non-optimal cost in case of the arrival of best-effort request: When a best-effort request arrives at queue, if a new schedule including the request has non-optimal cost at that level, it could be removed.

We assume that the seek cost is exactly proportional to the seek distance. In case that a newly arrived request does not have deadline, i.e. it is a best-effort request, only the schedule which has minimal seek cost is worthy of expanding further. Let T_{max} and T_{min} be the maximum track location and the minimum track location of requests in queue, respectively. When a new request arrives, there are three track location of new request, T_{new} as follows:

(1) T_{new} is between T_{max} and T_{min},
(2) T_{new} is above T_{max},
(3) T_{new} is below T_{min}

Fig. 3 illustrates first two cases of three cases when new request, R_{new} is inserted into $S_i(R_1,R_2,R_3)$ schedule. Case (3) is not shown because it is similar to case (2). All possible schedules are $S_1(R_{new},R_1,R_2,R_3)$, $S_2(R_1,R_{new},R_2,R_3)$, $S_3(R_1,R_2,R_{new},R_3)$, $S_4(R_1,R_2,R_3,R_{new})$. In fig. 3(a), the costs for schedule S_2 and schedule S_3 are minimal among four



(a) T_{new} is between T_{max} and T_{min}. (b) T_{new} is above T_{max}.

Fig. 3 An example of inserting new request R_{new}. In (a), R_{new} is inserted in *b* or *c*. In (b), R_{new} is inserted in *a*.

schedules. It is because R_{new} is on the way from R_1 to R_2, from R_2 to R_3, respectively, and thus R_{new} could be serviced without additional seek cost. In fig. 3(b), the track location of R_{new} is not between T_{max} and T_{min}. In this example, since S_1 has minimal seek cost among four schedules, it is reasonable to insert R_{new} into location "a" in fig. 3(b). Note that the arrival of real-time request is not relevant to this heuristics. Whether a real-time request could meet its own deadline or not is largely dependent on the request schedule. Hence, it is difficult to decide an optimal insertion location of new real-time request without knowledge about the arrival pattern of requests in the future. The number of schedules removed by this heuristic is denoted by K_b.

In fig. 2, we illustrated that the next state becomes ($N+1$, $(N+1)*M–K$) when the current state is (N, M) and a new request arrives. Note that K in the second term of the next state is equivalent to the sum of K_d, K_s, and K_b. There is possibility that all schedules are removed by above three heuristics. To resolve this problem, in case there is no schedule satisfying three heuristics, our algorithm choose one schedule which has minimal seek overhead.

Heuristic 4. 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, HB-EDF 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 removed. The number of schedules removed by this heuristic is denoted by D.

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 by using these heuristics.

3.3 Design and Implementation of HB-EDF

Since our algorithm deals with a significant amount of schedules, it should be carefully designed. When new request arrives at queue, HB-EDF expands the existing schedules by inserting new request and all new schedules should be efficiently sorted by their seek costs. Hence, we employ min-heap data structure in which the key value is the total seek time costs of each schedule. The pseudo code of our algorithm is listed as follows. When a new request *R* arrives at queue and when a request leaves from queue to be serviced, HBEDF_add_request(R) and HBEDF_service_request are invoked, respectively.

The HB-EDF algorithm

```
struct schedule
{
    request_order: set of requests;
    cost: integer;
}
var $H_{old}$, $H_{new:}$: pointers to heap structure;
$H_{old}$ := creation of heap structure;

/* when a new request R arrives at queue */

procedure HBEDF_add_request(R)
var S: schedule;
    $H_{new}$ := creation of heap structure;
    if $H_{old}$ is empty then
        S := a new schedule including R;
        insert S into $H_{new}$;
    else
        while (S := root schedule from $H_{old}$)
            expand_schedule(S,R);
        end while;
    end if;
    free the allocated memory for $H_{old}$;
    $H_{old}$ := $H_{new}$;
end procedure HBEDF_add_request;

procedure expand_schedule(S,R)
var $S_{new}$, $S_i$: schedule;
var $Q_{seq}$: the list of schedules;
var validate: integer;
    validate := 0;
    for (i:=0; i$\leq$|S.request_order|;i++) do
        $S_{new}$ := creation of schedule including R from S;
        if deadline_check($S_{new}$) is TRUE then
            if optimal_check($S_{new}$)) is TRUE then
                validate := 1;
                insert $S_{new}$ into $Q_{seq}$;
            end if;
        end if;
    end for;
    while ($S_i$ := a schedule in $Q_{seq}$)
        if R is real-time request then
            insert $S_i$ into $H_{new}$;
        else
            if $S_i$ is an optimal schedule in $Q_{seq}$ then
                insert $S_i$ into $H_{new}$;
            else
                free the allocated memory for $S_i$;
            end if;
        end if;
    end while;
end procedure expand_schedule;

procedure deadline_check(S)
    executetime := 0;
    for (i:=0; i<|S.request_order|; i++) do
        $R_i$ := $i_{th}$ request in S
        executetime := executetime + servicetime of $R_i$;
        if executetime > deadline of $R_i$ then
            return FALSE;
        end if
    end for
    return TRUE;
end procedure deadline_check;

procedure optimal_check(S)
    $S_{opt}$ := the optimal schedule;
    $C_{sweep}$ := the cost of disk full sweep;
    if S.cost - $S_{opt}$.cost > $C_{sweep}$ then
        return FALSE;
    end if;
    return TRUE;
end procedure optimal_check;
```

/* when a request leaves from queue to be serviced */

```
procedure HBEDF_service_request
    $H_{new}$ := the creation of heap structure;
    $S_{ser}$ := get root from $H_{old}$ to be serviced;
    while (S := get root from $H_{old}$)
        if $1_{st}$ request of $S_{ser}$ = $1_{st}$ request of S then
            $S_{new}$ := S.request_order - $1_{st}$ request;
            insert $S_{new}$ into $H_{new}$;
        else
            free the allocated memory for S;
        end if
    end while
    free the allocated memory for $H_{old}$;
    $H_{old}$ = $H_{new}$;
    return $1_{st}$ request of $S_{ser}$;
end procedure HBEDF_service_request;
```

4. Experimental Evaluation

4.1 Experimental Methodology

In this section, we present the performance analysis of our HB-EDF 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 FIFO, C-SCAN, EDF, and SCAN-EDF 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 three cases in the simulation. They are listed in table 1. Both 1 and 2 emulate the mixed-media workload environment, while 3 emulates homogeneous workload of multimedia request type.

4.2 Performance Evaluation

Table 1. The characteristics of workloads used

work-load	access pattern	# of tasks	bandwidth (if periodic task) / inter-arrival time (if random task)	IO Size (KB)
1	periodic	1	12 Mbps	64
	random	1	20ms	4 ~ 128
2	periodic	3	5 Mbps	64
	random	1	20ms	4 ~ 128
3	periodic	7	5 Mbps	64

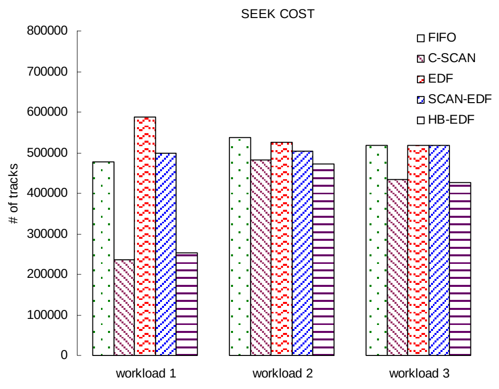


Fig. 4 The seek costs of scheduling algorithms

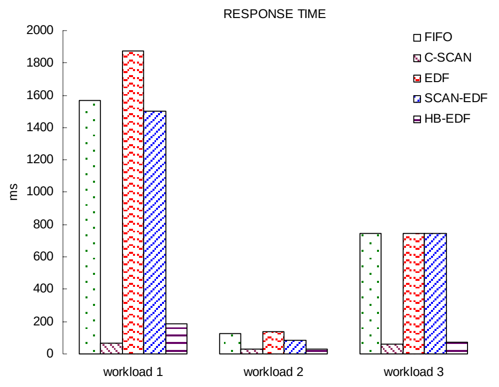


Fig. 5 The average response time of scheduling algorithms

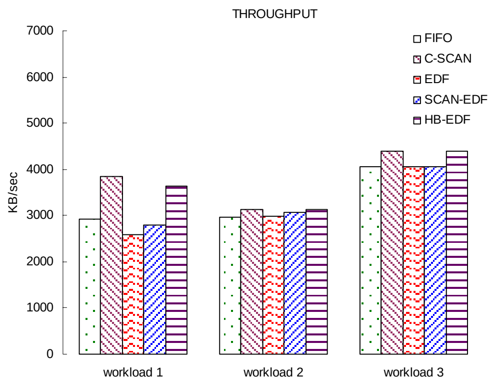


Fig. 6 The throughput of scheduling algorithms

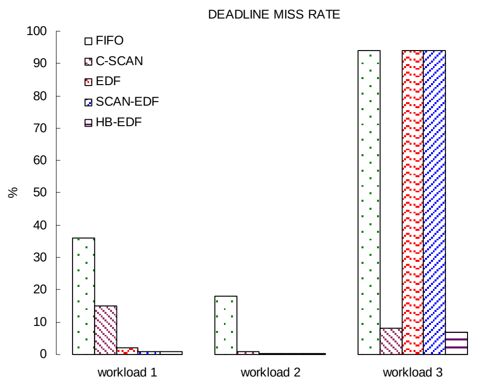


Fig. 7 The deadline miss rate of scheduling algorithms

We first investigated the disk head movement behavior of proposed algorithms. As shown in fig. 4, C-SCAN exhibits relatively low seek overhead. However, our HB-EDF algorithm also shows the performance as good as that of C-SCAN in terms of seek costs. As illustrated, HB-EDF is little better than C-SCAN in experiments with workload 2 and 3. Note that C-SCAN is not an algorithm which guarantees the deadlines of real-time requests. Fig. 5 and fig. 6 show the average response time and the throughput of different algorithms, respectively. In fig. 5, we observed that the average response times of FIFO, EDF, and SCAN-EDF in experiment with workload 1 is much larger than those in experiment with workload 2. This is because the deadlines of real-time requests in workload 1 is shorter than those in workload 2 and thus many best-effort requests experience relatively larger delay for trying to meet closer deadline of real-time requests in experiment with workload 2. For both response time and throughput metrics, our algorithm exhibits superior performance with C-SCAN, too. Fig. 7 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 and 2. 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 3). All the requests in workload 3 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. In all case, our HB-EDF algorithm exhibits the lowest deadline miss rate.

Finally, we show that the overhead of HB-EDF is not excessive. To this end, we measured the number of schedules actually expanded in our experiments. In fig. 8, 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. 8 is plotted on a log scale. The results show that our HB-EDF generates only a fraction of schedules to obtain an optimal schedule. In experiment with workload 1, the average number of schedules actually expanded is 39272. And the average of K_d, K_s, K_b in abovementioned heuristics are 15068, 5593, 17290, respectively. The overhead in experiments with the other two workloads were negligible. The average number of schedules actually expanded with workload 2 and 3 are 11 and 2, respectively.

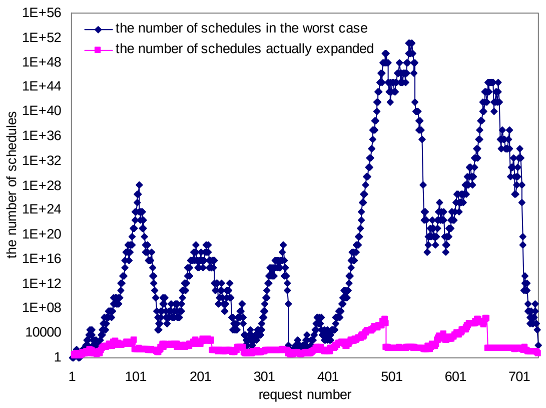


Fig. 8 The number of schedules generated in HB-EDF

5. Conclusion

In this paper, we propose a novel real-time disk scheduling algorithm in support of requests with different QoS requirements. We first recognize 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 employing several heuristics. In addition to that, we illustrate that our algorithm is based on on-line method. Through extensive simulation, we demonstrate that our algorithm outperforms other scheduling algorithms in terms of throughput, response time, while meeting the deadlines of real-time requests. We also demonstrate 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*, October 2000, 462-464.

[2] P. Shenoy, Cello: a disk scheduling framework for next generation operating system, *Real Time Systems Journal*, January 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] T. Plageman, V. Goebel, P. Halvorsen, and O. J. Anshus, Operating system support for multimedia system, *Computer communications*, 2000, 267-289.

[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] Huang Yin-Fu and Jiing-Maw Huang, Disk Scheduling on Multimedia Storage Servers, *IEEE Transactions on Computers, Vol. 53(1)*, 2004, 77-82.

[8] Joel C. Wu, Scott Banachowski, Scott A. Brandt, Hierarchical Disk Sharing for Multimedia Systems and Servers, *Proc. NOSSDAV2005*, WA, USA, 2005, 189-194.