Robust, Portable I/O Scheduling with the Disk Mimic

Florentina I. Popovici, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau
Computer Sciences Department, University of Wisconsin, Madison

Abstract

We propose a new approach for I/O scheduling that performs on-line simulation of the underlying disk. When simulation is integrated within a system, three key challenges must be addressed: first, the simulator must be portable across the full range of devices; second, all configuration must be automatic; third, the computation and memory overheads must be low. Our simulator, the Disk Mimic, achieves these goals by building a table-based model of the disk as it observes the times for previous requests. We show that a shortest-mimicked-time-first (SMTF) scheduler performs nearly as well as an approach with perfect knowledge of the underlying device and that it is superior to traditional scheduling algorithms such as C-LOOK and SSTF; our results hold as the seek and rotational characteristics of the disk are varied.

1 Introduction

High-performance disk schedulers explored in the research literature are becoming progressively more tuned to the performance characteristics of the underlying disks. Each generation of disk schedulers has accounted for more of the behavior of storage devices at the time. For example, disk schedulers analyzed in the 1970s and 1980s focused on minimizing seek time, given that seek time was often an order of magnitude greater than the expected rotational delay [10, 26, 29]. In the early 1990s, the focus of disk schedulers shifted to take rotational delay into account, as rotational delays and seek costs became more balanced [13, 21, 31].

At the next level of sophistication, a disk scheduler takes all aspects of the underlying disk into account: track and cylinder switch costs, cache replacement policies, mappings from logical block number to physical block number, and zero-latency writes. For example, Worthington *et al.* demonstrate that algorithms that effectively utilize a prefetching disk cache perform better than those that do not [31].

However, the more intricate the knowledge a scheduler has of the disk, the more barriers there are to its realization within operating system kernels. Specifically, there are three obstacles that must be overcome. First, the scheduler must discover detailed knowledge of the underlying disk. Although a variety of tools have been described that automatically acquire portions of this knowledge [19, 25, 32], it must still be embedded into the disk model employed by the scheduler; the resulting scheduler is then configured to handle only a single disk with those specific characteristics. Second, the disk scheduler must also have knowledge of the current state of the disk, such as the exact position of the disk head. Given that head position is not exposed by current disk controllers and its position is not predictable due to low-level disk techniques such as wear leveling, predictive failure analysis, and log updates, the scheduler must control the current position using non-trivial techniques [11, 33]. Finally, the computational costs of detailed modeling can be quite high [31]; it is not uncommon for the time to model request time to be larger than the time to service the request [4].

Due to these difficulties, few disk schedulers that leverage more than basic seek costs have been implemented for real disks. When considering rotational position, most previous work has been performed within simulation environments [13, 21, 23, 31]. The schedulers that have recently been implemented by researchers have either contained substantial simplifications [12] or have been painstakingly tuned for a small group of disks [11, 33]. Not surprisingly, the disk schedulers found in modern operating systems such as Linux, NetBSD, and Solaris, attempt only to minimize seek time.

1.1 A Different Approach

We believe that a promising alternative approach to embedding detailed knowledge of the disk into the scheduler is to embed an *on-line simulator* of the disk into the scheduler. An I/O scheduler is able to use on-line simulation of the underlying storage device to predict which request in its queue will have the shortest positioning time. Although a variety of disk simulators exist [4, 14, 30], most are targeted for performing traditional, off-line simulations, and unfortunately, the infrastructure for performing on-line simulation is fundamentally different.

In many respects, the requirements of an on-line simulator are more stringent than those of an off-line simulator. First, the on-line simulator must be *portable*; that is, the simulator must be able to model the behavior of any disk drive that could be used in practice. Second, the on-line simulator must have *automatic run-time configuration*, since one cannot know the precise characteristics of the underlying device when constructing the simulator; it is highly undesirable if a human administrator must interact with the simulator. Finally, the on-line simulator must have *low overhead*; the computation and memory overheads of an on-line simulator must be minimized such that the simulator does not adversely impact system performance.

In addition to the complexity it introduces, an on-line simulator also provides ample opportunities for simplification. First, the on-line simulator has the opportunity to observe the run-time behavior of the device; not only does this allow the simulator to configure itself on the fly, it also allows the simulator to adjust to changes in the behavior of the device over time. Second, the on-line simulator can be specialized for the problem domain in question. Finally, the on-line simulator does not need to be parameterizable; that is, since an on-line simulator is not exploring different versions of the device itself, the simulator does not need to contain a functional model of the device.

1.2 Contributions

We address how to implement an I/O scheduler that is aware of the underlying disk technology in a simple, portable, and robust manner. To achieve this goal, we introduce the Disk Mimic, which meets the requirements of an on-line simulator for disk scheduling. The Disk Mimic is based upon a simple table-based approach, in which input parameters to the simulated device are used to index into a table; the corresponding entry in the table gives the predicted output for the device. A table-based approach is appropriate for on-line simulation because it can portably capture the behavior of a variety of devices, requires no manual configuration, and can be performed with little computational overhead. However, there is a significant challenge as well: to keep the size of the table tractable, one must identify the input parameters that significantly impact the desired outputs. The method for reducing this input space depends largely upon the domain in which the on-line simulator is deployed.

We show that for disk scheduling, two input parameters are sufficient for predicting the positioning time: the logical distance between two requests and the request type. However, when using inter-request distance for prediction, two issues must be resolved. First, inter-request distance is a fairly coarse predictor of positioning time; as a result, there is high variability in the times for different requests with the same distance. The implication is that the Disk Mimic must observe many instances for a given distance and use an appropriate summary metric for the distribution; experimentally, we have found that summarizing a small number of samples with the mean works well. Second, given the large number of possible inter-request distances on a modern disk drive, the Disk Mimic cannot record all distances in a table of a reasonable size. We show that simple linear interpolation can be used to represent ranges of missing distances, as long as some number of the interpolations within each range are checked against measured values.

We propose a new disk scheduling algorithm, shortest-mimicked-time-first (SMTF), which picks the request that is predicted by the Disk Mimic to have the shortest positioning time. We demonstrate that the SMTF scheduler can utilize the Disk Mimic in two different ways; specifically, the Disk Mimic can either be configured off-line or on-line, and both approaches can be performed automatically. When the Disk Mimic is configured off-line, it performs a series of probes to the disk with different inter-request distances and records the resulting times; in this scenario, the Disk Mimic has complete control over which inter-request distances are observed and which are interpolated. When the Disk Mimic is configured on-line, it records the requests sent by the running workload and their resulting times. Note that regardless of whether the Disk Mimic is configured off-line or on-line, the simulation itself is always performed on-line, within an active system.

We show that the Disk Mimic can be used to significantly improve the throughput of disks with high utilization. Specifically, for a variety of simulated and real disks, C-LOOK and SSTF perform between 10% and 50% slower than SMTF. Further, we demonstrate that the Disk Mimic can be successfully configured on-line; we show that while the Disk Mimic learns about the storage device, SMTF performs no worse than a base scheduling algorithm (*e.g.*, C-LOOK or SSTF) and quickly performs close to the off-line configuration (*i.e.*, after approximately 750,000 requests).

The rest of the paper is organized as follows. In Section 2 we describe the SMTF scheduler in more detail and in Section 3 we describe the Disk Mimic. We describe our basic methodology for evaluation in Section 4. Next, we investigate the issues of configuring the Disk Mimic off-line in Section 5. We then describe the additional complexities of configuring the Disk Mimic on-line and show its performance in Section 6. Finally, we describe related work in Section 7 and conclude in Section 8.

2 I/O Scheduler

Many modern disks implement scheduling in the device itself. While this might suggest that file system I/O scheduling is obsolete, there are several reasons why the file system should perform scheduling. First, disks are usually able to schedule only a limited number of simultaneous requests since they have more restrictive space and computational power constraints. Second, there are instances when increased functionality requires the scheduling to be done at file system level. For example, Iyer and Druschel introduce short waiting times in the scheduler to preserve the continuity of a stream of requests from a single process rather than interleaving streams from different processes [12]. Further, Shenoy and Vin implement different service requirements for applications by implementing a scheduling framework in the file system [23].

We now briefly describe the approach of a new file system I/O scheduler that leverages the Disk Mimic. We refer to the algorithm implemented by this scheduler as shortest-mimicked-time-first, or SMTF. The basic function that SMTF performs is to order the queue of requests such that the request with the shortest positioning time, as determined by the Disk Mimic, is scheduled next. However, given this basic role, there are different optimizations that can be made. The assumptions that we use for this paper are as follows.

First, we assume that the goal of the I/O scheduler is to optimize the *throughput* of the storage system. We do not consider the fairness of the scheduler. We believe that the known techniques for achieving fairness (*e.g.*, weighting each request by its age [13, 21]) can be added to SMTF as well.

Second, we assume that the I/O scheduler is operating in an environment with heavy disk traffic. Given that the queues at the disk may contain hundreds or even thousands of requests [13, 21], the computational complexity of the scheduling algorithm is an important issue [2]. Given these large queue lengths, it is not feasible to perform an optimal scheduling decision that considers all possible combinations of requests. Therefore, we consider a greedy approach, in which only the time for the next request is minimized [13].

To evaluate the performance of SMTF, we compare to the algorithms most often used in practice: first-come-first-served (FCFS), shortest-seek-time-first (SSTF), and C-LOOK. FCFS simply schedules requests in the order they were issued. SSTF selects the request that has the smallest difference from the last logical block number (LBN) accessed on disk. C-LOOK is a variation of SSTF where requests are still serviced according to their LBN proximity to the last request serviced, but the scheduler picks requests only in ascending LBN order. When there are no more such requests to be serviced, the algorithm picks the request in the queue with the lowest LBN and then continues to service requests in ascending order.

To compare our performance to the best possible case, we have also implemented a best-case-greedy scheduler for our simulated disks; this best-case scheduler knows exactly how long each request will take on the simulated disk and greedily picks the request with the shortest positioning time next. We refer to this scheduler as the greedy-optimal scheduler.

3 The Disk Mimic

The Disk Mimic is able to capture the behavior of a disk drive in a portable, robust, and efficient manner. To predict the performance of a disk, the Disk Mimic uses a simple table, indexed by the relevant input parameters to the disk. Thus, the Disk Mimic does not attempt to simulate the mechanisms or components internal to the disk; instead, it simply reproduces the output as a function of the inputs it has observed.

3.1 Reducing Input Parameters

Given that the Disk Mimic uses a table-driven approach to predict the time for a request as a function of the observable inputs, the fundamental issue is reducing the number of inputs to the table to a tractable number. If the I/O device is treated as a true black box, in which one knows nothing about the internal behavior of the device, then the Disk Mimic must assume that the service time for each request is a function of all previous requests. Given that each request is defined by many parameters (*i.e.*, whether it is a read or a write, its block number, its size, the time of the request, and even its data value), this leads to a prohibitively large number of input parameters as indices to the table.

Therefore, the only tractable approach is to make assumptions about the behavior of the I/O device for the problem domain of interest [3]. Given that our goal is for the I/O scheduler to be portable across the realistic range of disk drives, and not to necessarily work on any hypothetical storage device, we can use high-level assumptions of how disks behave to eliminate a significant number of input parameters; however, the Disk Mimic will make as few assumptions as possible.

Our current implementation of the Disk Mimic predicts the time for a request from two input parameters: the *request type* and the *inter-request distance*. We define inter-request distance as the logical distance from the first block of the current request to the last block of the previous request. The conclusion that request type and inter-request distance are key parameters agrees with that of previous researchers [18, 27].

We now briefly argue why inter-request distance and request type are suitable parameters in our domain. We begin by summarizing the characteristics of modern disk drives. Much of this discussion is taken from the classic paper by Ruemmler and Wilkes [18]; the interested reader is referred to their paper for more details.

3.1.1 Background

A disk drive contains one or more *platters*, where each platter *surface* has an associated disk head for reading and writing. Each surface has data stored in a series of concentric circles, or *tracks*. A single stack of tracks at a common distance from the spindle is called a *cylinder*. Modern disks also contain RAM to perform caching; the caching algorithm is one of the most difficult aspects of the disk to capture and model [24, 32].

Accessing a block of data requires moving the disk head over the desired block. The time for this has two dominant components. The first component is *seek time*, moving the disk head over the desired track. The seek time for reads is likely to be less than that for writes, since reads can be performed more aggressively. A read can be performed when a block is not yet quite available because the read can be repeated if it was performed from the wrong sector; however, a write must first verify that it is at the right sector to avoid overwriting other data. The second component is *rotation latency*, waiting for the desired block to rotate under the disk head. The time for the platter to rotate is roughly constant, but it may vary around 0.5 to 1% of the nominal rate; as a result, it is difficult to predict the location of the disk head after the disk has been idle for many revolutions. Besides these two important positioning components there are other mechanical movements that need to be accounted for: head and track switch time. A head switch is the time it takes for the mechanisms in the disk to activate a different disk head to access a different platter surface. A track switch is the time it takes to move a disk head from the last track of a cylinder to the first one of the next.

The disk appears to its client as a linear array of logical blocks; these logical blocks are then mapped to physical sectors on the platters. This indirection has the advantage that the disk can reorganize blocks to avoid bad sectors and to improve performance, but it has the disadvantage that the client does not know where a particular logical block is located. If a client wants to derive this mapping, there are multiple sources of complexity. First, different tracks have different numbers of sectors; specifically, due to zoning, tracks near the outside of a platter have more sectors (and subsequently deliver higher bandwidth) than tracks near the spindle. Second, consecutive sectors across track and cylinder boundaries are skewed to adjust for head and track switch times; the skewing factor differs across zones as well. Third, flawed sectors are remapped through sparing; sparing may be done by remapping a bad sector (or track) to a fixed alternate location or by slipping the sector (or track) and all subsequent ones to the next sector (or track).

3.1.2 Input Parameters

As previously explained, read and write operations take different times to execute. In addition, the type of the last operation issued also influences service time [4, 18]. To account for these factors in our table-based model, we record the request type (read or write) of the current and previous requests as one of the input parameters.

The other input parameter is the inter-request distance between logical block addresses, which captures some of the aforementioned underlying characteristics of the disk, while missing others. We note that ordering requests based on the time for a given distance is significantly different than using the distance itself. Due to the complexity of disk geometry, some requests that are separated by a larger logical distance can be positioned more rapidly; the relationship between the logical block address distance and positioning time is not linear.

In the opinion of Ruemmler and Wilkes [18], the following aspects of the disk should be modeled for the best accuracy: seek time (calculated with two separate functions depending upon the seek distance from the current and final cylinder position of the disk head and different for reads and writes), head and track switches, rotation latency, data layout (including reserved sparing areas, zoning, and track and cylinder skew), and data caching (both read-ahead and write-behind). We briefly discuss the extent to which each of these components is captured with our approach.

Our approach accounts for the combined costs of seek time, head and track switches, and rotation layout, but in a probabilistic manner. That is, for a given inter-request distance, there is some probability that a request crosses track or even cylinder boundaries. Requests of a given distance that cross the same number of boundaries have the same total positioning time: the same number of track seeks, the same number of head and/or track switches, and the same amount of rotation.

We note that the table-based method for tracking positioning time can be *more* accurate than that advocated by Ruemmler and Wilkes; instead of expressing positioning time as a value computed as a sum of functions (seek time, rotation time, caching, etc.), the Disk Mimic records the precise positioning time for each distance.

The cost incurred by the rotation of the disk has two components: the rotational distance between the previous and current request, and the elapsed time between the two requests (and thus, the amount of rotation that has already occurred). Although using inter-request distance probabilistically captures the rotational distance, the Disk Mimic does not record the amount of time that has elapsed since the last request. This omission is not an issue for disk scheduling in the presence of a full queue of requests; in this case, the inter-arrival time between requests at the disk is negligible and, thus, can be ignored. Ignoring time causes inaccuracies when scheduling the first request after an idle period; however, if the disk is often idle, then I/O scheduling is not an important problem.

Data layout is incorporated fairly well by the Disk Mimic as well. The number of sectors per track and number of cylinders impact our measured values in that these sizes determine the probability that a request of a given inter-request distance crosses a boundary; thus, these sizes impact the probability of each observed time in the distri-

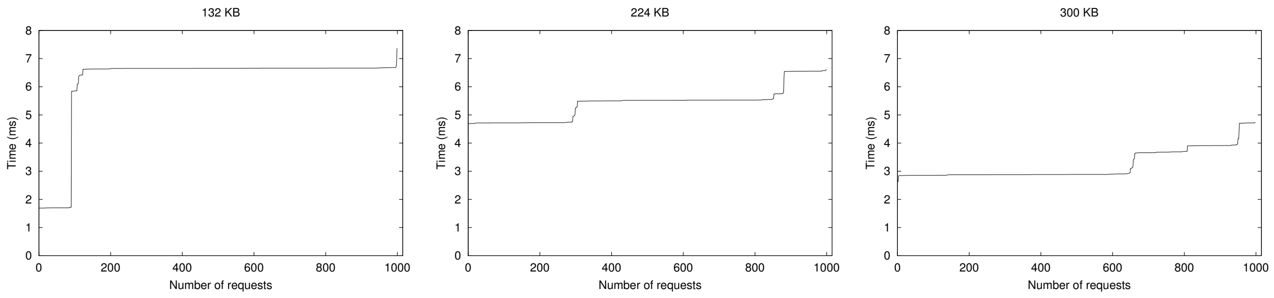


Figure 1: **Distribution of Off-Line Probe Times for Three Inter-Request Distances.** *Each graph shows a different inter-request distance: 132 KB, 224 KB, and 300 KB. Along the x-axis, we show each of the 1000 probes performed (sorted by time) and along the y-axis we show the time taken by that probe. These times are for the IBM 9LZX disk.*

bution. Although zoning behavior and bad sectors are not tracked by our model, previous research has shown that this level of detail does not help with scheduling [31].

The aspect which we model the least directly is that of general caching. However, the Disk Mimic will capture the effects of simple prefetching, which is the most important aspect of caching for scheduling [31]. For example, if a read of one sector causes the entire track to be cached, then the Disk Mimic will observe the faster performance of accesses with distances less than that of a track. In this respect, configuring the Disk Mimic on-line by observing the actual workload could be more accurate than configuring off-line, since the locality of the workload is captured.

Given the complexity associated with the inter-request distance, we concentrate on the issues related to this input parameter. For different values of the request type, the output of the Disk Mimic has the same characteristics, and thus we do not need to explore all the possible combinations of the two input parameters in our further discussions. Hence when we refer to inter-request distance we assume the request type is fixed.

3.2 Results

To illustrate some of the complexity of using inter-request distance as predictor of request time, we show the distribution of times observed. For these experiments, we configure the Disk Mimic off-line as follows.

The Disk Mimic configures itself by probing the I/O device using fixed-size requests (*e.g.*, 1 KB). For each of the possible inter-request distances covering the disk (both negative and positive), the Disk Mimic samples a number of points of the same distance: it accesses a block the specified distance from the previous block. To avoid any caching or prefetching performed by the disk, the Disk Mimic accesses a random location before each new probe of the required distance. The observed times are recorded in a table, indexed by the inter-request distance and the corresponding operation type.

In Figure 1 we show a small subset of the data collected on an IBM 9LZX disk. The figure shows the distribution of 1000 samples for three inter-request distances of 132 KB, 224 KB, and 300 KB. In each case, the y-axis shows the request time of a sample and the points along the x-axis represent each sample, sorted by increasing request time.

We make two important observations from the sampled times. First, for a given inter-request distance, the observed request time is not constant; for example, at a distance of 132 KB, about 10% of requests require 1.8 ms, about 90% require 6.8 ms, and a few require almost 8 ms. Given this multi-modal behavior, the time for a single request cannot be reliably predicted from only the inter-request distance; thus, one cannot usually predict whether a request of one distance will be faster or slower than a request of a different distance. Nevertheless, it is often possible to make reasonable predictions based upon the probabilities: for example, from this data, one can conclude that a request of distance 132 KB is likely to take longer than one of 224 KB.

Second, from examining distributions for different inter-request distances, one can observe that the number of transitions and the percentage of samples with each time value varies across inter-request distances. The number of transitions in each graph corresponds roughly to the number of track (or cylinder) boundaries that can be crossed for this inter-request distance.

This data shows that a number of important issues remain regarding the configuration of the Disk Mimic. First, since there may be significant variation in request times for a single inter-request distance, what summary metric should be used to summarize the distribution? Second, how many samples are required to adequately capture the behavior of this distribution? Third, must each inter-request distance be sampled, or is it possible to interpolate intermediate distances? We investigate these issues in Section 5.

4 Methodology

To evaluate the performance of SMTF scheduling, we consider a range of disk drive technology, presented in

Configuration	**rotation time**	**seek 1 cyl**	**seek 400**	**seek 3000**	**head switch**	**cyl switch**	**track skew**	**cyl skew**	**sectors per track**	**num heads**
1 **Base**	6	0.8	6.0	8	0.79	1.78	36	84	272	10
2 **Fast seek**	6	0.16	1.32	1.6	0.79	1.00	36	46	272	10
3 **Slow seek**	6	2.0	33.0	40.0	0.79	2.80	36	127	272	10
4 **Fast rotate**	2	0.8	6.0	8	0.79	1.78	108	243	272	10
5 **Slow rotate**	12	0.8	6.0	8	0.79	1.78	18	41	272	10
6 **Fast seek+rot**	2	0.160	1.32	1.6	0.79	1.00	108	136	272	10
7 **More capacity**	6	0.8	6.0	8	0.79	1.78	36	84	544	20
8 **Less capacity**	6	0.8	6.0	8	0.79	1.78	36	84	136	5

Table 1: **Disk Characteristics.** *Configurations of eight simulated disks. Times for rotation, seek, and head and cylinder switch are in milliseconds, the cylinder and track skews are expressed in sectors. In most experiments, the base disk is used.*

Table 1. We have implemented a disk simulator that accurately models seek time, fixed rotation latency, track and cylinder skewing, and a simple segmented cache. The first disk, also named the *base disk*, simulates a disk with performance characteristics similar to an IBM 9LZX disk. The seek times, cache size and number of segments, head and cylinder switch times, track and cylinder skewing and rotation times are either measured by issuing SCSI commands and measuring the elapsed time, or directly querying the disk, similar to the approach used by Schindler and Ganger [19], or by using the values provided by the manufacturer. The curve corresponding to the seek time is modeled by probing an IBM 9LZX disk for a range of seek distances (measured as the distance in cylinders from the previous cylinder position to the current one) and then curve fitting the values to use the two-function equation proposed by Ruemmler and Wilkes [18]. For short seek distances the seek time is proportional to the square root of the cylinder distance, and for longer distances the seek time is proportional to the cylinder distance. The middle value in the seek column represents the cylinder distance where the switch between the two functions occurs. For example, for the base disk, if the seek distance is smaller than 400 cylinders, we use the square root function.

For the other disk configurations we simulate, we start from the base disk and vary different parameters that influence the positioning time. For example, disk configuration number 2 (*Fast seek*) represents a disk that has a fast seek time and the numbers used to compute the seek curve are adjusted accordingly, as well as the number of sectors that constitute the cylinder skew. Similarly for disk configuration number 4 (*Fast rotate*) the time to execute a rotation is decreased by a factor of three and the number of track and cylinder skew sectors are increased. The other disk configurations account for disks that have a slower seek time, slower rotation time, faster seek time, faster rotation time and more or less capacity than the base disk. In addition to using the described simulated disks we also run our experiments on an IBM 9LZX disk.

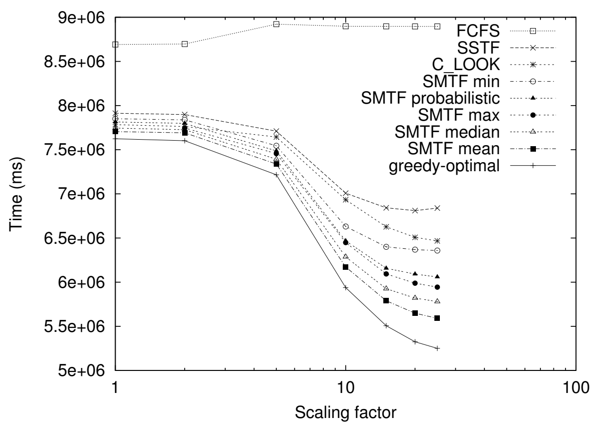


Figure 2: **Sensitivity to Summary Metrics.** *This graph compares the performance of a variety of scheduling algorithms on the base simulated disk and the week-long HP trace. For the SMTF schedulers, no interpolation is performed and 100 samples are obtained for each data point. The x-axis shows the compression factor applied to the workload. The y-axis reports the time spent at the disk.*

To evaluate scheduling performance, we show results from a set of traces collected at HP Labs [17]; in most cases, we focus on the trace for the busiest disk from the week of 5/30/92 to 6/5/92. For our performance metric, we report the time the workload spent at the disk. To consider the impact of heavier workloads and longer queue lengths, we compress the inter-arrival time between requests. When scaling time, we attempt to preserve the dependencies across requests in the workload by observing the blocks being requested; we assume that if a request is repeated to a block that has not yet been serviced, that this request is dependent on the previous request first completing. Thus, we hold repeated requests, and all subsequent requests, until the previous identical request completes.

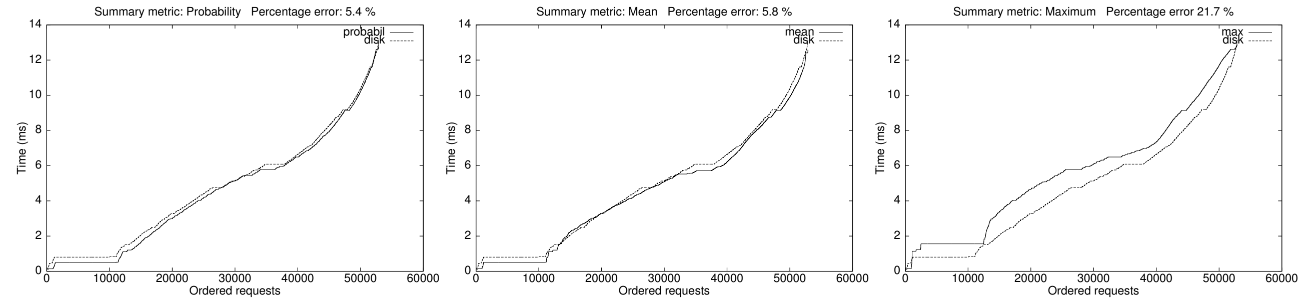


Figure 3: **Demerit Figures for SMTF with Probability, Mean, and Maximum Summary Metrics.** *Each graph shows the demerit figure for a different summary metric. These distributions correspond to the one day from the experiments shown in Figure 2 with a compression factor of 20.*

5 Off-Line Configuration

The SMTF scheduler can be configured both on-line and off-line. We now explore the case when the Disk Mimic has been configured off-line; again, although the Disk Mimic is configured off-line, the simulation and predictions required by the scheduler are still performed on-line within the system. As described previously, configuring the Disk Mimic off-line involves probing the underlying disk with requests that have a range of inter-request distances. We note that even when the model is configured off-line, the process of configuring SMTF remains entirely automatic and portable across a range of disk drives. The main drawback to configuring the Disk Mimic off-line is a longer installation time when a new device is added to the system: the disk must be probed before it can be used for workload traffic.

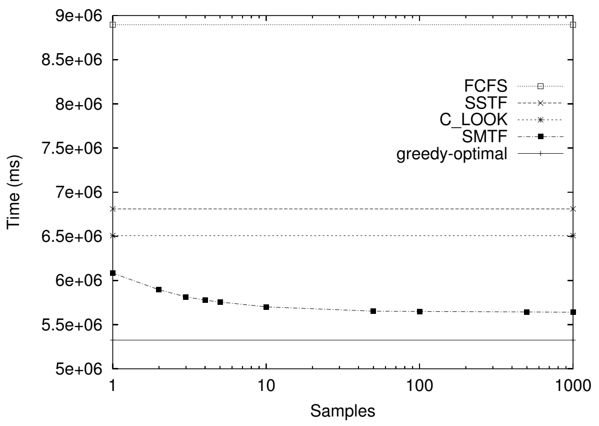


Figure 4: **Sensitivity to Number of Samples.** *The graph shows that the performance of SMTF improves with more samples. The results are on the simulated disk and the week-long HP trace with a compression factor of 20. The x-axis indicates the number of samples used for SMTF. The y-axis shows the time spent at the disk.*

5.1 Summary Data

To enable the SMTF scheduler to easily compare the expected time of all of the requests in the queue, the Disk Mimic must supply a summary value for each distribution as a function of the inter-request distance. Given the multi-modal characteristics of these distributions, the choice of a summary metric is not obvious. Therefore, we evaluate five different summary metrics: `mean`, `median`, `maximum`, `minimum`, and `probabilistic`, which randomly picks a value from the sampled distribution according to its probability.

The results for each of these summary metrics on the base simulated disk are shown in Figure 2. For the workload, we consider the week-long HP trace, scaled by the compression factor noted on the x-axis. The graph shows that FCFS, SSTF, and C-LOOK all perform worse than each of the SMTF schedulers; as expected, the SMTF schedulers perform worse than the greedy-optimal scheduler, but the best approach is always within 7% for this workload. These results show that using inter-request distance to predict positioning time merits further attention.

Comparing performance across the different SMTF approaches, we see that each summary metric performs quite differently. The ordering of performance from best to worse is: `mean`, `median`, `maximum`, `probabilistic`, and `minimum`. It is interesting to note that the scheduling performance of each summary metric is not correlated with its accuracy. The accuracy of disk models is often evaluated according to its *demerit figure* [18], which is defined as the root mean square of the horizontal distance between the time distributions for the model and the real disk. This point is briefly illustrated in Figure 3, which shows the distribution of actual times versus the predicted times for three different metrics: `probabilistic`, `mean`, and `maximum`.

As expected, the `probabilistic` model has the best demerit figure; with many requests, the distribution it predicts is expected to match that of the real device. However, the `probabilistic` model performs relatively poorly within SMTF because the time it predicts for any one request may differ significantly from the ac-

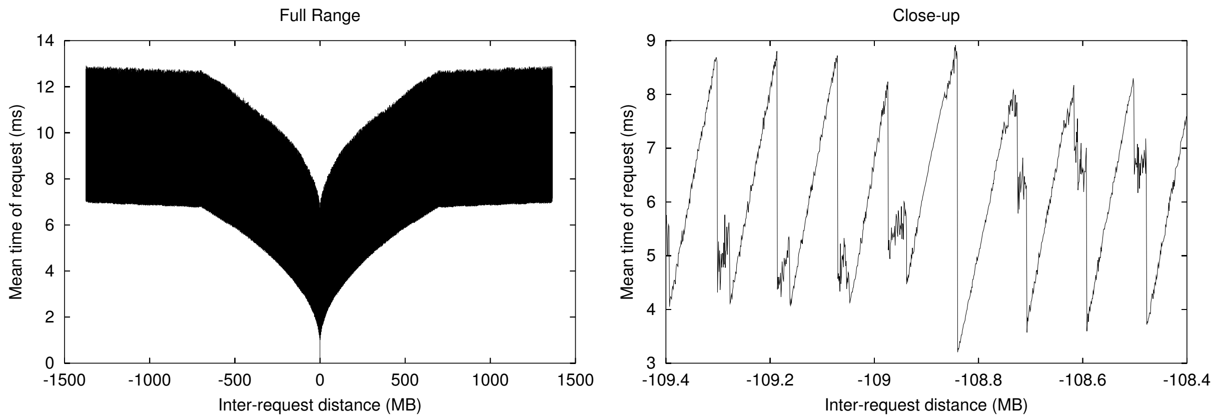


Figure 5: **Mean Values for Samples as a Function of Inter-request Distance.** *The graph on the left shows the mean time for the entire set of inter-request distances on our simulated disk. The graph on the right shows a close-up for inter-request distances; other distances have qualitatively similar saw-tooth behavior.*

tual time for that request. Conversely, although the `maximum` value results in a poor demerit figure, it performs adequately for scheduling; in fact, SMTF with `maximum` performs significantly better than with `minimum`, even though both have similar demerit figures. Finally, using the `mean` as a summary of the distribution achieves the best performance, even though it does not result in the best demerit figure; we have found that `mean` performs best for all other days from the HP traces we have examined as well. Thus, for the remainder of our experiments, we use the mean of the observed samples as the summary data for each inter-request distance.

5.2 Number of Samples

Given the large variation in times for a single inter-request distance, the Disk Mimic must perform a large number of probe samples to find the true mean of the distribution. However, to reduce the time required to configure the Disk Mimic off-line, we would like to perform as few samples as possible. Thus, we now evaluate the impact of the number of samples on SMTF performance.

Figure 4 compares the performance of SMTF as a function of the number of samples to the performance of FCFS, C-LOOK, SSTF, and optimal. As expected, the performance of SMTF increases with more samples; on this workload and disk, the performance of SMTF continues to improve up to approximately 10 samples. However, most interestingly, even with a single sample for each inter-request distance, the Disk Mimic performs better than FCFS, C-LOOK, and SSTF.

5.3 Interpolation

Although the number of samples performed for each inter-request distance impacts the running time of the off-line probe process, an even greater issue is whether each distance must be explicitly probed or if some can be interpolated from other distances. Due to the large number of potential inter-request distances on a modern storage device (*i.e.*, two times the number of sectors for both negative and positive distances), not only does performing all of the probes take a significant amount of time, but storing each of the mean values is prohibitive as well. For example, given a disk of size 10 GB, the amount of memory required for the table can exceed 800 MB. Therefore, we explore how some distances can be interpolated without making detailed assumptions about the underlying disk.

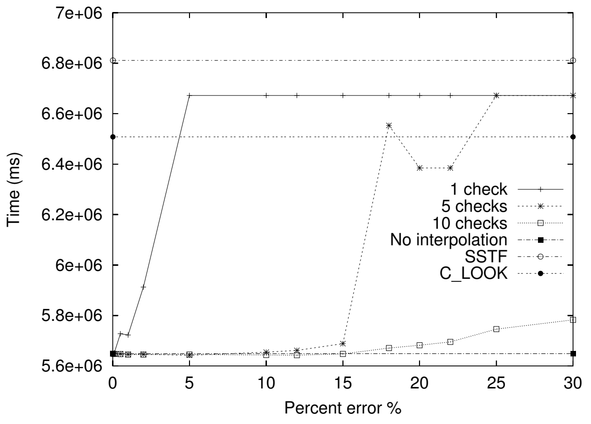


Figure 6: **Sensitivity to Interpolation.** *The graph shows performance with interpolation as a function of the percent of allowable error. Different lines correspond to different numbers of check points, N. The x-axis is the percent of allowable error and the y-axis is the time spent at the disk. These results use the base simulated disk and the week-long HP trace with a compression factor of 20.*

To illustrate the potential for performing simple interpolations, we show the mean value as a function of the inter-request distance in Figure 5. The graph on the left

Check Points N	**Acceptable Error**
1	1 %
2	2 %
3	5 %
4	10 %
5	15 %
10	20 %

Table 2: **Allowable Error for Interpolation.** *The table summarizes the percentage within which an interpolated value must be relative to the probed value in order to infer that the interpolation is successful. As more check points are performed between two inter-request distances, the allowable error increases. The numbers were gathered by running a number of different workloads on the simulated disks and observing the point at which performance with interpolation degrades relative to that with no interpolation.*

shows the mean values for all inter-request distances on our simulated disk. The curve of the two bands emanating from the middle point corresponds to the seek curve of the disk (*i.e.*, for short seeks, the time is proportional to the square root of the distance, whereas for long, the time is linear with distance); the width of the bands is relatively constant and corresponds to the rotation latency of the disk. The graph on the right shows a close-up of the inter-request distances. This graph shows that the times follow a distinct saw-tooth pattern; as a result, a simple linear model can likely be used to interpolate some distances, but care must be taken to ensure that this model is applied to only relatively short distances.

Given that the length of the linear regions varies across different disks (as a function of the track and cylinder size), our goal is not to determine the particular distances that can be interpolated successfully. Instead, our challenge is to determine when an interpolated value is "close enough" to the actual mean such that scheduling performance is impacted only negligibly.

Our basic off-line interpolation algorithm is as follows. After the Disk Mimic performs S samples of two inter-request distances *left* and *right*, it chooses a random distance *middle* between *left* and *right*; it then linearly interpolates the mean value for *middle* from the means for *left* and *right*. If the interpolated value for *middle* is within *error* percent of the probed value for *middle*, then the interpolation is considered successful and all the distances between *left* and *right* are interpolated. If the interpolation is not successful, the Disk Mimic recursively checks the two smaller ranges (*i.e.*, the distances between *left* and *middle* and between *middle* and *right*) until either the intermediate points are successfully interpolated or until all points are probed.

For additional confidence that linear interpolation is valid in a region, we consider a slight variation in which N points between *left* and *right* are interpolated and checked. Only if all N points are predicted with the desired level of accuracy is the interpolation considered successful. The intuition of performing more check points is that a higher error rate can be used and interpolation can still be successful.

Figure 6 shows the performance of SMTF when distances are interpolated; the graph shows the effect of increasing the number of intermediate points N that are checked, as well as increasing the acceptable error, *error*, of the interpolation. We make two observations from this graph.

First, SMTF performance decreases as the allowable error of the check points increases. Although this result is to be expected, we note that performance decreases dramatically with the error not because the error of the checked distances is increased, but because the interpolated distances are inaccurate by much more. For example, with a single check point (*i.e.*, $N = 1$) and an error level of 5%, we have found that only 20% of the interpolated values are actually accurate to that level and the average error of all interpolated values increases to 25% (not shown). In summary, when error increases significantly, there is not a linear relationship for the distances between *left* and *right* and interpolation should not be performed.

Second, SMTF performance for a fixed error increases with the number of intermediate check points N. The effect of performing more checks is to confirm that linear interpolation across these distances is valid. For example, with $N = 10$ check points and *error* = 5%, almost all interpolated points are accurate to that level and the average error is less than 1% (also not shown).

Table 2 summarizes our findings for a wider number of check points. The table shows the allowable error percentage as a function of the number of check points, N, to achieve scheduling performance that is very similar to that with all probes. Thus, the final probe process can operate as follows. If the interpolation of one distance between *left* and *right* has an error less than 1%, it is deemed successful. Otherwise, if two distances between *left* and *right* have errors less than 2%, the interpolation is successful as well. Thus, progressively more check points can be made with higher error rates to be successful. With this approach, 90% of the distances on the disk are interpolated instead of probed, and yet scheduling performance is virtually unchanged; thus, interpolation leads to a 10-fold memory savings.

5.4 Disk Characteristics

To demonstrate the robustness and portability of the Disk Mimic and SMTF scheduling, we now consider the full range of simulated disks described in Table 1. The performance of FCFS, C-LOOK, SSTF, and SMTF relative

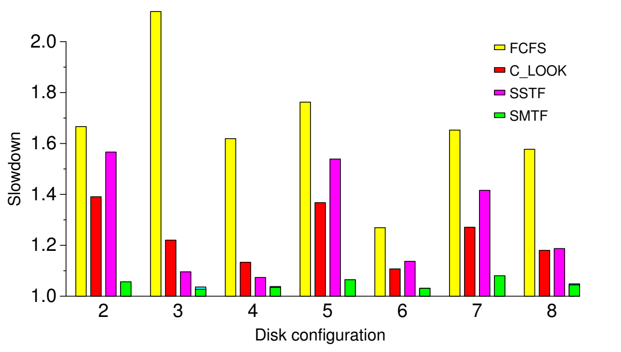


Figure 7: **Sensitivity to Disk Characteristics.** *This figure explores the sensitivity of scheduling performance to the disk characteristics shown in Table 1. Performance is shown relative to greedy-optimal. We report values for SMTF using interpolation. The performance of SMTF without interpolation (i.e., all probes) is very similar.*

to greedy-optimal for each of the seven new disks is summarized in Figure 7. We show the performance of SMTF with interpolation. The performance of SMTF with and without interpolation is nearly identical. As expected, FCFS performs the worst across the entire range of disks, sometimes performing more than a factor of two slower than greedy-optimal. C-LOOK and SSTF perform relatively well when seek time dominates performance (*e.g.*, disks 3 and 4); SSTF performs better than C-LOOK in these cases as well. Finally, SMTF performs very well when rotational latency is a significant component of request positioning (*e.g.*, disks 2 and 5). In summary, across this range of disks, SMTF always performs better than both C-LOOK and SSTF scheduling and within 8% of the greedy-optimal algorithm.

To show that SMTF can handle the performance variation of real disks, we compare the performance of our implementation of SMTF to that of C-LOOK when run on the IBM 9LZX disk. On the one week HP trace, we achieve a performance improvement of 8% for SMTF compared C-LOOK and an improvement of 12% if idle time is removed from the trace. This performance improvement is not as significant as it could be for two reasons. First, the IBM 9LZX disk has a relatively high ratio of seek to rotation time; the performance improvement of SMTF relative to C-LOOK is greater when rotation time is a more significant component of positioning. Second, the HP trace exercises a large amount of data on the disk; when the locality of the workload is low as in this trace, seek time further dominates positioning time.

To explore the effect of workload locality we create a synthetic workload of random 1 KB reads and writes with no idle time; the maximum inter-request distance is varied, as specified on the x-axis of Figure 8. This graph shows that the performance improvement of SMTF relative to C-LOOK varies between 32% and 8% as the inter-request distance varies from 25 MB to 1.3 GB. Given that most file systems (*e.g.*, Linux ext2) try to optimize locality by placing related files in the same cylinder group, SMTF can optimize accesses better than C-LOOK in practice. Thus, we believe that SMTF is a viable option for scheduling on real disks.

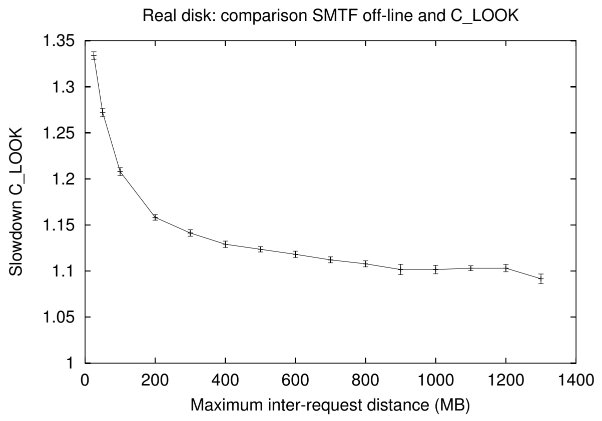


Figure 8: **Real Disk Performance.** *This graph shows the slowdown of C-LOOK when compared to the SMTF configured off-line. The workload is a synthetically generated trace and the numbers are averages over 20 runs. The standard deviation is also reported. The x-axis shows the maximum inter-request distance existent in the trace and the y-axis reports the percentage slowdown of the C-LOOK algorithm.*

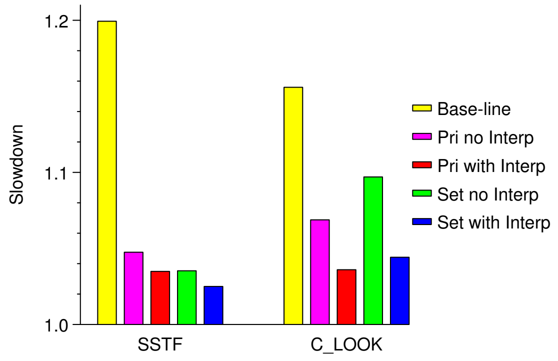
6 On-Line Configuration

We now explore the SMTF scheduler when all configuration is performed on-line. With this approach, there is no overhead at installation time to probe the disk drive; instead, the Disk Mimic observes the behavior of the disk as the workload runs. As in the off-line version, the Disk Mimic records the observed disk times as a function of its inter-request distance, but in this case it has no control over the inter-request distances it observes.

6.1 General Approach

For the on-line version, we assume that many of the lessons learned from off-line configuration hold. First, we continue to use the mean to represent the distribution of times for a given inter-request distance. Second, we continue to rely upon interpolation; note that when the Disk Mimic is configured on-line, interpolation is useful not only for saving space, but also for providing new information about distances that have not been observed.

The primary challenge that SMTF must address in this situation is how to schedule requests when some of the inter-request distances have unknown times (*i.e.*, this inter-request distance has not yet been observed by the



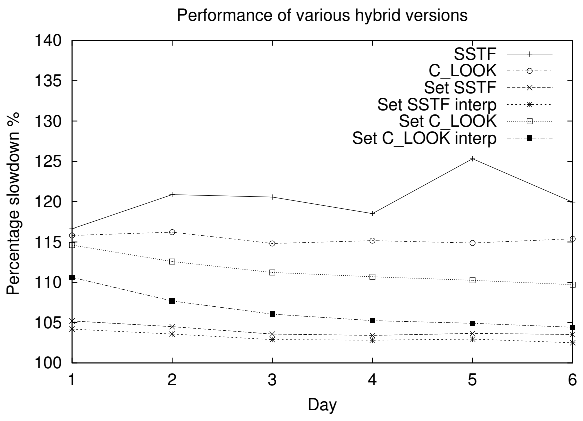


Figure 9: **Performance of On-Line SMTF.** *The first graph compares the performance of different variations of on-line SMTF; the performance of the last day of the week-long HP trace is shown relative to off-line SMTF. The second graph shows that the performance of Online-Set improves over time as more inter-request distances are observed.*

Disk Mimic and the Disk Mimic is unable to confirm that it can be interpolated successfully). We consider two algorithms for comparison. Both algorithms assume that there is a base scheduler (either C-LOOK or SSTF) which is used when the Disk Mimic does not have sufficient information.

The first algorithm, *Online-Priority*, schedules only those requests for which the Disk Mimic has information. Specifically, *Online-Priority* gives strict priority to those requests in the queue that have an inter-request distance with a known time; among those requests with known times, the request with the minimum mean time is picked. With *Online-Priority*, the base scheduler (*e.g.*, C-LOOK or SSTF) is only used when no inter-request distances for the current queue are known. There are two problems with this approach. First, given its preference for scheduling already known inter-request distances, *Online-Priority* may perform worse than its base scheduler. Second, schedules with a diversity of distances may never be produced and thus the Disk Mimic may never observe some of the most efficient distances.

The second algorithm, *Online-Set*, improves on both of these limitations by using the decision of the base scheduler as its starting point, and scheduling a different request only when the Disk Mimic has knowledge that performance can be improved. Specifically, *Online-Set* first considers the request that the base scheduler would pick. If the time for the corresponding distance is not known by the Disk Mimic, then this request is scheduled. However, if the time is known, then all of the requests with known inter-request distances are considered and the one with the fastest mean is chosen. Thus, *Online-Set* should only improve on the performance of the base scheduler and it is likely to schedule a variety of inter-request distances when it is still learning.

6.2 Experimental Results

To evaluate the performance of the on-line algorithms, we return to the base simulated disk. The left-most graph of Figure 9 compares the performance of *Online-Priority* and *Online-Set*, when either C-LOOK or SSTF is used as the baseline algorithm and both with and without interpolation. Performance is expressed in terms of slowdown relative to the off-line version of SMTF. We make three observations from this graph.

First, and somewhat surprising, although C-LOOK performs better than SSTF for this workload and disk, SMTF performs noticeably better with SSTF than with C-LOOK as a base; with C-LOOK, the Disk Mimic is not able to observe inter-request distances that are negative (*i.e.*, backward) and thus does not discover distances that are close together. Second, *Online-Set* performs better than *Online-Priority* with SSTF as the base scheduler. Third, although interpolation does significantly improve the performance of *Online-Priority* and of *Online-Set* with C-LOOK, it leads to only a small improvement with *Online-Set* and SSTF. Thus, as with off-line configuration, the primary benefit of interpolation is to reduce the memory requirements of the Disk Mimic, as opposed to improving performance.

The right-most graph of Figure 9 illustrates how the performance of *Online-Set* improves over time as more inter-request distances are observed. We see that the performance of the *Online-Set* algorithms (with and without interpolation) is better than the base-line schedulers of SSTF and C-LOOK even after one day of the original trace (*i.e.*, approximately 150,000 requests). The performance of *Online-Set* with SSTF converges to within 3% of the off-line version after four days, or only about 750,000 requests.

At this point, we feel that there are two opportunities for further improving the performance of on-line SMTF relative to off-line SMTF. First, in our current on-line implementations, if a slow time for a particular distance is observed initially, the scheduler will avoid that distance even if the mean is much faster. One can address this by requiring that a distance has a minimum number of samples before being classified as known. Second, our current algorithm does not leverage idle time [6]. One can perform probes of unknown inter-request distances during idle times so that the Disk Mimic can learn more of the characteristics of the disk.

7 Related Work

The approach we propose brings together two areas of study: disk modeling and disk scheduling. We present related work in both areas and compare it to our method.

7.1 Disk Modeling

The classic paper describing models of disk drives is that by Ruemmler and Wilkes [18]. The main focus of this work is to enable an informed trade-off between simulation effort and the resulting accuracy of the model. Ruemmler and Wilkes evaluate the aspects of a disk that should be modeled for a high level of accuracy, using the *demerit figure*. Other researchers have noted that additional non-trivial assumptions must be made to model disks to the desired accuracy level [14]; modeling cache behavior is a particularly challenging aspect [24].

Given that the detailed knowledge for modeling disks is not available from documentation, researchers have developed innovative methods to acquire the information. For example, Worthington *et al.* describe techniques for SCSI drives that extract time parameters such as the seek curve, rotation speed, and command overheads as well as information about the data layout on disk and the caching and prefetching characteristics [32]; many of these techniques are automated in later work [19].

Modeling storage devices using tables of past performance has also been explored in previous work; in most previous work [1, 7], high-level system parameters (*e.g.*, load, number of disks, and operation type) are used as indices into the table. Anderson [1] also uses the results on-line, to assist in the reconfiguration of disk arrays. The approach most similar to ours is that of Thornock *et al.* [27]. In this work, the authors use stochastic methods to build a model of the underlying drive. However, the application of this model is to standard, off-line simulation; specifically, the authors study block reorganization, similar to earlier work by Ruemmler and Wilkes [16].

At a higher level, Seltzer and Small suggest *in situ* simulation as a method for building more adaptive operating systems [22]. In this work, the authors suggest that operating systems can utilize in-kernel monitoring and adaptation to make more informed policy decisions. By tracing application activity, the VINO system can determine whether the current policy is behaving as expected or if another policy should be switched into place. However, actual simulations of system behavior are performed off-line, as a "last resort" when poor performance is detected.

7.2 Disk Scheduling

Disk scheduling has long been a topic of study in computer science [29]. Rotationally-aware schedulers came into existence in the early 1990's, through the work of Seltzer *et al.* [21] and Jacobson and Wilkes [13]. However, perhaps due the difficulty of implementation, those early works focused solely upon simulation to explore the basic ideas. Only recently have implementations of rotationally-aware schedulers been described within the literature, and those are crafted with extreme care [11, 33].

More recently, Worthington *et al.* [31] examine the benefits of even more detailed knowledge of disk drives within OS-level disk schedulers. They find that algorithms that mesh well with the modern prefetching caches perform best, but that detailed logical-to-physical mapping information is not currently useful.

Anticipatory scheduling is a recent scheduling development that is complementary to our on-line simulation-based approach [12]. An anticipatory scheduler makes the assumption that there is likely to be locality in a stream of requests from a given process; by waiting for the next request (instead of servicing a request from a different process), performance can be improved. The authors also note the difficulty of building a rotationally-aware scheduler, and instead use an empirically-generated curve-fitted estimate of disk access-time costs; the Disk Mimic would yield a performance benefit over this simplified approach.

8 Conclusions

In this paper, we have explored some of the issues of using simulation within the system to make run-time scheduling decisions; in particular, we have focused on how a disk simulator can automatically model a range of disks without human intervention. We have shown that the Disk Mimic can model the time of a request by simply observing the request type and the logical distance from the previous request and predicting that it will behave similarly to past requests with the same parameters. The Disk Mimic can configure itself for a given disk by either probing the disk off-line or, at a slight performance cost, by observing requests sent to the disk on-line. We have demonstrated that a shortest-mimicked-time-first (SMTF) disk scheduler can significantly improve disk performance relative to FCFS, SSTF, and C-LOOK for a range of disk characteristics.

In the future, we plan to show that SMTF scheduling is appropriate for a range of storage devices other than disk drives. For example, RAID systems [15], network-attached storage devices [5], MEMS-based devices [20], tapes [9], and non-volatile memory [28] may all be used as building blocks in a storage system. Each of these devices has its own complex performance characteristics and it would be ideal if the I/O scheduler could automatically adapt to any of these devices.

Acknowledgments

We would like to thank Nathan Burnett, Timothy Denehy, Brian Forney, and Muthian Sivathanu for their feedback on this paper. We also would like to thank Vern Paxson, our shepherd, and the anonymous reviewers for their thoughtful suggestions, all of which have greatly improved the content of this paper. Finally, we thank the Computer Systems Lab for their tireless assistance in providing a terrific environment for computer science research. This work is sponsored by NSF CCR-0092840, CCR-0133456, CCR-0098274, NGS-0103670, ITR-0086044, an IBM Faculty Award, and the Wisconsin Alumni Research Foundation.

References

[1] E. Anderson. Simple table-based modeling of storage devices. Technical Report HPL-SSP-2001-04, HP Laboratories, July 2001.

[2] M. Andrews, M. Bender, and L. Zhang. New algorithms for the disk scheduling problem. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 550-559, 1996.

[3] A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau. Information and Control in Gray-Box Systems. In *The 18th Symposium on Operating Systems Principles (SOSP)*, pages 43–56, October 2001.

[4] G. R. Ganger, B. L. Worthington, and Y. N. Patt. The DiskSim Simulation Environment - Version 2.0 Reference Manual. http://citeseer.nj.nec.com/article/ganger99disksim.html.

[5] G. A. Gibson, D. F. Nagle, K. Amiri, F. W. Chang, E. M. Feinberg, H. Gobioff, C. Lee, B. Ozceri, E. Riedel, D. Rochberg, and J. Zelenka. File server Scaling with Network-Attached Secure Disks. In *Proceedings of the 1997 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 272–284, Seattle, WA, June 1997.

[6] R. Golding, P. Bosch, C. Staelin, T. Sullivan, and J. Wilkes. Idleness is Not Sloth. In *Proceedings of the Winter '95 USENIX Technical Conference*, pages 201–212, New Orleans, Louisiana, January 1995.

[7] C. Gotlieb and G. MacEwen. Performance of movable-head disk storage devices. *Journal of the Association for Computing Machinery*, 20(4):604–623, 1973.

[8] J. L. Griffin, J. Schindler, S. W. Schlosser, J. S. Bucy, and G. R. Ganger. Timing-accurate Storage Emulation. In *Proceedings of the First USENIX Conference on File and Storage Technologies (FAST '02)*, pages 75–88, Monterey, CA, January 2002.

[9] B. Hillyer and A. Silberschatz. On the modeling and performance characteristics of a serpentine tape drive. In *Proceedings of 1996 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 170–179, 1996.

[10] M. Hofri. Disk scheduling: FCFS vs. SSTF revisited. *Communications of the ACM*, 23(11):645–653, 1980.

[11] L. Huang and T. Chiueh. Implementation of a rotation latency sensitive disk scheduler. Technical Report ECSL-TR81, SUNY, Stony Brook, March 2000.

[12] S. Iyer and P. Druschel. Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O. In *18th ACM Symposium on Operating Systems Principles*, pages 117–130, October 2001.

[13] D. M. Jacobson and J. Wilkes. Disk scheduling algorithms based on rotational position. Technical Report HPL-CSP-91-7, HP Laboratories, 1991.

[14] D. Kotz, S. B. Toh, and S. Radhakrishnan. A detailed simulation model of the HP 97560 disk drive. Technical Report TR94-220, Dartmouth College, 1994.

[15] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 17(3):109–116, September 1988.

[16] C. Ruemmler and J. Wilkes. Disk Shuffling. Technical Report HPL-91-156, Hewlett Packard Laboratories, October 1991.

[17] C. Ruemmler and J. Wilkes. Unix disk access patterns. In *Proceedings of the USENIX Winter 1993 Technical Conference*, pages 405–420, 1993.

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

[19] J. Schindler and G. Ganger. Automated disk drive characterization. Technical Report CMU-CS-99-176, Carnegie Mellon University, November 1999.

[20] S. W. Schlosser, J. L. Griffin, D. Nagle, and G. R. Ganger. Designing computer systems with MEMS-based storage. In *Architectural Support for Programming Languages and Operating Systems*, pages 1–12, 2000.

[21] M. Seltzer, P. Chen, and J. Ousterhout. Disk scheduling revisited. In *Proceedings of the USENIX Winter 1990 Technical Conference*, pages 313–324, Berkeley, CA, 1990.

[22] M. I. Seltzer and C. Small. Self-Monitoring and Self-Adapting Systems. In *Proceedings of the 1997 Workshop on Hot Topics on Operating Systems*, pages 124–129, Chatham, MA, May 1997.

[23] P. Shenoy and H. Vin. Cello: A Disk Scheduling Framework for Next-generation Operating Systems. In *Proceedings of the 1998 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 44–55, June 1998.

[24] E. A. M. Shriver, A. Merchant, and J. Wilkes. An analytic behavior model for disk drives with readahead caches and request reordering. In *Proceedings of 1998 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 182–191, 1998.

[25] N. Talagala, R. H. Arpaci-Dusseau, and D. Patterson. Microbenchmark-based Extraction of Local and Global Disk Characteristics. Technical Report CSD-99-1063, University of California, Berkeley, 1999.

[26] T. J. Teorey and T. B. Pinkerton. A comparative analysis of disk scheduling policies. *Communications of the ACM*, 15(3):177–184, 1972.

[27] N. C. Thornock, X.-H. Tu, and J. K. Flanagan. A stochastic Disk I/O Simulation Technique. In *Proceedings of the 1997 Winter Simulation Conference*, pages 1079–1086, 1997.

[28] A.-I. A. Wang, P. Reiher, G. J. Popek, and G. H. Kuenning. Conquest: better performance through a disk/persistent-RAM hybrid file system. In *The Proceedings of the USENIX Annual Technical Conference (USENIX '02)*, pages 15–28, Monterey, CA, June 2002.

[29] N. C. Wilhelm. An anomaly in disk scheduling: a comparison of FCFS and SSTF seek scheduling using an empirical model for disk accesses. *Communications of the ACM*, 19(1):13–17, 1976.

[30] J. Wilkes. The Pantheon storage-system simulator. Technical Report HPL-SSP-95-14, HP Laboratories, Palo Alto, CA, December 1995.

[31] B. L. Worthington, G. R. Ganger, and Y. N. Patt. Scheduling algorithms for modern disk drives. In *Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 241–251, Nashville, TN, USA, 1994.

[32] B. L. Worthington, G. R. Ganger, Y. N. Patt, and J. Wilkes. On-line extraction of SCSI disk drive parameters. Technical Report CSE-TR-323-96, Carnegie Mellon University, 1996.

[33] X. Yu, B. Gum, Y. Chen, R. Y. Wang, K. Li, A. Krishnamurthy, and T. E. Anderson. Trading capacity for performance in a disk array. In *Proceedings of the 2000 Symposium on Operating Systems Design and Implementation*, pages 243–258, San Diego, 2000. USENIX Association.