Scalable QoS-Aware Disk-Scheduling

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

[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

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

Abstract

A new quality of service (QoS) aware disk scheduling algorithm is presented. It is applicable in environments where data requests arrive with different 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. A unified framework for QoS disk scheduling is presented that scales with the number of scheduling parameters. The general idea is based on modeling the disk scheduler requests as points in the multi-dimensional space, 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.). Then the disk scheduling problem reduces to the problem of finding a linear order to traverse these multi-dimensional points. Space-filling curves are adopted to define a linear order for sorting and scheduling objects that lie in the multi-dimensional space. This generalizes the one-dimensional disk scheduling algorithms (e.g., EDF, SATF, FIFO). Several techniques are presented to show how a QoS-aware disk scheduler deals with the progressive arrival of requests over time. Simulation experiments are presented to show a comparison of the alternative techniques and to demonstrate the scalability of the proposed QoS-aware disk scheduling algorithm over other traditional approaches.

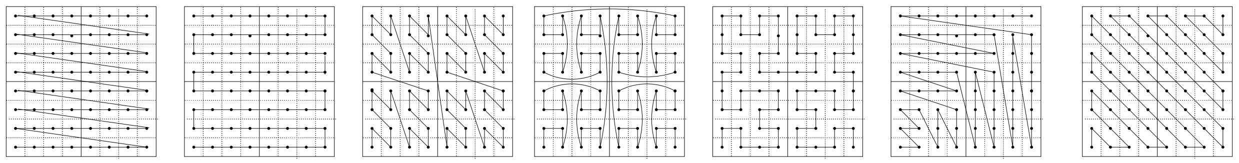
1. Introduction

Building reliable and efficient disk schedulers has always been a very challenging task. It has become even more so with today's complex systems and demanding applications. As applications grow in complexity, more requirements are imposed on disk schedulers, for example, the problem of disk scheduling in multimedia servers. In addition to maximizing the bandwidth of the disk, the disk scheduler has to take into consideration the real-time deadline constraints of the page requests, e.g., as in the case of video streaming. If clients are prioritized based on quality of service guarantees, then the disk scheduler might as well consider the priority of the requests in its disk queue. Writing a disk scheduler that handles real-time and quality of service constraints in addition to maximizing the disk bandwidth is a challenging and a hard task [2]. Similar issues arise when designing schedulers for multi-threaded CPUs, network-attached storage devices (NASDs) [9, 16], etc.

In the attempt to satisfy these concurrent and conflicting requirements, scheduler designers and algorithm developers depend mainly on heuristics to code such schedulers. It is not always clear that these schedulers are fair to all aspects of the system, or controllable in a measurable way to favor one aspect of the system over the other. The target of this paper is to revolutionize the way disk schedulers are developed. The general idea is based on modeling the disk requests as points in the multi-dimensional space where each dimension represents one of the parameters (e.g., one dimension represents the request deadline, another represents the disk cylinder number and the third dimension represents the priority of the request, etc.). Then the scheduler problem reduces to finding a linear order to traverse these multi-dimensional points.

The underlying theory is based on space-filling curves (SFCs). A space-filling curve maps the multi-dimensional space into the one-dimensional space. It 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. For example, in a QoS-aware disk scheduler, when a request arrives to the disk queue, the request's parameters (e.g., its disk cylinder number, its real-time deadline, etc.) are passed as arguments to the space-filling curve function, which returns a one-dimensional value that represents the location of the re-

*This reasearch is supported by the National Science Foundation NSF under Grant No. IIS-0093116



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

Figure 1. Two-dimensional Space-Filling Curves.

quest in the disk queue. As a result, the disk queue is always sorted in the specified space-filling curve order. Using space-filling curves as the bases for multi-parameter disk scheduling has numerous advantages, including scalability (in terms of the number of scheduling parameters), ease of code development, ease of code maintenance, the ability to analyze the quality of the schedules generated, and the ability to automate the scheduler development process in a fashion similar to automatic generation of programming language compilers.

The rest of this paper is organized as follows. Section 2 discusses the related work of disk scheduling and the use of space-filling curves in different applications. In Section 3, we develop new space-filling curve based disk scheduling algorithms. Section 4 adopts the notion of *irregularity* as a measure of the quality of the scheduled order provided by a space-filling curve. Section 5 presents a comprehensive study of the developed algorithms on different space-filling curves. Finally, Section 6 concludes the paper.

2. Related Work

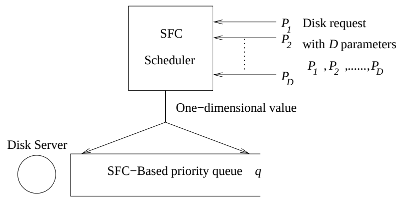
The problem of scheduling a set of tasks with time and resource constraints is known to be NP-complete [19]. Several heuristics have been developed to approximately optimize the scheduling problem. Traditional disk scheduling algorithms [5, 8, 12, 28] are optimized for aggregate throughput. These algorithms, including SCAN, LOOK, C-SCAN, and SATF (Shortest Access Time First), aim to minimize seek time and/or rotational latency overheads. They offer no QoS assurance other than perhaps absence of starvation. Deadline-based scheduling algorithms [1, 4, 15, 25] have built on the basic earliest deadline first (EDF) schedule of requests to ensure that deadlines are met. These algorithms, including SCAN-EDF and feasible-deadline EDF, perform restricted reorderings within the basic EDF schedule to reduce disk head movements while preserving the deadline constraints.

Like previous work on QoS-aware disk scheduling, space-filling curves explicitly recognize the existence of multiple and sometimes antagonistic service objectives in the scheduling problem. Unlike previous work that focuses on specific problem instances, we use a more general model of mapping service requests in the multi-dimensional space into a linear order that balances between the different dimensions. Disk schedulers based on space-filling curves generalize traditional disk schedulers. For example, SATF can be modeled by the Sweep SFC (Figure 1a) by assigning the access time to the vertical dimension. Similarly, EDF is modeled by the Sweep SFC by assigning the deadline to the vertical dimension.

Space-Filling curves are first discovered by Peano [24] where he introduced a mapping from the unit interval to the unit square (Figure 1c). Hilbert [11] generalizes the idea for a mapping of the whole space (Figure 1e). Following the Peano and Hilbert curves, many space-filling curves have been proposed, e.g., see [3, 6]. In this paper, we focus on the space-filling curves shown in Figure 1, namely the Sweep, C-Scan, Peano, Hilbert, Gray, Spiral, and Diagonal SFCs. However, the developed theory and scheduling algorithms apply to other space-filling curves. Space-filling curves are used in many applications in computer science and engineering fields, e.g., spatial join [22], range queries [13], spatial access method [7], R-Tree [14], multi-dimensional indexing [18], and image processing [29]. Up to the authors' knowledge using space-filling curves as a scheduling tool is a novel application.

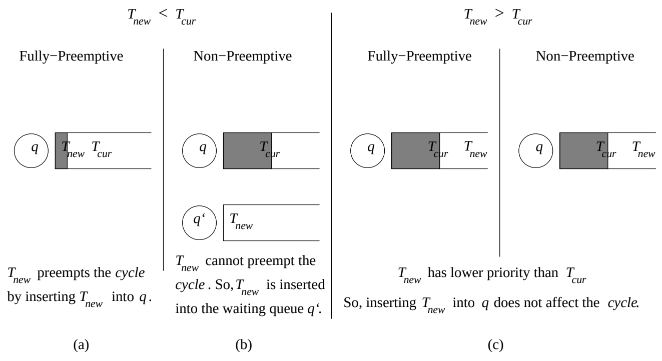
3. Disk-Scheduling Algorithms based on Space-Filling Curves

In the QoS-aware disk scheduler, a disk request is modeled by multiple parameters, (e.g., the disk cylinder, the real-time deadline, the priority, etc.) and represented as a point in the multi-dimensional space where each parameter corresponds to one dimension. Using a space-filling curve, the multi-dimensional disk request is converted to a one-dimensional value. Then, disk requests are inserted into a priority queue q according to their one-dimensional value with a lower value indicating a higher priority. Figure 2 gives an illustration of an SFC-based disk scheduler. To help in understanding the proposed algorithms, we present



SFC Scheduler
P_1
P_2
P_D
Disk request with D parameters
$P_1, P_2, \ldots\ldots, P_D$
One−dimensional value
Disk Server
SFC−Based priority queue q

Figure 2. SFC-based Disk Scheduler



$T_{new} < T_{cur}$
$T_{new} > T_{cur}$
Fully−Preemptive
Non−Preemptive
Fully−Preemptive
Non−Preemptive
q
T_{new} T_{cur}
q'
T_{new}
T_{new} preempts the cycle by inserting T_{new} into q.
T_{new} cannot preempt the cycle. So, T_{new} is inserted into the waiting queue q'.
T_{new} has lower priority than T_{cur}
So, inserting T_{new} into q does not affect the cycle.
(a)
(b)
(c)

Figure 3. The Non-Preemptive and Fully-Preemptive SFC disk schedulers.

the notion of a full *cycle* in a space-filling curve.

Definition 3.1: *A full cycle in a space-filling curve with N priority levels in each dimension of a D-dimensional space is a contiguous move from the first point, say point 0, to the last point, say point N^D, passing through all the points in the space exactly once*.

A disk request T takes a position in the *cycle* according to its space-filling curve value. Disk requests are stored in the priority queue q according to their *cycle* position. The disk server walks through a *cycle* by serving all disk requests in q according to their *cycle* position. Figure 3 presents two straightforward approaches of using space-filling curves in disk-scheduling.

The Non-Preemptive SFC Disk Scheduler: In this approach, once the disk server starts to walk through a full *cycle* of a space-filling curve, the *cycle* is never preempted. A newly arrived request T_{new} is inserted in the disk queue q if and only if it will not preempt the current *cycle* (Figure 3c). If T_{new} needs to preempt the *cycle* (i.e., T_{new} has higher priority than T_{cur}), then it is inserted in a waiting queue q' (Figure 3b). The *cycle* is finished when the disk request with the lowest priority in q is served, then, all requests from q' are moved to q and a new *cycle* is generated.

The Fully-Preemptive SFC Disk Scheduler: This is the simplest approach. All requests are inserted into a single disk queue q according to their space-filling curve priority. This scheduler is fully-preemptive in the sense that any incoming request T_{new} with higher priority than T_{cur} preempts the current *cycle* and starts a new one (Figure 3a). However, when T_{new} has lower priority than T_{cur}, it is inserted in q without affecting the current *cycle* (Figure 3c).

The fully-preemptive SFC 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 SFC disk scheduler does not lead to starvation since it guarantees that lower priority disk requests in a certain *cycle* will be served before starting a new *cycle*. However, a priority inversion takes place where higher priority disk requests may wait for lower priority disk requests to be served. The drawbacks of the two approaches raise the motivation for having a combined disk scheduler that has the merits of both schedulers. In the following section, we present a novel disk scheduling algorithm that avoids the drawbacks of these algorithms, i.e., respects the disk request priority and avoids starvation.

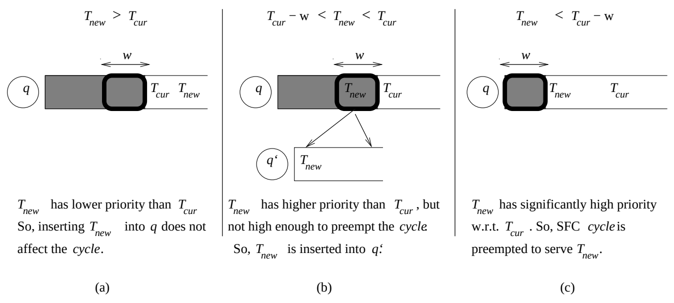
3.1. The Conditionally-Preemptive SFC Disk Scheduler Algorithm

As a trade-off between the fully-preemptive and the non-preemptive disk schedulers, in the conditionally-preemptive disk scheduling algorithm, a newly arrived disk request T_{new} preempts the process of walking through a full *cycle* if and only if it has significantly higher priority than the currently served disk request T_{cur}. To quantify the meaning of significantly higher priority, we define a blocking window with size w (the rounded box with thick border in Figure 4) that slides with T_{cur} in q. Then, T_{new} is considered as a priority significantly higher 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. When T_{new} arrives while the scheduler is going to serve T_{cur}, then one of the following three cases takes place:

1. $T_{cur} < T_{new}$ (Figure 4a). This means that T_{new} has lower priority than T_{cur}. Hence, T_{new} is inserted into q as inserting it into q will not preempt the *cycle*.

2. $T_{cur} - w < T_{new} < T_{cur}$ (Figure 4b). This means that T_{new} lies inside the blocking window w. Although T_{new} has a priority higher than that of T_{cur}, but it is not high enough to preempt the space-filling curve *cycle*. So, T_{new} is inserted in the waiting queue q'.

3. $T_{new} < T_{cur} - w$ (Figure 4c). This means that T_{new} has a priority that is significantly higher than that of



T_new > T_cur
w
q
T_cur T_new
T_new has lower priority than T_cur
So, inserting T_new into q does not affect the cycle.
(a)
T_cur − w < T_new < T_cur
w
q
T_new T_cur
q'
T_new
T_new has higher priority than T_cur, but not high enough to preempt the cycle
So, T_new is inserted into q'
(b)
T_new < T_cur − w
w
q
T_new T_cur
T_new has significantly high priority w.r.t. T_cur. So, SFC cycle is preempted to serve T_new.
(c)

Figure 4. The Conditionally-Preemptive SFC Disk Scheduler.

T_{cur}. So, it is worth to preempt the space-filling curve *cycle* by inserting T_{new} in q.

There are two issues that need to be addressed; first, how to deal with the occurrence of priority inversion that result from disk requests that lie inside the blocking window w (stored in q‘) and have higher priority than some requests in q. Second, with any value of w less than N^D (the last point in a *cycle*), there is still a chance of starvation, where a continuous stream of very high priority requests may arrive. The next two sections propose alternative approaches for dealing with these two problems.

3.2. Minimizing Priority Inversion

The disk requests that lie in window w are stored in q‘. This results in priority inversion as the blocked requests have higher priority than T_{cur}. In this section, we propose three alternatives to deal with this situation. Figure 5 gives an example that demonstrates the difference among the three proposed scheduling policies. Assume that while T_1 is being served, all the other disk requests T_2, T_3, T_4, T_5, T_6, and T_7 have arrived. Notice that T_2, T_3, and T_4 are inserted in q‘ since they lie inside the window w. T_6 and T_7 are inserted in q since they have lower priority than T_1. T_5 is inserted in q since it has a significantly higher priority than T_1.

Serve and Resume (SR): The space-filling curve *cycle* is preempted only by inserting the newly arriving request T_{new} of significant high priority into q. After preempting the *cycle* and serving T_{new}, the process of serving the *cycle* is resumed.

As in Figure 5, after serving T_1, following the *cycle* order would result in serving T_6. However, the *cycle* is preempted to serve T_5 (T_5 has a significantly higher priority than T_6). After serving T_5, the *cycle* is resumed to serve the disk requests in q (T_6 and T_7). Finally, the next *cycle* (waiting) queue q‘ is considered and is served. Hence, the final order is T_1, T_5, T_6, T_7, T_2, T_3, T_4.

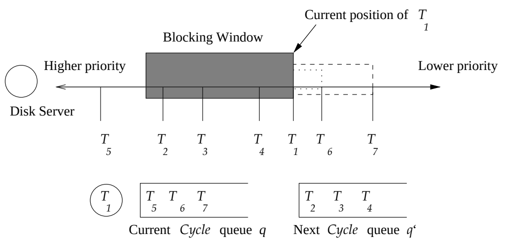


Figure 5. Example of Conditionally-Preemptive SFC Disk Scheduler.

Serve, Resume and Promote (SRP): SRP acts exactly as SR. In addition, before the disk starts to serve a request from q, it checks q‘ for any request that becomes with a significantly higher priority. If such a request is found, SRP promotes this request and inserts it in q. So, the space-filling curve *cycle* can be preempted either by a newly arrived request or by an old request that eventually becomes of significant higher priority.

In Figure 5, after serving T_1, the *cycle* is preempted to serve T_5. Then, before serving T_6, SRP detects that T_2 now lies outside the window of T_6. Hence, T_2 is served before T_6. Continuing in this way, the final order will be T_1, T_5, T_2, T_6, T_3, T_7, T_4.

Serve and Scan (SS): When the *cycle* is preempted due to the arrival of a new disk request T_{new}, all the requests in q‘ are scanned and served in their priority order.

In Figure 5, when the *cycle* is preempted to serve T_5, all the disk requests inside the window (next *cycle* queue q‘) are served before returning to the current *cycle* queue. Hence, the final order will be T_1, T_5, T_2, T_3, T_4, T_6, T_7.

3.3. Starvation Avoidance

If the window size w remains fixed, an adversary would still select disk requests in a manner that results in a starvation of other disk requests. To avoid starvation, we propose to expand the window size w during the course of executing the scheduling algorithm. As w increases, it eventually becomes large enough to prevent preemption and hence avoids starvation. In this section, we propose two policies for expanding the window size w.

Always Expand (AE): In AE, the window size w is increased by a constant factor, expansion factor e, with any preemption of the space-filling curve *cycle*. Eventually, w will be large enough to prevent any *cycle* preemption and hence, the disk scheduler works as the non-preemptive disk scheduling algorithm which avoids starvation.

Expand and Reset (ER): ER is the same as AE where we increase the window size w by a constant factor e. However, when a disk request is served and another disk request from

q is dispatched, ER resets w to its original value. The objective is to achieve a balance between the non-preemptive and the fully-preemptive schedulers. While in AE, once a scheduler becomes a non-preemptive one (due to the increase of w), it continues to work as the non-preemptive scheduler, in ER, the scheduler moves back and forth between working as the non-preemptive scheduler and as the conditionally-preemptive scheduler with different values of w.

4. The Quality of Space-Filling Curves

An optimal space-filling curve is one that sorts points in space in ascending order for all dimensions. In reality, when a space-filling curve attempts to sort the points in ascending order according to one dimension, it fails to do the same for the other dimensions. A good space-filling curve for one dimension is not necessarily good for the other dimensions. In this section, we introduce the concept of *irregularity* as a measure of goodness of space-filling curves [20]. Then, we show how the irregularity can be used as an indicator for the practical performance measures, e.g., disk utilization, priority inversion, and deadline losses.

4.1. Irregularity in Space-Filling Curves

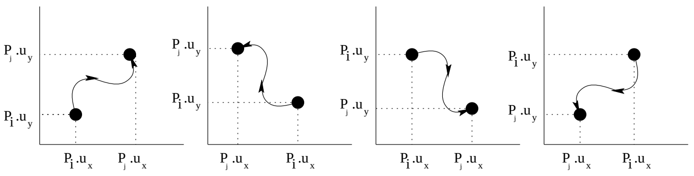
In order to measure the scheduling quality of a space-filling curve, we introduce the concept of irregularity as a measure of goodness for the scheduling order imposed by a space-filling curve. Irregularity is measured for each dimension separately. It gives an indicator of how a space-filling curve is far from the optimal. The lower the irregularity, the better the space-filling curve is.

Definition 4.1: *For any two points, P_i and P_j, in the D-dimensional space with coordinates $(P_i.u_1, P_i.u_2, \ldots, P_i.u_D), (P_j.u_1, P_j.u_2, \ldots, P_j.u_D)$, respectively, and for a given space-filling curve, if P_i is visited before P_j, we say that an irregularity occurs between P_i and P_j in dimension k iff $P_j.u_k < P_i.u_k$.*

Figure 6 demonstrates all possible scenarios that can lead to an irregularity in the two-dimensional space, where the arrows in the curves indicate the order imposed by the underlying space-filling curve, i.e., point P_i is visited before point P_j. Formally, for a given space-filling curve in the D-dimensional space with grid size N, the number of irregularities for any dimension k is:

$$I(k,N,D) = \sum_{j=1}^{N^D} \sum_{i=1}^{j-1} f_{ij} \ , \quad f_{ij} = 1 \ iff \ P_i.u_k > P_j.u_k$$

An optimal schedule for any dimension k would have no irregularity. In contrast, the worst-case schedule for any dimension k is to sort all the requests in reverse order with respect to k.



(a) No Irregularity in x, y (b) Irregularity in x only (c) Irregularity in y only (d) Irregularity in x, y

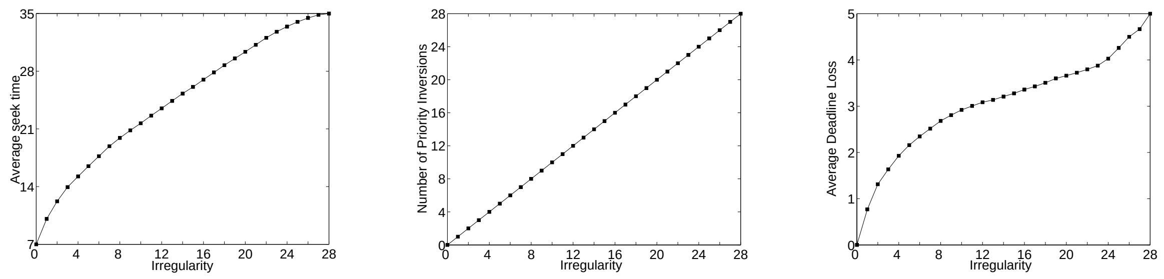
Figure 6. Irregularity in 2D space.

4.2. Irregularity as a Measure of Performance

In this section, we show that the irregularity can be used as a practical measure of performance. Three experiments have been conducted to show the effect of lower irregularity on disk utilization, priority inversion, and deadline losses. In the experiments, we assume eight priority levels with one disk request for each level. This results in 8! possible different schedules. The optimal schedule would have no irregularity while the worst-case schedule would have 28 irregularities [20].

4.2.1 Disk Utilization

In this section, we investigate the correlation between irregularity as a measure of goodness and disk utilization. We conduct the following experiment. Assume that we have a disk with eight consecutive disk cylinder zones, say C_1 to C_8. We map these consecutive cylinder zones to eight levels of priority in the irregularity frame of work. Assume that each cylinder zone contains one disk request. The objective is to serve all disk requests while minimizing seek time overhead (i.e., increasing the disk utilization). The disk head is initially located at the first cylinder C_1. The seek time between any two consecutive cylinders is T_s. Irregularity is computed based on the shortest possible seek time from the current location. For example, the best schedule is $C_1C_2C_3C_4C_5C_6C_7C_8$ which results in a seek time of $7T_s$ and has zero irregularity. The worst-case schedule is $C_8C_1C_7C_2C_6C_3C_5C_4$ where each time the scheduler chooses the furthest cylinder to serve, this results in 28 irregularities and a seek time of $35T_s$. Figure 7a gives the relation between irregularity and disk utilization, where for each possible number of irregularities I (varies from 0, the optimal, to 28, the worst), we compute the average seek time over all schedules that result in I irregularity. From Figure 7a, we notice that the average seek time and the irregularity in a sequence of disk request schedule are almost linearly correlated, i.e., the lower the irregularity the better the disk utilization is and vise versa. Therefore, we can deduce that irregularity can be used as a measure of goodness for disk performance. The advantage of using irregularity as a measure of goodness is that we can compute it analytically, and hence be able to analytically quantize the



(a) Seek time (b) Priority (c) Deadline

Figure 7. Irregularity as a practical measures.

scheduling quality for a given scheduling policy.

4.2.2 Priority Inversion

Priority inversion takes place when a higher priority disk request is waiting for a lower priority disk request to be served. In this experiment, we assume that all disk requests lie in the same cylinder, so there is no seek time overhead. There are eight levels of priorities P_1 to P_8, and one disk request per priority level. When a disk request with priority P_i is served, we compute the priority inversion as the number of disk requests with priority $P_j > P_i$ that are waiting for P_i to be served. Figure 7b gives the effect of irregularity on priority inversion. As can be seen from the figure, irregularity and priority inversion are linearly correlated. The lower the irregularity, the lower the occurrence of priority inversion is and vise versa. Hence, irregularity may be used as a good performance measure that reflects the quality of a schedule generated by a given scheduler w.r.t. priority inversion.

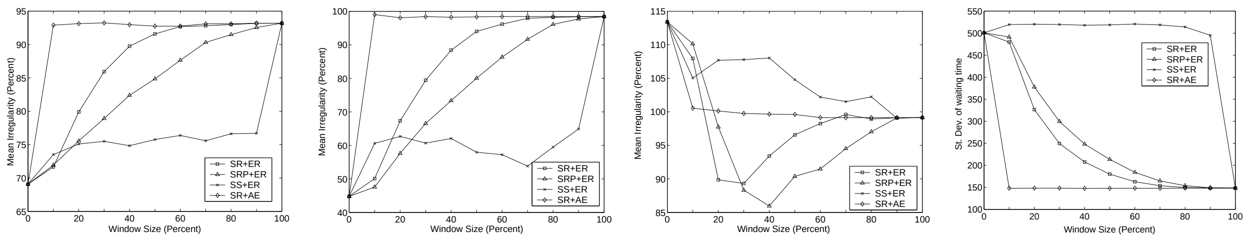
4.2.3 Deadline Loss

In this experiment, we assume that we have eight priority levels from $P_1 = 1$ to $P_8 = 8$, and there is one disk request per priority level. Assume that each disk request needs constant service time, say a msec. Assume further that higher priority requests have more tight deadlines, so, the deadline for each request is bP_i where $b > a$. Notice that $b \leq a$ means relaxed deadlines, and hence no deadline violation would take place. We conduct this experiment in the following way: For each possible number of irregularities I (varies from 0, the optimal, to 28, the worst), we compute the average deadline losses over all schedules that result in I irregularity. Figure 7c gives the relationship between irregularity and the number of deadline losses where we set a=20 msec and b=25 msec. The same figure is obtained for any values for a, b where $b > a$. From the figure, notice that the lower the irregularity, the lower the deadline losses, and vise versa. Also, the figure demonstrates a linear correlation between irregularity and the number of deadline losses. Hence, irregularity can be used as a measure of goodness for deadline loss performance as well. Therefore, lowering the irregularity is favorable in the case of real time applications.

5. Performance Evaluation

The SFC-based disk scheduler has three major components; the irregularity policy, the starvation policy, and the underlying space-filling curve, and two parameters; the window size w and the expansion factor e. In this section, we perform comprehensive experiments to construct an SFC-based disk scheduler by appropriately choosing its components and parameters. In Section 5.1, we perform experiments to evaluate all the proposed policies for minimizing irregularity and avoiding starvation In Section 5.2, we study the effect of each space-filling curve on the scheduler In Section 5.3, we study the effect of the initial window size w For all experiments we set the expansion factor e to be 5% of w. All experiments in this section are performed with the disk simulation model developed at Dartmouth College [17]. It simulates the Hewlett Packard 97560 disk drive [23] that is described in detail in [27]. This disk simulation model has been widely used in many projects, e.g., in the SimOS project at Stanford University [10, 26] and in the Galley project at Dartmouth College [21]. The HP 97560 disk drive contains 1962 cylinder with 19 tracks per cylinder. Each track contains 72 sectors with 512 bytes each. The revolution speed is 4002 rpm and the disk has a SCSI-II controller interface.

To reflect the irregularity of the SFC scheduler, we measure the mean irregularity over all the space dimensions. The standard deviation of request waiting time is considered as a measure of starvation, the higher the standard deviation the higher the chance that starvation may occur. For



(a) Sweep SFC (b) Diagonal SFC (c) Gray SFC (d) Sweep SFC

Figure 8. Comparison among different policies.

comparison purposes, we use the FCFS scheduler as our base point. The irregularity and the standard deviation of the waiting time are presented as a ratio to the irregularity and the standard deviation of the waiting time caused by FCFS, respectively. Recall that traditional disk schedulers, e.g., EDF and SATF can be modeled as special cases of an SFC-based disk scheduler. Hence, we do not have to compare with each one of them separately.

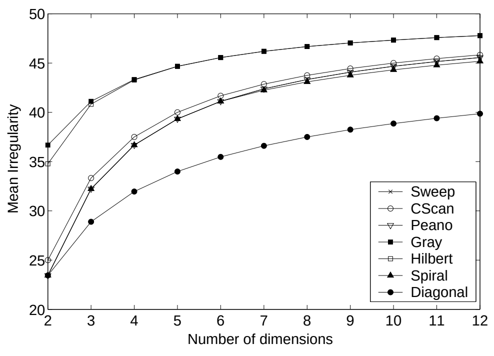
5.1. Selecting the Policy

In this section, we compare the proposed algorithms in Sections 3.2 and 3.3. All experiments consider disk requests with four QoS parameters that arrive exponentially with mean interarrival time 25 msec. The initial window size w is expressed as a percentage of the total points in the space. The expansion factor e is set to 5% of w. The notation ”+” is used to indicate a combination of two disk scheduling policies. For example SR+AE indicates that the disk scheduler uses Policies SR (Serve and Resume) to handle irregularity and AE (Always Expand) to handle starvation. Figures 8a, 8b, and 8c give the mean irregularity for the Sweep, Diagonal, and Gray SFCs, respectively at different values for w. To simplify the graph, we only plot SR+AE as a representative of disk schedulers that use policy AE. Other disk schedulers that use the same policy (AE) give the same performance as SR+AE. At w=0, all the schedulers degenerate to the Fully-Preemptive SFC disk scheduler. Similarly, at w=100, all the schedulers degenerate to the Non-Preemptive SFC disk scheduler. Except for the case where w=0, the AE (Always Expand) Policy results in very high irregularity even with small window size. The reason is that w is always increasing and eventually it becomes large enough to block all incoming disk requests as in the non-preemptive scheduler.

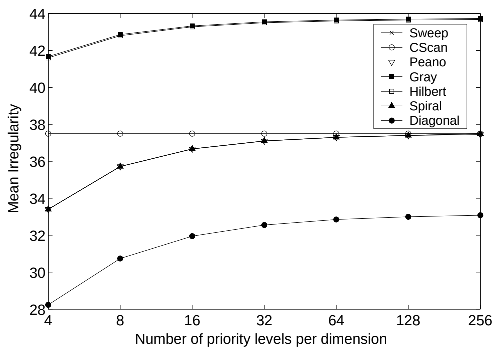
In Figures 8a and 8b, Policy SS gives the lowest irregularity when $w > 25\%$. As w increases, more disk requests are blocked and stored in the disk queue q‘. The blocked disk requests are served according to their SFC order. Thus, respecting the SFC-order lowers the irregularity. SR+ER and SRP+ER give reasonable increase in irregularity as w increases. The Spiral and Peano SFCs exhibit similar behavior as the Sweep and Diagonal SFCs. The Diagonal SFC gives the lowest irregularity for any window size w. However, the choice of the appropriate space-filling curve does not rely only on irregularity. Other aspects that control the choice of a space-filling curve are investigated in the next section. Figure 8c represents the performance of the Gray, C-Scan, and Hilbert SFCs. Unlike the Sweep and Diagonal SFCs, in SR+ER and SRP+ER policies, increasing w from 0 to 40 results in lowering the irregularity.

Figure 8d gives the standard deviation of the waiting time for the Sweep SFC. All space-filling curves give the same curve for waiting time. SS+ER gives very high standard deviation, which indicates a high possibility of starvation. In Policy SS, disk requests that are blocked by window w are accumulated and stored in the queue q‘. So serving them in one scan may result in starving lower priority requests. SR+AE works as the non-preemptive scheduler. SRP+ER and SR+ER give lower starvation as w increases.

As can be seen from the experiments, SR+ER and SRP+ER give the best scheduling performance where they result in a moderate schedule that balances between irregularity and starvation. However, Policy SR has a vital drawback, that it penalizes disk requests for their early arrival. Assume that a disk request T arrives within window w, then it is stored in q‘. The service of T is postponed till all disk requests in q are served. If T was smart enough to delay itself so that it arrives when the blocking window w slides ahead so that T becomes outside w and stored in q, then it will be eligible for being served immediately. This scenario highlights the fact that T may be served better if it arrives late. This problem is dealt with in Policy SRP, that after serving each request, SRP checks the queue q‘ for those requests that become eligible to service and move (promote) them to q. For the rest of experiments in the following sections, we use the Policies SRP+ER in the Conditionally-Preemptive SFC disk scheduler.



(a) Dimensions



(b) Priority levels

Figure 9. Scalability of SFC Scheduler.

5.2. Selecting the Space-Filling Curve

In this section, we perform comprehensive comparison between the seven space-filling curves in Figure 1. The objective is to determine which space-filling curve will best fit in the SFC-based disk scheduler. All experiments in this section are performed with SRP+ER policies.

5.2.1 Scalability of SFC-based Schedulers

In this section, we address the issue of scalability of SFC-based schedulers, e.g., when the number of dimensions (schedule parameters) increases or when the number of priority levels per dimension increases. The experiments in this section are performed with SRP+ER policies with $w = 50\%$, and the mean interarrival time of disk requests is 25 msec. In Figure 9a, we measure the irregularity of the SFC-based disk scheduler using different space-filling curve for up to 12 QoS parameters (scheduling dimensions) where each dimension has 16 priority levels. The Diagonal SFC gives the best performance especially with higher dimensions. The Sweep, Peano, and Spiral SFCs have almost the same performance.

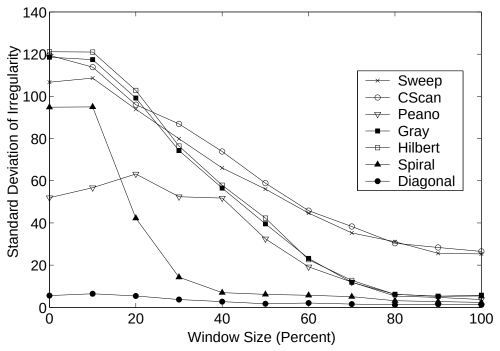
Figure 9b compares the space-filing curves in the four-dimensional space, while the number of priority levels varies from 4 to 256. After 16 priority levels, all space-filling curves tend to exhibit constant behavior. The Diagonal SFC gives the best performance. The Hilbert and Gray SFCs have the worst performance with respect to irregularity. The Sweep, Peano, and Spiral SFCs have similar performance that tends to be equal to the performance of the C-Scan SFC in high priority levels. The C-Scan SFC has constant performance regardless of the number of priority levels.

From the experiments, it can be seen that the SFC-based disk scheduler scales easily and without additional coding difficulty to higher QoS parameters. Also when using the appropriate SFC, the SFC-based scheduler can exhibit low irregularity even at higher dimensions. The time complexity for converting a point in the D-dimensional space into the one-dimensional space is $O(D)$ [20].

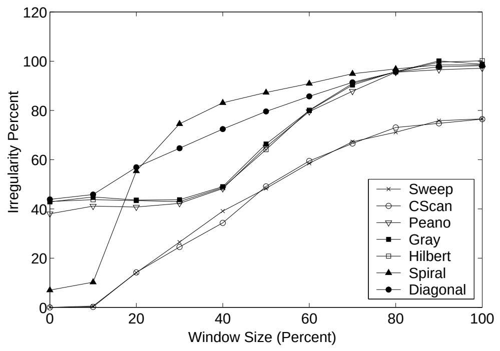
5.2.2 Fairness of SFC-based Schedulers

A very critical point for SFC-based disk schedulers is how to assign the QoS disk request parameters (i.e., the deadline, priority, etc.) to the dimensions of the space-filling curve. For example, the EDF disk scheduler can be modeled by the Sweep SFC when assigning the vertical dimension (Figure 1a) to the deadline parameter. Also SATF can be modeled using the Sweep or the C-Scan SFC by assigning the vertical dimension to the access time. We say that a space-filling curve is biased to dimension k if it results in low irregularity in k relative to the other dimensions. Also, we say that a space-filling curve is fair if it results in similar irregularity for all dimensions. In this section, we use the standard deviation of irregularity over all the dimensions as a measure of the fairness of space-filling curves. The experiment in this section is performed on four QoS parameters using SRP+ER policies with $w = 50\%$, and the interarrival time of disk requests is 25 msec. In Figure 10a, we measure the standard deviation of irregularity over all dimensions. A low standard deviation indicates more fairness. 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. This is because they have no irregularity in the last dimension while having high irregularity in the other dimensions.

Some applications may have only one important dimension, while the other dimensions are not with the same sig-



(a) Standard Deviation



(b) Favored Dimension

Figure 10. Fairness of SFC Scheduler.

nificant importance. One example is optimizing the mechanical movements of the disk head over the cylinders. Another example is the real time requests, where in some applications the most important factor would be to meet the request deadline, and then the other parameters can be scheduled. EDF favors the deadline, while ignoring all other dimensions. CSCAN favors the cylinder dimension. SATF favors the access time dimension. For these applications, we develop the experiment given in Figure 10b. Although, we run the experiment in a four-dimensional QoS space, we plot only the most favored dimension for each space-filling curve. Figure 10b shows that the C-Scan and the Sweep SFCs are always the best for a small window size. They have no irregularity in small window sizes. This is also an interpretation of why they have very high standard deviation (Figure 10a).

5.3. Selecting the Initial Window Size

In this set of experiments, we investigate how the value of the window w can be determined. We develop a design curve for each space-filling curve that demonstrates the effect of changing w on the irregularity and starvation in the three-, four-, and five-dimensional spaces. We use the same experiment as in Section 5.1 while varying the number of dimensions from three to five. Figure 11 gives the design curves for the Peano, Hilbert, and Diagonal SFCs, respectively. The C-Scan and the Gray SFCs have similar shapes as that of the Hilbert SFC. All other SFCs have similar design curves as that of the Peano and Diagonal SFCs with different irregularity values.

Determining the value of the window size w depends on the space-filling curve. For the Peano SFC, setting w=35% results in the best trade-off between the irregularity and the standard deviation of waiting time. For the Hilbert and Diagonal SFCs, setting w=35%, 40%, respectively would result in the best trade-off.

6. Conclusion

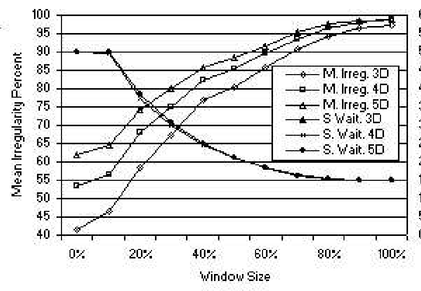
In this paper, we have proposed a new scalable disk scheduling algorithm for serving requests that require QoS parameters (i.e., deadline, priority, etc.). The idea is to map the multiple QoS parameters into the one-dimensional space. Then, we use the ordering imposed by space-filling curves to serve the disk requests. We introduce the *irregularity* as a measure of quality of the space-filling curve order. We show how irregularity is linearly correlated with other measures of goodness for scheduler performance, e.g., disk utilization, deadline losses, and priority inversion. The window size tuning parameter w is introduced to tune the irregularity and starvation of an SFC-based disk scheduler. Our comprehensive simulation experiments show that using the disk-scheduling algorithm SRP+ER achieves the best performance for any space-filling curve. From the set of the discussed space-filling curves, we show the different properties that motivates the use of each space-filling curve.

References

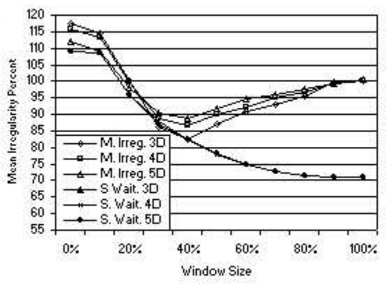
[1] R. K. Abbot and H. Garcia-Molina. Scheduling i/o requests with deadlines: A performance evaluation. In *Proc. of the IEEE Real-Time Systems Symp., RTSS*, pages 113–125, Florida, Dec. 1990.

[2] 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.

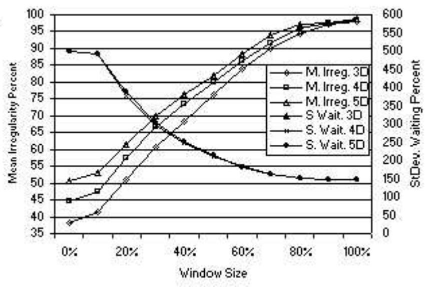
[3] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmayer. Space-fi lling curves and their use in the design of geometric data structures. *Theoretical Computer Science, TCS*, 181(1):3–15, 1997.



(a) Peano



(b) Hilbert



(c) Diagonal

Figure 11. Design Curves.

[4] 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.

[5] E. G. Coffman, L. Klimko, and B. Ryan. Analysis of scanning policies for reducing seek times. *SIAM Journal on Computing*, 1(3):269–279, Sept. 1972.

[6] C. Faloutsos. Multiattribute hashing using gray codes. In *Proc. of the Int. Conf. on Management of data, ACM SIGMOD*, pages 227–238, Washington D.C., May 1986.

[7] C. Faloutsos and Y. Rong. Dot: A spatial access method using fractals. In *Proc. of the Int. Conf. on Data Engineering, ICDE*, pages 152–159, Kobe, Japan, Apr. 1991.

[8] R. Geist and S. Daniel. A continuum of disk scheduling algorithms. *ACM Trans. of Computer Systems, TOCS*, 5(1):77–92, Feb. 1987.

[9] G. Gibson, D. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka. File server scaling with network-attached secure disks. In *In Proc. of the ACM Int. Conf. on Measurement and Modeling of Computer Systems, SIGMETRICS*, pages 272–284, Seatle, Washington, June 1997.

[10] S. A. Herrod. *Using Complete Machine Simulation to Understand Computer System Behaviour*. PhD thesis, Stanford University, Feb. 1998.

[11] D. Hilbert. Ueber stetige abbildung einer linie auf ein flshenstuck. *Mathematishe Annalen*, pages 459–460, 1891.

[12] M. Hofri. Disk scheduling: Fcfs vs sstf revisited. *Communications of the ACM, CACM*, 23(11):645–653, Nov. 1980.

[13] H. V. Jagadish. Linear clustering of objects with multiple attributes. In *Proc. of the Int. Conf. on Management of data, ACM SIGMOD*, pages 332–342, Atlantic City, NJ, June 1990.

[14] I. Kamel and C. Faloutsos. Hilbert r-tree: An improved r-tree using fractals. In *Proc. of the 20th Int. Conf. on Very Large Data Bases, VLDB*, pages 500–509, Santiago, Chile, Sept. 1994.

[15] I. Kamel, T. Niranjan, and S. Ghandeharizedah. A novel deadline driven disk scheduling algorithm for multi-priority multimedia objects. In *Int. Conf. on Data Engineering, ICDE*, pages 349–358, San Diego, CA, Mar. 2000.

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

[17] D. Kotz, S. Toh, and S. Radhakrishnan. A detailed simulation model of the hp97560 disk drive. Technical Report PCS-TR94-220, Department of Computer Science, Dartmouth College, 1994.

[18] J. K. Lawder and P. J. H. King. Querying multi-dimensional data indexed using the hilbert space fi lling curve. *SIGMOD Record*, 30(1), Mar. 2001.

[19] J. K. Lenstra, A. R. Kan, and P.Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[20] M. F. Mokbel and W. G. Aref. Irregularity in multi-dimensional space-fi lling curves with applications in multimedia databases. In *Proc. of the Int. Conf. on Information and knowledge Management, CIKM*, Atlanta, GA, Nov. 2001.

[21] N. Nieuwejaar. *Galley: A New Parallel File System for Scientifi c Applications*. PhD thesis, Computer Science Department, Dartmouth College, 1996.

[22] J. A. Orenstein. Spatial query processing in an object-oriented database system. In *Proc. of the Int. Conf. on Management of data, ACM SIGMOD*, pages 326–336, Washington D.C., May 1986.

[23] H. Packard". *HP97556/58/60 5.25 inch SCSI Disk Drive*, technical reference manual, 2nd edition edition, June 1991.

[24] G. Peano. Sur une courbe qui remplit toute une air plaine. *Mathematishe Annalen*, 36:157–160, 1890.

[25] A. Reddy and J. C. Wyille. Disk scheduling in multimedia i/o systems. In *Proc. of the 1st ACM Multimedia*, pages 225–233, Anaheim, CA, Aug. 1993.

[26] M. Rosenblum, S. Herrod, E. Witchel, and A. Gupta. Complete computer simulation: The simos approach. In *Proc. IEEE Parallel and Distributed Technology*, 1995.

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

[28] A. Silberchatz and P. Galvin. *Operating System Conceps*. Addison-Wesley, 5th edition, 1998.

[29] L. Velho and J. Gomes. Digital halftoning with space fi lling curves. *Computer Graphics*, 25(4):81–90, July 1991.