The International Journal of Time-Critical Computing Systems, 19, 149–168 (2000)

Real-Time Disk Scheduling for Multimedia Applications with Deadline-Modification-Scan Scheme

RAY-I CHANG william@iis.sinica.edu.tw
Institute of Information Science, Academia Sinica, Taipei, Taiwan, ROC

WEI-KUAN SHIH wshih@cs.nthu.edu.tw
Department of Computer Science, National Tsing Hua University, Hsin-chu, Taiwan, ROC

RUEI-CHUAN CHANG rc@cc.nctu.edu.tw
Department of Computer & Information Science, National Chiao Tung University, Hsin-chu, Taiwan, ROC

Abstract. Real-time disk scheduling (RTDS) is important for time-critical multimedia applications. In conventional approaches of real-time disk scheduling, such as SCAN-EDF, the seek-optimizing SCAN scheme is applied to reduce the disk service time of the real-time EDF schedule. Since only tasks with the same deadline are seek-optimized, the obtained improvement of SCAN-EDF is limited. In this paper, based on the *maximum-scannable-group* (MSG) concept, a *deadline-modification-scan* (DMS or DM-SCAN) algorithm is proposed. Our algorithm uses MSG to automatically decide the suitable task groups for seek-optimizing. Based on the MSG concept, we repeatedly apply DMS to further increase disk throughput to support more tasks. We have implemented the proposed algorithm on UnixWare 2.01. The appropriate problem behaviors and parameter values to yield good solutions are investigated. Experiments show that DMS is better than conventional approaches in both the obtained disk throughput and the supported tasks. Moreover, our proposed approach can schedule task sets that are not schedulable by EDF and SCAN-EDF.

Keywords: RTDS (real-time disk scheduling), multimedia applications, SCAN, EDF (earliest-deadline-first), operating systems, maximum-scannable-group (MSG), deadline-modification-scan (DMS)

1. Introduction

Multimedia applications have become increasingly popular. To support these time-critical applications, the scheduler serves disk tasks in real-time manners (Terry and Swinehart, 1988; Anderson et al., 1992). In a real-time disk scheduler, we need to consider two problem parameters (release time r_i and deadline d_i) for each task T_i to decide the scheduled start-time e_i and fulfill-time f_i.

- *Release time*: the earliest time at which the task can be started.
- *Deadline*: the latest time at which the task must be completed.
- *Start-time*: the time at which the task is actually started.
- *Fulfill-time*: the time at which the task is actually completed.

To satisfy the real-time constraints, we must guarantee $r_i \leq e_i$ and $f_i \leq d_i$. A simple example to demonstrate the terminology in a real-time disk task is shown in Fig. 1. Notice

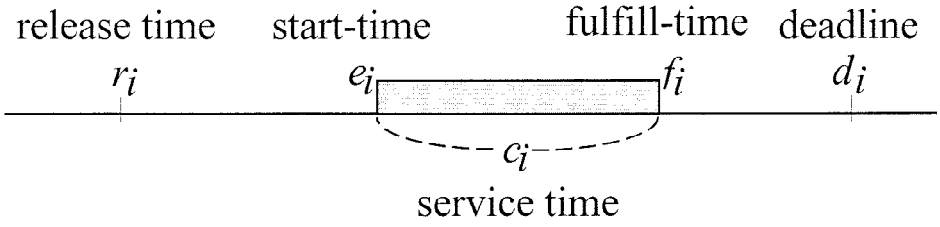


Figure 1. The terminology used for a real-time disk task.

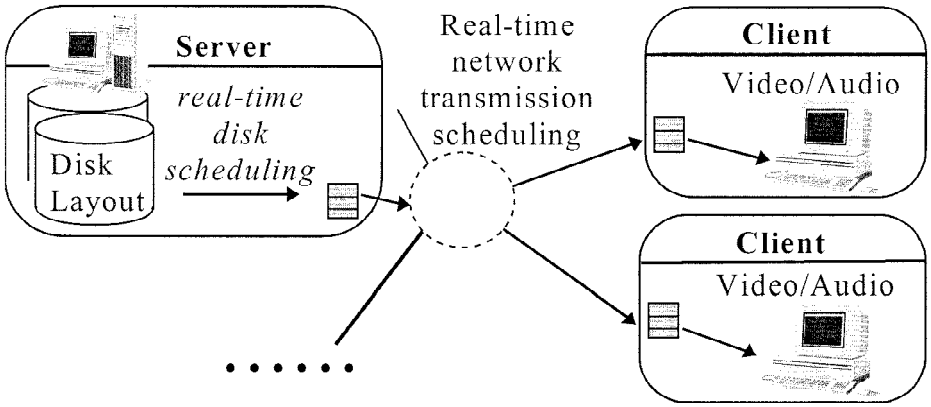


Figure 2. The basic architecture of distributed multimedia system.

that the disk task is not preemptive. Therefore, the required service time is defined as follows: $c_i = f_i - e_i$. Considering a distributed multimedia system as shown in Fig. 2, the media data are stored in disks at the server site (Vin and Rangan, 1991; Lougher and Shepherd, 1993; Wang et al., 1997). To support continuous playback, media data must be retrieved and transmitted to the client site under guaranteed timing constraints (Rangan and Vin, 1993; Gemmell and Han, 1994). In such a time-critical system, disk tasks need to be served not only with high throughput, but also with guaranteed real-time requirements (Lin and Tarng, 1991; Dan et al., 1994). It is different from the conventional system in which we focus only on the processor performance (Peterson and Silberschatz, 1985). Since a good real-time scheduler is important for disk retrieving, we propose a new RTDS (real-time disk scheduling) algorithm to support real-time disk tasks. Studies about the disk layout, real-time network transmission, and admission control problems can be found in other papers (Chang et al., 1997; Wang et al., 1997; Chang et al., 1998; Chang et al., 1999).

The volumes of media data are usually very large. The real-time disk scheduling has a pronounced effect on the system performance. For scheduling conventional disk tasks, the SCAN algorithm (as shown in Appendix 1, which scans the disk surface to retrieve the required data block under disk-head) has been proved as an optimal algorithm under

the amortized analysis and probability models (Chen et al., 1992; Chen and Yang, 1992). However, in a hard real-time system, the service of disk task will become meaningless and useless if the specified timing constraints are missed (Gemmell and Christodoulakis, 1992). Since the SCAN algorithm does not consider real-time constraints, it is not suitable for scheduling real-time disk tasks (Rangan and Vin, 1991; Gemmell et al., 1995). In the conventional real-time system (which models real-time tasks as independent tasks with related deadlines), EDF (earliest-deadline-first) is known as a good real-time scheduler (Liu and Layland, 1973; Lehoczky, 1990; Lin and Tarng, 1991). However, in the real-time disk scheduling problem, the required service time of the real-time disk task depends on the current position of the disk-head. Since the assumption of task-independence no longer holds, the employment of EDF will result in poor disk throughput (Yee and Varaiya, 1991; Reddy and Wyllie, 1994).

In past years, different RTDS methods have been proposed to combine the schemes of the seek-optimizing SCAN method and the real-time EDF scheduling method. For example, in the best-known SCAN-EDF approach (Reddy and Wyllie, 1993), the real-time disk tasks are sorted according to their deadlines by the EDF scheme. Based on the EDF schedule, the real-time requirements can be guaranteed. Since the movement cost of disk-head is not considered in scheduling, the obtained throughput would be very low. To reduce the disk service time, SCAN- EDF tries to reschedule the EDF tasks with the same deadlines by SCAN. Notice that, the efficiency of SCAN-EDF depends on the number of tasks with the same deadlines.

To resolve the drawback of SCAN-EDF, we propose a new RTDS algorithm. Our algorithm automatically selects suitable task groups for rescheduling. Given an EDF schedule as the input schedule, we first define the *maximum-scannable-group* (MSG) as a set of consequent tasks that can be successfully rescheduled by SCAN. Thus, by seek-optimizing tasks in MSG, disk throughput can be improved with guaranteed real-time requirements. Given an EDF schedule as the input schedule, our obtained disk throughput is intuitively better than that of SCAN-EDF. In this paper, we introduce a formal definition of the MSG concept. Then, an O(n) algorithm (where n is the number of input tasks) is proposed to identify MSG. The theoretical proof of the correctness of the proposed algorithm is also presented. Notice that, the initial input and its rescheduled result may not be an EDF schedule. We propose a *deadline-modification-scan* (DMS or DM-SCAN) algorithm to modify tasks' deadlines to satisfy the EDF property. In this way, the problem input would always be an EDF schedule and the proposed MSG scheme can be repeatedly applied to further improve the disk throughput.

Comparing with SCAN-EDF, our proposed DMS method can achieve over 14% more improvements in the obtained disk throughput. Nearly 10% of the improvements are obtained at the first iteration of DMS (called *DMS*-1) under the same computation complexity of SCAN-EDF. Notice that, in conventional approaches, the input task sets are feasible EDF schedules. In this paper, we also consider the input task sets that are not schedulable by EDF. The ratio of task sets that can not be scheduled by EDF but can be scheduled by our proposed approach is over 12. This paper is organized as follows. In section 2, we introduce the real-time disk scheduling (RTDS) problem. Section 3 gives mathematical definitions of the proposed MSG concept, and gives a detail description of the proposed

DMS algorithm. The experimental environments and our obtained results are shown in Section 4. Conclusive remarks and future work are presented in Section 5.

2. Problem Definition for Real-Time Disk Scheduling

2.1. Conventional Disk Service Model and Real-Time Service Model

A magnetic disk is made up of the rotating disks, the heads to access disks, and an assembly arm that moves the heads into the specified cylinder. Generally, a disk task would specify the track location of data we want to retrieve. When serving the disk task, a *seek-time* is taken to move the disk arm to the appropriate cylinder. Then, the system must wait until the desired block rotates under its read-write head by a *latency-time*. Besides, a *switch-time* is taken for switching the controller's attention from one disk-head to the other within the same cylinder. Finally, the data is transferred from disk to buffer by a *transfer-time*. The required disk service time is the summation of the related seek-time, latency-time, switch-time and transfer-time. Conventionally, the problem objective of disk scheduling is to maximize the disk throughput in serving the presented disk tasks. However, in a time-critical application, the first goal of disk scheduling is to meet the specified real-time constraints (Shih et al., 1992; Mok, 1983). Maximizing the disk throughput is also important, but the schedule is meaningless if the specified real-time constraints are not satisfied.

When considering a real-time disk task T_i, the release time r_i and the deadline d_i are presented ($r_i < d_i$, from the definitions) to compute the scheduled start-time e_i and the required fulfill-time f_i. To satisfy the real-time constraints, the start-time e_i is not earlier than the release time r_i. Besides, the computed fulfill-time f_i is not later than the given deadline d_i (Lehoczky, 1990; Jeffay et al., 1991; Stankovic and Buttazzo, 1995). Define c_i as the required service time. In conventional real-time service models, the required service time can be specified before scheduling. However, in the RTDS problem, the required service time depends not only on the data location of the present task, but also on the current location of the disk-head (the data location of the previous task). It is determined by the scheduling result and can not be specified before scheduling. In the past years, some approaches were presented to calculate the service time by the mean distance or the worst movement distance of disk-head (Steinmetz, 1995). However, the real-time requirements will not be guaranteed if the approaches with mean movement distance of disk-head are applied. Although the timing requirements are guaranteed if we apply the worst movement distance of disk-head to each task, the obtained disk throughputs are very low. A comprehensive survey for this real-time disk scheduling problem can be found in Daigle and Srosnider, 1996.

2.2. Formulating the Real-Time Disk Scheduling Problem

A real-time disk task T_i can be denoted by four parameters (r_i, d_i, a_i, b_i). The release time r_i and the deadline d_i are the timing requirements (Stankovic and Buttazzo, 1995). The track location a_i and the data capacity b_i represent the disk service requirements. Define the

problem input as a set of real-time disk tasks $\boldsymbol{T} = \{T_0, T_1, \ldots, T_n\}$ where n is the number of tasks. $T_0 = (0, 0, 0, 0)$ represents the initial condition of disk arm (at the outermost track 0). In this paper, a scheduler $\boldsymbol{Z}$ for problem input $\boldsymbol{T}$ is defined as a procedure that produces the schedule sequence $\boldsymbol{T}_Z = T_{Z(0)} T_{Z(1)} \ldots T_{Z(n)}$. The sequence of subscripts $Z(i)$, for $i = 0$ to n, is a permutation of index $\{0, 1, \ldots, n\}$. Notice that, when serving task T_i by the scheduled order $T_{i-1} T_i$, the disk-head should move from the track location a_{i-1} to the track location a_i to retrieve the required data block b_i. The scheduled start-time e_i and the required fulfill-time f_i of task T_i can be computed by the following equations:

$$\begin{aligned} e_i &= \begin{cases} 0 & \text{if } (i = 0) \\ \max\{r_i,\ f_{i-1}\} & \text{otherwise,} \end{cases} \\ f_i &= e_i + c_{i-1,i}. \end{aligned} \tag{1}$$

The variable $c_i = c_{i-1,i}$ is the required service time for serving T_i after serving T_{i-1}. In a HP 97560 hard disk (Reddy and Wyllie, 1993), the service time $c_{i-1,i}$ can be approximated by

$$c_{i-1,i} = \begin{cases} 8.00 + 0.008 \times D_{i-1,i} & \text{if } D_{i-1,i} > 383 \\ 3.24 + 0.4 \times \sqrt{D_{i-1,i}} & \text{Otherwise} \end{cases} \tag{2}$$

where the variable $D_{i-1,i} = |a_i - a_{i-1}|$ is the movement distance of disk-head. As shown in the above equation, the required service time is dependent on the schedule result. Although the disk driver is specified, the schedule result is unknown before scheduling.

In this paper, if all the scheduled tasks $T_{Z(i)}$ (for $i = 0$ to n) satisfy their real-time requirements $r_{Z(i)} \le e_{Z(i)}$ and $f_{Z(i)} \le d_{Z(i)}$, this schedule $\boldsymbol{T}_Z$ is called a *feasible* schedule. For the feasible schedule $\boldsymbol{T}_Z$, the time that it takes to serve all the scheduled tasks $T_{Z(0)} T_{Z(1)} \cdots T_{Z(n)}$ is called the *schedule fulfill-time*. Since the initial release time $r_{Z(0)} = 0$, the schedule fulfill-time ($f_{Z(n)} - r_{Z(0)}$) of $\boldsymbol{T}_Z$ is the same as the fulfill-time of the latest scheduled task $f_{Z(n)}$. Define the total retrieved data size as $B = \sum_{i=1}^{n} b_i$, the disk throughput of schedule $\boldsymbol{T}_Z$ is

$$B / f_{Z(n)}. \tag{3}$$

The obtained disk throughput is just related to the inverse of the schedule fulfill-time. When comparing the scheduler $\boldsymbol{Z}$ with another scheduler $\boldsymbol{X}$, the improvement of disk throughput (called *improvement* for short) obtained by $\boldsymbol{Z}$ can be measured by

$$(1 - f_{Z(n)} / f_{X(n)}) \times 100\%. \tag{4}$$

As shown in Eq. (3) and (4), the problem objective defined to maximize the disk throughput can be achieved by minimizing the schedule fulfill-time. We formally formulate the real-time disk scheduling (RTDS) problem as follows.

Definition. ***Real-Time Disk Scheduling (RTDS)*** Given a set of real-time disk tasks $\boldsymbol{T} = \{T_0, T_1, \ldots, T_n\}$, find a feasible schedule $\boldsymbol{T}_Z = T_{Z(0)} T_{Z(1)} \cdots T_{Z(n)}$ that resolves $\mathbf{min}\{f_{Z(n)}\}$ under $r_{Z(i)} \le e_{Z(i)}$ and $f_{Z(i)} \le d_{Z(i)}$ for $i = 0$ to n.

This problem definition can be modeled by a fully connected graph $G = (V, E)$ with vertices $V = \boldsymbol{T}$ and edges E. In this graph, there are two cost-bounds (r_i and d_i) on each vertex T_i and an arbitrary transition cost $c_{i,j}$ on the edge from vertex T_i to vertex T_j. The

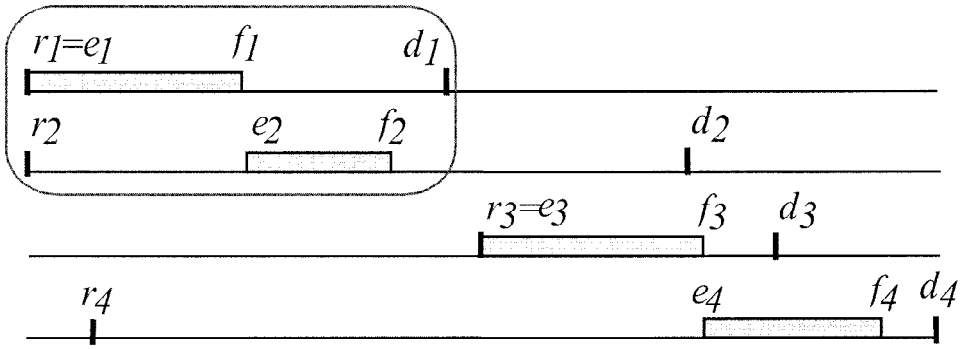


Figure 3. A simple example to demonstrate the identification procedure of MSG.

goal is to minimize the final tour cost (starting from vertex T_0 and visiting each vertex in G exactly once) without violating the cost-bound constraints (real-time constraints) of r_i and d_i. This problem has been proved to be NP-complete (Wong, 1980; Vin et al., 1995). In recent years, different approaches have been proposed to combine the seek-optimizing SCAN method and the real-time EDF scheduling method (i.e. SCAN-EDF) to resolve the RTDS problem. The obtained efficiency greatly depends upon *how often the SCAN algorithm can be applied*. Notably, the best task groups for seek-optimizing (called *the best task groups*, for short) may have varying sizes for different task groups.

3. Proposed Approach to Reschedule MSG by the DMS Scheme

3.1. Definition of Maximum-Scannable-Group

Generally, doing seek-optimization on the maximum scannable task groups would provide a large benefit to disk throughput. To avoid the incorrect partitioning of the task groups, we should identify the scannable task groups as large as possible. In this paper, an MSG (maximum-scannable-group) concept is introduced as follows.

Definition. ***MSG (maximum-scannable-group)*** Given an EDF schedule $\boldsymbol{T}_e = T_{E(0)}T_{E(1)} \cdots T_{E(n)}$, the maximum-scannable-group G_i started from task T_i is defined as the maximum consequent task group $G_i = T_{E(i)}T_{E(i+1)} \cdots T_{E(i+m)}$ which satisfies: $f_{E(k)} \leq d_{E(i)}$ and $r_{E(k)} \leq e_{E(i)}$ for $k = i$ to $i + m$.

A simple example to demonstrate the proposed MSG concept is shown in Fig. 3. Assume that the input schedule is $T_0T_1T_2T_3T_4$. Considering the group G_1 that started from T_1, we have $f_2 \leq d_1$ and $r_2 \leq e_1$, but $f_3 > d_1$ and $r_3 > e_1$. Thus, T_2 is in G_1, but T_3 is not in G_1. Based on the above definition, we can construct all MSG groups ($G_0 = T_0$, $G_1 = T_1T_2$, $G_2 = T_2$, $G_3 = T_3$, and $G_4 = T_4$). The theoretical proof for the correctness of our MSG definition is given as follows.

THEOREM *$\boldsymbol{T}_E = T_{E(0)}T_{E(1)}\cdots T_{E(n)}$ is an EDF schedule with MSG $G_i = T_{E(i)}T_{E(i+1)}\cdots T_{E(i+m)}$. Assume that $S_i = T_{S(i)}T_{S(i+1)}\cdots T_{S(i+m)}$ is the seek-optimized reschedule result of G_i for $\boldsymbol{T}_S = T_{E(0)}T_{E(1)}\cdots T_{E(i-1)}T_{S(i)}T_{S(i+1)}\cdots T_{S(i+m)}T_{E(i+m+1)}\cdots T_{E(n)}$. If $\boldsymbol{T}_E$ is feasible, $\boldsymbol{T}_E$ can be rescheduled as $\boldsymbol{T}_S$ to improve the disk throughput under guaranteed real-time requirements.*

Proof: (a) In the input schedule $\boldsymbol{T}_E$, from the definition of the fulfill-time, we have

$$f_{E(i+m)} = \mathbf{max}\{\cdots \mathbf{max}\{e_{E(i)} + c_{E(i-1),E(i)}, r_{E(i+1)}\}\cdots\} + c_{E(i+m-1),E(i+m)}. \tag{5}$$

As G_i is an MSG, we have $r_{E(k)} \le e_{E(i)}$ for $k = i$ to $i+m$. Eq. (5) can be rewritten as

$$f_{E(i+m)} = e_{E(i)} + c_{E(i-1),E(i)} + c_{E(i),E(i+1)} + \cdots + c_{E(i+m-1),E(i+m)}.$$

In schedule $\boldsymbol{T}_S$, the related fulfill-time of S_i is

$$f_{S(i+m)} = \mathbf{max}\{\cdots \mathbf{max}\{e_{S(i)} + c_{S(i-1),S(i)}, r_{S(i+1)}\}\cdots\} + c_{S(i+m-1),S(i+m)}. \tag{6}$$

From the definition of start-time $e_{E(i)} = \mathbf{max}\{r_{E(i)},\ f_{E(i-1)}\}$, we have $e_{S(i)} \le e_{E(i)}$. Since S_i is a reschedule of G_i, they contain the same tasks. Given a task $T_{S(x)}$ in S_i, we have $r_{S(x)} = r_{E(k)}$ for an arbitrary k in $[i, i+m]$. Thus, $r_{S(x)} \le e_{E(i)}$ for $x = i$ to $i+m$. Eq. (6) can be rewritten as

$$f_{S(i+m)} \le e_{E(i)} + c_{S(i-1),S(i)} + c_{S(i),S(i+1)} + \cdots + c_{S(i+m-1),S(i+m)}. \tag{7}$$

Given S_i as a seek-optimized reschedule result of G_i for $\boldsymbol{T}_S$, we have

$$c_{S(i-1),S(i)} + c_{S(i),S(i+1)} + \cdots + c_{S(i+m-1),S(i+m)} \le c_{E(i-1),E(i)} + c_{E(i),E(i+1)} + \cdots + c_{E(i+m-1),E(i+m)}. \tag{8}$$

$\Rightarrow$ From Eq. (7) and (8), the fulfill-time is $f_{S(i+m)} \le f_{E(i+m)}$. The obtained disk throughput in the rescheduled group S_i would not be smaller than that of the input group G_i.

(b) From the definition of MSG G_i, we have $f_{E(k)} \le d_{E(i)}$ for all $k = i$ to $i+m$. As G_i is an EDF schedule, we also have $f_{E(k)} \le f_{E(k+1)}$ and $d_{E(k)} \le d_{E(k+1)}$ for all $k = i$ to $i+m-1$. These relations can be rewritten as

$$f_{E(i)} \le f_{E(i+1)} \le \cdots \le f_{E(i+m)} \le d_{E(i)} \le d_{E(i+1)} \le \cdots \le d_{E(i+m)}. \tag{9}$$

From the definition of S_i, the fulfill-time $f_{S(k)} \le f_{S(k+1)} \le f_{E(i+m)}$ for all $k = i$ to $i+m$. These relations can be written as

$$f_{S(i)} \le f_{S(i+1)} \le \cdots \le f_{S(i+m)} \le f_{E(i+m)}. \tag{10}$$

$\Rightarrow$ Note that, $\mathbf{min}\{d_{S(k)} \mid \text{for all } k = i \text{ to } i+m\} = d_{E(i)}$. From Eq. (9) and (10), we can guarantee that the real-time constraints for S_i are satisfied. If $\boldsymbol{T}_E$ is feasible, $\boldsymbol{T}_E$ can be rescheduled as $\boldsymbol{T}_S$ to improve the disk throughput under guaranteed real-time requirements. ■

We prove that the seek-optimized result of MSG can improve the disk throughput with guaranteed real-time requirements. As the seek-optimized fulfill-time of the rescheduling result would not be larger than that of the input schedule, we can focus only on the real-time requirements. In our proposed MSG definition, we use $f_{E(k)} \leq d_{E(i)}$ and $r_{E(k)} \leq e_{E(i)}$ (for $k = i$ to $i + m$) to guarantee real-time requirements.

3.2. Identification of Maximum-Scannable-Group

Define G_i as the maximum scannable group starting from T_i. There are $n + 1$ MSG groups considered $(G_0, G_1, \ldots, G_n)$. For these successive MSG groups, a useful *sub-group* property is introduced as follows.

Property *Give an EDF schedule* $\boldsymbol{T}_E = T_{E(0)}T_{E(1)}\cdots T_{E(n)}$. *For each MSG* $G_i = T_{E(i)}$ $T_{E(i+1)}\cdots T_{E(i+m)}$, *its sub-group* $T_{E(x)}T_{E(x+1)}\cdots T_{E(i+m)}$ *must be a part of MSG* G_x *for* $x = i + 1$ *to* $i + m$.

Proof: (a) Since $T_{E(0)}T_{E(1)}\cdots T_{E(n)}$ is an EDF schedule, $d_{E(i)} \leq d_{E(i+1)}$ and $e_{E(i)} \leq e_{E(i+1)}$ for all i.

(b) As $G_i = T_{E(i)}T_{E(i+1)}\cdots T_{E(i+m)}$ is an MSG, we have $f_{E(k)} \leq d_{E(i)}$ and $r_{E(k)} \leq e_{E(i)}$ for $k = i$ to $i + m$.

$\Rightarrow$ From (a) and (b), it can be easily derived that $f_{E(k)} \leq d_{E(i)} \leq d_{E(x)}$ and $r_{E(k)} \leq e_{E(i)} \leq e_{E(x)}$ for $x = i + 1$ to $i + m$. Thus, the sub-group $T_{E(x)}T_{E(x+1)}\cdots T_{E(i+m)}$ of G_i must be a part of G_x. ■

By applying the above sub-group property, $n + 1$ MSG groups can be easily identified by an one-pass $O(n)$ algorithm. Define *Start_*G_i and *End_*G_i as the start and the end index of the task group G_i. The proposed MSG identification algorithm is shown as follows.

Algorithm: ***MSG (maximum-scannable-group)***

1. /* INPUT: an EDF schedule $T_0T_1\cdots T_n$.
   OUTPUT: MSG groups $G_0, G_1, \ldots,$ and G_n. */
2. $k := 1$;
3. **for** $i := 0$ **to** n **do begin** /* n is the number of input tasks */
4. **while** $((r_k \leq e_i)$ **and** $(f_k \leq d_i)$ **and** $(k \leq n))$ **do** $k := k + 1$;
5. /* G_i is defined as $T_iT_{i+1}\cdots T_{k-1}$ */
6. *Start_*$G_i := i$; /* *Start_*G_i is the start index of group G_i */
7. *End_*$G_i := k - 1$; /* *End_*G_i is the end index of group G_i */
8. **end** /* next MSG group */

Notably, tasks in MSG may have the same scan direction (called *scanned MSG*). Besides, these MSG groups may not be mutually-exclusive. In this paper, we consider only the un-scanned and mutually-exclusive groups for rescheduling. Define the forward scan direction as $+1$ and the backward scan direction as -1. We represent the cumulated scan direction of tasks in G_i by variable $Z[i]$. Notably, the task group G_i is scanned if and only if $|Z[i]|$ is equal to the group size. The detail description of an $O(n)$ algorithm to identify these un-scanned and mutually-exclusive MSG groups is illustrated as follows.

Algorithm: ***MU_MSG (mutually-exclusive and un-scanned MSG groups)***

1. /* INPUT: MSG groups G_0, G_1, ..., and G_n.
   OUTPUT: MU_MSG, a subset of MSG. */
2. $Z[0] := 0$; /* $Z[i]$ is the cumulated scan direction of G_i */
3. **for** $k := 1$ **to** n **do begin**
4. **if** ($a_{k-1} \leq a_k$) then *scan_direction* $:= 1$; **else** *scan_direction* $:= -1$;
5. /* MSG is scanned if and only if the scan directions are $+1, +1, \ldots, +1$ or $-1, -1, \ldots, -1$ */
6. $Z[k] := Z[k-1] +$ *scan_direction*;
7. **end** /* next task */
8. $i := 0$;
9. **while** ($i < n$) **do begin**
10. **if** $((|Z[End\_G_i] - Z[Start\_G_i]| + 1) =$ *size of* G_i) /* if G_i is seek-optimized */
11. **then** $i := i + 1$; /* next group */
12. **else begin**
13. G_i is defined as a MU_MSG and can be rescheduled by SCAN;
14. $i := End\_G_i + 1$; /* next mutually exclusive group */
15. **end** /* else */
16. **end** /* group */

In this paper, these MU_MSG groups are identified by a first-come-first-serve (FCFS) policy. As these MU_MSG groups are mutually-exclusive, tasks in different MU_MSG groups can be seek- optimized by SCAN at the same time.

3.3. Deadline-Modification-Scan for Maximum-Scannable-Group

As shown in the previous sub-section, our MSG concept is based on an EDF input schedule. However, in real-world systems, the problem input may not be an EDF schedule. To resolve this drawback, a DMS (*deadline-modification-scan*) algorithm is proposed to modify the original problem input and transfer it into an EDF input schedule. For example, given the schedule sequence T_iT_j, we can simply modify the deadline $d_i = \mathbf{min}\{d_i, d_j\}$ to satisfy the EDF requirements $d_i \leq d_j$. This O(n) procedure is called *deadline-modification (DM)*. After executing the DM algorithm, we can prove that the new problem input is an EDF schedule. The MSG concept can be applied to this new problem input to further improve the disk throughput. Notice that, the rescheduled result of MSG may violate the original EDF property. We can execute the DM algorithm again to produce a new EDF problem input. Thus, the MSG concept can be applied repeatedly to further improve the disk throughput. This DMS (DM-SCAN) algorithm is described as follows.

Algorithm: ***DMS (deadline-modification-scan)***

1. /* INPUT: the initial schedule. OUTPUT: the obtained schedule. */
2. Store $d'_k := d_k$ for all tasks T_k /* use d'_k to backup real deadline d_k */
3. **repeat**
4. **for** $k := n - 1$ **down to** 1 **do** /* deadline modification */
5. **if** $(d_k > d_k + 1)$ **then** $d_k := d_{k+1}$;
6. Identify all the MSG groups from input schedule
7. Identify all the MU_MSG groups from MSG groups
8. **for** $i := 1$ **to** n **do** /* group seek-optimization */
9. **if** (G_i is MU_MSG) **then** reschedule G_i by SCAN;
10. **until** (no deadline is modified); /* no more change */
11. Recover $d_k := d'k$ for all tasks T_k. /* restore real deadlines */

A simple example to demonstrate the processing steps of DMS is shown in Fig. 4. Given an input schedule $T_0T_1T_2T_3T_4$ (as $d_3 > d_4$, it is not an EDF schedule), we can modify deadline $d_3 = d_4$ to satisfy the EDF property between T_3 and T_4. By following the same idea, d_2 is not changed and d_1 is modified as d_2. As the modified deadlines satisfy the EDF requirements ($d_0 \leq d_1 \leq d_2 \leq d_3 \leq d_4$), the proposed DMS algorithm is correct for iterative computing. Experiments show that the results of DMS can be convergent within a small iteration number ($\ll n$). The theoretical proof of convergence easily can be shown by the monotonic-decreasing property of modified deadlines (Chang, 1996).

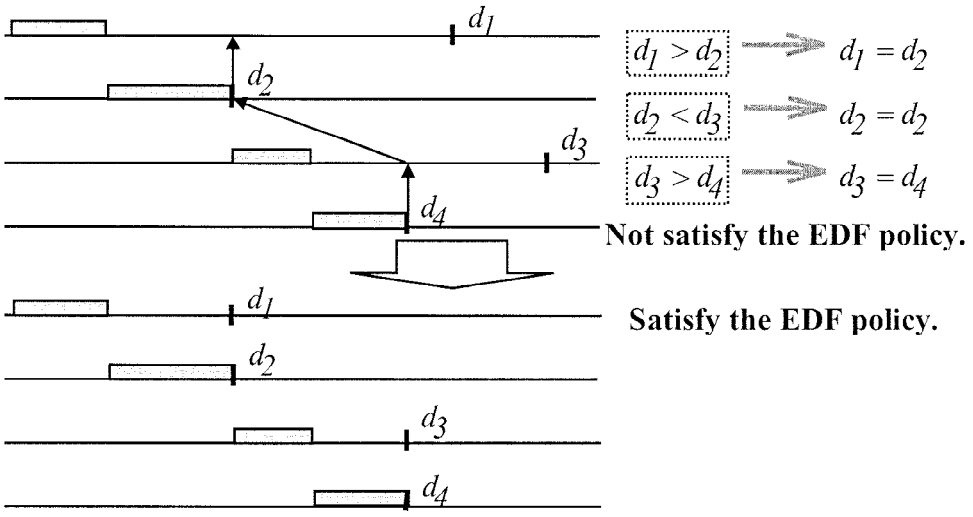


Figure 4. A simple example to demonstrate the proposed DMS scheme.

Notably, if the input schedule is feasible for real-time requirements, we have $f_i \leq f_{i+1}$ and $f_{i+1} \leq d_{i+1}$ for each input task T_i ($i = 0$ to n). Thus, after executing the DM algorithm, the obtained EDF schedule is also a feasible schedule (Chang, 1996). We can conclude that *the obtained result will be a feasible RTDS schedule if the input schedule is feasible.* Notably, in our MSG definition, the input schedule is not necessary to be feasible. Besides, the input schedule is not necessary to be an EDF schedule. However, if the input schedule is infeasible, our obtained result may not be a feasible RTDS schedule. We try to demonstrate this schedulability ratio by testing infeasible input schedules in our experiments. Many task sets that are not schedulable by EDF can be scheduled by our proposed approach.

4. Experimental Results

In this paper, we apply the HP 97560 hard disk for performance evaluations. HP 97560 contains 1962 cylinders. It has 19 tracks-per-cylinder and 72 sectors-per-track with 512 bytes sector size. The rotation speed is 4002 RPM (rotations-per-minute). In this paper, we apply this disk to model the disk service time $c_{i,j}$. Given the seek distance $D_{i,j} = |a_j - a_i|$, the value of $c_{i,j}$ can be computed by Eq. (2). We assume that the data access for each input task is one disk track (36 KB in HP 97560). The workloads of input tasks are uniformly distributed on the disk surface. Besides, tasks' release times are generated with proper hard-deadlines assigned. In our experiments, the applied release times of tasks are uniformly distributed among 0 and 240 ms (milliseconds). The related deadline is formulated by the release time with an additional period time varied from 120 to 480 ms. More detail descriptions about the disk layout scheme supported for multimedia applications can be found in Wang et al., 1997.

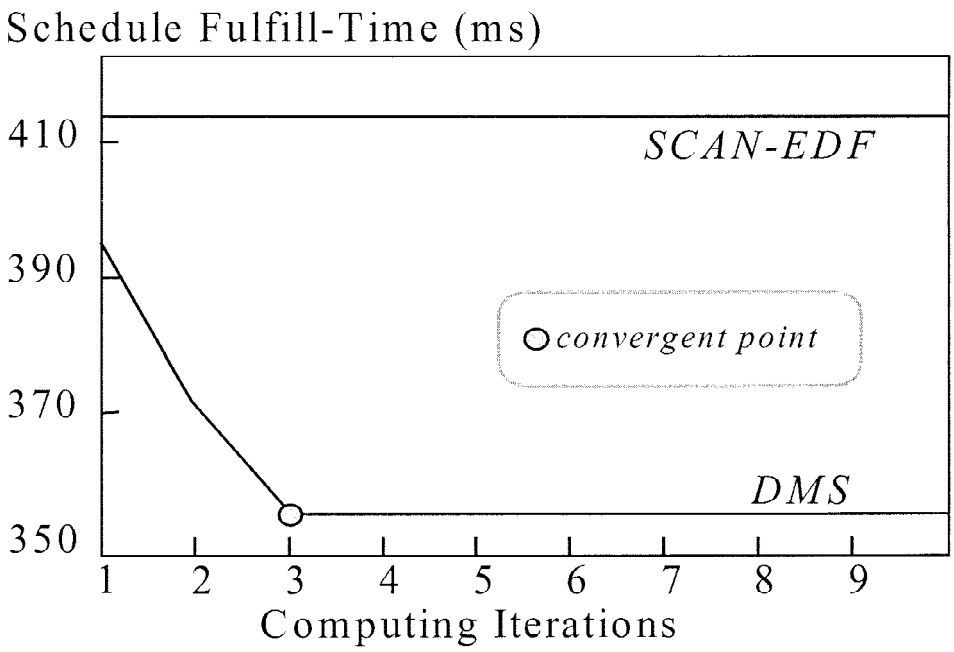


Figure 5. The convergence property of SCAN-EDF and the proposed DMS approach.

4.1. Number of Iterations to Convergence

For an iterative algorithm such as DMS, the iteration number required to convergence is an important factor to measure system performance. In this paper, we evaluate the convergence property of DMS by the obtained schedule fulfill-time under different iteration numbers. The obtained results of SCAN-EDF are also presented for comparisons. For each approach, the same 100 test examples with 10 random-generated disk tasks are applied. The mean value of the obtained schedule fulfill-time is plotted as shown in Fig. 5. The gray circles, called *convergent points*, mark the iteration number required to convergence. Notably, as SCAN-EDF is a non-iterative approach, extra iterations do not change the obtained results. The required schedule fulfill-time is 414.8 ms. Different from SCAN-EDF, our proposed DMS method is an iterative approach. The increasing of applied iterations will improve the obtained results until the convergence is met. By applying only 3 computing iterations, our method can obtain 356.7 ms in the schedule fulfill-time. The required iteration time to convergence is small and the obtained results are highly improved. Moreover, our proposed method can achieve better results than SCAN-EDF in the first iteration (called *DMS-1*). The required computation complexity of the DMS-1 method is $O(n \log n)$ (the same as SCAN-EDF). Table 1 summarizes the minimum, the maximum, and the average schedule fulfill-time obtained for different test approaches. We apply the same 100 test examples for all test approaches to keep the comparison as fair as possible. Each test example contains

Table 1. The obtained schedule fulfill-time and the related improvement.

Algorithms	Schedule Fulfill-Time (ms: millisecond)			
	minimum	maximum	average	improvement
DMS	289.27 ms	402.36 ms	356.74 ms	14.00%
DMS-1	317.24 ms	460.18 ms	375.12 ms	9.58%
SCAN-EDF	377.06 ms	462.70 ms	414.84 ms	≈ 0.00%

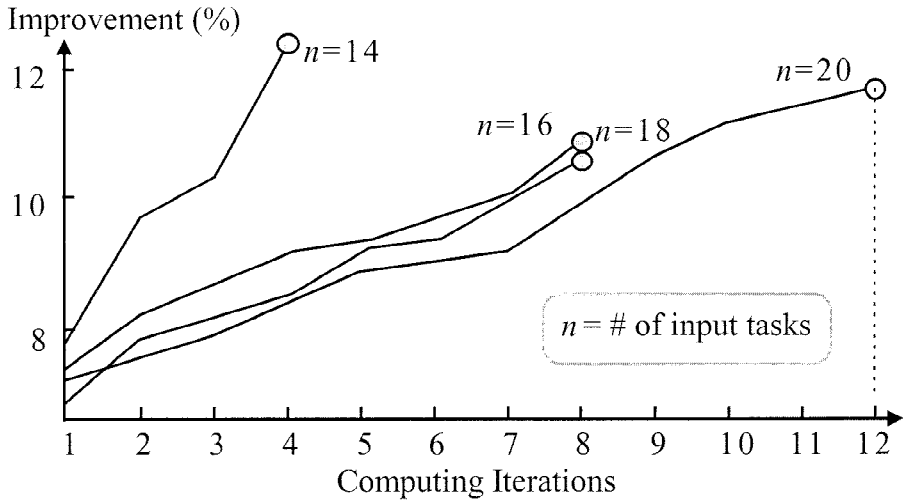


Figure 6. The performance improvements of DMS with different input tasks.

10 random-generated disk tasks and has a feasible EDF schedule. Comparing the obtained improvements defined in Eq. (4), our DMS approach can achieve over improvements that are 14% higher (9% at the first iteration) than SCAN-EDF.

4.2. Number of Input Tasks

To further demonstrate the capability of DMS, we compare the obtained improvements for different computing iterations under four different problem sizes (14, 16, 18, and 20 input tasks) as shown in Fig. 6. In each point, the mean value of the performance improvements for 100 test examples is presented. Experiments show that all the obtained results have improvements that are over 10%. Besides, all these test examples are convergent before n computing iterations (n is the number of tasks). This iteration bound can be easily proved by the limited number of input tasks and the definition of applied MSG (Chang, 1996). In Fig. 7, DMS is applied to compare with conventional approaches by the obtained improvements under different number of input tasks. We increase the test problem sizes from

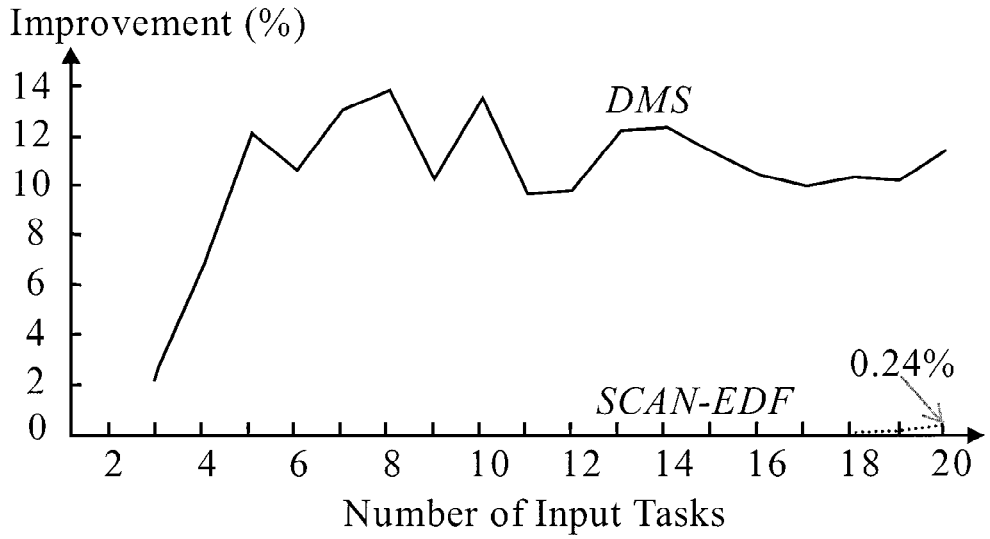


Figure 7. Experiments for DMS with different numbers of input tasks.

Table 2. The numbers of task supported.

Algorithms	**Number of Supported Tasks**		
	minimum	**maximum**	**average**
DMS	7	25	13.8
SCAN-EDF	4	24	10.2
FCFS	1	10	4.1
SCAN	1	4	3.3

3 to 20 input tasks. In each test problem size, 100 test examples are applied. Experiments show that the proposed DMS approach is better than these conventional approaches in all the test examples. Our proposed DMS method can achieve improvements over 11% in disk throughput when the problem size is increased to 20 tasks. It demonstrates that DMS can handle large-sized problems.

Table 2 shows the minimum, the maximum and the average number of input tasks that can be supported by different approaches. For each test approach, the same 100 test examples are applied. In each test example, we try to increase the number of input tasks until the obtained schedule is not feasible. The applied problem size is recorded as the supported number of requests in this test example. Experiments show that our proposed DMS approach can support more tasks than conventional approaches. Comparing with EDF, our proposed method achieves over 1.35 in the ratio of the supported average number of tasks. This ratio is defined as follows:

$$(\text{tasks schedulable by DMS})/(\text{tasks schedulable by EDF}). \tag{11}$$

Notice that, as the real-time constraints are not considered in SCAN scheduling, the SCAN method has the worst results in the supported number of tasks. Although SCAN is a seek-optimization method for disk scheduling, these seek-optimized schedules are not acceptable for time-critical applications.

4.3. Ratio of Schedulability

In conventional approaches, the input task set must be a feasible EDF schedule. As the input task set is schedulable by EDF, these approaches just have a minor improvement to obtain a better fulfill-time. In this paper, our proposed DMS method can schedule task sets that are not schedulable by EDF. As shown in the left side of Fig. 8(a), the EDF schedule $T_0T_1T_2T_3T_4$ is not feasible for real-time requirements because task T_4 misses its deadline ($d_4 < f_4$). This task set is not schedulable by EDF and SCAN-EDF. In this paper, by applying our DMS algorithm to $T_0T_1T_2T_3T_4$, we obtain the MSG group $T_0T_1T_2T_3$ (see Fig. 8(a)). As shown in Section 3, this MSG group can be successfully seek-optimized as $T_0T_1T_3T_2$ to reduce the related fulfill-time. Thus, the next task T_4 can be started earlier than that in the original schedule. The new fulfill-time (called f_4^*) is smaller than f_4. As show in the right side of Fig. 8(a), we have $f_4^* \leq d_4 < f_4$. The output schedule of DMS ($T_0T_1T_3T_2T_4$) is feasible for real-time requirements even when the input EDF schedule $T_0T_1T_2T_3T_4$ is infeasible.

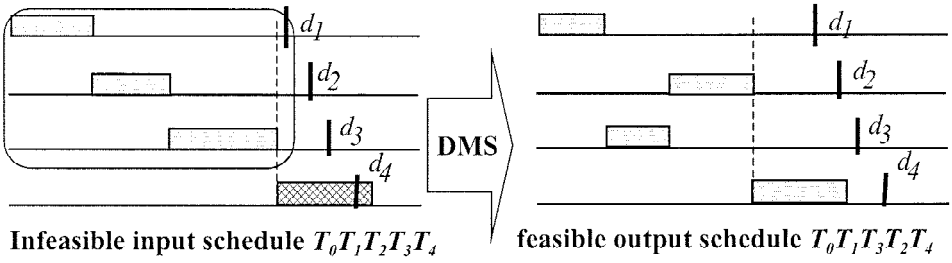
Although our proposed algorithm can schedule task sets that are not schedulable by EDF, the real-time requirements could be guaranteed only when the input schedule is feasible. In another words, reschedule results may be infeasible for real-time requirements if the input schedule is infeasible. We define the ratio of schedulability as follows:

$$\text{(task sets schedulable by DMS)}/\text{(task sets schedulable by EDF)}. \tag{12}$$

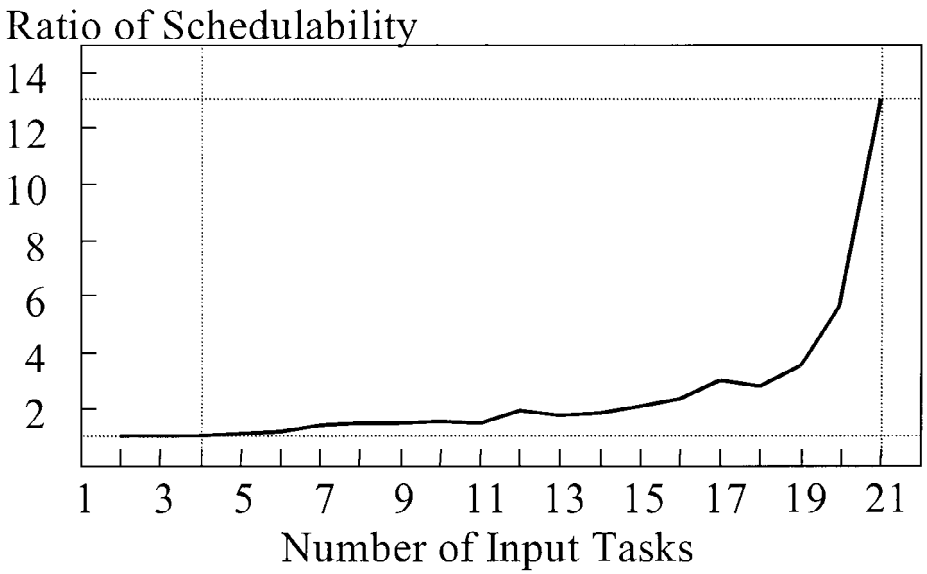
Fig. 8(b) shows the obtained ratio of schedulability under different problem sizes. For each problem size, the same 100 test examples are applied to both the EDF and DMS approaches for fair comparisons. Experiments show that the obtained ratio will be 1 if the number of input tasks is less than 5. When the problem size is increased to 21 tasks, the obtained ratio is increased to 13. Our proposed approach can schedule the task sets that are not schedulable by EDF. The ratio of task sets that are not schedulable by EDF but schedulable by DMS is over 12.

4.4. Implementation Results

In this paper, we also implement our proposed DMS scheme on a real system to evaluate the obtained performance. The best-known SCAN-EDF scheme is implemented for comparisons. At the server site, we design a parent thread to deal with arrival media services. These presented media services are random-arrived with uniformly distributed data allocations. After receiving the requests of media services, a child thread is created (by the parent thread) to decide the data location and send the real-time disk tasks. Our algorithm is implemented on the system kernel to handle these real-time disk tasks. Thus, the real-time network transmission scheduler (Chang et al., 1997) can send the retrieved media data to



(a)



(b)

Figure 8. (a) By applying the proposed DMS method, an infeasible input schedule may produce a feasible output schedule. (b) The ratio of schedulability obtained.

the clients for continuous playback. Our implementation currently runs on a platform with UnixWare 2.01 operating system. The applied storage device is the Seagate ST-31200N hard disk with the Adaptec SCSI-2742AT control card. In this paper, we consider a disk array system with 20 identical disk devices. For each hard disk, 20 KB/s bandwidth is required for serving one MPEG-I video stream. We implement the real-time disk scheduler on the *portable device interface* (PDI) of the SCSI hard disk driver. It takes the advantage

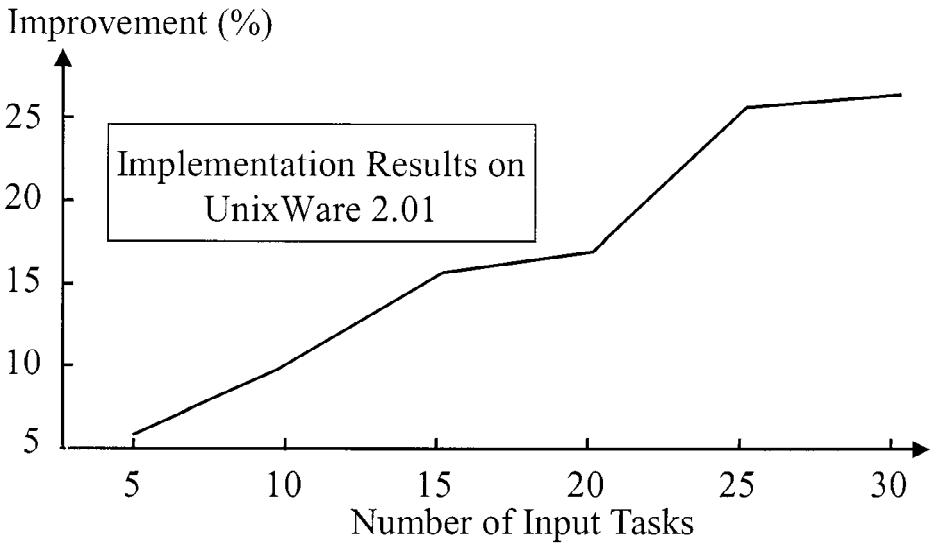


Figure 9. The implementation results of the proposed DMS method on UnixWare 2.01.

to organize and standardize the processing flow of device drivers. Thus, our implementation can be easily ported to other systems. More detail discussions about the top-down hierarchical structure of PDI, the original design of PDI for disk scheduling, and the detail implementation of the proposed algorithms can be found in (Chang, 1996). Fig. 9 shows the obtained improvements of implementation results for different number of input tasks (task arrival rates). In each point, the mean value of improvements obtained for 100 test examples is presented. Our implementation results show that the proposed DMS method can achieve better results than the well-known SCAN-EDF approach.

5. Conclusion

In this paper, based on the MSG (maximum-scannable-group) concept, we propose a DMS (deadline-modification-scan) algorithm to resolve the RTDS (real-time disk scheduling) problem. In conventional RTDS methods, the seek-optimization procedure is applied only for the tasks with the same deadline. Thus, the obtained improvement is limited. Our DMS algorithm can automatically identify MSG to iteratively reduce the disk service time. Thus, the obtained disk throughput can be highly improved. In this paper, both the experiment results and the implementation results are presented. The investigations of the appropriate problem behaviors and parameter values to yield good solutions are also proposed. Experiments show that our obtained approach can achieve higher disk throughput and support more real-time disk tasks than the conventional approaches. Different from the conventional SCAN-EDF approach, our problem input is not necessary to be a feasible EDF schedule. Many task sets that are not schedulable by EDF can be schedulable by our proposed ap-

proach. In this paper, the introduced MSG and DMS are two general concepts for real-time disk scheduling. Based on different problem configurations, we can define different MSG formulations and different DMS algorithms. How to further extend these concepts to other real-time applications is an interesting research problem. We will investigate this problem in our future works.

Appendix 1

Algorithm: SCAN

1. /* Input: a set of tasks $\boldsymbol{T} = \{T_0, T_1, \ldots, T_n\}$.
   Output: the schedule $\boldsymbol{T}_S = T_{S(0)} T_{S(1)} \cdots T_{S(n)}$. */
2. Assume that the initial disk-head is at track 0 (the data location of task T_0 is $a_0 = 0$).
3. Sort the input tasks in the ascending order of their data locations.
4. Assume that $a_{S(0)} a_{S(1)} \cdots a_{S(n)}$. The schedule result is $\boldsymbol{T}_S = T_{S(0)} T_{S(1)} \cdots T_{S(n)}$.
5. /* The sequence of subscripts $S(i)(i = 0 \text{ to } n)$ is a permutation of task index $\{0, 1, \ldots, n\}$. */

Acknowledgments

The authors would like to thank anonymous referees for their valuable comments and suggestions about this paper. This work was partially supported by NSC under grants NSC88-2213-E-001-018 and NSC88-2213-E-001-019.

References

Anderson, D. P. 1993. Metascheduling for continuous media. *ACM Trans. Computer Systems* 11(3): 226–252.

Anderson, D. P., Osawa, Y. and Govindan, R. 1991. Real-time disk storage and retrieval of digital audio/video data. Tech. Report, Univ. of California, Berkeley, Dept. of Computer Science.

Anderson, D. P., Osawa, Y. and Govindan, R. 1992. A file system for continuous media. *ACM Trans. Computer Systems* 10(4): 311–337.

Chang, R. I. 1996. Real-time disk scheduling in multimedia systems. Ph.D. Thesis, College of EE and CS, Dept. of CIS, NCTU, Taiwan, ROC.

Chang, R. I., Chen, M. C., Ho, J. M. and Ko, M. T. 1997. Optimizations of stored VBR video transmission on CBR channel. *Proc. SPIE VVDC: Performance and Control of Network Systems*, pp. 382–392.

Chang, R. I., Chen, M. C., Ho, J. M. and Ko, M. T. 1998. Characterize the minimum required resources for admission control of pre-recorded VBR video transmission by an $O(n \log n)$ algorithm. *Proc. IEEE Int. Conf. on Computer Communications and Networks*.

Chang, R. I., Chen, M. C., Ho, J. M. and Ko, M. T. 1999. An Effect and Efficient Traffic-Smoothing Scheme for Delivery of Online VBR Media Streams. *Proc. IEEE INFOCOM*.

Chang, R. I., Shih, W. K. and Chang, R. C. 1998. Deadline-modification-scan with maximum-scannable-groups for multimedia real-time disk scheduling. *Proc. IEEE RTSS*.

Chen, M., Kandlur, D. D. and Yu, P. S. 1993. Optimization of the grouped sweeping scheduling (GSS) with heterogeneous multimedia streams. *Proc. ACM Multimedia Conf.*, pp. 235–242.

Chen, S. and Thapar, M. 1996. I/O channel and real-time disk scheduling for video servers. *Proc. NOSSDAV*, pp. 113–122.

Chen, T. S. and Yang, W. P. 1992. Amortized analysis of disk scheduling algorithm V(R)*. *Journal of Inform. Science and Eng*. 8: 223–242.

Chen, T. S., Yang, W. P. and Lee, R. C. T. 1992. Amortized analysis of some disk scheduling algorithms: SSTF, SCAN, and N-Step SCAN. *BIT* 32: 546–558.

Daigle, S. J. and Srosnider, J. K. 1996. Disk scheduling for continuous media data streams. *Proc. SPIE Conference on High-Speed Networking and Multimedia Computing*.

Dan, A., Sitaram, D. and Shahabuddin, P. 1994. Scheduling policies for an on-demand video server with batching. *Proc. ACM Multimedia Conf.*, pp. 15–22.

Gemmell, D. J. and Christodoulakis, S. 1992. Principles of delay sensitive multimedia data storage and retrieval. *ACM Trans. Information Systems* 10(1): 51–90.

Gemmell, D. J. and Han, J. 1994. Multimedia network file servers: multichannel delay sensitive data retrieval. *Multimedia Systems* 1(6): 240–252.

Gemmell, D. J., Vin, H. M., Kandlur, D. D., Rangan, P. V. and Rowe, L. A. 1995. Multimedia storage servers: a tutorial. *IEEE Computers*, pp. 40–49.

Jeffay, K., Stanat, D. F. and Martel, C. U. 1991. On nonpreemptive scheduling of periodic and sporadic tasks. *Proc. of Real-Time Systems Symp.*, pp. 129–139.

King, R. P. 1990. Disk arm movement in anticipation of future requests. *ACM Trans. Computer Systems* 8(3): 214–229.

Lehoczky, J. P. 1990. Fixed priority scheduling of periodic task sets with arbitrary deadlines. *Proc. Real-Time Systems Symp.*, pp. 201–212.

Lin, T. H. and Tarng, W. 1991. Scheduling periodic and aperiodic tasks in hard real-time computing systems. *Proc. SIMMetrics Conf.*, pp. 31–38.

Liu, C. L. and Layland, J. W. 1973. Scheduling algorithms for multiprogramming in a hard real- time environment. *Journal of ACM*, pp. 46–61.

Lougher, P. and Shepherd, D. 1993. The design of a storage server for continuous media. *The Computer Journal* 36(1): 32–42.

Mok, A. 1983. Fundamental design problems for the hard real-time environment. MIT Ph.D. Dissertation, Cambridge, MA.

Peterson, J. L. and Silberschatz, A. 1985. *Operating System Concepts*. Addison-Wesley.

Rangan, P. V. and Vin, H. M. 1991. Designing file systems for digital video and audio. *Proc. ACM Symp. Operating Systems*, pp. 81–94.

Rangan, P. V. and Vin, H. M. 1993. Efficient storage techniques for digital continuous multimedia. *IEEE Trans. Knowledge and Data Engineering*, 5(4): 564–573.

Reddy, A. L. N. and Wyllie, J. 1993. Disk scheduling in a multimedia I/O system. *Proc. ACM Multimedia Conf.*, pp. 225–233.

Reddy, A. L. N. and Wyllie, J. 1994. I/O issues in a multimedia system. *IEEE Computers.*, pp. 69–74.

Ruemmler, C. and Wilkes, J. 1994. An introduction to disk drive modeling. *IEEE Computers*, pp. 16–28.

Shih, W. K., Liu, J. W. S. and Liu, C. L. 1992. Modified rate monotone algorithm for scheduling periodic jobs with deferred deadlines. Tech. Report, Univ. of Illinois, Urbana-Champaign, Dept. of Computer Science.

Stankovic, J. A. and Buttazzo, G. C. 1995. Implications of classical scheduling results for real- time systems. *IEEE Computer*, pp. 16–25.

Steinmetz, R. 1995. Multimedia file systems survey: approaches for continuous media disk scheduling. *Computer Communication* 18(3): 133–144.

Terry, D. B. and Swinehart, D. C. 1988. Managing stored voice in the etherphone system. *ACM Trans. Computer Systems* 6(1): 3–27.

Tindell, K. and Burns, A. 1994. Fixed priority scheduling of hard real-time multi-media disk traffic. *The Computer Journal* 37(8): 691–697.

Vin, H. M., Goyal, A. and Goyal, P. 1995. Algorithms for designing multimedia servers. *Computer Comm*. 18(3): 192–203.

Vin, H. M. and Rangan, P. V. 1991. Designing file systems for digital video and audio. *Proc. ACM Symp. Operating System Principles*, pp. 81–94.

Wang, Y. C., Tsao, S. L., Chang, R. I., Chen, M. C., Ho, J. M. and Ko, M. T. 1997. A fast data placement scheme for video server with zoned-disks. *Proc. SPIE VVDC*, pp. 92–102.

Wong, C. K. 1980. Minimizing expected head movement in one dimension and two dimension mass storage system. *Comput. Survey* 12(2): 167–178.

Yee, J. and Varaiya, P. 1991. Disk scheduling policies for real-time multimedia applications. Tech. Report, Univ. of California, Berkeley, Dept. of Computer Science.

Yee, J. and Varaiya, P. 1995. Modeling and performance of real-time disk access policies. *Computer Communications* 18(10).