# Scalable Multimedia Disk Scheduling

Mohamed F. Mokbel[1] Walid G. Aref[1] Khaled El-Bassyouni[2] Ibrahim Kamel[2]

[1]Department of Computer Sciences, Purdue University, West Lafayette, IN 47907-1398

[2]Panasonic Information and Networking Technologies Laboratory. Two Research Way Princeton, NJ 08540

{mokbel,aref}@cs.purdue.edu, ibrahim@research.panasonic.com

## Abstract

*A new multimedia disk scheduling algorithm, termed Cascaded-SFC, is presented. The Cascaded-SFC multimedia disk scheduler is applicable in environments where multimedia data requests arrive with different quality of service (QoS) requirements such as real-time deadline and user priority. Previous work on disk scheduling has focused on optimizing the seek times and/or meeting the real-time deadlines. The Cascaded-SFC disk scheduler provides a unified framework for multimedia disk scheduling that scales with the number of scheduling parameters. The general idea is based on modeling the multimedia disk requests as points in multiple multi-dimensional sub-spaces, where each of the dimensions represents one of the parameters (e.g., one dimension represents the request deadline, another represents the disk cylinder number, and a third dimension represents the priority of the request, etc.). Each multi-dimensional sub-space represents a subset of the QoS parameters that share some common scheduling characteristics. Then the multimedia disk scheduling problem reduces to the problem of finding a linear order to traverse the multi-dimensional points in each sub-space. Multiple space-filling curves are selected to fit the scheduling needs of the QoS parameters in each sub-space. The orders in each sub-space are integrated in a cascaded way to provide a total order for the whole space. Comprehensive experiments demonstrate the efficiency and scalability of the Cascaded-SFC disk scheduling algorithm over other disk schedulers.*

## 1 Introduction

Multimedia applications are growing rapidly due to the advances in computer hardware and software technologies. In particular, the advancement in mass storage, video compression, and high-speed networks have made it feasible to provide multimedia servers (e.g., news broadcasting, digital libraries, and Video on Demand). With these applications, disk bandwidth is a scarce resource that needs to be managed intelligently through the underlying disk scheduling algorithm. In addition to maximizing the disk bandwidth, the disk scheduler must take into consideration the real-time deadline constraints of each disk request. If the multimedia servers' clients are prioritized based on certain quality of service (QoS) guarantees, then the multimedia disk scheduler should consider the priority of disk requests in its disk queue. Designing a multimedia disk scheduler that handles real-time and QoS constraints in addition to maximizing the disk bandwidth is a difficult task [3].

In attempting to satisfy multiple concurrent and conflicting requirements (e.g., disk bandwidth, deadline, and priority), scheduler designers and algorithm developers depend mainly on heuristics to code the schedulers (e.g., see [1, 2, 3, 5, 6, 9, 12, 14, 18, 19]). It is not always clear that these schedulers are equitable to all aspects of the system, or are controllable in a measurable way to favor one aspect of the system over the other. For example, a scheduler that minimizes the overhead of seek time may violate deadlines of some high priority requests. Another scheduler that deals with requests based on their priorities may incur excessive seek time overhead, which leads to deadline violations. In the case of disk requests with multiple priorities (i.e., QoS requirements), a disk scheduler that handles requests based on one type of priority will fail to adequately handle other types of priorities.

In this paper, we introduce a *scalable* and *generic* multimedia disk scheduler, termed the *Cascaded-SFC* disk scheduler, that takes all the requirements of a multimedia server into consideration. The *Cascaded-SFC* disk scheduler can be tuned by simple parameters to emulate the operation of other disk schedulers, e.g., FCFS, EDF, SSTF and SCAN-EDF. In addition to disk scheduling, *Cascaded-SFC* can be used in other applications that have multiple concurrent and conflicting requirement, e.g., schedulers for multi-threaded CPUs [21], network-attached storage devices (NASDs) [8, 13], operating systems schedulers [15].

The main idea of the *Cascaded-SFC* disk scheduler is to model the multimedia disk requests as points in multiple multi-dimensional sub-spaces. Each sub-space encompasses a group of QoS parameters. Each dimension in a sub-space represents one of the parameters (e.g., one dimension represents the request deadline, another dimension represents the disk cylinder number, a third dimension represents the priority of the request, etc.). Each multi-dimensional sub-space represents a subset of the QoS parameters that share some common scheduling characteristics. The scheduler problem is thus reduced to finding a linear-order to traverse the multi-dimensional points in each sub-space. We distinguish among three categories of QoS requirements for disk requests: (1) Priority-like requirements (e.g., user priority, request value, request size, arrival time). The objective of this category is to respect the order of each priority type. Serving two disk requests in reverse order with respect to any of these priorities is considered a priority inversion. (2) Deadline-like requirements. Unlike the first category, there is no penalty when serving disk requests in reverse order as long as the disk requests are served before their deadlines. (3) Disk utilization requirements (e.g., the seek and latency time). Unlike the first two categories, the value of the third category is continuously changing with each scheduling instance. Based on these categories, a scalable multimedia disk scheduler should achieve the following goals:

1. **Minimizing Priority Inversion.** For each priority-like parameter, a multimedia disk scheduler should minimize the number of priority inversions.

2. **Scalability.** The efficiency of a multimedia disk scheduler should not affected by the number of QoS requirements (i.e., the dimensionality of the scheduling problem).

3. **Fairness.** A multimedia disk scheduler should be fair to all QoS requirements i.e., it should demonstrate similar behavior with respect to all dimensions (QoS requirements).

4. **Minimizing Deadline Loss.** A multimedia disk scheduler should minimize the number of deadline losses.

5. **Selectivity.** If a deadline miss is a must, a multimedia disk scheduler should be able to select the disk requests with lowest priority to miss.

6. **Maximizing Disk Utilization.** A multimedia disk scheduler should minimize the seek and latency times, thus maximizing the disk utilization.

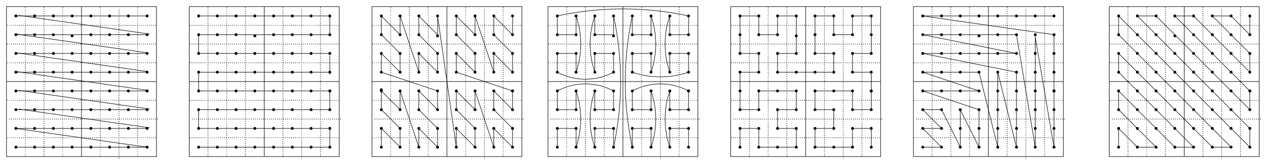
The *Cascaded-SFC* multimedia disk scheduler presented in this paper achieves these goals. The underlying theory of *Cascaded-SFC* is based on space-filling curves (SFCs). A space-filling curve maps the multi-dimensional space into the one-dimensional space. An SFC acts like a thread that passes through every cell element (or pixel) in the multi-dimensional space so that every cell is visited exactly once. Thus, space-filling curves are adopted to define a linear order for sorting and scheduling objects that lie in the multi-dimensional space. Figure 1 gives examples of seven two-dimensional space-filling curves. Using space-filling curves as the basis for multimedia disk scheduling has numerous advantages, including (1) Scalability in terms of the number of scheduling parameters, (2) Ease of code development and maintenance, (3) The ability to analyze the quality of the schedules generated, and (4) The ability to automate the scheduler development process in a fashion similar to automatic generation of programming language compilers.

The remainder of this paper is organized as follows: Section 2 discusses related work. The *Cascaded-SFC* disk scheduler is presented in Section 3. Section 4 presents interested features of the *Cascaded-SFC* disk scheduler. Section 5 conducts a comprehensive study of the *Cascaded-SFC* disk scheduler. Practical applications of the *Cascaded-SFC* disk scheduler are addressed in Section 6. Finally, Section 7 concludes with a summary.

## 2 Related Work

Scheduler designers and algorithm developers continuously develop new disk scheduling algorithms to cope with the growing complexity of applications. Traditional disk scheduling algorithms (i.e., FCFS and SSTF) focus on only one aspect of the system. For example, FCFS (First-Come First-Service) aims to achieve fairness among user requests while ignoring disk utilization. On the other hand, SSTF (Shortest Seek Time First) aims to maximize disk utilization.

Real-time applications pose new challenges for scheduler designers. A real-time disk scheduler should take the deadline of each request into account. A straightforward algorithm to handle real-time disk requests is EDF (Earliest Deadline First) [16]. In EDF, disk requests are served based on their deadlines. Although EDF achieves good results in terms of minimizing the deadline losses, it degrades the disk utilization,



(a) Sweep (b) C-Scan (c) Peano (d) Gray (e) Hilbert (f) Spiral (g) Diagonal

**Figure 1. Two-dimensional Space-Filling Curves.**

where it does not take into account the cylinder position of disk requests. Many variants of the traditional SCAN algorithm [7] are proposed (e.g., see [1, 11, 18]) to handle real-time disk requests. The FD-SCAN (Feasible Deadline SCAN) algorithm [1] differs from SCAN in the way that it dynamically adapts the scan direction towards the request with the earliest feasible deadline. A deadline is said to be feasible if FD-SCAN estimates that the deadline can be met. SCAN-EDF [18] achieves a trade-off between the SCAN and EDF algorithms. where requests are served according to their deadlines. Requests with the same deadline are served in SCAN order. In the SCAN-RT algorithm [11], when a request arrives, it is inserted in the disk queue in the SCAN order, only if the insertion does not potentially violate the deadlines of other pending requests. Otherwise, the request is appended to the end of the queue. Other heuristic disk scheduling algorithms that are not based on SCAN include SSEDO [5] and SSEDV [5] (Shortest Seek and Earliest Deadline by Ordering/Value). These algorithms give the disk request with the earliest deadline/value a high priority. However, if a request with a larger deadline is *very* close to the current arm position, then it may be assigned the highest priority.

Scheduling prioritized user requests calls for another set of complicated algorithms. In the multi-queue disk scheduling algorithm [4], each queue represents a priority level. Disk requests in higher priority queues are served first. Inside each queue, requests are served in the SCAN order. Haritsa et. al. [9] add a new dimension for the scheduling problem, where they assume that each user request has a value in addition to its deadline. So, the objective of the scheduler is to maximize the sum of the satisfied user requests while minimizing the number of missed deadlines. They introduce the BUCKET algorithm that computes the priority of a user request using a mapping function that takes the value and deadline as input, and outputs one value indicating the priority of the user request. User requests are served based on their computed priority value. Since the BUCKET algorithm is designed for scheduling transactions, it does not take disk utilization into account. Multiple priorities disk scheduling is presented in [2]. The main idea is similar to the BUCKET where all the priorities are converted to only one value representing the absolute priority of the request. However, this algorithm does not take in consideration the real-time and disk utilization issues.

Combining disk utilization with deadline and priority in one disk scheduling algorithm is addressed in [12]. The objective is to propose a disk scheduling algorithm that serves disk requests based on their priority and minimizes the number of deadline misses while maximizing the disk utilization. The idea of the algorithm is to insert the new disk request in the SCAN order, only if the insertion does not potentially violate the deadline of other pending request. Otherwise, the scheduler chooses the lowest priority disk request in the queue and moves it to the tail of the queue. Other heuristic algorithms in disk scheduling that depend on the nature of disk requests are proposed in [3, 14, 20].

## 3 The Cascaded-SFC Disk-Scheduling Algorithm

It is obvious from the discussion in Section 2 that adding more dimensions (e.g., user priority, request value, request size, arrival time) to the disk scheduling problem results in more complicated disk scheduler algorithms. This paper proposes the *Cascaded-SFC* algorithm as a scalable multimedia disk scheduler that can scale with any number of dimensions. The basic idea of the *Cascaded-SFC* disk scheduler is to use space-filling curves to convert a multi-dimensional disk request into a one-dimensional value. Disk requests are modeled by multiple parameters (the disk cylinder, the real-time deadline, and the multiple priorities) and are represented as points in the multi-dimensional space, where each parameter corresponds to one dimension. Using space-filling curves, the multi-dimensional disk request is converted to a one-dimensional value. Then, disk requests are inserted into a priority queue

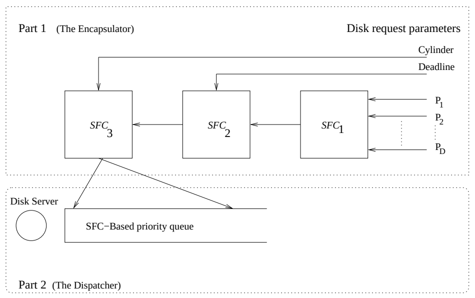


**Figure 2. Modeling of the *Cascaded-SFC* Disk Scheduler**

according to their one-dimensional value, with a lower value indicating a higher priority. Because of the different natures of the multimedia scheduling parameters and the different characteristics of these parameters, *Cascaded-SFC* groups the scheduling parameters into stages and schedules the disk requests in a cascaded fashion. Figure 2 illustrates the *Cascaded-SFC* disk scheduler with three cascaded stages (sub-spaces). Multiple space-filling curves ($SFC_1$, $SFC_2$, and $SFC_3$) are selected to fit the scheduling needs of the QoS parameters in each sub-space. A real-time disk request $T$ with $D$ different types of priorities (user priority, value, arrival time, etc.) is modeled as a point in the $(D+2)$-dimensional space. $D$ dimensions represent the different types of priorities, one dimension represents deadline, another dimension represents cylinder position. The mapping from the $(D+2)$-dimensional space into the one dimensional space is performed with three stages of space-filling curves. The first stage takes the $D$ types of priorities as input and outputs a one-dimensional value, using a $D$-dimensional space-filling curve ($SFC_1$) that aims to minimize priority inversion. The second stage takes the output of the first stage and the disk request deadline as input, and applies a two-dimensional space-filling curve ($SFC_2$) that aims to minimize deadline violations. In addition, if a deadline has to be missed, $SFC_2$ chooses a low priority request as a victim. Similarly, the third stage takes the output of the second stage and the cylinder position of the disk request, and applies a two-dimensional space-filling curve ($SFC_3$) that optimizes the seek time. The output from the third stage is a one-dimensional value, the *characterization value* $v_c$, that encapsulates all the properties of the disk request. A disk request $T$ is inserted in a priority queue $q$ based on its characterization value $v_c$. The lower the value of $v_c$ the higher the priority of the disk request. In the remainder of this paper, we use the terms $T_i$ and the characterization value $v_c$ of request number $i$ as synonymous.

The *Cascaded-SFC* disk scheduling algorithm is divided into two parts. The first part, the *encapsulator*, takes the multi-dimensional disk requests as input, and outputs a one-dimension value $v_c$. The second part, the *dispatcher*, takes $v_c$ as input, and outputs the scheduling order of disk requests. The appropriate choice of the cascaded space-filling curves $SFC_1$, $SFC_2$, and $SFC_3$ in the *encapsulator* is investigated in Section 5, along with tunable parameters that reflect the relative importance of deadline and seek time optimization. The remainder of this section focuses on the *dispatcher*, that is, the order by which the disk requests are dispatched from the priority queue $q$. We start by describing two straightforward approaches for managing the *Cascaded-SFC* priority queue, namely, the *Non-Preemptive* Disk Scheduler and the *Fully-Preemptive* Disk Scheduler.

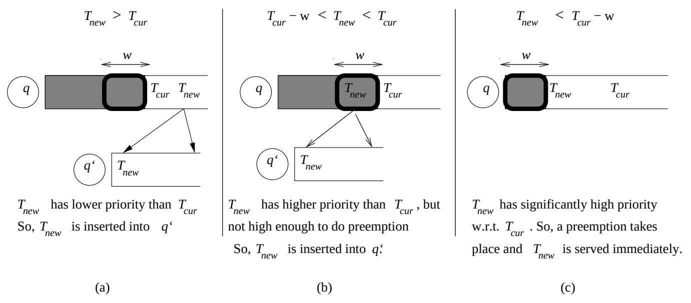
**The Non-Preemptive Disk Scheduler:** Using this approach, once the disk server starts to serve the disk requests from the priority queue $q$, no more requests are inserted in $q$. Arriving requests are grouped together in another priority queue $q$‘. Once $q$ is empty, $q$ and $q$‘ are swapped, and the disk server starts serving disk requests from $q$ again. This scheduler is non-preemptive in the sense that the process of serving disk requests from $q$ is not preempted by the arrival of new requests.

**The Fully-Preemptive Disk Scheduler:** This is a trivial approach. All requests are inserted in the same priority queue $q$ regardless of their arrival times. This scheduler is fully-preemptive in the sense that the process of serving disk requests from $q$ is preempted by the arrival of any new request.

The fully-preemptive disk scheduler serves all disk requests according to their priority. Low priority requests may starve due to the continuous arrival of high priority requests. On the other hand, the non-preemptive disk scheduler does not lead to starvation since it groups requests according to their arrival times. However, priority inversion is obvious since higher priority requests inserted in $q$‘ must wait for all lower priority requests in $q$ to be served. The drawbacks of the two approaches motivate the concept of a combined disk scheduler that has the merits of both schedulers. In the following section, we present a new multimedia disk scheduling algorithm that avoids the drawbacks of these algorithms, that is, it respects the disk request priority and avoids starvation.

## 3.1 The Conditionally-Preemptive Disk Scheduler

As a trade-off between the fully-preemptive and the non-preemptive disk schedulers, the conditionally-



$T_{new} > T_{cur}$
$w$
$q$
$T_{cur}$ $T_{new}$
$q$'
$T_{new}$
$T_{new}$ has lower priority than $T_{cur}$
So, $T_{new}$ is inserted into $q$'
(a)
$T_{cur}$ − w < $T_{new}$ < $T_{cur}$
$T_{new}$ has higher priority than $T_{cur}$, but not high enough to do preemption
So, $T_{new}$ is inserted into $q$'
(b)
$T_{new}$ < $T_{cur}$ − w
$T_{new}$ has significantly high priority w.r.t. $T_{cur}$. So, a preemption takes place and $T_{new}$ is served immediately.
(c)

**Figure 3. The Conditionally-Preemptive Disk Scheduler.**

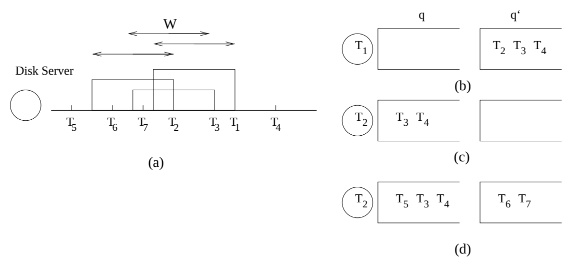


**Figure 4. Example of the Conditionally-Preemptive Disk Scheduler**

preemptive disk scheduler allows a newly arrived disk request $T_{new}$ to preempt the process of serving disk requests in $q$ if and only if it is has *significantly higher* priority than the currently served disk request $T_{cur}$. To quantify "significantly higher", we define a blocking window with size $w$ (the rounded box with thick border in Figure 3) that slides with $T_{cur}$ in $q$. $T_{new}$ is considered to have significantly higher priority than $T_{cur}$ if and only if $T_{new} < T_{cur} - w$. The window size $w$ is a compromise between the fully-preemptive and the non-preemptive disk schedulers. Setting $w$=0 corresponds to the fully-preemptive disk scheduler, while setting $w$ to a very large value corresponds to the non-preemptive disk scheduler. If $T_{new}$ arrives while the scheduler is ready to serve $T_{cur}$, then one of the following takes place:

1. $T_{cur} < T_{new}$ (Figure 3a). $T_{new}$ has lower priority than $T_{cur}$. Hence, $T_{new}$ is inserted into $q$', as it is not more important than $T_{cur}$.

2. $T_{cur} - w < T_{new} < T_{cur}$ (Figure 3b). $T_{new}$ lies inside the blocking window $w$. Although $T_{new}$ has a priority higher than that of $T_{cur}$, it is not high enough to preempt the process of serving disk requests in $q$. So, $T_{new}$ is inserted in the waiting queue $q$'.

3. $T_{new} < T_{cur} - w$ (Figure 3c). $T_{new}$ has a priority that is significantly higher than that of $T_{cur}$. So, it is worth preempting the process of serving requests in $q$ by inserting $T_{new}$ in $q$.

There are two issues that need to be addressed: (1) Priority inversion results from disk requests that lie inside the blocking window $w$ (stored in $q$') and have higher priority than some requests in $q$, and (2) With any value of $w < MAX(v_c)$, there is still a chance of starvation when a continuous stream of very high priority requests arrive. The next two sections propose two approaches for tuning priority inversion and starvation.

## 3.2 Tuning Priority Inversion

Storing disk requests that lie within window $w$ in $q$' results in priority inversion, as the blocked requests have higher priorities than that of the disk request that is currently being served. Thus, we propose the SP (Serve and Promote) policy. In the SP policy, before the disk starts to serve a request from $q$, it checks $q$' for any disk requests that eventually attain significantly higher priority. If such a request is found, SP promotes this request and insert it in $q$. Thus, serving disk requests in $q$ can be preempted either by a newly arrived request or by an old request that eventually attains significantly higher priority.

Figure 4 gives an example of applying the conditionally preemptive *Cascaded-SFC* disk scheduling with the SP policy. Disk requests $T_1, T_2, \cdots, T_7$ are numbered according to their arrival times and are drawn into a priority line, where $T_5$ has the highest priority while $T_4$ has the lowest priority (Figure 4(a)). When $T_1$ arrives, the disk is idle, so $T_1$ is served immediately. While $T_1$ is served, $T_2$, $T_3$, and $T_4$ arrive and are inserted in $q$', since none of them have significantly higher priorities than $T_1$ (Figure 4(b)) (note that $T_2$ and $T_3$ have higher priorities than $T_1$, but they are still within the blocking window $w$). When $T_1$ is served, $q$ is empty, so $q$ and $q$' are swapped and $T_2$ is served, since it has the highest priority among requests in $q$ (Figure 4(c)). While $T_2$ is served, $T_5$, $T_6$, and $T_7$ arrive. Only $T_5$ has significant importance, and it is inserted in $q$ (Figure 4(d)). After serving $T_5$ and before serving $T_3$, the scheduler takes into consideration that $T_6$ has significantly higher priority than $T_3$ ($T_6$ lies outside the window $w$ w.r.t. $T_3$). So the scheduler serves $T_6$ before $T_3$. The same applies to $T_4$ and $T_7$, before serving $T_4$, the scheduler recognizes that $T_7$ has significantly higher priority than $T_4$. So, the final order of serving disk requests is $T_1, T_2, T_5, T_6, T_3, T_7, T_4$.

### 3.3 Tuning Starvation Avoidance

If the window size $w$ remains fixed, an adversary would still select disk requests in a manner that results in the starvation of other disk requests. To completely avoid starvation, we propose the ER (Expand and Reset) policy. ER *expands* the window size $w$ by a constant, termed the *expansion* factor $e$, with any preemption of the process of serving disk requests from $q$. However, when a disk request is served and another disk request from $q$ is dispatched, ER *resets* $w$ to its original value. ER avoids starvation, where after a finite number of high priority requests, $W$ will be eventually very large, and hence the scheduler will work in the non-preemptive mode, which is free of starvation. The scheduler moves back and forth between being non-preemptive or conditionally-preemptive as the window $w$ expands or shrinks.

## 4 Features of the Cascaded-SFC Disk Scheduler Algorithm

The idea of using three cascaded space-filling curves instead of only one space-filling curve in the *Cascaded-SFC* scheduler results in many good features. In this section, we focus on three main features: (1) Flexibility: The *Cascaded-SFC* can work in several environments. (2) Generalization: The *Cascaded-SFC* can be considered as a generalization of many other disk schedulers. (3) Extensibility: The *Cascaded-SFC* can be used to extend other disk schedulers to deal with different environments. Other features that include scalability, modularity of the design, and fairness are investigated through comprehensive experiments in Section 5.

### 4.1 Flexibility

The *Cascaded-SFC* multimedia disk scheduler is flexible as it can fit in environments that have different requirements with very slight modifications. For example, in environments where deadlines are relaxed, $SFC_2$ can be skipped and the output of $SFC_1$ is entered directly as input to $SFC_3$. If the scheduling problem does not need to optimize for disk utilization (e.g., CPU scheduling, thread scheduling), then $SFC_3$ can be skipped, and the output from $SFC_2$ is entered directly to the priority queue. In the case of applications with only one priority type, $SFC_1$ is ignored and the priority is entered directly to $SFC_2$.

### 4.2 Generalization

The *Cascaded-SFC* disk scheduler is a generalization of many disk schedulers. Ignoring the three stages of space-filling curves and setting $w = 0$ in the priority queue makes the *Cascaded-SFC* work as any one-dimensional disk scheduler (e.g., FIFO, SSTF, EDF, SCAN). Notice that the criteria of inserting disk requests in the queue depends on the algorithm. Two-dimensional disk schedulers are special cases from the *Cascaded-SFC* disk scheduler by ignoring two of the three stages. For example, FD-SCAN [1], SCAN-EDF [18], multi-queue disk scheduler [4] can be realized by the *Cascaded-SFC* disk scheduler when using only $SFC_3$. The Sweep SFC is used as $SFC_3$ where the $y$ dimension represents the deadline (for FD-SCAN and SCAN-EDF) or the request priority (for multi-priority scheduling) while the $x$ dimension represents the difference in cylinders between the current cylinder position and the disk request in any direction (for FD-SCAN) or in the current direction (for SCAN-EDF and multi-priority scheduler). Multiple priorities disk scheduler [2] is realized by the *Cascaded-SFC* disk scheduler when using only $SFC_1$ and ignoring $SFC_2$ and $SFC_3$. In this case, the multiple priorities will be entered to $SFC_1$ and the output is a one-dimensional value that entered to the priority queue.

### 4.3 Extensibility

The idea of the *Cascaded-SFC* disk scheduler can be used to extend current disk schedulers to deal with multiple priorities and/or real-time systems. For example, the disk scheduler in [12] deals with real-time disk requests where each request has only one type of priority. To extend the functionality of this algorithm to deal with multiple types of priorities, we use the $SFC_1$ from the *Cascaded-SFC* disk scheduler. The multiple priorities is entered to $SFC_1$ and the output is considered the absolute priority of the disk request that is entered into the algorithm in [12]. Similarly to extend the BUCKET algorithm [9] to deal with disk utilization, we take the output of the BUCKET algorithm and enter it into $SFC_3$ of the *Cascaded-SFC* disk scheduler with the cylinder position of the disk request. Similar ideas may be used to extend the FD-SCAN, SCAN-EDF to deal with prioritized requests and for multiple queue disk scheduler to deal with real-time disk requests.

## 5 Performance Analysis

To evaluate the performance of the *Cascaded-SFC* multimedia disk scheduler, we perform several simu-

| Disk Parameter | Value |
|---|---|
| No. of Cylinders | 3832 |
| Type | Quantum XP32150 |
| Tracks/Cylinder | 10 |
| No. of Zones | 16 |
| Sector Size | 512 |
| Rotation Speed | 7200 RPM |
| Average Seek | 8.5 mSec. |
| Max. Seek | 18 mSec. |
| Seek cost function | $0.8 + 0.002372(d) + 0.12581(\sqrt{d})$ |
| Disk Size | 2.1 GBytes |
| File Block Size | 64 KBytes |
| Transfer Speed | 4.9 - 8.2 MBytes/sec |
| Disks / RAID | 5 (4 data 1 Parity) |

**Table 1. Disk Model**

lation experiments using a simulator that models the PanaViss server. PanaViss is a video server research prototype [10] for video broadcasting applications. For architecture details of the PanaViss video server, the reader is referred to [12]. The parameters of the disks in PanaViss server are given in Table 1.

A milestone in the *Cascaded-SFC* multimedia disk scheduler is the choice of three space-filling curves for the three stages of Figure 2. If we limit ourselves to the seven space-filling curves given in Figure 1, we will have $7^3$ different versions of the *Cascaded-SFC* disk scheduler. In this section, we will not show an exhaustive comparison for all different versions. Instead, we will give the comparison between different space-filling curves for $SFC_1$ only. For $SFC_2$ and $SFC_3$, we give the space-filling curve that gives best performance and compare to other conventional disk schedulers.

## 5.1 Priority Inversion

The number of priority inversions for dimension $k$ that results from serving a disk request $T_i$ is the number of disk requests $|T_w|$ in the waiting queue that have higher priority in dimension $k$ than $T_i$. The total number of priority inversions is the sum of all priority inversions over all dimensions. The number of priority inversions is continuously increasing with the time of the simulation run. Thus, for comparison purposes, we represent the number of priority inversions as a percentage of the number of priority inversions that occurs in the FIFO policy. To be able to evaluate $SFC_1$ independently of the rest of the scheduler, we assume that disk requests have relaxed deadlines (So, $SFC_2$ can be eliminated) and the disk block size is large enough to make the transfer time of disk requests dominate the seek time (So, $SFC_3$ can be eliminated). Thus, the output of $SFC_1$ is entered directly to the priority queue.

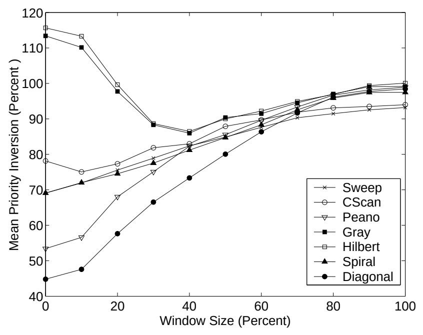


**Figure 5. Minimizing Priority Inversion.**

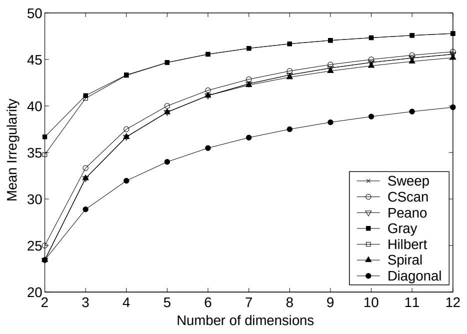
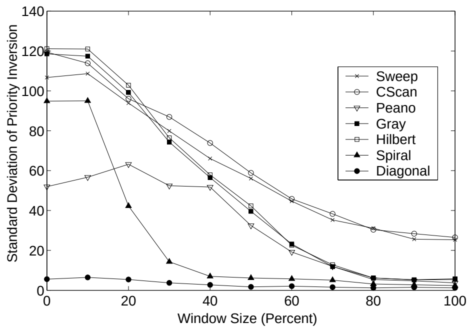


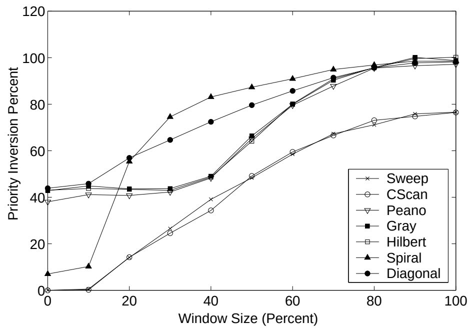
**Figure 6. Scalability of the *Cascaded-SFC* disk scheduler.**

**Minimizing Priority Inversion.** Figure 5 gives the priority inversion that results from the seven space-filling curves given in Figure 1 for normal and high system load. The window size $w$ is varied from 0% of the size of the scheduling space (the fully-preemptive scheduler) to 100% of the size of the scheduling space (the non-preemptive scheduler). In Figure 5a, disk requests arrive exponentially with mean interarrival time 25 msec. The Diagonal SFC gives the best performance (lowest priority inversion) at $w < 60\%$, with around 10% lower than the Peano SFC. The Gray and Hilbert SFCs have very high priority inversion. For large window sizes, the Sweep and C-Scan SFCs are the best curves. This is due to the nature of these curves that is suitable to the non-preemptive scheduler.

**Scalability.** Figure 6a gives the number of priority inversion when running the *Cascaded-SFC* disk scheduler for up to 12 QoS parameters (i.e., 12 dimensions). Each QoS parameter (dimension) has 16 priority levels. The experiment is performed with window size $w = 50\%$ and mean interarrival time 25 msec. The Diagonal SFC gives the best performance especially with higher dimensions. The Sweep, Peano, and Spiral SFCs have almost the same performance.



(a) Standard Deviation



(b) Favored Dimension

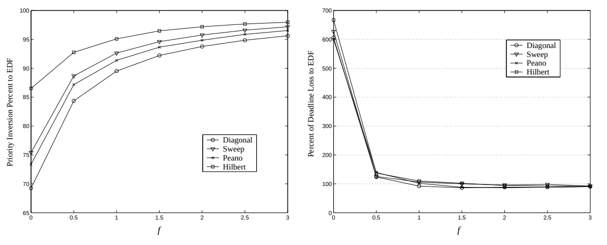
**Figure 7. Fairness of the *Cascaded-SFC* disk scheduler.**

**Fairness.** A very critical point for $SFC_1$ in the *Cascaded-SFC* disk scheduler is how to assign the disk request parameters (i.e., value, user priority, etc.) to the dimensions of the space-filling curve. Depending on the nature of the application, we may require that $SFC_1$ should be either fair to all dimensions or biased to some of the dimensions. In this experiment, we use the standard deviation of priority inversion over all dimensions as a measure of the fairness of space-filling curves. The lower the standard deviation the more fair the space-filling curve. The experiment is performed in the four-dimensional space with $w = 50\%$, and the interarrival time of disk requests is 25 msec. In Figure 7a, the Diagonal SFC is the most fair space-filling curve among the space-filling curves we consider in this study (the standard deviation is less than 10%). For a medium window size, the Spiral SFC has a very low standard deviation. The C-Scan and Sweep SFCs give the worst performance because they have no priority inversion in the last dimension while having high priority inversion in all the other dimensions. Figure 7b plots only the most favored dimension (the one with lowest priority inversion) from the same experiment. The C-Scan and Sweep SFCs are always the best where they have no priority inversion in small window sizes, which interprets why they have very high standard deviation (Figure 7a). This result is very beneficial in applications that have only one important dimension, while the other dimensions are not with the same significant importance. For example, some applications that schedule over user priority, value, and request size may decide to maximize the profit. So, the value dimension would be the most important dimension while other dimensions may not have the same importance. Other applications may decide to strictly enforce the user priority. In this case, a space-filling curve that favors one of the dimensions over the others will be chosen.

## 5.2 Missing Deadlines

In the following experiments, we consider real-time multi-priority disk requests. Each disk request has three types of priorities and a deadline selected randomly in the range 500-700 msec. In a typical application, such as nonlinear editing, low priority requests can be represented by ftp transfers of large files while higher priority requests can be represented by the playing of Audio and Video files, which are usually retrieved in chunks of small size blocks. Thus it is reasonable to assume that the service time for high priority requests is smaller than that of lower priority requests. Also, we assume that the disk block size is large enough to make the transfer time of disk requests dominate the seek time. So, $SFC_3$ in Figure 2 is ignored and the output from $SFC_2$ is entered directly to the priority queue.

$SFC_2$ is a two-dimensional space-filling curve that has the deadline as one dimension and priority (the output from $SFC_1$) as the second dimension. We propose a variable $f$ that balances between the two dimensions. In this case, the characterization value $v_c$ is computed from $v_c = Priority + f * deadline$. Setting $f = 0$ and solving the tie by serving the earliest deadline corresponds to the Sweep SFC where the $X$ dimension represents the deadline while the $Y$ dimension represents the priority. Setting $f = 1$ while solving the tie randomly corresponds to the Diagonal SFC. Also, setting $f$ to a very large value while solving the tie by serving the higher priority request corresponds to the Sweep SFC where the $X$ dimension represents the priority and the $Y$ dimension represents the deadline. In



(a) Priority Inversion (b) Deadline Misses

**Figure 8. The effect of $f$ in $SFC_2$.**

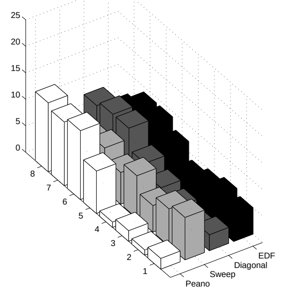
general, if $f < 1$, then the scheduler pays more attention to minimizing priority inversion than satisfying the deadline. If $f > 1$, then serving disk requests according to deadline is more important than minimizing priority inversion. For comparison purposes, we normalize all the results in this section to the EDF disk scheduler. In the following experiments, we show the effect of changing $f$ on the *Cascaded-SFC* disk scheduler.

**Minimizing Deadline Misses.** Figures 8a and 8b give the effect of changing $f$ on priority inversion and the number of deadline misses (normalized to EDF), respectively. When $f = 0$, the scheduler ignores the deadline in order to minimize the priority inversion. This results in a very high ratio (from six to seven times the EDF) of deadline misses and low priority inversion. As $f$ increases, the *Cascaded-SFC* disk scheduler gives more attention to deadline. Thus, the ratio of deadline misses decreases while the priority inversion increases. The Diagonal SFC gives better results in both deadline misses and priority inversion. Setting $f = 1$ results in a reasonable trade-off between priority inversion and deadline misses, where the Diagonal SFC achieves the same number of deadline misses as the EDF while minimizing the priority inversion to 90%.

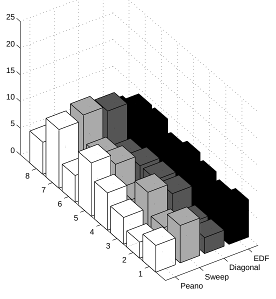
**Selectivity.** One may ask why bother with a complicated space-filling curve to achieve only 90% of priority inversion, while the deadline misses is the same as EDF. However, this is not everything that space-filling curves can achieve. The *Cascaded-SFC* disk scheduler does not only minimize the deadline misses, it also sacrifices the low priority disk requests when missing a deadline is a must. Figure 9 gives the number of deadline misses in each priority level in three-dimensional disk requests. We compare the EDF scheduler with the *Cascaded-SFC* disk schedulers based on the Diagonal, Sweep, and Peano SFCs. Each dimension of the three-dimensional space is plotted separately. Each column represents the number of deadline misses of a certain priority level, where we have eight priority levels for each dimension. The objective from this experiment is to see how the deadline misses are scattered over the different dimensions and different priority levels. The ideal scheduler would have all its deadline misses in the lowest priority level (level 8) for all dimensions. EDF does not take into consideration the priority level of the disk request. Thus, EDF misses disk requests randomly from each priority level and it may happen that higher priority requests are lost in favor of serving lower priority requests. Each space-filling curve uses its own properties [17] to select the disk request to sacrifice. For example, the Diagonal SFC minimizes the deadline misses of high priority disk requests over all dimensions. Also, the Diagonal SFC treats all dimensions fairly where it has approximately the same pattern of deadline misses over the three space dimensions. The Sweep SFC favors the third dimension in the sense that it does not miss any requests with high priority in this dimension. However, for the other two dimensions, Sweep SFC treats disk requests as the EDF scheduler does. In applications that have priorities with very high importance, it is better to use the Sweep SFC and assign this type of priority to the last dimension. The Peano SFC favors the first and the last dimensions since it misses only low priority disk requests in these dimensions. Due to the significance of these results, we perform these experiments in a practical environment, and describe our results in Section 6.

## 5.3 Maximizing Disk Utilization

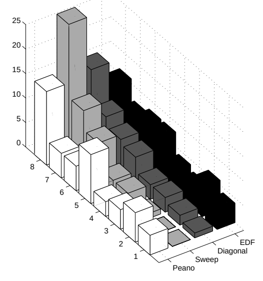
In the following experiments, we use the same parameters and assumptions as in the previous section, except that we consider disk blocks with small size to analyze the effect of seek time optimization. Experimental results show that using the two-dimensional Sweep SFC as $SFC_3$ gives the best results that combine the cylinder dimension with the priority-deadline dimension that comes form $SFC_2$. The $X$ dimension in the Sweep SFC is assigned to the priority-deadline dimension, while the $Y$ dimension is assigned to the nearest disk request. In this case, the characterization value $v_c$ would be: $v_c = Y_v \times Max_x + X_v$, where $Y_v$ is the value of the $Y$ dimension which is the difference in cylinders between the current position of disk head and the disk request. $Max_x$ corresponds to the maximum possible value in the $X$ dimension. $X_v$ is the value of $X$ dimension which is the difference between the priority-deadline value of the disk request and the minimum possible priority-deadline value of any disk request. With such value of $v_c$, disk requests are inserted in the disk queue $q$ in the order of their $Y_v$ value, and in case of a tie (two disk requests lie



(a)1st Dimension



(b)2nd Dimension



(c)3rd Dimension

**Figure 9. Number of deadline misses in all priority levels.**

in the same cylinder), the disk request with lower $X_v$ (priority-deadline value) would be served first. Then, all disk requests in $q$ can be served in only one disk scan.

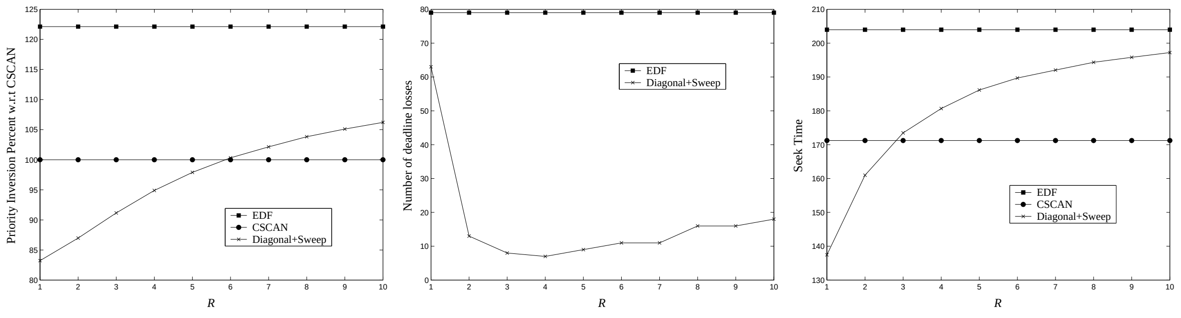
To tune the effect of seek time optimization, we use a factor $R$ that takes an integer value. $R$ represents the number of disk scans that a disk head is allowed to take to serve all disk requests in $q$. The idea of $R$ is to partition the two-dimensional space vertically into $R$ partitions numbered from 0 to $R-1$ where partition size is $P_s = \frac{Max_x}{R}$. In the above discussion we set $R = 1$ which allows only one partition (hence, one single scan) that is visited by the Sweep SFC. For any $R > 1$, we serve all disk requests that lie in partition $R$ before starting to serve disk requests from partition $R+1$. Each partition is served separately using the Sweep SFC. A disk request with $(X_v, Y_v)$ lies in the partition number $P_n = \lfloor \frac{X_v R}{Max_x} \rfloor$. Then, $v_c = Max_y \times P_s \times P_n + Y_v \times P_s + X_v + P_s \times P_n$ where $Max_y$ is the number of disk cylinders. Note that setting $R = 1$ would result in $P_s = Max_x$ and $P_n = 0$. This makes $v_{c\,R=1} = Y_v \times Max_x + X_v$ which is the same as using the Sweep SFC in one partition. In general, $R = i$ corresponds to passing through $i$ two-dimensional Sweep space-filling curves glued together horizontally.

The effect of $R$ on the number of priority inversions, deadline misses, and seek time, is given in Figures 10 (a),(b), and (c), respectively. In all experiments we compare the CSCAN, EDF schedulers with the *Cascaded-SFC* disk scheduler where $SFC_1$ and $SFC_2$ are represented by the Diagonal SFC while $SFC_3$ is represented by the Sweep SFC tuned by $R$. In Figure 10(b), the number of deadline misses is normalized to the CSCAN disk scheduler. The *Cascaded-SFC* scheduler algorithm always gives lower deadline misses than EDF and CSCAN. At $R = 1$, the deadline loss is still large; this is because $R = 1$ ignores the priority and hence ignores the deadline and sorts on seek time only. However, it still gives lower misses than EDF because sorting on seek time increases disk utilization, allowing the disk to serve more disk requests before their deadlines. When $R$ reaches 4, the number of deadline misses decreases; this is because we take the deadline and priority into consideration. However, increasing $R$ again results in more deadline losses. In this case, we give more attention to priority and ignore the seek time, which results in poor disk utilization and hence loss of requests.

Figure 10(c) shows that when $R < 3$, the seek time of the *Cascaded-SFC* disk scheduler is less than the CSCAN. Figure 10(a) shows that the *Cascaded-SFC* disk scheduler has better priority inversion than CSCAN when $R < 7$. From these figures, we can conclude that $R = 3$ is the best value for $R$, where it beats CSCAN in terms of number of deadline losses, disk utilization and priority inversion.

## 6 Practical Considerations and Applications

We apply the *Cascaded-SFC* disk scheduler to the design of a scheduler for a real-time file system that will be used in the NewsByte50. NewsByte50 is a non-linear editing server that can use both MPEG2 and high quality DVD formats. Non-linear editing servers are multimedia servers that are commonly used by the broadcast industry. These systems can edit raw films before broadcasting them. One or more audio-video clips are retrieved and then are edited. Several editing operations can be done to the clips, such as remove/add shots, remove/add audio channels, cut/paste, reorder, etc. The editor can process the clips in real-time by using patches via an Edit Decision List (EDL). EDL is a program that describes a list of operations to be exe-



(a) Priority Inversion (b) Deadline (c) Seek Time

**Figure 10. Effect of $R$ on $SFC_3$.**

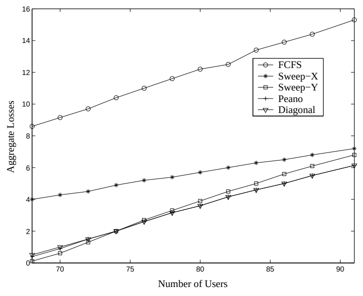


**Figure 11. The aggregate losses.**

cuted in a predefined order. A non-linear video editing server is similar to a video on demand (VoD) system in that both playback multiple streams with soft or hard real-time guarantees. However, a non-linear editing system supports a real-time disk write operation. Reading from an archive and non real-time FTP are also supported by a non-linear editing system.

In the experiments, we assume that 68 to 91 users are simultaneously accessing each disk in the server. Each user requests either to retrieve or download an MPEG-1 stream at 1.5 Mbps. Users send read or write requests periodically, and we assume that these requests arrive in bursts. This means that the disk scheduler serves the incoming requests in batches. The experiments are conducted using eight priority levels, with a normal distribution of requests across the different levels. We assume that each request must be serviced by the disk server before a certain deadline, selected randomly in the range (750-1500) milliseconds. A request not serviced prior to this deadline is considered lost.

The main performance metric that we measure is the fraction of read/write requests that are lost by the disk server. Since a scheduling algorithm may reduce the number of losses in one priority level at the expense of increasing this number in other levels, we use a cost function that combines the losses in all priority levels to compare the performance of the different algorithms. Our cost function is the weighted sum of the miss ratios (that is, the ratio of the number of misses to the total number of requests) in the different priority levels. The weights are selected to reflect the relative cost of missing deadlines across the different levels: $f = \sum_{i=1}^{P} w_i \frac{m_i}{r_i}$ where $p$ is the number priorities, $m_i$ is the number of misses at level $i$, $r_i$ is the total number of requests at level $i$, and $w_i$ is the weight of level $i$. In our experiment, we have selected the weights to decrease linearly with priority, such that data lost in the highest level has a cost 11 times that of the lowest level.

Figure 11 summarizes our results for measuring the request losses in the system. In the figure, we show the aggregate losses as the number of users increases, for five different scheduling algorithms: (1) FCFS, (2) Sweep-X; the space-filling curve scheduling algorithm where the Sweep curve is used with the priority assigned to the X-axis and the deadline assigned to the Y-axis, (3) Sweep-Y; the space-filling curve scheduling algorithm where Sweep is used with the deadline assigned to the X-axis and the priority assigned the Y-axis, (4) the space filling algorithm where the Peano curve is used with the priority assigned to the X-axis and the deadline assigned to the Y-axis, and (5) the space filling algorithm where the Diagonal curve is used with the priority assigned to the X-axis and the deadline assigned to the Y-axis. Note that Sweep-X is essentially the traditional Earliest Deadline First algorithm, which serves requests in increasing order of their deadlines regardless of the priority, whereas Sweep-Y corresponds to the multi-queue algorithm, which serves the requests in the highest-priority level according to their deadlines, followed by requests in the next lower level, and so on.

From the plot, we see that Diagonal and Peano SFCs behave the same. The Sweep-Y algorithm behaves better than all the other techniques, including Peano and Diagonal, when the load on the system is light. However, as the load on the system increases, meeting the deadline of all requests might not be possible, and the scheduler must make wise decisions on which requests to lose. When this is the case, Peano starts to outperform Sweep-X, and the gap between the two techniques increases with increasing the numbers of users. The Peano and the Diagonal SFCs can be thought of as a trade-off between the Sweep-X and Sweep-Y algorithms, as they try to balance the number of misses across the different levels, at the same time tending to favor the higher priority requests.

## 7 Conclusion

A new scalable multimedia disk scheduler, termed the *Cascaded-SFC*, is presented. The *Cascaded-SFC* multimedia disk scheduler is applicable in environments where disk requests have multiple QoS requirements. The main goal of *Cascaded-SFC* is to respect the priorities of different disk requests. In addition, *Cascaded-SFC* tries to minimize the number of deadline losses while maximizing the disk utilization. If a deadline loss should occur, *Cascaded-SFC* chooses a lower priority request as a victim. *Cascaded-SFC* is scalable in terms of the number of QoS parameters. Unlike previous disk scheduling algorithms, *Cascaded-SFC* is generic in the sense that it can be applied regardless of the number of parameters in the system. Furthermore, *Cascaded-SFC* can be tuned by simple parameters to emulate many other scheduling algorithms. Comprehensive experiments are presented to show the applicability and scalability of the *Cascaded-SFC* over other disk schedulers.

## References

[1] R. K. Abbot and H. Garcia-Molina. Scheduling i/o requests with deadlines: A performance evaluation. In *Proceedings of the IEEE Real-Time Systems Symbosium, RTSS*, pages 113–125, Dec. 1990.

[2] W. G. Aref, K. El-Bassyouni, I. Kamel, and M. F. Mokbel. Scalable qos-aware disk-scheduling. In *International Database Engineering and Applications Symposium, IDEAS*, July 2002.

[3] W. G. Aref, I. Kamel, and S. Ghandeharizadeh. Disk scheduling in video editing systems. *IEEE Trans. on Knowledge and Data Engineering, TKDE*, 13(6):933–950, 2001.

[4] M. A. Carey, R. Jauhari, and M. Livny. Priority in dbms resourse scheduling. In *VLDB*, pages 397–410, Aug. 1989.

[5] S. Chen, J. Stankovic, J. Krouse, and D. Towsley. Performance evaluaion of two new disk scheduling algorithms for real-time systems. *Journal of Real-Time Systems*, 3:307–336, 1991.

[6] S. J. Daigle. Disk scheduling for multimedia data streams. In *Proc. of SPIE Conf.on High-Speed Networking and Multimedia Computing*, volume 2188, pages 212–223, Apr. 1994.

[7] P. J. Denning. Effects of scheduling on file memory operations. *AFIPS Spring Joint Computer Conference*, pages 9–21, Apr. 1967.

[8] G. G. et. al. File server scaling with network-attached secure disks. In *SIGMETRICS*, pages 272–284, June 1997.

[9] J. R. Haritsa, M. J. Carey, and M. Livny. Value-based scheduling in real-time database systems. *VLDB Journal*, 2(2):117–152, 1993.

[10] Y. Ito and T. Tanaka. A video server using atm switching technology. In *5th International Workshop on Multimedia Communication*, pages 341–346, May 1994.

[11] I. Kamel and Y. Ito. Disk bandwidth study for video servers. Technical report, Matsushita Information Technology Laboratory, 1996.

[12] I. Kamel, T. Niranjan, and S. Ghandeharizedah. A novel deadline driven disk scheduling algorithm for multi-priority multimedia objects. In *ICDE*, pages 349–358, Mar. 2000.

[13] R. H. Katz. High performance network- and channel-attached storage. *Proceedings of IEEE*, 80(8), Aug. 1992.

[14] M. F. Khan, A. Ghafoor, and M. Ayyaz. Design and evaluation of disk scheduling policies for high-demand multimedia servers. In *ICDE*, pages 592–599, Mar. 1999.

[15] S. Khanna, M. Sebree, and J. Zolnowsky. Real-time scheduling in sunos 5.0. *In Proceedings of Winter USENIX Conference*, Jan. 1992.

[16] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environments. *Journal of ACM, JACM*, 20(1):46–61, Jan. 1973.

[17] M. F. Mokbel and W. G. Aref. Irregularity in multi-dimensional space-filling curves with applications in multimedia databases. In *CIKM*, Nov. 2001.

[18] A. Reddy and J. C. Wyille. Disk scheduling in multimedia i/o systems. In *Proceedings of the First ACM Multimedia*, pages 225–233, Aug. 1993.

[19] P. J. Shenoy and H. Vin. Cello: A disk scheduling framework for next generation operating systems. In *SIGMETRICS*, pages 44–55, June 1998.

[20] R. Wijayaratne and A. L. Reddy. Integrated qos management for disk i/o. In *ICMCS*, pages 487–492, Jan. 1999.

[21] D. K. Yau and S. S. Lam. Adaptive rate-controlled scheduling for multimedia applications. *IEEE/ACM Transactions on Networking*, 5(4):475–588, Aug. 1997.