The Computer Journal, Vol. 47 No. 5, © The British Computer Society; all rights reserved

Cache-Aware Real-Time Disk Scheduling

HSUNG-PIN CHANG[1], RAY-I CHANG[2], WEI-KUAN SHIH[3] AND RUEI-CHUAN CHANG[4]

[1]*Department of Computer Science, National Chung Hsing University, Taichung, Taiwan, R.O.C.*
[2]*Department of Information Management, National Central University, Jhongli City, Taoyuan, Taiwan, R.O.C.*
[3]*Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, R.O.C.*
[4]*Department of Computer & Information Science, National Chiao Tung University, Hsinchu, Taiwan, R.O.C.*
Email: hpchang@cs.nchu.edu.tw

Previous real-time disk scheduling algorithms assume that each disk request incurs a physical disk mechanical operation and only consider how to move the disk head under real-time constraints. However, with the increased capacity of on-disk cache, modern disk drives read-ahead data aggressively. Thus, the on-disk cache may service many disk requests without incurring physical disk access. By exploring the design methodology of on-disk cache, in this paper, we propose cache-aware real-time disk scheduling algorithms that take the on-disk cache into consideration during scheduling. Therefore, the scheduling algorithm can help the cache replacement scheme to minimize the cache miss ratio. Besides, the service timing estimation is more accurate in schedulability analysis since the cache effect is considered during scheduling. A simulation-based evaluation shows the proposed scheduling algorithms to be highly successful as compared with the classical real-time disk scheduling algorithms. For example, under sequential workload with 10 sequential streams, the data throughput of our scheme is 1.1 times that of DM-SCAN.

Received 16 April 2003; revised 3 February 2004

1. INTRODUCTION

1.1. Motivation

With the immense popularity of the Web or broadcasting servers, the world is witnessing an unprecedented demand for data services. Nevertheless, in some cases, the delivery of data may have real-time constraints. For example, a radar system would need to compare images of objects against a database of known aircraft type. In the stock trading system, the stock price would be recorded and retrieved to broadcast to the subscribed clients. At the same time, the development of a speech-based information retrieval system for the blind, in which users can talk into the microphone and find the intended audio book, news or music directly, is under way. In this system they can hear the streaming audio from the server. All these applications have the characteristics that the requested data must be retrieved and delivered before a deadline; otherwise the data will be meaningless and even damaged. In this paper, we address real-time data retrieval by taking advantage of the on-disk cache and seek-optimizing the real-time disk scheduling scheme. The techniques for the delivery of real-time data can be found in [1, 2].

In a computer system, after disk scheduling, the scheduled requests are sent to and served by the disk drive [3]. However, because of the excess delay caused by the disk mechanical operation, a random access memory, i.e. an on-disk cache, is provided in disk controllers to bridge the speed gap between the main memory and disk and acts as a speed-matching buffer. Nevertheless, in the last couple of years, the drastic improvement in hardware technology has caused an increase in the capacity of the on-disk cache. Therefore, when a disk task is sent to the disk drive, the disk drive retrieves the requested data item to the on-disk cache. Furthermore, if read-ahead is enabled, the disk drive does read-ahead data following the requested data item to the on-disk cache. Accordingly, the data in the on-disk cache consist of both the read-ahead data and data read by previous requests.

However, there exists a read-ahead overhead since the disk head and track switches waste time and cannot be aborted once initiated. Nevertheless, under sequential workload, a disk with read-ahead caching still outperforms the disk without read-ahead caching [3, 4]. This is because, without read-ahead, if two back-to-back disk tasks are issued, the disk and host processing time for initiating the second request would be larger than the inter-sector gap. As a result, the second request would be delayed by almost a full revolution [3]. Since caches work on the premise that the issued tasks have spatial and temporal locality, with the hope of repeated or sequential access patterns, the on-disk cache can service many requests without incurring physical disk operations. If the majority of the accesses to disk are serviced by the on-disk cache, the I/O delay will be reduced significantly [5, 6, 7].

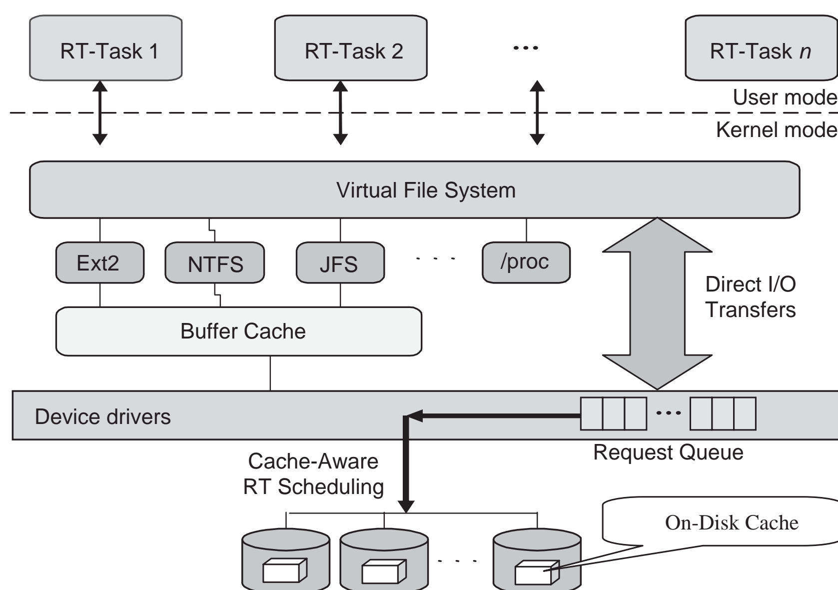

FIGURE 1. The system architecture.

The cache design methodology gives cache designers a competitive edge in the market. Therefore, manufacturers either patent them or consider their implementation a trade secret. However, if the parameters of the on-disk cache were disclosed, the caching effect would be taken into consideration during the disk scheduling. Consequently, not just the cache replacement scheme, but the scheduling algorithm also can help in preserving the principles of spatial and temporal locality. Besides, the service timing estimation is more accurate in schedulability analysis since a task's service time is accounted as cache transfer time if a cache hit occurred. Otherwise, a task's execution time must assume, in the worst case, that a mechanical disk access is incurred. As a result, during the schedulability testing, we must make worst-case assumptions of physical disk access time if we ignore the on-disk cache influence. This results in an overestimation of system resource usage and a decrease in system performance.

The idea of taking the on-disk cache into account in disk scheduling is also seen in [8]. The authors mentioned that requests that can be satisfied by the cache should be given higher priority to be accessed from the disk cache. However, they only simulate the caching effect for the performance evaluations of conventional disk scheduling algorithms, which have no timing requirements.

1.2. System architecture

Figure 1 shows the system architecture. In Unix-like systems, e.g. Linux, all system calls related to a standard file system are directed to the virtual file system (VFS). After receiving the issued system call, VFS invokes the appropriate file system function, say Ext2, to service this request. However, before accessing the disk, the system would look up the buffer cache to determine if the requested data are in the buffer cache or not. If the data are in the buffer cache, the system performance can be improved by reducing disk access. If the data are not in the buffer cache, a disk request is issued to the device driver.

Furthermore, file systems would also read ahead of files. In addition to the requested data, several adjacent blocks of data are also read. Nevertheless, some application programs would like to have full control of the whole I/O data transfer mechanism. Therefore, the operating system also offers direct I/O transfer to bypass the buffer cache. Thus, when a real-time task wishes to retrieve data from disks, it may access them through the file system or use direct I/O transfer to bypass the buffer cache.

Below the file system is the device driver. All disk access requests are sent to the device driver and queued in the request queue. Then, an appropriate disk scheduling algorithm selects one of the requests from the queue and sends it to the disk. As mentioned above, since modern disks are equipped with considerable on-disk cache, in this paper, we propose the on-disk cache-aware real-time disk scheduling algorithm that reduces the disk access time while guaranteeing tasks' real-time constraints.

Notably, both the file systems and disks have their own buffers and may issue read-ahead respectively. In this paper, the proposed algorithms only address the on-disk cache and the disk's read-ahead mechanism. In other words, we assume the real-time disk requests are from the direct I/O transfer. A combinational consideration of on-disk cache and buffer cache will be the subject of future work.

1.3. Contributions

On the basis of an existing real-time disk scheduling algorithm, DM-SCAN [9], three different cache-aware algorithms are proposed in this paper. They are the deadline shift scheme, the collaboration scheme, and the CARDS (Cache-Aware Real-Time Disk Scheduling) scheme.

In the deadline shift scheme, if requests are ordered in Earliest Deadline First (EDF) sequence, the tasks' deadlines are shifted if they are to be served by the on-disk cache, i.e. having spatial locality. Then, tasks with their shifted deadline are scheduled by the DM-SCAN scheme. In other words, disk tasks whose accesses have spatial locality are brought closer to meet their temporal locality by the deadline shift operation, thus increasing the cache hit probability.

In contrast, the collaboration scheme directly considers the on-disk cache effect in the DM-SCAN algorithm. First, the selection scheme of a reschedulable group (called MSG in DM-SCAN) is extended to consider the on-disk cache. Therefore, in addition to the original tasks identified by DM-SCAN, tasks that have the chance to be cache hits are also grouped into a reschedulable group. Besides, a cache-aware rescheduling scheme is proposed to augment the seek-optimizing SCAN rescheduling scheme to increase the cache hit ratio. Note that, reordering tasks may cause the schedule derived by DM-SCAN to become infeasible; therefore, feasibility checking must be performed for each reordering operation, and this consumes significant computational overhead. In this paper, techniques to accelerate the checking operation are also proposed.

Different from the two schemes above, the CARDS scheme is proposed that reorders the tasks if a cache hit is guaranteed after such a reordering, after the completion of the DM-SCAN algorithm. Experimental results shows that, under sequential accesses, our proposed cache-aware algorithms obtain larger data throughput than DM-SCAN, because of the increased cache hit ratio. For example, under sequential workload with 10 sequential streams, the data throughput of the CARDS scheme is 1.1 times that of DM-SCAN. Note that, although the proposed cache-aware real-time disk scheduling schemes are related tightly to the caching strategy and underlying on-disk cache characteristics, the proposed algorithms do not depend on any specific on-disk cache. That is, if the parameters of a different on-disk cache are disclosed, the proposed algorithms can be adapted easily to this disk.

In the rest of this paper, we shall first introduce the disk service model in a real-time environment, including on-disk cache design methodology, the timing characteristics of real-time tasks and the objective of a real-time disk scheduling algorithm in Section 2. Section 3 reviews the related work. In Section 4, we introduce the terms used in this paper. Section 5 presents the proposed three cache-aware real-time disk scheduling algorithms. The experimental results are shown in Section 6. Section 7 discusses some related issues about this paper. Finally, Section 8 summarizes this paper.

2. BACKGROUND

2.1. Design methodology of on-disk cache

Many applications process data sequentially, i.e. the next request will be for data following the current request. As a result, in addition to the service of requested data blocks, most disks, based on analyzing the access and usage pattern of recent requests, also perform the read-ahead. By reading-ahead, the requested data of subsequent accesses will reside in the on-disk cache and shorten the service time. Furthermore, some disk controllers even read-ahead aggressively to cross the track and cylinder boundaries. Nevertheless, since the on-disk cache has limited size, if very large read requests are issued, they may bypass the cache.

A single read-ahead cache can only support a single sequential stream. As a result, if two or more sequential streams are interleaved, this single read-ahead cache is no benefit at all. To remedy this pitfall, nowadays, the on-disk cache is often organized as a number of segments. A segment is a sequence of data blocks managed as a unit; i.e. each segment contains data that is disjointed from all other segments. Therefore, several unrelated data items can be cached at different segments. Some disk drives even dynamically resize the number (and size) of cache segments based on the recent access characteristics to ensure greater cache utilization. More details on the caching algorithm can be found in [3, 10, 11].

Compared with the capacity of a disk drive, the on-disk cache is small. Consequently, a segment replacement occurs when the cache is full of data and a new data block is requested. Note that, the replacement algorithm has a profound impact on the cache performance. A good replacement scheme should evict the segment that has no immediate access and retain the data more likely to be accessed soon. For example, random replacement (RR), least recently used (LRU) and least frequently used (LFU) are some of the well-known cache replacement algorithms [11, 12, 13, 14].

2.2. Real-time system

Assume that the start-time and finish-time denote the actual times at which a task is started and completed respectively. To characterize the timing characteristics of a real-time task, two parameters are associated with it to determine the proper start-time and finish-time.

- *Ready time*: the earliest time at which a task can start.
- *Deadline*: the latest time at which a task must be completed.

To satisfy the real-time requirements, the start-time of a task should not be earlier than its ready time. Additionally, its finish-time should not be later than the related deadline [15]. Depending on the consequence of a missed deadline, real-time tasks are further classified into hard and soft. A real-time task is said to be hard if missing its timing constraints will cause serious damage and the system will misbehave. In contrast, a real-time task is said to be soft if meeting its timing constraints is desirable for performance, but a missed deadline does not influence the correctness of system behavior. A schedule of real-time tasks is said to be feasible if all tasks can be sequentially served according to the specified real-time requirements. In this paper, we address hard real-time systems.

2.3. Real-time disk scheduling problem

As stated above, tasks in a real-time system must be associated with timing characteristics to describe their timing

constraints. Accordingly, a real-time disk task T_i is denoted by five parameters $(t_i, l_i, b_i, r_i, d_i)$, where t_i is the track location, l_i the sector number, b_i the data size, r_i the ready time and d_i its deadline. Assume that the schedule sequence is $T_j T_i$. Because disk tasks are non-preemptive, the start-time s_i and finish-time f_i of a real-time task T_i with schedule $T_j T_i$ are thus computed by $s_i = \max\{r_i, f_j\}$ and $f_i = s_i + c_{j,i}$ respectively, where $c_{j,i}$ denotes the service time of task T_i with schedule sequence $T_j T_i$. If T_i is a cache hit, $c_{j,i}$ is the value of cache access time. Otherwise, $c_{j,i}$ represents the time spent to access the physical disk.

Given a set of real-time disk tasks $\boldsymbol{T} = \{T_1, T_2, \ldots, T_n\}$, where n is the number of input disk tasks and the ith disk task T_i is denoted by $(r_i, d_i, t_i, l_i, b_i)$, the objective of a real-time disk scheduling algorithm is to find a feasible schedule $\boldsymbol{T_Z} = T_{z(1)} T_{z(2)} \cdots T_{z(n)}$ with maximal throughput. The index function $Z(i)$, for $i = 1$ to n, is a permutation of $\{1, 2, \ldots, n\}$. Define the schedule finish-time as the finish-time it takes to serve all input tasks according to their respective timing constraints. Clearly, this is the finish-time of the latest task $f_{z(n)}$. Therefore, the disk throughput is calculated as follows:

$$\text{Throughput} = \sum_{i=1}^{n} b_{z(i)} \Big/ f_{z(n)} \propto (f_{z(n)})^{-1} \qquad (1)$$

The obtained disk throughput is related to the inverse of the schedule finish-time. If the input schedule is completed earlier, more data throughput is obtained. The data throughput improvement of scheduler $\boldsymbol{Z}$ compared with scheduler $\boldsymbol{X}$ can be computed as

$$\text{Throughput improvement} = (1 - f_{z(n)}/f_{x(n)}) * 100\% \quad (2)$$

Therefore, the problem objective defined to maximize throughput can be achieved by minimizing the schedule finish-time. We formally define the real-time disk scheduling problem as follows.

DEFINITION 1. *Real-time disk scheduling. Given a set of n real-time disk tasks $\boldsymbol{T} = \{T_1, T_2, \ldots, T_n\}$ where the ith task T_i is $(r_i, d_i, t_i, l_i, b_i)$, find a feasible schedule $\boldsymbol{T_Z} = T_{Z(1)} T_{Z(2)} \ldots T_{Z(n)}$ that resolves $\min_{\forall Z}\{f_{Z(n)}\}$ under $r_{Z(i)} \le s_{Z(i)}$ and $f_{Z(i)} \le d_{Z(i)}$ for $1 \le z(i) \le n$.*

3. RELATED WORK

3.1. Real-time disk scheduling algorithms

In this subsection, previous real-time disk scheduling algorithms are investigated. EDF is a well-known real-time scheduling algorithm [16, 17]. By scheduling requests in the order of their deadlines, EDF has been shown as optimal if tasks are independent. However, for disk scheduling, the service time of a disk task depends on its previous task's location. The assumption that tasks are independent does not hold. Actually, taking only deadlines into account without considering service time, EDF incurs excessive seek-time costs and results in poor disk throughput [3].

Actually, real-time disk scheduling has been shown to be NP-complete [18]. Consequently, various approaches have been dedicated to combine the seek-optimizing SCAN scheme [19] with real-time characteristics of the EDF method. These real-time disk scheduling algorithms start from an EDF schedule and then reschedule requests so as to reduce seek and/or rotational latency overhead under the real-time constraints [20, 21, 22].

In [23], the authors proposed the Earliest Deadline SCAN (D-SCAN) that uses the location of the task with the earliest deadline to determine the scan direction. While moving the disk head to the track of the task that has the earliest deadline, the requests whose access data are along the path are also served. Chen *et al.* [24] proposed the Shortest Seek Earliest Deadline by Order (SSEDO) and Shortest Seek Earliest Deadline by Value (SSEDV). Both algorithms start from an EDF schedule and then SSEDO uses a weighted seek distance for rescheduling, while SSEDV checks whether deadlines are feasible prior to service.

In 1993, the well-known SCAN-EDF scheme was proposed that first schedules disk tasks with the earliest deadlines [21, 22]. If two or more disk tasks have the same deadline, these tasks are served according to their relative track locations, i.e. by the SCAN algorithm. Since only tasks with the same deadline are seek-optimized, the obtained data throughput improvement is limited.

To increase the probability of applying the SCAN algorithm to reschedule input tasks, DM-SCAN (Deadline Modification-SCAN) proposed the concept of a maximum-scannable-group (MSG) [9]. An MSG is a set of continuous tasks that can be rescheduled by SCAN without missing their respective timing constraints. Given an EDF schedule $T = T_1, T_2 \cdots T_n$, an MSG G_i starting from task T_i is defined as the sequential tasks $G_i = T_i T_{i+1} T_{i+2} \cdots T_{i+m}$ where task T_j satisfies the following criteria:

$$f_j \le d_i \quad \text{and} \quad r_j \le s_i \quad \text{for } j = i \text{ to } i+m \qquad (3)$$

A simple example to demonstrate the identification of MSGs is shown in Figure 2. Given an EDF schedule $T = T_1 T_2 T_3 T_4 T_5$, to calculate MSG G_2, we have $f_2 \le d_2, r_2 \le s_2$ and $f_3 \le d_2, r_3 \le s_2$, but $f_4 > d_2$ although $r_4 \le s_2$. Thus, $G_2 = T_2 T_3$. Following the same approach, other MSGs can be obtained as $G_1 = T_1$, $G_3 = T_3 T_4$, $G_4 = T_4 T_5$ and $G_5 = T_5$ respectively.

After the identification of MSGs, DM-SCAN reschedules tasks in each MSG by the seek-optimizing SCAN scheme to minimize service time. Note that the rescheduled result will destroy the EDF sequence. Because DM-SCAN requires the input tasks based on an EDF order, a deadline modification scheme is proposed to modify the tasks' deadlines and transfers the rescheduled non-EDF sequence into a pseudo EDF order. Here, 'pseudo' means that the tasks are ordered by the modified deadlines. For example, given the schedule sequence $T_i T_j$, a pseudo deadline $d_{s(i)}$is derived as $d_{s(i)} = \min\{d_i, d_{s(j)}\}$. Figure 3 presents a simple example to illustrate the deadline modification scheme. The original input $\boldsymbol{T} = T_1 T_2 T_3 T_4 T_5$ is not an EDF schedule because we have $d_2 > d_3$ and $d_4 > d_5$. Traversing from the last task T_5 to the first task T_1, if any task has its deadline larger than that of its previous task, the deadline modification scheme is applied.

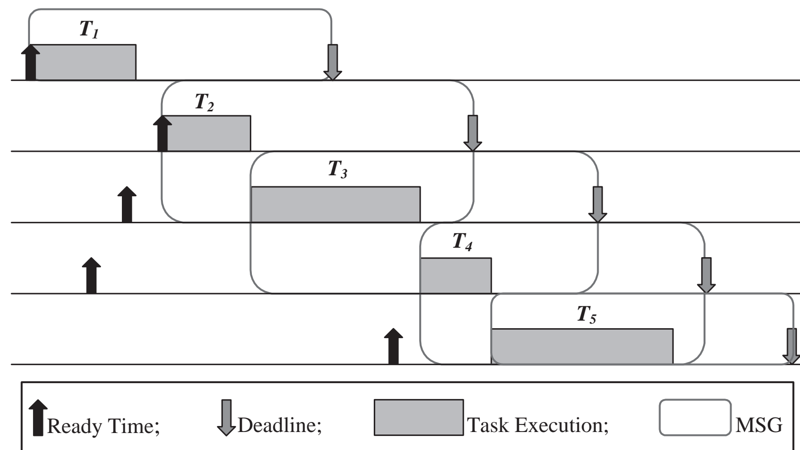

FIGURE 2. An example to demonstrate the identification of MSG.

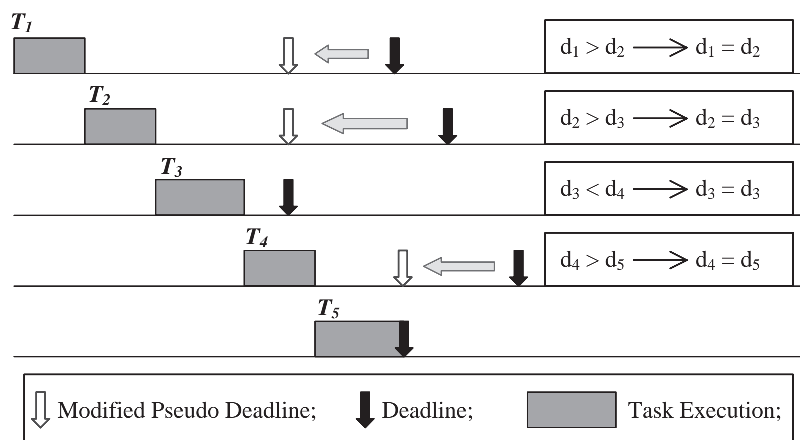

FIGURE 3. A simple example to illustrate the deadline modification scheme.

For example, d_4 is larger than d_5 and is modified equal to d_5 in order to satisfy the EDF requirement. Following the same procedure, d_2 and d_1 are also modified. Note that, although $d_1 < d_2$ in the original input schedule, d_1 is also modified as the value of d_1 is larger than that of the modified pseudo deadline d_2. By the deadline modification scheme, DM-SCAN reschedules tasks iteratively from the derived pseudo EDF schedule to obtain more data throughput.

3.2. Mixed-workload disk scheduling algorithms

Some researchers address the simultaneous support of mixed-media disk scheduling [25, 26, 27]. In [27], Cello is a two-level disk scheduling framework that consists of a class-independent scheduler and a set of class-specific schedulers. Therefore, a number of class-dependent disk schedulers schedule requests according to the application's needs. For example, in Cello, the soft real-time disk requests are scheduled by SCAN-EDF and the interactive best-effort disk requests are scheduled by classic slack stealing techniques [28]. Then the class-independent scheduler determines when and how many requests are moved from the class-dependent queues into the class-independent queue, which employs a First Come First Serve (FCFS) queuing discipline. For example, the class-independent scheduler employs a just-in-time scheduler (which schedules requests just prior to their deadlines) that moves the real-time disk requests to the class-independent queue at their latest start time.

However, the Cello disk scheduling framework could suffer from missing deadlines because of its just-in-time approach. Thus, in [25], the authors proposed the ΔL scheduler that, instead of using the just-in-time manner, extends Jeffay *et al.*'s [29] non-preemptive resource scheduler and uses the slack time (the minimum time between the end of any executed real-time request and its deadline) to service the best-effort requests. Notably, Jeffay *et al.*'s [29] non-preemptive schedule is a deadline-dynamic scheduling algorithm and ΔL employs the EDF scheduler to schedule the real-time requests.

Cello and ΔL use SCAN-EDF and EDF separately as their real-time disk scheduling algorithms. Nevertheless, taking only deadlines into account during disk scheduling without considering the seek-time latency, EDF incurs excessive seek-time costs and results in poor disk throughput

[21, 22]. Furthermore, in the performance evaluation of the ΔLscheduler, they found that the disk cache is of limited use because of its small size: the advantages of disk read-ahead for one stream are nullified by the transfer of other streams. This is because they read and write blocks of 1 MB each for a high bandwidth application, e.g. video stream playback. However, in this paper, we address other real-time applications, as stated in Section 1, which use small block sizes. Furthermore, the issues concerning the number of streams and on-disk cache will be addressed in Section 6.2.3.

3.3. Cache-aware CPU scheduling algorithms

Taking the cache effect into consideration during the scheduling has been done in CPU scheduling. To analyze the schedulability of a given CPU task set, the estimation of a task's execution is simplified at the cost of a number of worst case assumptions. In order to obtain more accurate timing for schedulability analysis, Lee *et al*. [30, 31] analyzed the cache-related preemption delays of tasks in the context of fixed-priority preemptive scheduling. An enhanced technique for analyzing the cache-related preemption delay is proposed in [32].

With the knowledge of the number of useful cache blocks, a scheduling scheme, called limited preemptive scheduling (LPS), which limits preemptions to occur at time points with the smallest cache-related preemption costs is introduced in [33]. Therefore, the overall task switching costs are reduced. However, this results in an increase in block delay of higher priority tasks; thus LPS makes an optimal trade-off in decreasing task switching costs and increasing blocking latency.

The cache miss can be classified as intrinsic and extrinsic. Extrinsic misses are caused by the interaction of different tasks on the cache; i.e., is due to preemptions in multi-processes systems. Intrinsic cache misses are caused by the contention on cache blocks of different codes in a program. In [30, 31, 32], only the extrinsic cache behavior is addressed. In reality, the cache misses caused by context switches (extrinsic cache miss) influence the cache miss probability during the internal execution of a program (intrinsic cache miss). Therefore, an integrated analysis of the intrinsic and extrinsic cache misses is proposed in [34]. The authors proposed a CPU scheduling methodology that integrates scheduling and timing analyses, and takes the caching effect into consideration. By offline analyzing the control flow graph (CFG) of a task, they derived a schedule by using a simulated annealing algorithm to minimize the cache miss probability.

From the discussion above, the cache-aware CPU scheduling either extends the previous classical schedulability analysis of EDF to include the cache effect or analyzes the CFG to determine the schedule that minimizes the cache miss probability. However, for disk scheduling, no optimal scheduling scheme (like EDF for CUP scheduling) is available. In addition, it is difficult to analyze the CFG since the interference of file system's buffer cache. Furthermore, because of the different characteristics of CPU and disk mechanisms, new on-disk cache-aware real-time disk scheduling algorithms must be devised.

3.4. Applications-aware buffer cache management

To make efficient use of system resources, in [35] the authors present proactive mechanisms to tailor file system resources to the needs of I/O-intensive applications. The user applications would provide 'hints' and the file system would then use these hints to prefetch and cache data aggressively into the buffer cache. Therefore, the file system resources would be tailored to the needs of user applications. However, this requires the modification of user application programs to disclose their future resource requirement to the operating system. Besides, if the idea is to be applied to the disk subsystem, it also requires the knowledge of the detailed behavior of the on-disk cache. Then, we know how to manage the on-disk cache on the basis of the disclosed hints. Furthermore, the disk subsystem must modify its interface with the host computer to distinguish the actual data requests from the prefetching and caching hinted data requests. In contrast, by our proposed scheme, we do not require the modification of user applications and the disk subsystem interface.

4. PRELIMINARIES

In this section, we describe the terms used in this paper. Given a set of n real-time disk tasks, assume that for each disk access T_i, $1 \leq i \leq n$, if a cache miss occurs, the cache logic will bring a data size of x into the on-disk cache and the content of data blocks brought into the cache is denoted by E_i. Thus, $\text{size}(E_i) = x$. Note that, the value of x depends on the cache segment size, and if read-ahead is performed, also on the read-ahead size. To distinguish a set of tasks whose accesses exhibit spatial locality, we define the principal task and the cached task as follows.

DEFINITION 2. *Principal task and cached task. Given a set of real-time disk tasks $T_1T_2 \cdots T_n$, if T_j's requested data block b_j is included in E_i, where $1 \leq i \leq j \leq n$, then T_i is called the principal task of T_j and denoted as $P(T_j) = T_i$. In addition, T_j is called the cached task of T_i and denoted as $C(T_i) = T_j$.*

DEFINITION 3. *Immediate principal task and immediate cached task. Given a set of real-time disk tasks $T_1T_2 \cdots T_n$, assume that $P(T_j) = T_i$ (i.e. $C(T_i) = T_j$), where $1 \leq i < j \leq n$. If there exist no principal tasks of T_j (or T_i's cached tasks) between T_i and T_j, then task T_i is called the immediate principal task of T_j and denoted as $G(T_j) = T_i$. In addition, task T_j is called the immediate cached task of T_i and denoted as $H(T_i) = T_j$.*

Therefore, T_j is a cache hit if $E_{G(T_j)}$ is resident in the on-disk cache when T_j is issued. In other words, a cache hit occurs for T_j if the cached data of $G(T_j)$ remains in the cache, i.e. has not yet been replaced when T_j is issued. Consequently, if T_j and $G(T_j)$ are scheduled close enough such that the cached data of $G(T_j)$ have not yet been flushed when T_j is issued,

FIGURE 4. The execution of T_k must be between T_i and T_j.

then T_j can be serviced by the on-disk cache and shorten its access time.

However, in a real-time system, a derived schedule must be feasible. Therefore, scheduling T_j and $G(T_j)$ to be closer must not violate both T_j and $G(T_j)$'s timing constraints. In addition, since other tasks may be influenced by this cache-aware rescheduling, the deadlines of the influenced tasks should not be missed to guarantee a feasible schedule. Therefore, when and how to perform such a cache-aware scheduling scheme under real-time constraints poses a challenge in the design of our scheduling algorithm.

5. CACHE-AWARE REAL-TIME DISK SCHEDULING ALGORITHMS

On the basis of DM-SCAN, we introduce in this section the three proposed cache-aware real-time disk scheduling algorithms. In Section 5.1, we introduce the deadline shift scheme. Then, the collaboration scheme and the CARDS scheme are presented in Section 5.2 and 5.3 respectively.

5.1. Deadline shift algorithm

As described in Section 4, to increase the cache hit ratio, T_i and $G(T_i)$ must be close enough to prevent $E_{G(Ti)}$ from being replaced when T_i is executed. If the input tasks are ordered in an EDF sequence, then a task's deadline determines its relative positions in a schedule. Accordingly, making T_i and $G(T_i)$ closer to one another such that T_i can be a cache hit implies that their deadlines should not be far away. This motivates the idea of a deadline shift scheme.

5.1.1. Deadline shift algorithm

Given an EDF schedule, the deadline shift scheme first identifies pairs of the cached tasks and their immediate principal tasks. Then, for each pair of the cached task $T_i, i \in [1, n]$, and its immediate principal task $G(T_i)$, the deadline of T_i is shifted to be close to that of $G(T_i)$. As a result, the number of tasks between T_i and $G(T_i)$ is decreased and then T_i has an opportunity to be a cache hit. After this deadline shifting process, tasks with their shifted deadlines are scheduled by DM-SCAN to minimize seek-time overhead under real-time constraints.

However, in some cases, it is impossible for a cached task to be a cache hit because it is too far away from its corresponding immediate principal task. Therefore, shifting these cached tasks' deadlines is unnecessary. To identify these cached tasks, we define the concept of task dimension.

DEFINITION 4. *Task dimension. Given a schedule $T_1 T_2 \cdots T_n$, the task dimension between T_i and T_j, $1 \le i < j \le n$, which is denoted as $D_{i,j}$, is defined as the number of non-cached tasks that must be executed after T_i and before T_j. Notably, a non-cached task means a task that has no corresponding principal task, i.e. $P(T_k) = \Psi$ if T_k is a non-cached task. Consequently, the access of a non-cached task always results in a cache miss.*

As shown in Figure 4, for task T_k that must be executed after T_i and before T_j, its ready time has to be after d_i and its deadline must be before r_j. Let $A_{i,j}$ denote the set of tasks that must be executed after T_i and before T_j.Thus, $A_{i,j}$ is computed as

$$A_{i,j} = \{T_k \mid d_i < r_k < d_k < r_j \quad \text{and} \quad P(T_k) = \Psi; 1 \le i < j < k \le n\} \tag{4}$$

From the definition of task dimension, $D_{i,j} = |A_{i,j}|$ if for all $T_k \in A_{i,j}$, T_k is a non-cached task. Suppose that the number of cache segments is m and LRU is used as the cache replacement algorithm. The following lemma identifies the cached tasks that have no way of being cache hits and thus have no need to shift their deadlines.

LEMMA 1. *Given an EDF schedule $S = T_1 T_2 \cdots T_n$, for any cached task T_j and its associated immediate principal task T_i, $1 \le i \le j \le n$, if $D_{i,j} \ge m$, then the access of T_j always results in a cache miss.*

Proof. A cache segment is replaced when a cache miss occurs, i.e. the requested data of an issued task is not in the cache. Since the task dimension between T_i and T_j is $D_{i,j}$, at least $D_{i,j}$ cache replacements will occur between T_i and T_j. Because $D_{i,j} > m$ and LRU is used as replacement scheme, the cached data $E_{T(i)}$ must be flushed out from the on-disk cache when T_j is issued. As a result, the access of T_j results in a cache miss and, therefore, it is not necessary to shift T_j's deadline. □

After eliminating the cached tasks that have no way of being cache hits, we must determine the proper value of the shifted deadline. Here, 'proper' means that a schedule based on the shifted deadlines should maximize the hit ratio while still being feasible. Assume that T_j is a cached task and $G(T_j) = T_i$. Denote the value of the shifted deadline of the cached task T_j as x_j. Depending on the relative position of T_i and T_j, there are three situations wherein the deadline of T_j could be shifted.

(a) If $d_i \le r_j$, as shown in Figure 5a, then the value of T_j's shifted deadline must be between r_j and d_j, i.e. $r_j \le x_j \le d_j$. Therefore, the value of the shifted deadline can be chosen as $x_j = r_j + c_j$, where c_j is a parameter depending on the characteristics of the input task set and the properties of the on-disk cache. Note that, $0 < c_j \le d_j$.
(b) If $r_i \le r_j \le d_i$, as shown in Figure 5b, then the value of the shifted deadline of T_j must be between d_i and

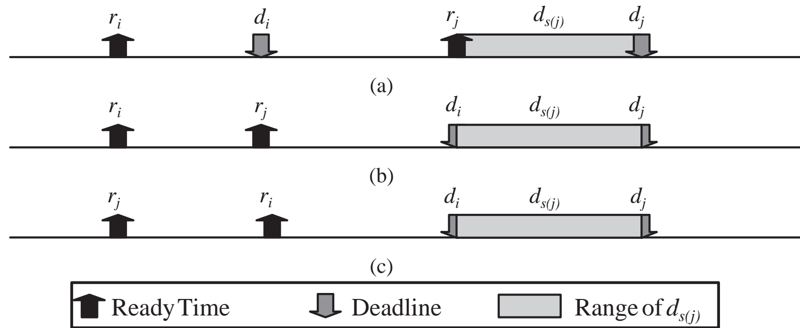

FIGURE 5. The three cases that a shifted deadline x_j could fall through.

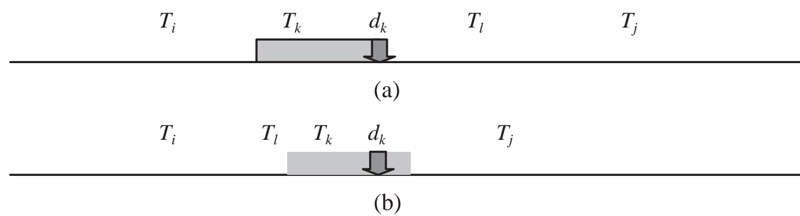

FIGURE 6. A simple example to illustrate that tasks could be influenced by a rescheduling of a task for hitting cache. (a) Original schedule. (b) To be cache hit, T_l is advanced to execute before T_k. However, T_k misses its deadline because of the advance of T_l's execution, even though T_l's access time is equal to the cache access time and is quite short.

d_j, i.e. $d_i \leq x_j \leq d_j$. Consequently, the value of the shifted deadline can be chosen as $x_j = d_i + c_j$. Note that, $0 \leq c_j \leq d_j - d_i$.

(c) Finally, if $r_j \leq r_i \leq d_i$, as shown in Figure 5c, then the value of the shifted deadline of task T_j must also fall between d_i and d_j, i.e. $d_i \leq x_j \leq d_j$, the same situation as Case (b).

After the deadline shifting process, tasks with their new shifted deadlines are scheduled by the DM-SCAN algorithm. Accordingly, the deadline shifting scheme modifies the tasks' deadlines to force a different execution order under the EDF schedule since EDF determines the tasks' execution order based on their deadlines.

5.1.2. Feasibility checking

Assume that the input schedule is a feasible schedule. Shifting a task's deadline will advance its execution and other tasks may be delayed and miss their deadlines. As shown in Figure 6, T_k misses its deadline due to T_l's advanced execution, even though the new access time of T_l is equal to the cache transfer time, which is significantly shorter than the physical disk access. Although DM-SCAN may produce a feasible rescheduled result even if an infeasible EDF schedule is given, this is not guaranteed.

Therefore, with the progress of the DM-SCAN algorithm, we have to check the schedule's feasibility. Once a task T_i misses its deadline, the shifted deadlines of the cached tasks before T_i are restored to their original values and these tasks must be put into their correct positions in the schedule, where correct positions means the locations scheduled by DM-SCAN using their original deadlines. Assume that the cached task T_j is before T_i and its deadline is shifted. If T_j has not yet been scheduled by DM-SCAN, we just restore T_j's deadline to its original value. If T_j has been scheduled by DM-SCAN, then T_j's deadline is first restored to its original value. In addition, DM-SCAN must reschedule the task set from T_j. This process continues until T_i meets its deadline.

5.1.3. Discussion

Although the deadline shift scheme considers the caching effect during disk scheduling to increase the cache hit ratio, it suffers from some drawbacks as discussed below.

- It is not easy to determine the value of a shifted deadline. A proper value of the shifted deadline, which depends on the characteristics of input tasks and the behavior of on-disk cache, must maximize the cache hit ratio while guaranteeing the schedule's feasibility. However, it is not easy to derive a suitable shifted deadline value for a cached task to meet the above criteria.
- The deadline shift scheme just tries to increase the cache hit probability, but does not guarantee it. By means of shifting T_j's deadline to be closer to the deadline of its immediate principal task T_i, the deadline shift scheme expects that T_j can then be served before the cached data of T_i has been replaced. However, it is not guaranteed that a cache hit occurs for T_j. In addition, the DM-SCAN algorithm, which runs after the deadline shifting process, will reschedule tasks without considering the caching effect. As a result, the shifted cached tasks may be pulled apart from their corresponding immediate principal tasks by DM-SCAN.

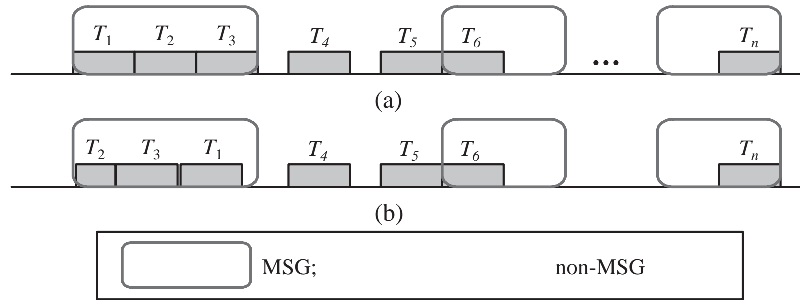

FIGURE 7. A simple example to demonstrate the DM-SCAN algorithm. (a) The input tasks are grouped into consecutive MSGs or non-MSGs. (b) Tasks in each MSG are rescheduled by the SCAN algorithm.

- The deadline shift scheme requires that the input tasks must be in EDF sequence. However, the input task set may not be in EDF order. Although a deadline modification scheme can be applied transferring a non-EDF ordered task set into a 'pseudo' EDF sequence, the modified pseudo deadlines have tighter deadlines than their original one to guarantee feasibility. Consequently, the modified schedule is stricter than the previous one. As a result, the schedulability of a given task set is influenced.

To resolve the above drawbacks of the deadline shift scheme, we propose two additional cache-aware real-time disk scheduling algorithms: the collaboration scheme and the CARDS scheme, which are described in turn in the following sections.

5.2. Collaboration scheme

To resolve the problems of the deadline shift scheme, the collaboration scheme is proposed, which considers the caching effect during the scheduling of DM-SCAN. However, rescheduling a task to increase the cache hit ratio may cause other tasks to miss their deadlines and a feasibility check must be performed for each rescheduling operation. Therefore, we also present the techniques to reduce the checking overhead.

5.2.1. *Collaboration scheme*

As stated in Section 3, DM-SCAN identifies MSGs that consist of a number of continuous tasks that can be rescheduled under real-time constraints (identification process). Continuous tasks that cannot be rescheduled seek-optimizingly belong to a non-MSG group. As a result, as shown in Figure 7a, an input task set is divided into a number of groups, either an MSG or a non-MSG group. After that, as shown in Figure 7b, tasks in each MSG are rescheduled by the SCAN algorithm (rescheduling process). Accordingly, the collaboration scheme considers the caching effect during both the identification process and the rescheduling process. Assume that LRU is used as the cache replacement scheme and a number of m cache segments are in the on-disk cache. Given a set of real-time disk tasks $\boldsymbol{T} = T_1 T_2 \cdots T_n$, for each group $G_i = T_i T_{i+1} \cdots T_{i+m}$, $1 \leq i \leq n$, the steps to perform the collaboration scheme are described in the following:

(i) If G_i is a MSG, tasks within G_i are first seek-optimized by the SCAN scheme.
(ii) Then, for each task T_l within $G_i, l \in [i, i+m]$, calculate its corresponding cached tasks, if they exist, after G_i. Remove the cached tasks that have no way of being a cache hit using the concept of task dimension, which is described in Section 5.1.1.
(iii) For each remaining cached task T_j, $j \in [i+m+1, n]$, assume that its principal task is $T_k, k \in [i, i+m]$. Depending on T_j's ready time r_j, the following shows the different steps to be performed.
    (a) If $r_j \leq r_l$, then T_j can be scheduled into G_i. Therefore, T_j is moved to the $(k+1)$th location to be immediately after T_k. As a result, T_j could hit the cached data of T_k when it is executed. However, feasibility checking must be performed to prevent other tasks from missing their deadlines as described in Section 5.2.2. Note that, T_j is not selected into G_i by DM-SCAN because $f_j > d_m$ in the input task set.
    (b) If $r_l < r_j < d_m$, T_j cannot be directly scheduled into G_i. To be a cache hit, principal task T_k is rescheduled, if feasible, such that $f_k \leq r_i$. Then T_j is rescheduled immediately after T_k.
    (c) This process is continued until all the remaining cached tasks are rescheduled.

The collaboration scheme thus directly adapts DM-SCAN to be cache-aware. First, the task set in an R-Group G_i is enlarged with the cached tasks, whose principal tasks are in G_i, that are guaranteed to be cache hits. Then, the rescheduling scheme also considers the caching effect. In addition to the movement of the cached tasks, principal tasks are also rescheduled to be closer to their cached tasks to increase the probability of cache hit.

5.2.2. *Feasibility checking*

As shown in Figure 8, when task T_i is rescheduled from position α to β, other tasks may be influenced by an increased or decreased delay of finish-time. Therefore, feasibility

FIGURE 8. The condition when a task T is moved from α to β. Tasks in the region A are not influenced. However, tasks in the region B may be delayed. Besides, tasks within the region C may be delayed or advanced for execution depending on whether T's access results in a cache hit or miss at location β.

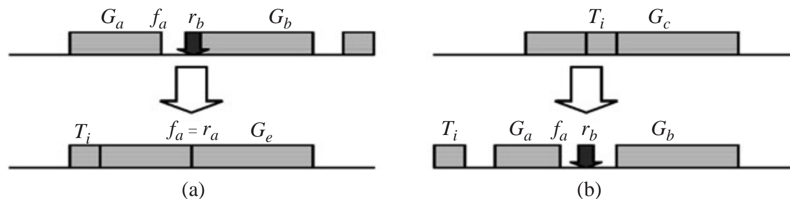

FIGURE 9. Conjunction groups may be merged or split when a rescheduling operation occurs. (a) T_i is rescheduled to the front of G_a. As a result, conjunction groups G_a and G_b are merged into G_c since $r_b \leq f_a$. (b) T_i is rescheduled out from G_c. As a result, conjunction group G_c is split into G_a and G_b since $f_a < r_b$.

checking must be performed when rescheduling a task and, if an infeasible schedule is produced, this rescheduling operation cannot be activated. The feasibility checking involves computing the start-time and finish-time for each request and thus a naive computation algorithm has $O(n)$ complexity. To accelerate the checking process, the concept of a conjunction group is introduced.

DEFINITION 5. *Conjunction group. Given a set of real-time disk tasks* $\boldsymbol{T} = T_1 T_2 \cdots T_n$, *a conjunction group* G_i *is defined as a number of continuous tasks* $G_i = T_i T_{i+1} \cdots T_{i+m}$ *where each task* T_k *for* $k = i+1$ *to* $i+m$ *satisfies* $r_k \leq f_{k-1}$.

Therefore, tasks in a conjunction group will be executed one by one without any free time slice between them. Note that, as shown in Figure 9, conjunction groups may be merged or split when a rescheduling operation is taking place. By the idea of conjunction group, the following lemmas assist in simplifying the checking process.

LEMMA 2. *Assume that we are given a conjunction group* $G_k = T_k T_{k+1} \cdots T_l$ *and task* T_m *is rescheduled from position* α *to* β, *where position* α *is inside* G_k *while position* β *is outside* G_k. *If* $T_i,\ i \in [k, l-1]$, *is influenced by a delayed execution of* ε, *then for all tasks* $T_j,\ j \in [i+1, l]$, *their executions are also delayed by* ε.

Proof. For a real-time task T_{i+1}, $s_{i+1} = \max\{r_{i+1}, f_i\}$ and $f_{i+1} = s_{i+1} + e_{i+1}$, where e_{i+1} denotes T_{i+1}'s execution time. Since $T_{i+1} \in G_k$, from the definition of conjunction groups,

$$s_{i+1} = f_i \quad \text{and} \quad f_{i+1} = s_{i+1} + e_{i+1} = f_i + e_{i+1}. \tag{5}$$

Because T_i is delayed by ε, i.e. f_i is increased by ε, from Equation (5), s_{i+1} and f_{i+1} are also delayed by ε. Following the same arguments, task $T_j,\ j \in [i+2, l]$, is also influenced by a delayed execution of ε. □

LEMMA 3. *Assume that we are given a conjunction group* $G_k = T_k T_{k+1} \cdots T_l$ *and a task* T_m *is rescheduled from position* α *to* β. *If* $T_i,\ i \in [k, l-1]$, *i.e.* T_i *is within* G_k, *is thus influenced by an advanced execution of* ε, *then for all tasks* $T_j,\ j \in [i+1, l]$, *their executions are also advanced by* ε, *if* G_k *is not split.*

Proof. The proof can be derived in the same way as the proof of Lemma 2. □

Given the set of tasks in a schedule, we define the slack l_i of task T_i as follows:

$$l_i = d_i - f_i. \tag{6}$$

That is, the slack l_i represents the duration for which T_i can be delayed without violating its deadline. As Lemmas 2 and 3 show, the increase/decrease in finish-time is the same for all tasks in a collaboration group. Accordingly, we only maintain the smallest value of slack for each collaboration group rather than maintaining it for individual requests. As a result, when a rescheduling operation is done, we only have to check the task with the smallest value of slack to see whether its deadline is missed, if a delayed execution occurs. Besides, the checking process is stopped when a free time slice is encountered. Note that, conjunction groups may be merged or split by a delayed or advanced execution, and thus the slack value should be updated correspondingly. From above, the overhead of feasibility checking is reduced significantly by the introduction of slack and conjunction groups. Therefore, the collaboration scheme can verify quickly whether a rescheduling is feasible or not.

5.2.3. Discussion

From Section 5.2.1, the collaboration scheme thus resolves the drawbacks of the deadline shift scheme. First, without the need to shift the tasks' deadlines, it is not necessary to decide the values of the shifted deadlines. Second, the collaboration scheme works for any input sequence and thus requires no deadline modification scheme. Third, the rescheduling of a cached task is only performed when a cache hit is guaranteed.

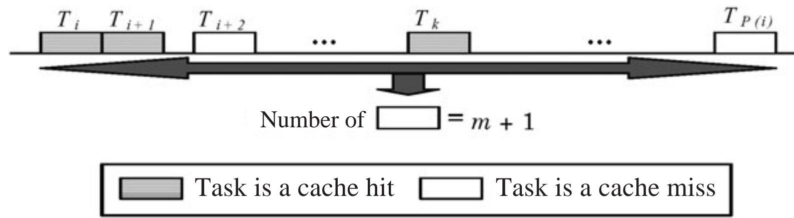

FIGURE 10. The identification of a flush point.

Finally, the schedulability of a given task set is not influenced since no task's deadline is shifted.

However, the collaboration scheme also suffers from some disadvantages. First, as presented in Section 5.2.1, when a cached task T_j has its ready time after the finish-time of its principal task T_k, i.e. $r_j > f_k$, the collaboration scheme does not perform cache-aware rescheduling for T_j or T_k. This is because the collaboration scheme works with the progress of the DM-SCAN scheme and thus the scheduling sequence between T_k and T_j is not yet determined when the group containing T_k is encountered. As a result, the collaboration scheme cannot perform cache-aware rescheduling for T_j or T_k. In addition, when T_j is rescheduled to hit the cached data of its principal task T_k, the collaboration scheme reschedules T_j to be immediately after T_k, which results in a maximum number of tasks being influenced (from T_{k+1} to T_n). However, since the cached data of T_k will remain in the on-disk cache until they are evicted, it is not necessary to reschedule T_j immediately after T_k, except in the case when only one cache segment exists in the on-disk cache. Besides, if T_j is also a principal task of T_l, this could result in a larger distance between T_j and T_l and increase the difficulty for T_l to be a cache hit. In the following section, the CARDS scheme is, therefore, introduced to rectify the above drawbacks.

5.3. CARDS scheme

As stated in Section 5.1, the deadline shift scheme first performs cache-aware scheduling to a given task set. Then, DM-SCAN is applied for seek-optimizing input tasks. Section 5.2 introduces the collaboration scheme that considers the caching effect during the scheduling of DM-SCAN. In this section, a new cache-aware real-time disk scheduling algorithm, the CARDS scheme, is proposed that considers the caching effect after the DM-SCAN scheme.

Before describing the CARDS scheme, for task T_k, we first introduce the miss function $g(k)$ as:

$$g(k) = \begin{cases} 1 & \text{if } T_k \text{ introduces a cache miss} \\ 0 & \text{if } T_k \text{ introduces a cache hit} \end{cases} \quad (7)$$

By the miss function, the concept of flush point of T_i, $P(i)$, is introduced such that

$$\sum_{l=i}^{P(i)} g(l) = m + 1 \quad \text{or} \quad P(i) = n \quad \text{if } n \text{ is reached} \quad (8)$$

As shown in Figure 10, $P(i)$ represents the position that the cached data of T_i will be flushed to the disk. As a result, T_j should be executed before $T_{P(i)}$, if possible, to be a cache hit.

In contrast to the collaboration scheme, which schedules a cached task T_j immediately after its principal task T_i, the CARDS scheme schedules T_j just immediately before the flush point of T_i, $P(i)$. Thus, a cache hit is also guaranteed for T_j while minimizing the number of tasks being influenced.

Suppose that the number of cache segments is m and LRU is used as the cache replacement algorithm. Assume that after the running of DM-SCAN, the derived schedule $S = T_1 T_2 \cdots T_n$. Then, the CARDS scheme identifies pairs of cached tasks and their immediate principal tasks. For each pair of cached task T_j, $j \in [1, n]$, and its immediate principal task $T_i (= G(T_j))$, the CARDS scheme must decide whether T_j should be scheduled to be closer to T_i and, if so, which position is suitable for T_j to be scheduled. The steps that are performed by the CARDS scheme for each pair of the cached task T_j and its immediate principal task T_i are shown in the following:

- (i) Calculate the value of $P(i)$ by Equations (7) and (8).
- (ii) If T_j is in front of $T_{P(i)}$, as shown in Figure 11a, T_j can be serviced by the on-disk cache with the cached data of T_i. Therefore, no rescheduling is needed for T_j.
- (iii) However, if T_j is after or equal to $T_{P(i)}$, i.e. $P(i) \leq j$, then a cache miss will occur when T_j is issued. Consequently, the CARDS scheme tries to schedule T_j to execute before $T_{P(i)}$. Depending on the values of r_j, the ready time of T_j, and $s_{P(i)}$, the start-time of $T_{P(i)}$, two different cases may exist:
  - (a) If $s_{P(i)} \leq r_j$, as shown in Figure 11b, T_j cannot be advanced to execute before $T_{P(i)}$ since its ready time falls behind the start-time of $T_{P(i)}$. Consequently, no reordering is performed for T_j.
  - (b) If $s_{P(i)} > r_j$, as shown in Figure 11c, T_j can be advanced to execute before $T_{P(i)}$. Although the time at which T_j could be started is between $\max(d_i, r_j)$ and $s_{P(i)}$, the CARDS scheme reschedules T_j into the $(P(i) - 1)$th position, i.e. immediately before $T_{P(i)}$. Accordingly, the number of influenced tasks (from $T_{P(i)}$ to T_n) is minimal. Note that the rescheduling of T_j may result in an infeasible schedule. Therefore, a feasibility checking must be performed for each rescheduling operation by the techniques described in Section 5.2.2.

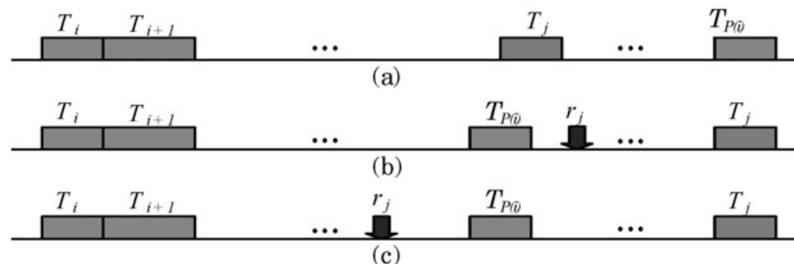

FIGURE 11. Three cases for the CARDS scheme. (a) T_j is guaranteed to be cache hit and thus no movement is needed as it is scheduled before the $T_{P(i)}$. (b) No movement is needed for T_j because its ready time is after the start time of $T_{P(i)}$. (c) By moving T_j in front of $T_{P(i)}$, T_j thus can be cache hit.

TABLE 1. Quantum Atlas 10K: MAG 3091 disk parameters.

Year	1999
Capacity	9.1 GB
Number of cylinders	10,042
Number of surface	6
Number of sectors per track	334
Sector size	512 bytes
Revolution speed	10,000 rpm

TABLE 2. Quantum Atlas 10K: MAG 3091 disk cache parameters.

Size	2 MB
Number of buffer segments	10
Segment size	374 sectors
Transfer time	0.184 ms

From the above, the increase in cache hit probability is realized with the CARDS scheme by rescheduling tasks that have the opportunity to be a cache hit after the DM-SCAN scheme.

6. EXPERIMENTAL RESULTS

In this section, the performances of the three cache-aware real-time disk scheduling algorithms are evaluated. Section 6.1 shows the platform used for our experiments and the characteristics of the input workload. In Section 6.2, the experimental results of the three cache-aware real-time disk scheduling algorithms are presented to compare their performance.

6.1. Experiment platform

As stated above, the characteristics of the on-disk cache must be explored so that a cache-aware scheduling scheme can be applied. Because disk manufacturers consider their on-disk cache implementation scheme a technical secret, we use the disk drive parameters derived from [36], which uses the techniques of on-line extraction [37, 38, 39]. Table 1 shows some important parameters of the Quantum Altas 10K MAG 3091, which is used as the target disk in our experiments [36, 40]. The seek-time cost is calculated by the extracted data from [36]. Rotational latency is assumed to be half of the time of a full track revolution. The on-disk cache parameters of Quantum Altas 10K MAG 3091, which are based on the extracted data of [36], are shown in Table 2.

There are two kinds of workloads in our experiments, one is random and the other is sequential. The workload of random tasks is uniformly distributed over the disk surface. The sequential workload comprises a number of sequential streams and random requests. Each sequential stream in our simulations emulates the sequential access pattern. The accessed block of the first request is also randomly distributed over the disk surface. Then, the following requests access the block immediately after their previous tasks. In addition, the number of random requests in a sequential workload is selected as one-third of the total requests. The accessed blocks of these random tasks are also uniformly distributed over the disk surface. The size of data accessed by each request, either sequential or random, is normally distributed with a mean of 36 KB. For a random workload, if there are n random tasks, the ready times of tasks are randomly generated from 0 to $6 * n$ ms. After a random time interval, 0–$5 * n$ ms, the related deadlines are uniformly distributed within 0–$10 * n$ ms. For a sequential workload, if there are m sequential streams, the total number of input tasks $n = 1.5 * (5 * m)$. Since there are five sequential tasks in a stream, the ready time of each sequential task in a stream is randomly generated between 0 and $2 * n/5$ ms after its previous task and its deadline is uniformly distributed within 0–$20 * n/5$ ms after a random time interval, 0–$10 * n/5$ ms. For the random tasks in the sequential workload, their ready times are randomly generated between 0 and $2 * n$ ms. After a random time interval, 0–$10 * n$ ms, their related deadlines are uniformly distributed within 0–$20 * n$ ms. The cache replacement scheme is assumed to be LRU. If a cache miss occurs, the cache logic will read ahead a data size of 354 sectors (177 KB), including the requested one, into a least-recently used cache segment. In all the following experiments, 50 experiments are conducted with different seeds for random number generation and the average value is measured.

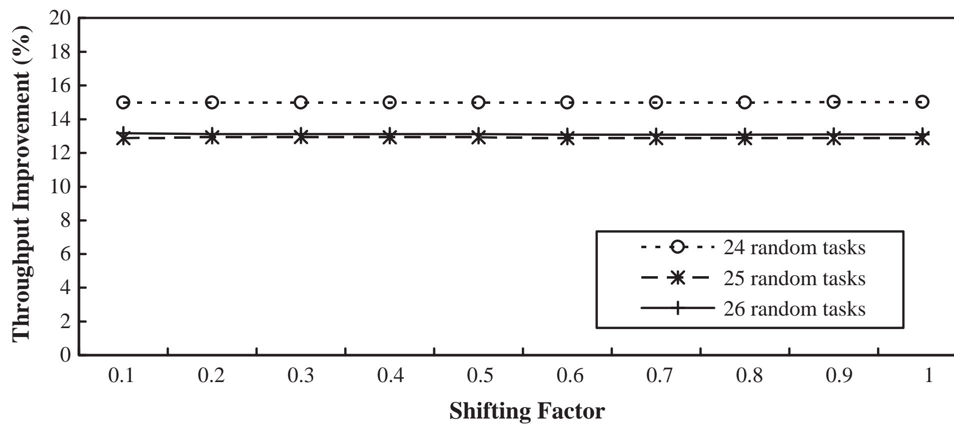

FIGURE 12. Throughput improvement for different values of shifting factors in random workload. The throughput improvement is compared with EDF.

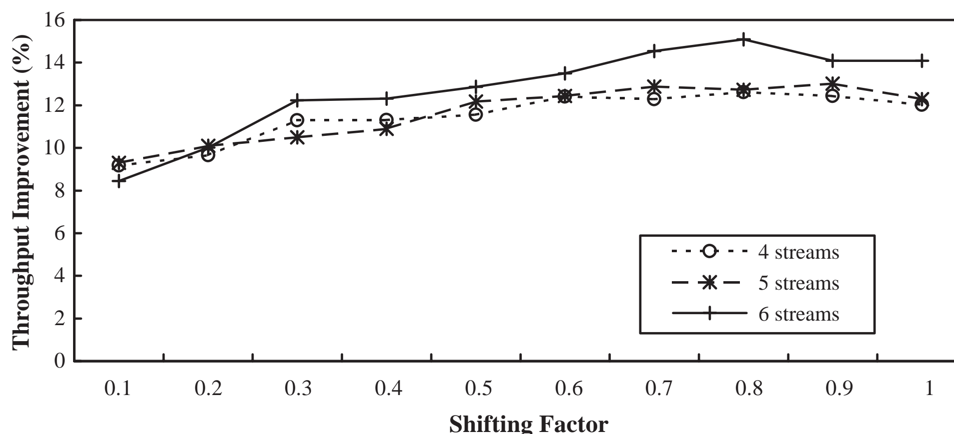

FIGURE 13. Throughput improvement for different values of shifting factors in sequential workload. The throughput improvement is compared with EDF.

6.2. Experimental results

6.2.1. Different values of the shifted deadlines

In this section, we first measure the data throughput under the different shifted values of the deadlines in the deadline shift scheme. Assume that $T_j$is a cached task and $G(T_j) = T_i$. Depending on the relative position of T_i and T_j in the input task set, T_j's deadline would be shifted such that $r_j \leq x_j \leq d_j$ or $d_i \leq x_j \leq d_j$. Thus, x_j would be selected as $x_j = r_j + c_j$ or $x_j = d_i + c_j$, where c_j is a parameter that depends on the characteristics of the input task set and the properties of the on-disk cache. We thus define a shifting factor w_j for T_j such that $c_j = (d_j - r_j) * w_j$ or $c_j = (d_j - d_i) * w_j$, $w_j \in [0, 1]$. Note that, x_j is increased with w_j, i.e. a larger value of the shifting factor results in a smaller value of the shifted deadline, which in turn incurs a smaller difference between the original deadline, d_j, and the shifted deadline, x_j.

Given a number of random tasks, Figure 12 plots the data throughput improvement compared with EDF for different values of shifting factors. It is observed that the value of the shifting factor, and thus the shifted deadline, has little impact on the obtained data throughput. This is because the input workload is random. Hence, the caching effect is insignificant since there is little possibility that the cached data of a task will be reused by another task. In contrast, Figure 13 demonstrates the same experiment under different sequential workloads. For example, the line of four sequential streams represents the workload that consists of four sequential streams and 10 random requests. Observe that, when the shifting factor is 0.9, the data throughput is best under four and five sequential streams. Thus, in the following experiments, $w_i = 0.9$ is used as the shifting factor in calculating the shifted deadlines for the deadline shift scheme.

6.2.2. Data throughput improvement

If the same number of real-time tasks is given, a well-behaved scheduling algorithm must maximize data throughput under guaranteed real-time constraints. Given random access workload, the data throughput improvements of DM-SCAN and three other disk scheduling schemes for different numbers of input tasks are shown in Figure 14. The

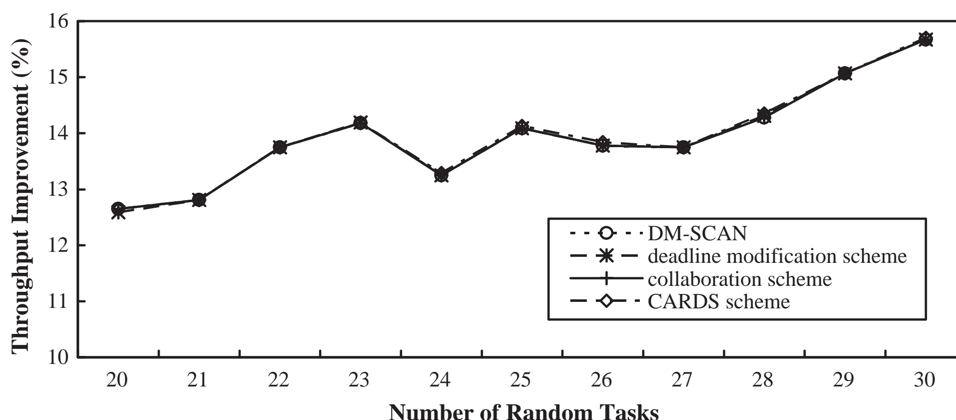

FIGURE 14. Throughput improvement of different schemes under different numbers of random tasks. The throughput improvement is compared with EDF.

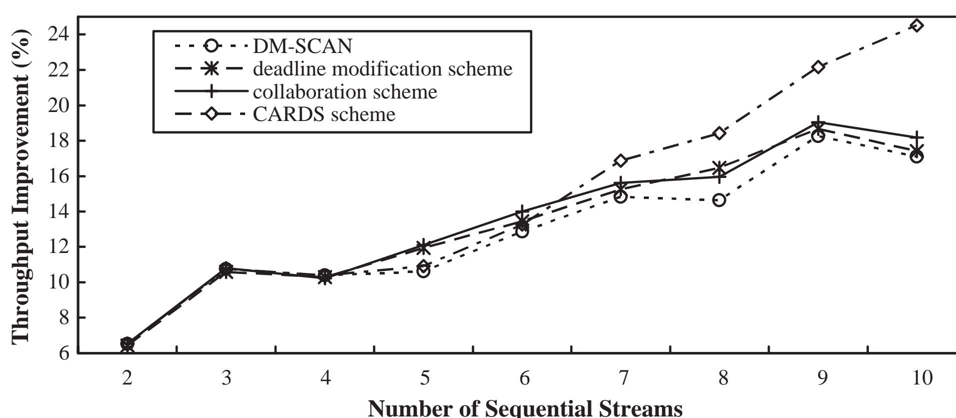

FIGURE 15. Throughput improvement of different schemes for sequential workload with different number of sequential streams. The throughput improvement is compared with EDF.

TABLE 3. Given 26 random tasks, the minimum, maximum, average schedule fulfill-time and throughput improvement compared with EDF for different schemes.

Algorithms	Schedule fulfill-time (ms)			
	Minimum	Maximum	Average	Improvement (%)
EDF	262.10	355.86	309.74	0.0
DM-SCAN	230.48	314.60	267.05	15.66
Deadline shift scheme	230.48	314.60	267.05	15.66
Collaboration scheme	230.48	314.60	267.05	15.68
CARDS scheme	230.48	314.60	266.88	15.69

TABLE 4. Under sequential workload with 10 sequential streams, the minimum, maximum, average schedule fulfill-time and throughput improvement compared with EDF for different schemes.

Algorithms	Schedule fulfill-time (ms)			
	Minimum	Maximum	Average	Improvement (%)
EDF	453.31	543.15	498.51	0.0
DM-SCAN	376.99	473.61	413.24	17.11
Deadline shift scheme	359.16	453.51	411.72	17.41
Collaboration scheme	373.39	461.79	407.81	18.19
CARDS scheme	327.04	442.02	376.32	24.51

derived throughput improvement is compared with EDF. Figure 15 presents the same experiment for different sequential workloads. The minimum, maximum and average schedule fulfill times of various approaches with a sequence of 25 random tasks are also presented in Table 3. Table 4 presents the same performance metrics but under a sequential workload with 10 streams. Note that the performance improvement is obtained because of the cache-aware rescheduling scheme, which leads to an increase in the cache hit ratio. It does not include the advantage when compared with systems that make worst-case assumptions about disk access time. As stated in Section 1.1, this would only influence the schedulability analysis.

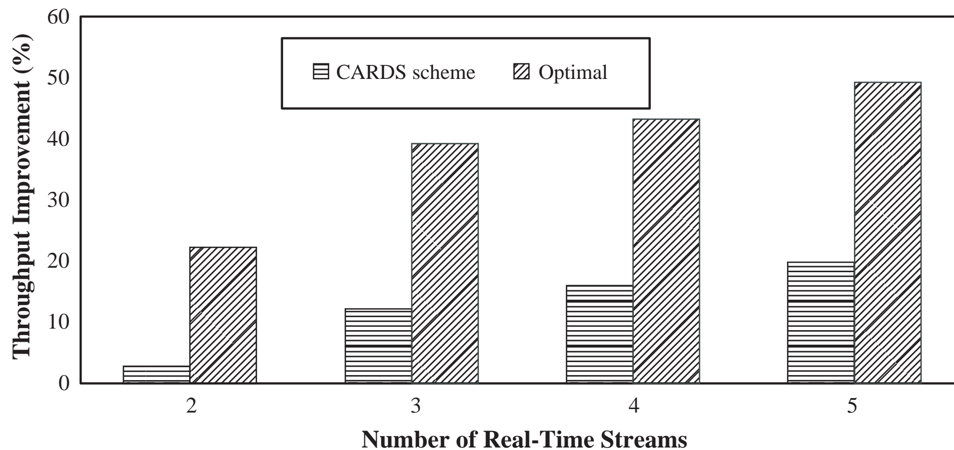

FIGURE 16. Throughput improvement of CARDS and optimal schemes for sequential workload with different numbers of sequential streams. The throughput improvement is compared with EDF.

On-disk caches work on the premise that the input workload follows the principles of temporal and spatial locality. Thus, given random tasks, the throughput improvements presented in Figure 14 show little difference between the proposed cache-aware scheduling schemes and DM-SCAN. There is little possibility that a random task will hit the data cached in the on-disk cache. Therefore, cache-aware scheduling has no means of increasing the cache hit probability.

In contrast, as shown in Figure 15, if input is sequential workload, both the collaboration scheme and the CARDS scheme obtain larger data throughput than DM-SCAN. However, in some ways, the data throughput of the deadline shift scheme is worse than that of DM-SCAN. This is because the deadline shift scheme suffers the drawbacks described in Section 5.1.3.

Thus, the performance of the deadline shift scheme is undetermined. Observe that the performance of the CARDS scheme is better than DM-SCAN with an increase in the number of sequential streams. Since the number of cache segments is 10, when the number of cache segments is considerably larger than that of the sequential streams, the on-disk cache capacity is large enough to sustain a great deal of blocks accessed by each sequential task. Thus, the derived throughput difference between DM-SCAN and the CARDS scheme is not significant. However, when the number of sequential streams is increased, the CARDS scheme can increase the on-disk cache utilization and obtain larger data throughput than DM-SCAN.

Furthermore, in Figure 16, we show the throughput improvement of CARDS compared with that of the optimal scheme. Because of the exponential time complexity of calculating the optimal solution, we only show the throughput performance under five sequential streams. From Figure 16, we see that the optimal solution outperforms the proposed CARDS scheme and obtains nearly two times the throughput of the CARDS approach.

To prove that taking into consideration the on-disk cache during disk scheduling indeed increases the cache hit ratio,

TABLE 5. Under sequential workload, the minimum, maximum, and average cache hit ratio.

No. of streams	Algorithm	Min. (%)	Max. (%)	Average (%)
6	DM-SCAN	15	22	19.0
	CARDS	16	24	19.5
7	DM-SCAN	16	26	20.0
	CARDS	17	28	22.0
8	DM-SCAN	16	27	21.9
	CARDS	20	29	23.8
9	DM-CAN	15	32	23.1
	CARDS	19	33	26.9
10	DM-CAN	18	32	22.9
	CARDS	23	34	29.2

Table 5 shows the minimum, maximum and average cache hit ratio under a sequential workload. As shown in Table 5, the CARDS scheme has a better cache hit ratio than the DM-SCAN. Because of the increased cache hit ratio in CARDS, the schedule fulfill-time of CARDS is shorter than that of DM-SCAN. Therefore, as shown in Figure 15, the CARDS scheme obtains larger data throughput than DM-SCAN.

6.2.3. *Throughput improvement versus number of cache segments*

In modern on-disk cache design technology, the number of cache segments is configurable. Thus, we conducted an experiment in which the performance of the three proposed cache-aware real-time disk scheduling schemes is measured for different numbers of cache segments. Given 25 random tasks, Figure 17 plots the throughput improvement compared with EDF for different numbers of cached segments. The figure shows that these three schemes yield almost the same throughput improvement. In addition, for each scheme, the throughput improvement is fixed in any number of cache segments. This is because the input tasks are randomly accessed. Therefore, the caching effect is negligible and, no

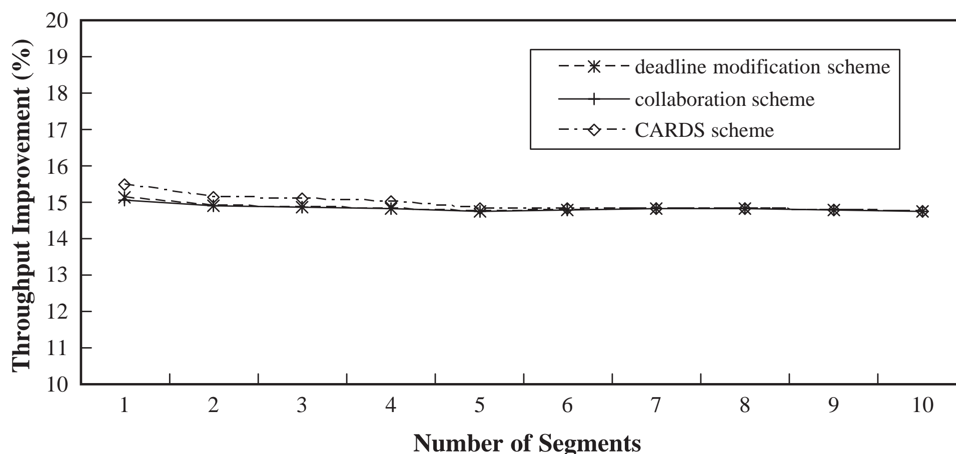

FIGURE 17. Given 25 random tasks, the throughput improvement of three schemes under different cache segment number. The throughput improvement is compared with EDF.

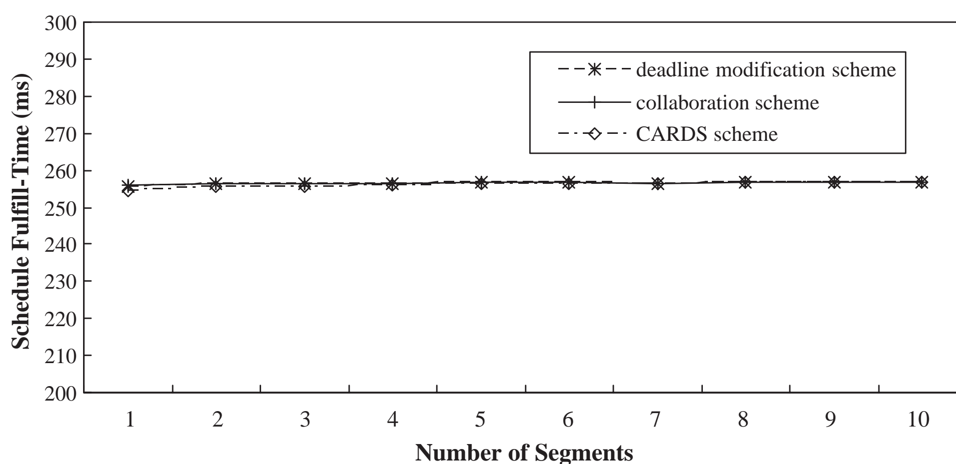

FIGURE 18. Given 25 random tasks, the schedule fulfill time of three schemes under different cache segment number.

matter how many cache segments are provided, most accesses result in physical disk mechanism operations. The schedule fulfill-times of the three schemes for different numbers of cache segments are also presented in Figure 18. The figure shows the same phenomenon as Figure 17.

Figure 19 presents the throughput improvement compared with EDF for different numbers of cache segments under sequential workload with five sequential streams. Observe that, the lines representing the three schemes' throughput improvement are flat when the number of cache segments is larger than seven. This is because there are five sequential streams in the input workload and the requested data size of each task (smaller than 36 KB) is smaller than that of a cache segment (larger than 167 KB). Thus, when the number of cache segments is larger than that of the sequential streams, cache replacement rarely occurs, even though a few random tasks are involved in the workload. Therefore, cache hits often occur for each sequential task, even without consideration of cache-aware scheduling. Figure 20, which plots the schedule fulfill-times for the workload in Figure 19, demonstrates that the schedule fulfill-times of the three schemes decrease with an increase in the number of cache segments. However, when the number of cache segments is larger than seven, the schedule fulfill-times of the three schemes are stable and almost the same.

Thus, when the number of cache segments matches that of the sequential streams, the on-disk cache behavior is aligned to the application's characteristics and obtains the largest data throughput. However, in a true system, the number of sequential streams is dynamic. Thus, it is difficult to determine a suitable number of cache segments. The proposed cache-aware scheduling schemes address this limitation by performing cache-aware scheduling to increase the cache hit probability.

7. DISCUSSIONS

Note that, in this paper, we assume that the on-disk cache replacement scheme is LRU. However, the three algorithms proposed in this paper can also be applied to other cache

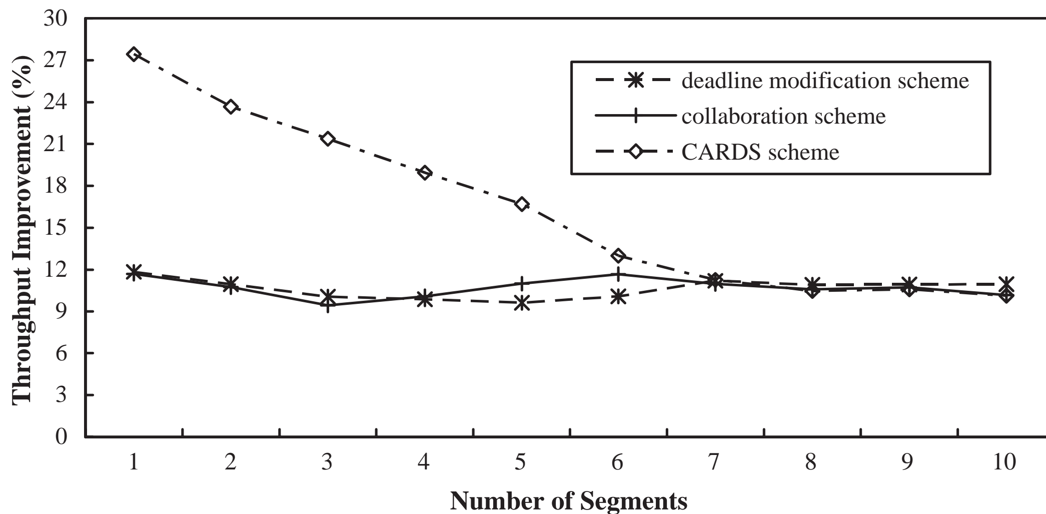

FIGURE 19. Under sequential workload with five sequential streams, the throughput improvement of three schemes under different cache segment number.

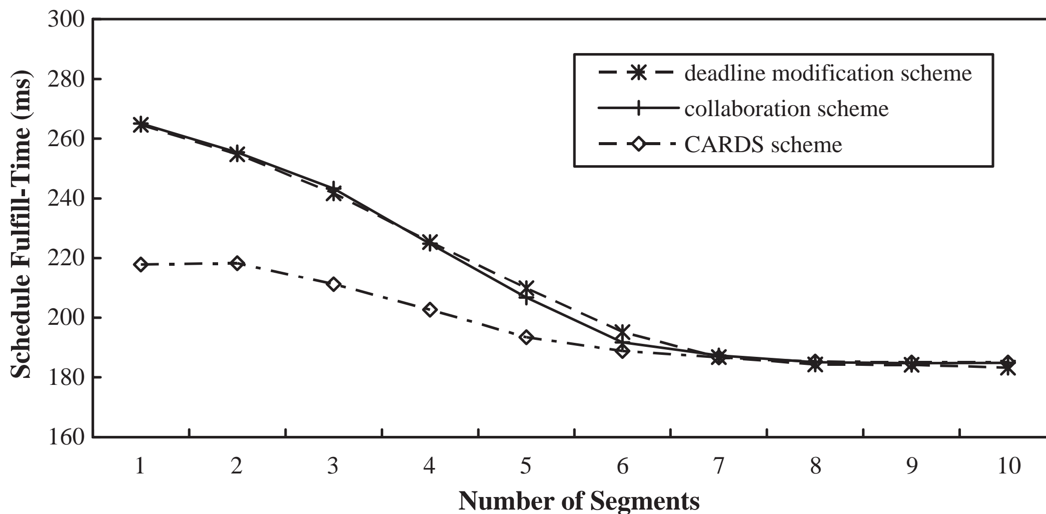

FIGURE 20. Under sequential workload with five sequential streams, the schedule fulfill-time of three schemes for different cache segment number.

replacement algorithms. As stated in Section 4, all our algorithms are based on the identification of the cached tasks and their immediate principal tasks. After that, each algorithm tries to reschedule pairs of the cached tasks and their immediate principal tasks close enough. As a result, once the cached tasks are issued, they will be cache hits since their requested data have been read ahead by their immediate principal tasks and have not yet been replaced.

Therefore, if another cache replacement scheme is adopted by disk drives, we only have to change the identification scheme of cached tasks and their immediate principal tasks. Besides, the identification scheme can be derived easily if we know the cache replacement scheme. For example, if the replacement scheme is changed from LRU to RR, we can easily trace the input disk tasks and identify each pair of the cached task and immediate principal tasks under the RR cache replacement scheme.

Furthermore, the performance of our proposed algorithms depends on how well the cache replacement scheme performs. If data cached in the on-disk cache are almost what the following disk tasks access, then our algorithms can obtain a higher performance improvement. In contrast, if the data item cached in the on-disk cache is not what the following disk tasks expect, since an ill-behaved cache replacement scheme evicts useful blocks from the on-disk cache, then our proposed algorithms will have no way of increasing the number of cache hits by cache-aware rescheduling. Fortunately, with an increase in on-disk cache size and an improvement in caching strategies, on-disk cache

and also our proposed scheme, benefit from such technology improvement.

8. CONCLUSIONS AND FUTURE WORK

8.1. Conclusions

To maintain their competitive edge in the market, disk manufacturers consider their disk implementation to be a technical secret. However, if the information of on-disk cache is explored, the disk scheduling can exploit this information to derive a schedule minimizing the cache miss probability. In this paper, we therefore propose cache-aware real-time disk scheduling algorithms that consider the caching effect during scheduling. As a result, the disk scheduling scheme can also be actively involved in reducing the cache miss ratio. In addition, the timing analysis is more accurate since the on-disk cache is considered during scheduling and thus, if a cache hit occurs, the cache transfer time is used as the task's execution time for schedulability analysis without assuming the worst case that each disk task incurs a physical disk mechanical operation. The experiments demonstrate that the proposed schemes indeed obtain larger data throughput than DM-SCAN. For example, under a sequential workload with 10 sequential streams, the data throughput of the CARDS scheme is 1.1 times that of DM-SCAN.

In addition, we investigate the influence of the number of cache segments on the performance of our proposed schemes. Experimental results show that the CARDS scheme can resolve the performance limitation of on-disk cache when the number of sequential streams is larger than that of cache segments.

8.2. Future work

The cache-aware real-time disk scheduling algorithms proposed in the paper are based on the static manner of an on-disk cache; i.e. the scheduling scheme is aligned to the on-disk cache's behavior. However, in the recent design of on-disk cache, the number (and hence the size) of the cache segment can be configured. In addition, the read-ahead can be enabled or disabled dynamically. As a result, with a knowledge of the application's access patterns, our future work will propose a more aggressive cache-aware real-time disk scheduling scheme that will also change the behavior of on-disk cache dynamically during scheduling. For example, if only a few concurrent processes exist in a system at a time, the segment number can be decreased and thus more data can be cached for each process. As a result, the on-disk cache can be aligned to the application requirements and be utilized more efficiently by such an aggressive scheduling scheme. Furthermore, the interaction between the on-disk cache and buffer cache is also a possible future work. For example, if the information of an on-disk cache is exported to the file system, we can manage both the buffer cache and on-disk cache more efficiently. In addition, by differentiating between requests whether they are read-ahead or real requests, the cache-aware real-time disk scheduling algorithm can manage the disk bandwidth more efficiently under the feasibility constraints.

REFERENCES

[1] Jain, P. and Lam, S. S. (1991) Specification of real-time broadcast networks. *IEEE Trans. Comput.*, **40**(4), 404–422.

[2] Xuan, P. *et al*. (1997) Broadcast on demand: efficient and timely dissemination of data in mobile environment. In *Proc. IEEE Real-Time Technology and Applications Symp.*, Montreal, Que., Canada, June 9–11, pp. 38–48. IEEE Computer Society Press, Los Alamitos, USA.

[3] Ruemmler, C. and Wilkes, J. C. (1994) An introduction to disk drive modeling. *IEEE Comput.*, **27**(3), 17–28.

[4] Chang, H. P., Chang, R. I., Shih, W. K. and Chang, R. C. (2001) Scheduling I/O requests in a multimedia-on-demand application. *J. Appl. Syst. Studies*, **2**(3), 151–166.

[5] Hospodor, A. (1992) Hit ratio of caching disk buffers. In *Proc. IEEE Computer Society International Conf.*, San Francisco, CA, USA, February 24–28, pp. 427–432. IEEE Computer Society Press, Los Alamitos, USA.

[6] IBM Corporation (1998) *Larger Disk Cache Improves Performance of Data-Intensive Applications.* White Paper, October, 1998.

[7] Shriver, E., Merchant, A. and Wilkes, J. (1998) An analytic behavior model for disk drives with readahead caches and request reordering. In *Proc. ACM SIGMETRICS*, Madison, WI, USA, June 22–26, pp. 182–191. ACM Press, USA.

[8] Worthington, B. L., Ganger, G. R. and Patt, Y. N. (1994) Scheduling algorithms for modern disk drives. In *Proc. ACM SIGMETRICS*, Nashville, TN, USA, May 16–20, pp. 241–251. ACM Press, USA.

[9] Chang, R. I., Shih, W. K. and Chang, R. C. (1998) Deadline-Modification-SCAN with maximum-scannable-groups for multimedia real-time disk scheduling. In *Proc. 19th IEEE Real-Time Systems Symp.*, Madrid, Spain, December 2–4, pp. 40–49. IEEE Computer Society Press, Los Alamitos, USA.

[10] Ganger, G. R. *et al*. (1999) The DiskSim Simulation Environment Version 2.0 Reference Manual. http://www.pdl.cmu.edu/DiskSim/disksim2.0.html.

[11] Karedla, R., Love, J. S. and Wherry, B. G. (1994) Caching strategies to improve disk system performance. *IEEE Comput.*, **27**(3), 38–46.

[12] Jalics, P. J. and Mcintype, D. R. (1989) Caching and other disk access avoidance techniques on personal computers. *Commun. ACM*, **32**(2), 246–255.

[13] Smith, A. J. (1985) Disk cache-miss ratio analysis and design considerations. *ACM Trans. Comput. Syst.*, **3**(3), 161–203.

[14] Thiebaut, D., Stone. S. H. and Wolf, J. L. (1992) Improving disk cache hit-ratios through cache partitioning. *IEEE Trans. Comput.*, **41**(6), 665–676.

[15] Stankovic, J. A. and Buttazzo, G. C. (1995) Implications of classical scheduling results for real-time systems. *IEEE Comput.*, **28**(6), 16–25.

[16] Lin, T. H. and Tarng, W. (1991) Scheduling periodic and aperiodic tasks in hard real-time computing systems. In *Proc. ACM SIGMETRICS*, San Diego, CA, USA, May 21–24, pp. 31–38. ACM Press, USA.

[17] Liu, C. L. and Layland, J. W. (1973) Scheduling algorithms for multiprogramming in a hard real-time environment. *J. ACM*, **20**(1), 46–61.

[18] Wong, C. K. (1980) Minimizing expected head movement in one dimension and two dimensions mass storage system. *Comput. Surv.*, **12**(2), 167–178.

[19] Denning, P. L. (1967) Effects of scheduling on file memory operations. In *Proc. of AFIPS SJCC*, pp. 9–21.

[20] Chang, H. P., Chang, R. I., Shih, W. K. and Chang, R. C. (2001) Real-time disk scheduling for multimedia applications by Enlarged-Maximum-Scannable-Groups. *J. Appl. Syst. Studies*, **2**(3), 151–166.

[21] Reddy, A. L. N. and Wyllie, J. C. (1993) Disk scheduling in a multimedia I/O system. In *Proc. ACM Int. Conf. on Multimedia*, Anaheim, CA, USA, August 2–6, pp. 225–233. ACM Press, USA.

[22] Reddy, A. L. N. and Wyllie, J. C. (1994) I/O issues in a multimedia system. *IEEE Comput.*, **27**(3), 69–76.

[23] Abbott, R. K. and Garcia-Molina, H. (1990) Scheduling I/O requests with deadlines: a performance evaluation. In *Proc. 11th IEEE Real-Time Systems Symp. (RTSS)*, Lake Buena Vista, FL, December 5–7, pp. 113–124. IEEE Computer Society Press, Los Alamitos, USA.

[24] Chen, S., Stankovic, J. A., Kurose, J. F. and Towsley, D. (1991) Performance evaluation of two new disk scheduling algorithms for real-time systems. *J. Real-Time Syst.*, **3**(3), 307–336.

[25] Bosch, P. and Mullender, S. J. (2000) Real-time disk scheduling in a mixed-media file system. In *Proc. Sixth IEEE Real-Time Technology and Applications Symp.*, Washington, DC, 31 May–2 June, pp. 23–32. IEEE Computer Society Press, Los Alamitos, USA.

[26] Chang, H. P., Chang, R. I., Shih, W. K. and Chang, R. C. (2001) Reschedulable-Group-SCAN scheme for mixed real-time/non-real-time disk scheduling in a multimedia system. *J. Syst. Softw.*, **59**(2), 143–152.

[27] Shenoy, P. and Vin, H. M. (1998) Cello: a disk scheduling framework for next generation operating systems. In *Proc. ACM SIGMETRICS*, Madison, WI, USA, June 22–26, pp. 44–55. ACM Press, USA.

[28] Lehoczky, J. P., Sha, L. and Stronsnider, J. K. (1987) Enhanced aperiodic responsiveness in hard real-time environments. In *Proc. 8th IEEE Real-Time System Symp.*, San Jose, CA, USA, December 1–3, pp. 261–270. IEEE Computer Society Press, Los Alamitos, USA.

[29] Jeffay, K., Stanat, D. F. and Martel, C. U. (1991) On non-preemptive scheduling on periodic and sporadic tasks. In *Proc. Real-Time System Symp.*, San Antonio, TX, USA, December 4–6, pp. 129–139. IEEE Computer Society Press, Los Alamitos, USA.

[30] Lee, C. G. *et al*. (1996) Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. In *Proc. 17th Real-Time Systems Symp.*, Los Alamitos, CA, USA, 4–6 December, pp. 264–274. IEEE Computer Society Press, Los Alamitos, USA.

[31] Lee, C. G. *et al*. (1998) Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. *IEEE Trans. Comput.*, **47**(6), 700–713.

[32] Lee, C. G. *et al*. (1997) Enhanced analysis of cache-related preemption delay in fixed-priority preemptive scheduling. In *Proc. 18th IEEE Real-Time Systems Symp.*, San Francisco, CA, USA, December 2–5, pp. 187–198. IEEE Computer Society Press, Los Alamitos, USA.

[33] Lee, S., Min, S. L., Kim, C. S., Lee, C. G. and Lee, M. (1999) Cache-conscious limited preemptive scheduling. *Real-Time Syst.*, **17**(2–3), 257–282.

[34] Luculli, G. and Netale, M. D. (1997) A cache-aware scheduling algorithm for embedded systems. In *Proc. 18th IEEE Real-Time Systems Symp.*, San Francisco, CA, USA, December 2–5, pp. 199–209. IEEE Computer Society Press, Los Alamitos, USA.

[35] Patterson, R., Hugo, G., Garth A., Ginting, E., Stodolsky, D. and Zelenka, J. (1995) Informed prefetching and caching. In *Proc. Fifteenth ACM Symp. on Operating System Principles*, Copper Mountain Resort, CO, USA, December 3–6, pp. 79–95. ACM Press, USA.

[36] Ganger, G. R. and Schindler, J. (1998) Database for Validated Disk Parameters for DiskSim. http://www.pdl.cmu.edu/DiskSim/diskspecs.html.

[37] Ganger, G. R. (1995) *System-Oriented Evaluation of Storage Subsystem Performance*. PhD Dissertation, CSE-TR243-95, University of Michigan, Ann Arbor, MI, USA.

[38] Schindler, J. and Ganger, G. R. (1999) *Automated Disk Drive Characterization.* Technical Report CMU-CS-99-176, School of Computer Science, University of Carnegie Mellon.

[39] Worthington, B. L., Ganger, G. R., Patt, Y. N. and Wilkes, J. (1995) On-line extraction of SCSI disk drive parameters. In *Proc. ACM SIGMETRICS*, Ottawa, Ont., Canada, May 15–19, pp. 136–145. ACM Press, USA.

[40] Quantum Corporation, Quantum Atlas 10K (1999) http://www.quantum.com/products/hdd/atlas_10k/atlas_10k_specs.htm.