Zone-Based Shortest Positioning Time First Scheduling for MEMS-Based Storage Devices[†]

Bo Hong, Scott A. Brandt, Darrell D. E. Long, Ethan L. Miller,
Karen A. Glocer and Zachary N. J. Peterson[‡]

Storage Systems Research Center
Jack Baskin School of Engineering
University of California, Santa Cruz
{hongbo, sbrandt, darrell, elm, kag}@cs.ucsc.edu, zachary@jhu.edu

Abstract

Access latency to secondary storage devices is frequently a limiting factor in computer system performance. New storage technologies promise to provide greater storage densities at lower latencies than is currently obtainable with hard disk drives. MEMS-based storage devices use orthogonal magnetic or physical recording techniques and thousands of simultaneously active MEMS-based read-write tips to provide high-density low-latency non-volatile storage. These devices promise seek times 10–20 times faster than hard drives, storage densities 10 times greater, and power consumption an order of magnitude lower. Previous research has examined data layout and request ordering algorithms that are analogs of those developed for hard drives. We present an analytical model of MEMS device performance that motivates a computationally simple MEMS-based request scheduling algorithm called ZSPTF, which has average response times comparable to Shortest Positioning Time First (SPTF) but with response time variability comparable to Circular Scan (C-SCAN).

1. Introduction

The huge disparity between memory access times and disk access times has been the subject of extensive research. CPU speed has been increasing rapidly but disk access latency has lagged behind—disk transfer rates have been increasing at 40% per year, while seek times and rotational latency have been increasing at less than 10% per year [5]. This disparity has created a performance bottleneck in computer systems. Many techniques based on limiting the seek and rotational latency of a disk drive have been developed to improve disk, and therefore system, performance [7, 11, 12, 17, 20].

A new class of secondary storage devices based on microelectromechanical systems (MEMS) [1, 10, 19] currently being developed promises seek times 10–20 times faster than hard drives, storage densities 10 times greater, and power consumption an order of magnitude lower. MEMS devices provide non-volatile storage using either physical [19] or magnetic [1] recording techniques to achieve extremely high-density storage. In order to achieve these high densities, MEMS-based storage designs use a non-rotating storage device with storage media on one surface and a large array of read/write heads on another surface directly above the storage media. By moving the surfaces relative to each other using MEMS actuators, each read/write head can access a region of the surface. MEMS-based storage devices are expected to have many other significant advantages over hard disks, including better I/O performance, higher throughput, smaller physical size, lower heat dissipation requirements, and integrated processing and storage [15]. For all of these reasons, MEMS-based storage devices are an appealing next-generation storage technology. However, the characteristics of these devices are very different from those of hard drives, and file system algorithms designed for hard drives are not likely to be optimal for MEMS-based storage devices.

Previous work by Griffin *et al.* [4] showed that standard disk request scheduling algorithms such as First-Come-First-Served (FCFS), Circular Look (C-LOOK), Shortest Seek Time First (SSTF), and Shortest Positioning Time First (SPTF) can be applied to MEMS-based storage devices. They found that, as with disks, SPTF generally provides the lowest response time but exhibits the greatest variation in response times, while FCFS has the least variation in response times but the highest average response time. Their results were obtained by a relatively direct application of the disk-based concepts of sector, track, and cylinder in which a cylinder consists of all of the sectors ac-

[†] This research is supported by the National Science Foundation under grant number CCR-073509 and the Institute for Scientific Computation Research at Lawrence Livermore National Laboratory under grant number SC-20010378.

[‡] Zachary Peterson is now at John Hopkins University.

This paper will appear at the 11th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '03), Orlando, FL, October 2003.

cessible without seeking in the higher-latency dimension. However, the fairly significant differences between MEMS-based storage and disks suggests that these disk-based algorithms are unlikely to be optimal for MEMS-based storage devices. In particular, cylinders in disks are areas of equal seek time, while the physically analogous regions of a MEMS device are not; the relative costs of seeks in the two dimensions are close enough that it is sometimes less costly to seek a short distance in the higher-latency dimension than to seek a long distance in the lower-latency dimension. The goal of this work is therefore to discover the logical analog of a disk cylinder in MEMS-based storage devices and use that knowledge to develop better MEMS-specific request scheduling algorithms.

To develop a better understanding of the access time characteristics of MEMS-based storage devices we created a device simulator that generates seek times between any two points on the device. This allowed us to observe the *seek time equivalence regions* or simply *equivalence regions*, areas of nearly equal seek time from any fixed point, analogous to cylinders in a hard drive. Our results show that unlike hard drives, the equivalence regions in MEMS devices are not a single sector wide but are in fact rectangular with a ratio of about 1:10, indicating that it is sometimes faster to move a short distance latitudinally than to move a long distance longitudinally. This is in direct contrast to hard drives, where most algorithms implicitly assume that it is faster to access any sector in the current track than to access any sector in any other track. These results suggest a new approach to request scheduling algorithms is warranted for such devices.

Based on our knowledge of the equivalence regions we have developed *ZSPTF*, a new MEMS-specific request scheduling algorithm. ZSPTF partitions the MEMS device into a two-dimensional array of *zones* based on equivalence regions. ZSPTF services requests by traversing the zones in order, servicing requests within each zone in Shortest Positioning Time First (SPTF) order before moving on to the next zone.

This simple algorithm is shown to have average seek times lower than almost all previously published MEMS scheduling algorithms including First Come First Served (FCFS), Circular Scan (C-SCAN), Shortest Seek Time First (SSTF), and Aged Shortest Positioning Time First (ASPTF). ZSPTF was only outperformed by SPTF in experiments that measure response times. However, ZSPTF's seek time variability is similar to C-SCAN's and significantly lower than SPTF's. Additionally, ZSPTF does not suffer from the starvation and computational complexity problems that plague SPTF — ZSPTF is easy to implement, provides comparable performance, and ensures fairness.

2. Background

It is important to note that because MEMS-based storage devices are still in their infancy, many of the details are still uncertain. There are several proposed architectures [3, 10, 15, 19], and we have based the physical parameters of our experimental model on the specification from Carnegie Mellon University (CMU) [3, 15]. While the exact performance numbers depend upon the details of that specification, the techniques themselves do not. The ZSPTF algorithm should perform well regardless of the exact characteristics of the eventual devices produced.

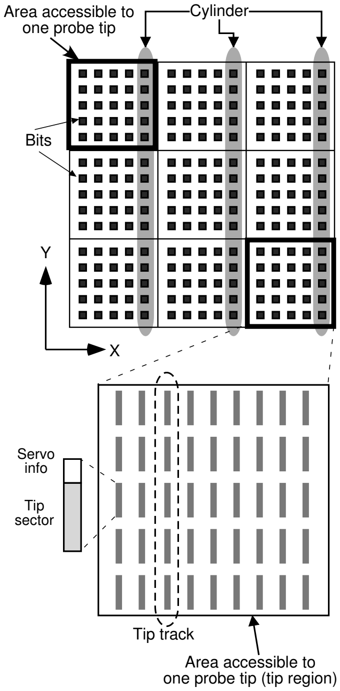

Area accessible to one probe tip
Cylinder
Bits
Y
X
Servo info
Tip sector
Tip track
Area accessible to one probe tip (tip region)

Figure 1. Data layout on a MEMS device.

A MEMS-based storage device is comprised of two main components: groups of probe tips called *tip arrays* that are used to access data on a movable *media sled*. In a modern disk drive, data is accessed by means of an arm that seeks in one dimension above a rotating platter. In a MEMS device, the entire media sled is positioned in the x and y directions by electrostatic forces while the heads remain stationary.[1] Another major difference between a MEMS-based storage device and a disk is that on a MEMS device, multiple tips can be active at the same time. Data can be then be striped across multiple tips, allowing a considerable amount of parallelism. However, power and heat considerations limit the number of probe tips that can be active simultaneously; it is estimated that 200 to 2000 probes will actually be active at once.

Figure 1 illustrates the low level data layout of a MEMS-based storage device. The media sled is logically broken into *tip regions*, defined by the area that is accessible by a single head, approximately 2000 by 2000 bits in size. Each tip in the MEMS device can only read the data in its own

[1] Some MEMS storage device designs, like the IBM Millipede, fix the sled and move the heads. The effect is the same—the heads move relative to the media.

tip region; this limits the maximum sled movement to the dimensions of a single tip region. The smallest unit of data in a MEMS-based storage device is called a *tip sector*. Each tip sector, identified by the tuple $\langle x, y, tip \rangle$, has its own servo information for positioning and its own error correction information. The set of bits accessible to a single tip with the same x coordinate is called a *tip track*, and the set of all bits (under all tips) with the same x coordinate is referred to as a *cylinder*. Also, the set of tip sectors that can be accessed by simultaneously active tips is known as a *logical sector*. For faster access, disk sectors can be striped across logical sectors.

3. Related Work

Researchers have been optimizing storage device seek algorithms for secondary storage devices for decades. Disk seek algorithms have been studied since the 1960s; Teorey and Pinkerton [18] discussed the state of the art in disk scheduling in 1972. Seltzer *et al.* [16] discussed disk scheduling, noting that Circular LOOK (C-LOOK) performed well under high load, and that SSTF was sufficient for low load. More generally, researchers have found that almost any disk seek algorithm will suffice under low load; it is only under moderately high load that seek algorithms are stressed sufficiently to exhibit much performance difference. Worthington *et al.* [20] conducted a more thorough study of disk drive positioning algorithms, including Shortest Positioning Time First (SPTF) which, for disk drives, includes both disk arm positioning and rotational latency.

Though it is possible to study seek algorithms by examining them in a live system, it is more feasible to explore them using an accurate simulation of a storage device. Ruemmler and Wilkes [14] developed an accurate disk drive model, which has since been used to study disk seek algorithms. DiskSim is another storage simulator that has been used to model system behavior [2], and has been adapted to include MEMS devices.

Recently, there has been interest in modeling the behavior of MEMS storage devices. Griffin, Schlosser, Ganger, and Nagle have published extensively on the modeling of MEMS-based storage devices and the optimization of request scheduling for such devices [3, 4, 15]. They showed that for a limited class of MEMS devices, one-dimensional placement and scheduling can be applied efficiently. In that work, data was placed on the MEMS-based device in longitudinally sequential tracks, similar to tracks on a disk drive. However, this method is preferable only if a long longitudinal seek takes less time than a smaller latitudinal seek. They found that for MEMS devices SPTF had the lowest average access time, but the variability of SPTF was very high, as would be expected for a greedy algorithm continually searching for the least expensive "next step." They also explored the use of standard disk-type algorithms such as SSTF and C-LOOK, with minor modifications, on MEMS devices. Their results show that such algorithms can be successfully adapted to MEMS storage.

Madhyastha and Yang [8] have also studied MEMS modeling, with an emphasis on creating more accurate models of MEMS devices. They developed a more realistic access time model that does not assume only an ideal acceleration and takes into account damping and restoring spring forces. Their work informed one analytical seek time analysis.

IBM has developed a prototype device, called *Millipede* [19] that, unlike the CMU model, has a media sled that also moves in the z direction. This enables data to be written using tiny physical marks on the media, as opposed to magnetic recording used in the CMU model. Additional hardware research is also being done at Hewlett Packard and Sandia National Laboratory.

4. Modeling Seek Time

An accurate and tractable model of seek time is important for understanding the seek time characteristics of MEMS-based storage devices. We used the positioning model and physical parameters provided by CMU [3, 4] as the basis for our analysis. The CMU positioning model takes into account the external force (constant but bidirectional, $\pm F$), the spring force, and the initial and final access velocities, which are opposite for odd and even-indexed bit columns. Griffin *et al.* [3] used an iterative approach to solve the model, which is difficult to apply in practice. In this section, we propose an analytic solution to the CMU model. A complete discussion is available in a technical report [6].

Because the actuation mechanisms and control loops for x and y positioning are independent in MEMS–based storage devices, positioning in the x and y dimensions can proceed in parallel. Therefore,

$$t_{seek} = \max(t_x, t_y), \qquad (1)$$

where t_{seek} is the seek time and t_x and t_y are the seek times in the x and y dimensions.

A seek in the x and y dimensions consists of a base seek plus a settling time in the x dimension and necessary turnaround times in the y dimension. t_x thus consists of a base seek plus a settling time, t_{settle}, a function of the resonant frequency of the system, and t_y consists of a base seek plus necessary turnaround times, $t_{turnaround}$, a function of the actuator and spring forces. Both t_{settle} and $t_{turnaround}$ can be easily calculated from the physical parameters of MEMS-based storage devices.

A base seek consists of two phases: acceleration and deceleration. The actuators accelerate the sled toward the destination in the acceleration phase and reverse polarity and decelerate the sled to its final destination and velocity in the deceleration phase. In addition to the actuator force, the sled springs constantly pull the sled toward its center-most position. Because the kinetic energy of the sled is unchanged at the beginning and the end of a base seek, we know when and where to reverse the polarity of the actuators:

$$x_m = \frac{x_0 + x_1}{2} + \frac{k}{4F}(x_1^2 - x_0^2), \qquad (2)$$

where x_m is the position at which actuators reverse polarity, from positive to negative, k is the spring constant, and F is the actuator force.

The phases of acceleration and deceleration in a base seek are described in Equations 3 and 4:

$$\ddot{x} = a - \frac{kx}{m}, \qquad (3)$$

$$\ddot{x} = -a - \frac{kx}{m}, \qquad (4)$$

where m is the sled mass, a is the acceleration by the actuators, and x is the sled displacement. Considering the marginal conditions of Equation 3 and 4, the seek times elapsed during seeking in the x and y dimensions, t_x and t_y, are given in Equations 5 and 6:

$$\begin{aligned} t_x &= \sqrt{\frac{m}{k}}\arccos\Big(\frac{x_m - \frac{ma}{k}}{x_0 - \frac{ma}{k}}\Big) && (5)\\ &\quad + \sqrt{\frac{m}{k}}\arccos\Big(\frac{x_m + \frac{ma}{k}}{x_1 + \frac{ma}{k}}\Big) + t_{settle}, \\ t_y &= \sqrt{\frac{m}{k}}\arcsin\Big(\frac{y_m - \frac{ma}{k}}{\sqrt{(y_0 - \frac{ma}{k})^2 + (v_0\sqrt{\frac{m}{k}})^2}}\Big) && (6)\\ &\quad - \sqrt{\frac{m}{k}}\arcsin\Big(\frac{y_0 - \frac{ma}{k}}{\sqrt{(y_0 - \frac{ma}{k})^2 + (v_0\sqrt{\frac{m}{k}})^2}}\Big) \\ &\quad + \sqrt{\frac{m}{k}}\arcsin\Big(\frac{y_1 + \frac{ma}{k}}{\sqrt{(y_1 + \frac{ma}{k})^2 + (v_1\sqrt{\frac{m}{k}})^2}}\Big) \\ &\quad - \sqrt{\frac{m}{k}}\arcsin\Big(\frac{y_m + \frac{ma}{k}}{\sqrt{(y_1 + \frac{ma}{k})^2 + (v_1\sqrt{\frac{m}{k}})^2}}\Big) \\ &\quad + t_{turnaround}. \end{aligned}$$

The physical constants in Equations 5 and 6 are given or can be easily derived from design parameters of MEMS-based storage devices. Using Equations 1, 2, 5, and 6, we can very accurately estimate the seek time between any two positions in the sled.

5. Seek Time Analysis

To gain a more complete understanding of the seek time characteristics of MEMS-based storage devices, we conducted experiments using Equations 5 and 6 and the physical parameters provided by CMU, shown in Table 1. From a given sector, we calculated the physical distance and seek time to all other sectors on the device. We examined two representative locations on the sled: the center and the top-right corner. Because the access velocities on even-indexed and odd-indexed bit columns are positive and negative, respectively, we examined two tip sectors on even-indexed and odd-indexed bit columns for each location. The sectors we examined are $(0,0)$ and $(25,0)$, which are in the center, and $(1250,1250)$ and $(1225,1250)$, which are in the top-right corner.

Table 1. Default MEMS-based storage device parameters.

device capacity	3.2 GB
number of tips	6400
maximum concurrent tips	1280
sled mobility in x and y	100 μm
sled acceleration in x and y	803.6 m/s^2
sled access speed	28 mm/s
sled resonant frequency	739.0 Hz
spring factor	75%
media bit cell size	40×40 nm
bits per tip region (M×N)	2500×2500

Figure 2(a) shows the seek times to every position on the sled from tip sector $(0,0)$. The seek times are shown in 0.1 ms increments, and larger seek times are indicated by darker colors. Seek times are independent of the y dimension movements when the x dimension movements are large because of the extra settling time in the x dimension. The jagged boundaries of the similarly colored regions indicate that seek times to neighboring tip sectors can differ due to the unequal numbers of turnarounds required to access the data. These interesting effects are discussed further below.

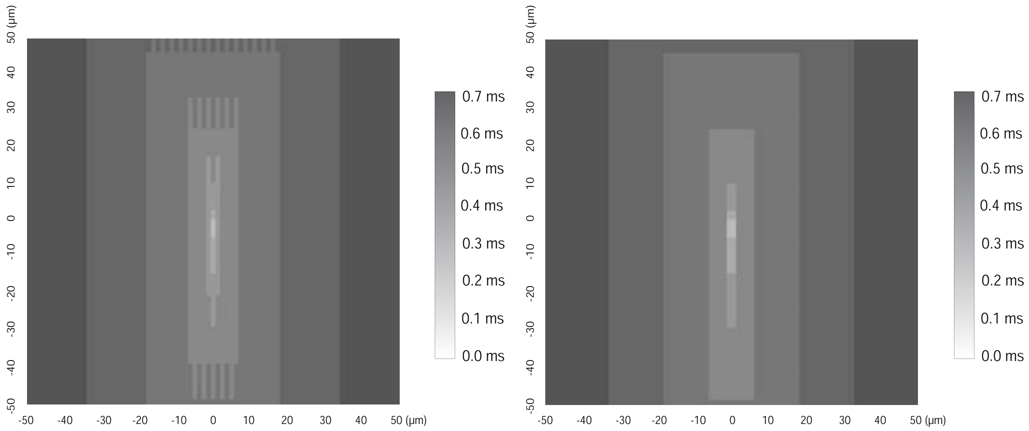
Because of different access velocities on tip sectors $(0,0)$ and $(25,0)$, the equivalence regions for these sectors are different although the starting locations of the seeks are close. Nevertheless, the shapes and sizes of equivalence regions of for these sectors are almost identical.

The shapes of the equivalence regions are rectangular, but with jagged top and bottom edges. This is due to the different numbers of turnarounds required to access even- and odd-indexed bit columns. Figures 2(b) show the equivalence regions from the center to even-indexed bit columns, demonstrating clearly that the non-uniformity in the previous graphs was due to this effect. The equivalence regions of tip sector $(1250,1250)$ and tip sector $(1225,1250)$ that are distant from the center are approximately the same size and shape as those of the sectors near the center.

The equivalence regions in the above analysis exhibit an x:y size ratio that is on average about 1:10. This means that it is cheaper to move one unit of distance in the x direction than to move more than ten units in the y direction. These results suggest that data layout and scheduling algorithms that take advantage of this ratio are likely to outperform those that are based on a disk-based model of tip cylinders, which assumes an implicit ratio of 1:∞.

6. The Zone-based Shortest Positioning Time First Algorithm

Although SPTF has very good average response times over a wide range of request rates, it suffers from a high co-



(a) Equivalence regions to all bit columns

(b) Equivalence regions to even-indexed bit columns

Figure 2. Seek time equivalence regions from tip sector $(0, 0)$.

efficient of variation in response times and can cause starvation at high request rates [4]. It can also be computationally intensive for large numbers of requests. Based on our seek time analysis we have developed zone-based Shortest Positioning Time First (ZSPTF), an algorithm that, like SPTF, groups nearby requests together to reduce average response time, but also dramatically reduces the coefficient of variation for response times.

ZSPTF divides the MEMS storage media into a set of *zones* based on seek time equivalence regions. Zones are serviced in a C-SCAN order and multiple requests within a zone are serviced in SPTF order. Once all of the pending requests in the current zone have been serviced, the algorithm moves on to the next zone with pending requests; zones in which there are no requests waiting for service are skipped (*i.e.*, the device doesn't actually seek to zones in which there are no requests pending).

Although ZSPTF provides a high degree of fairness in a way similar to C-SCAN by servicing zones in a fixed order, its variability tends to be like SPTF when a steady stream of requests keeps arriving to a single zone, resulting in potential starvation. To address the problem of potential starvation, we have also developed a variation of ZSPTF, Zone-based arrival-Time-constrained Shortest Positioning Time First (ZTSPTF). ZTSPTF improves the fairness of ZSPTF in a way similar to FCFS by only servicing requests that arrive before ZTSPTF begins scheduling requests in a zone.

The number of zones into which the device is divided will affect the performance of the ZSPTF and ZTSPTF algorithms. Fewer zones leads to longer average queue lengths for each zone, allowing SPTF to perform better within each zone but causing higher variability of seek times. Larger numbers of zones leads to lower variability, *i.e.* greater fairness and starvation resistance, but higher average seek times. The performance of ZSPTF and ZTSPTF with several zone sizes will be examined in Section 7.

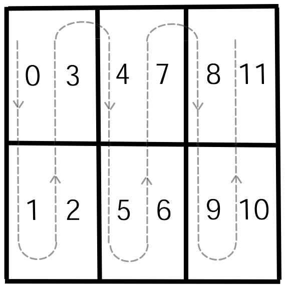


Figure 3. An example of partitioning the MEMS storage area and the traversed order. The sled is divided into 12 zones.

Figure 3 shows an example of partitioning the MEMS storage media into 12 zones and the order in which zones are traversed. Note that each cell in Figure 3 contains two zones because we divide even-indexed and odd-indexed bit columns into different zones.

Like SPTF, ZSPTF keeps the average service time low by grouping nearby requests together, thereby reducing average seek distance. Unlike SPTF, however, ZSPTF guarantees fairness in a way similar to C-SCAN by servicing zones in a fixed order. This servicing order prevents a large number of new requests in lower-numbered zones from indefinitely delaying the servicing of requests in higher-numbered zones. ZTSPTF can even provide better fairness by imposing a FCFS-like order on scheduling.

ZSPTF and ZTSPTF are computationally simpler than SPTF and its variants. SPTF must recompute the positioning time of each request in its queue after servicing each request, and the time needed to do so is proportional to queue length. With longer queue lengths, this repeated recomputation may not be practical. In ZSPTF and ZTSPTF, the average queue length of each zone will on average be much less than that of SPTF. Because the zones are small and based on seek time equivalence regions, this recalculation may not be necessary after each request.

Another advantage of ZSPTF is its customizability. The order in which the algorithm traverses the zones is not fixed; rather, the order can be stored in an array, allowing easy conversion from a "physical" zone number to a logical zone number in constant time. In the following experiments, we used a fixed order analogous to the one shown in Figure 3.

7. Experimental Analysis

Because MEMS storage devices are not readily available, we used DiskSim [2] to simulate requests and device service. This simulator has been used in previous studies of MEMS seek algorithms [3, 4, 15], making it a good choice to allow direct comparison with prior work in the area.

Many studies of access time optimization use traces of real file system requests to produce more realistic results. We used two traces, labeled *server* and *user*, that were collected from an HP-UX time-sharing system and an HP-UX workstation in 1999. These systems were also traced and studied in 1992 [13]. The server trace is a one-hour subset of the HP Cello news disk (Seagate Barracuda 9, 9.1 GB) trace with 118,760 requests. The user trace is a one-hour subset of the HP Hplajw user disk (Seagate Barracuda 4, 4.3 GB) trace with 75,304 requests. We believe that one-hour heavy workloads are long enough to exercise request scheduling algorithms and reveal their performance difference. The average request arrival rate is 33.0 requests per second for the server trace and 20.9 requests per second for the user trace. The logical sequentiality (the percentage of requests that are at adjacent disk addresses or addresses spaced by the file system interleave factor) of the server and user traces is 0.5% and 82.9%, respectively. In general, the user trace is much more sequential than the server trace, and 75.3% of the requests in the user trace have inter-arrival times less than 5 ms.

In order to explore a range of workload intensities, we scale the traced inter-arrival times to produce a range of average inter-arrival times. Hence, a scaling factor of one corresponds to replaying the trace at its original speed; a scaling factor of two corresponds to halving the traced inter-arrival times and replaying the trace twice as fast, and so on.

Because the capacities of the traced disks are larger than the default capacity of a MEMS media sled (3.2 GB), we used multiple media sleds in a MEMS-based storage device to bridge the gap in capacity. The numbers of media sleds used for the server and user traces are three and two, respectively. The low-range disk logical block numbers (LBN) are mapped to low-numbered media sleds. The sleds move simultaneously and their relative positions are unchanged.

The layout of blocks on the storage device is an important issue. Allocation algorithms that take storage device characteristics into account tend to outperform those that do not. For example, the Berkeley Fast File System [9] groups related data and metadata into cylinder groups in an effort to reduce seek time. Disk-based allocations are unlikely to be optimal for MEMS-based devices, but in these experiments no attempt has been made to optimize the layout for MEMS-based devices. Specifically, we used the disk-analogous MEMS data layout proposed by Griffin *et al.* [3, 4]. This layout favors sequential workloads, as we can see in Section 8.

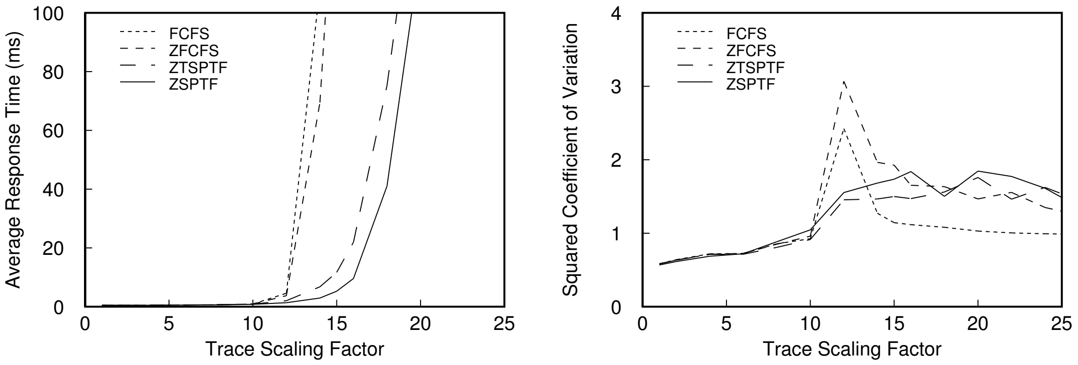
To generate the results we modified a version of DiskSim to include our seek algorithms, ZSPTF and ZTSPTF, and ran the new algorithms a well as the existing algorithms on the same request streams. All of the simulations in Section 8 used the default parameters reported in Griffin *et al.* [4], as shown in Table 1.

7.1. Comparison of In-Zone Algorithms

The essence of zone-based algorithms is to partition the MEMS storage media into a set of seek-time-constrained regions based on seek equivalence regions and service the regions in a fixed order. One question that arises is what scheduling algorithm to use within each zone. To determine this we implemented several in-zone scheduling algorithms and found that SPTF gives the best overall performance. For simplicity, we only discuss the results of ZSPTF, ZTSPTF, and Zone-based First Come First Served (ZFCFS). The zone size we used here is 120×1200 bits^2.

In addition to the average response time, another important factor for request scheduling algorithms is the squared coefficient of variation of response times (σ^2/μ^2), where σ is the standard deviation of response times and μ is the average response time. This metric measures the consistency of the response time for requests [18, 20]. A low coefficient of variation means that the service times for the requests are likely to be near the average, while a high coefficient of variation is an indication that some requests may get fast service at the expense of others that can suffer starvation. In other words, it is a measure of fairness and starvation resistance.

Figure 4 shows the average response times and squared coefficients of variation of response times of FCFS, ZFCFS, ZSPTF, and ZTSPTF with different trace scaling factors for the user trace. Although the regions in zone-based algorithms are seek-time-constrained, a good in-zone scheduling algorithm is still critical to avoid unnecessary settling and turnaround times. Both FCFS and ZFCFS have good performance only under light workloads. ZSPTF outperforms ZTSPTF by 31–122% under the moderate to heavy user workloads. The overall starvation resistance of ZSPTF and ZTSPTF are quite similar; ZSPTF has up to 15% higher squared coefficient of variation than ZTSPTF only under moderate workloads. ZSPTF and ZTSPTF have very similar average response times and squared coefficients of variation for all range of request arrival rates of the non-sequential server workloads. In summary, ZSPTF has the best performance with reasonable variability among these algorithms under different workloads.



(a) Average response times

(b) Response time variation

Figure 4. Performance comparison of different in-zone scheduling algorithms on the user trace.

7.2. Potential Zone Sizes

As discussed in Section 5, there is a rectangular relationship between equivalence regions and starting sectors if we study equivalence regions for even-indexed and odd-indexed bit columns separately. The sizes of equivalence regions of tip sectors near the center are a little different from those of tip sectors distant from the center, which suggests different zone sizes for different sectors. However, for simplicity, we used uniformly sized zones. As mentioned above, we found the best ratio of width and height of a zone to be around 1:10.

The size of a zone is determined by a seek time threshold such that the seek time between any two positions within a zone should be less than the threshold. The minimum threshold must be more than the settling time in the x dimension, which is 0.215 ms in our simulations. Using Equations 1, 5, and 6, a threshold of 0.3 ms gives us zones 60×600 bits^2 in size. A threshold of 0.4 ms gives us zones 120×1200 bits^2 in size. To make the MEMS media sled divisible by these zone sizes, we slightly change the sled mobility in x and y from 100 μm to 96 μm for simplicity (in practice, such a change is unnecessary because the $x : y$ size ratio of zones can be slightly different from 1:10). Therefore, the bits per tip region is 2400×2400. As an extreme case we will also examine zones 240×2400 bits^2 in size. For simplicity, we will refer to the ZSPTF and ZTSPTF algorithms with zone sizes of 60×600, 120×1200, and 240×2400 bits^2 as Z(T)SPTF(60, 600), Z(T)SPTF(120, 1200), and Z(T)SPTF(240, 2400).

7.3. Zone Size Results

From the simulations, we find that although ZSPTF(60, 600) has slightly better overall starvation resistance than ZSPTF(120, 1200), it has 12–21% greater response time under the heavy non-sequential server workloads and 69–118% greater response time under the moderate to heavy sequential user workloads. ZSPTF(120, 1200) and ZSPTF(240, 2400) have similar average response times and squared coefficients of variation for all range of request arrival rates of the non-sequential server workloads. ZSPTF(240, 2400) outperforms ZSPTF(120, 1200) by 18–42% under the moderate to heavy sequential user workloads. However, ZSPTF(240, 2400) suffers higher response time variation than ZSPTF(120, 1200). Its squared coefficient of variation is higher than that of ZSPTF(120, 1200) by 22–142% under the moderate to heavy sequential user workloads.

Our experiments show that the arrival time constraint of ZTSPTF and the choice of the zone size have no significant impact on the performance and variability of ZSPTF under non-sequential workloads. However, they do have great impact on ZSPTF under moderate to heavy sequential workloads. The arrival time constraint provides a higher degree of fairness by bounding the number of requests serviced within a zone to the size of the request queue when that zone was entered, but can degrade performance by failing to take full advantage of the sequentiality of the request stream. Larger zone sizes can improve performance by making the request queue longer and the scheduler more efficient but may result in higher response time variability. Based on its good performance across a wide range of request rates under different workloads, we used ZSPTF(120, 1200) as the basis for comparison against existing request scheduling algorithms.

8. Comparison of ZSPTF with Existing Algorithms

Many disk request scheduling algorithms have been proposed and studied over the years. These algorithms can be adapted to MEMS-based storage devices once these devices are mapped onto a disk-like interface. Our comparisons focuses on five: FCFS, C-SCAN, SSTF, SPTF, and ASPTF. FCFS has good performance only under light workloads, but we include it here as a baseline for comparison. C-SCAN services requests in ascending logical block number (LBN) order, starting over with the lowest LBN when the disk arms reach the edge of the disk. Our MEMS implementation of SSTF uses the number of tracks between the last accessed LBN and the desired LBN as an estimate of the seek time. SPTF always services the request with smallest positioning delay [7, 16] from the current position. SPTF explicitly considers both seek time and rotational latency for disks. For MEMS, SPTF considers seek time in both x and y dimensions. Although SPTF generally has the best performance, it suffers from high response time variability. To address this problem, ASPTF was proposed by Jacobson and Wilkes [7]. Besides seek time and rotational latency, ASPTF also considers the time that the request has been waiting for service. The resulting positioning delay, t_{eff}, is given in Equation 7:

$$t_{eff} = t_{pos} - \frac{w}{1000} \times t_{wait}, \tag{7}$$

where w is the aging factor, t_{pos} is the positioning time and t_{wait} is the waiting time.

The aging factor w can be varied from zero (pure SPTF) to infinity (pure FCFS). Experimentally, we chose $w = 5$ because ASPTF(5) exhibited good performance across a range of request rates. In fact, this factor is close to the aging factor, 6, proposed for the disk ASPTF algorithm in [7, 20]. The reason is that although MEMS-based storage devices are much faster, they have ratios of request throughput to data bandwidth similar to those of disks from the early 1990s [4]. Therefore, we compare the ZSPTF algorithm to ASPTF(5).

Figure 5(a) and 5(b) show the average response times of the different scheduling algorithms with different trace scaling factors for the server and user traces. As expected, FCFS has good performance only under light workloads. As the trace scaling factor increases, the performance of FCFS degrades dramatically. C-SCAN and SSTF also work well under light workloads but suffer under heavy workloads. SPTF always has the lowest average response time under all trace scaling factors.

ZSPTF has performance similar to that of C-SCAN and SSTF under light and moderate server workloads and significantly outperforms C-SCAN and SSTF, by as much as a factor of 2.8, under heavy server workloads. ZSPTF, C-SCAN, and SSTF perform similarly under light user workloads. However, ZSPTF exceeds C-SCAN and SSTF by almost a factor of 2–27 under moderate and heavy user workloads.

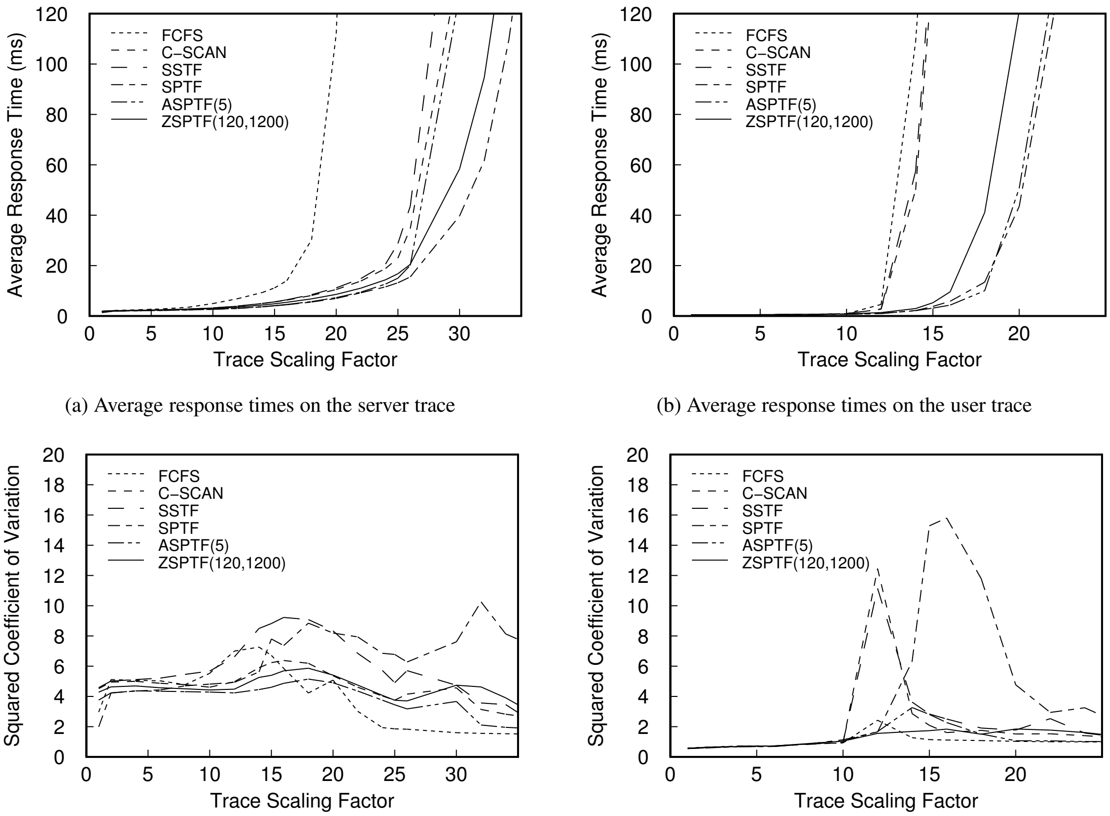
SPTF and ASPTF(5) perform slightly better than ZSPTF, by up to 7%, under light workloads. However, SPTF and ASPTF(5) perform better than ZSPTF, by 10–40% under moderate workloads. Under heavy server workloads, ZSPTF significantly outperforms ASPTF(5), by a factor of 1.2–2.6, whose performance degrades because of the aging effect, and performs almost as well as SPTF. However, SPTF and ASPTF(5) perform better than ZSPTF by 30–76% under heavy user workloads. This is because ZSPTF cannot take advantage of the high sequentiality of the user workloads as much as SPTF and ASPTF(5). The reason is that the data layout was done assuming a disk-based model rather than a MEMS-based model. This inherently punishes algorithms that employ a MEMS-based model. In the future, we will generate MEMS-specific layouts that will further highlight the benefits of MEMS-specific request scheduling.

When the workload is light, the queue lengths of SPTF, ASPTF, and ZSPTF are all small enough that no significant optimization can take place by reordering requests. And when the queue lengths are long, all three algorithms can achieve excellent performance by reordering requests. However, when the workload is moderate, the queue lengths of SPTF and ASPTF(5) are long enough to allow for good optimization through reordering, but the queue length of each zone in ZSPTF is too small to allow for any significant optimization. We propose a solution to this problem in Section 9.

Figure 5(c) and 5(d) show the squared coefficients of variation of response times of different scheduling algorithms under different trace scaling factors for the server and user traces. In general, FCFS has the overall lowest squared coefficients of variation under moderate and heavy workloads. However, we are not interested in it because of its poor average response time. C-SCAN and ZSPTF have similar squared coefficients of variation for all range of request arrival rates because ZSPTF traverses the media area in the order similar to that of C-SCAN. ZSPTF and ASPTF(5) have very similar squared coefficient of variation under light workloads. ZSPTF has 8–44% higher squared coefficient of variation than ASPTF(5) under moderate and heavy workloads. SSTF and SPTF suffer higher viability of response times than other scheduling algorithms for all range of request arrival rates. SPTF has 16–45% and 83–700% higher squared coefficient of variation than ZSPTF under moderate and heavy server and user workloads, respectively.

Although ASPTF(5) has the overall best performance in terms of average response time and squared coefficient of variation of response times under light and moderate workloads and heavy sequential workloads (the user trace), its high computational cost prevents its use in real systems. To quantify this, we ran some experiments on a Linux machine (Pentium III 700 MHz, 512 MB). We found that SPTF and ASPTF take 7.1 μs for each entry in the queue. Even with table-driven calculation of arcsin and arccos functions, they took about 5.1 μs for each entry in the queue. This means that for a queue length of 200 requests SPTF and ASPTF take more than 1 ms to determine which request to service next. This is longer than the maximum seek time of the MEMS-based storage device using the default parameters. Therefore, it is impractical to apply SPTF and ASPTF in real systems. By contrast, the queue length of each zone in the ZSPTF algorithm is on average 10–100 times smaller and the computational cost of computing SPTF within each zone is therefore limited and tolerable.

Based on its good average response time, very low coefficient of variation, and low computational cost, we believe



(a) Average response times on the server trace

(b) Average response times on the user trace

(c) Response time variation on the server trace

(d) Response time variation on the user trace

Figure 5. Performance comparison of different scheduling algorithms.

that ZSPTF is an ideal algorithm for use with MEMS-based storage devices.

9. Future Work

The ZSPTF algorithm improves MEMS scheduling by providing near-SPTF performance with much lower response time variability at high request rates. However, it performs less well at moderate request rates because there are too few requests in each zone to yield much optimization. To address this, we are exploring a variable-sized zoning technique called *pyramiding*. In Section 7, we showed that with larger zone sizes and appropriate width-height ratios, ZSPTF can achieve better performance. Instead of optimizing within fixed-size zones, pyramiding merges nearby zones with too few requests and schedules all requests from the merged zones together. Figure 6 illustrates an example of pyramiding in ZSPTF. In this example, ZSPTF can function as either a 8 × 1, 16 × 2, or 32 × 4 grid depending on the request rate. Although instantaneous request rates are difficult to measure, we can use the length of the request queue to approximate the request rate and to determine the granularity of the grid. Our preliminary experiments with pyramiding are promising: we have found that dynamically merging up to 16 neighboring zones eliminates half of the performance difference between ZSPTF and ASPTF(5) under moderate workloads.

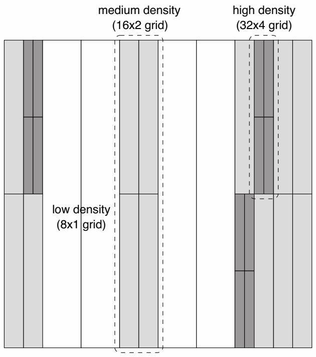


Figure 6. Fractal breakup of the MEMS device grid.

We are also exploring data layout issues for MEMS devices. File systems have long clustered related data together; for example, the Berkeley Fast File System [9] lays out data in cylinder groups. We believe that grouping related data within zones, similar to a method suggested by Schlosser *et al.* [4], and using a zone-based scheduling algorithm will provide considerable performance improvement over simply using mechanisms based on logical block numbers.

10. Conclusions

As new types of storage devices are developed, it is necessary to revisit the issue of scheduling to reduce access latency to data on the devices. We introduced a new scheduling algorithm, ZSPTF, for MEMS-based storage devices, and showed that ZSPTF exhibits a combination of high performance over a wide range of request rates while maintaining very low variability over the same range. We explored a design trade-off of zone size for the ZSPTF algorithm and showed that the zone size of 120×1200 bits^2 is the best among the sizes we tested.

Our results show that ZSPTF has better average response time than FCFS, C-SCAN, and SSTF and better than ASPTF on heavy non-sequential workloads. ZSPTF also has response time variability as much as 50–160% lower than of that of SPTF. Finally, ZSPTF avoids the practical implementation problems that plague SPTF and ASPTF, making ZSPTF an attractive choice for systems use in MEMS devices.

As MEMS-based storage devices are developed and put into general use, it will be necessary to modify file systems to take full advantage of their unique characteristics. The ZSPTF algorithm provides a combination of high performance, low variability, fairness, starvation avoidance, ease of implementation and customizability for varying device characteristics, making it an ideal seek algorithm for storage systems built around MEMS-based devices.

Acknowledgments

We are grateful to Greg Ganger for allowing us to use DiskSim to simulate MEMS behavior. We thank also John Wilkes of Hewlett-Packard Laboratories and his group for providing us traces.

References

[1] L. Carley, J. Bain, G. Fedder, D. Greve, D. Guillou, M. Lu, T. Mukherjee, S. Santhanam, L. Abelmann, and S. Min. Single-chip computers with microelectromechanical systems-based magnetic memory. *Journal of Applied Physics*, 87(9):6680–6685, May 2000.

[2] G. R. Ganger, B. L. Worthington, and Y. N. Patt. The DiskSim simulation environment version 2.0 reference manual. Technical report, Carnegie Mellon University / University of Michigan, Dec. 1999.

[3] J. L. Griffin, S. W. Schlosser, G. R. Ganger, and D. F. Nagle. Modeling and performance of MEMS-based storage devices. In *Proceedings of the 2000 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 56–65, June 2000.

[4] J. L. Griffin, S. W. Schlosser, G. R. Ganger, and D. F. Nagle. Operating system management of MEMS-based storage devices. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 227–242, Oct. 2000.

[5] J. L. Hennessy and D. A. Patterson. *Computer Architecture—A Quantitative Approach*. Morgan Kaufmann Publishers, 3rd edition, 2003.

[6] B. Hong and S. A. Brandt. An analytical solution to a MEMS seek time model. Technical Report UCSC-CRL-02-31, Storage Systems Research Center, University of California, Santa Cruz, Sept. 2002.

[7] D. M. Jacobson and J. Wilkes. Disk scheduling algorithms based on rotational position. Technical Report HPL-CSP-91-7rev1, Hewlett-Packard Laboratories, Concurrent Systems Project, Mar. 1992.

[8] T. Madhyastha and K. P. Yang. Physical modeling of probe-based storage. In *Proceedings of the 18th IEEE Symposium on Mass Storage Systems and Technologies*, pages 207–224, Apr. 2001.

[9] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry. A fast file system for UNIX. *ACM Transactions on Computer Systems*, 2(3):181–197, Aug. 1984.

[10] Nanochip Inc. Nanochip: Array nanoprobe mass storage IC. Nanochip web site, at http://www.nanochip.com/preshand.pdf, 1999.

[11] J. K. Peacock. The Counterpoint Fast File System. In *Proceedings of the Winter 1988 USENIX Technical Conference*, pages 243–249. USENIX, Jan. 1988.

[12] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems*, 10(1):26–52, Feb. 1992.

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

[14] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–29, Mar. 1994.

[15] S. W. Schlosser, J. L. Griffin, D. F. Nagle, and G. R. Ganger. Designing computer systems with MEMS-based storage. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 1–12, Cambridge, MA, Nov. 2000.

[16] M. Seltzer, P. Chen, and J. Ousterhout. Disk scheduling revisited. In *Proceedings of the Winter 1990 USENIX Technical Conference*, pages 313–323, Jan. 1990.

[17] E. Shriver, E. Gabber, L. Huang, and C. A. Stein. Storage management for web proxies. In *Proceedings of the 2001 USENIX Annual Technical Conference*, pages 203–216. USENIX, June 2001.

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

[19] P. Vettiger, M. Despont, U. Drechsler, U. Urig, W. Aberle, M. Lutwyche, H. Rothuizen, R. Stutz, R. Widmer, and G. Binnig. The "Millipede"—More than one thousand tips for future AFM data storage. *IBM Journal of Research and Development*, 44(3):323–340, 2000.

[20] B. Worthington, G. Ganger, and Y. Patt. Scheduling algorithms for modern disk drives. In *Proceedings of the 1994 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 241–251, May 1994.