Appears in Proceedings of the 3rd Conference on File and Storage Technologies, March 2004

MEMS-based storage devices and standard disk interfaces: A square peg in a round hole?

Steven W. Schlosser, Gregory R. Ganger
Carnegie Mellon University

Abstract

MEMS-based storage devices (MEMStores) are significantly different from both disk drives and semiconductor memories. The differences motivate the question of whether they need new abstractions to be utilized by systems, or if existing abstractions will be sufficient. This paper addresses this question by examining the fundamental reasons that the abstraction works for existing devices, and by showing that these reasons also hold for MEMStores. This result is shown to hold through several case studies of proposed roles MEMStores may take in future systems and potential policies that may be used to tailor systems' access to MEMStores. With one noted exception, today's storage interfaces and abstractions are as suitable for MEMStores as for disks.

1 Introduction

MEMS-based storage devices (MEMStores) offer an interesting new component for storage system designers. With tiny mechanical positioning components, MEMStores offer disk-like densities (which are consistently greater than FLASH or MRAM projections) with order of magnitude latency and power reductions relative to high-performance and low-power disks, respectively. These features make MEMStores worthy of exploration now, so that designers are ready when the devices become available.

A debate has arisen, during this exploration, about which form of algorithms and interfaces are appropriate for MEMStores. Early work [11, 28] mapped the linear logical block number (*LBN*) abstraction of standard storage interfaces (SCSI and IDE/ATA) onto MEMStores, and concluded that MEMStores looked much like disks. From anecdotal evidence, it is clear that many researchers are unhappy with this; since MEMStore mechanics are so different from disks, they assume that MEMStores **must** need a new abstraction. Several groups [14, 29, 35] are exploring more device-specific approaches. As is often the case with such debates, we believe that each "side" is right in some ways and wrong in others. There is clearly a need for careful, balanced development of input for this debate.

We divide the aspects of MEMStore use in systems into two categories: roles and policies. MEMStores can take on various *roles* in a system, such as disk replacement, cache for hot blocks, metadata-only storage, etc. For the debate at hand, the associated sub-question is whether a system using a MEMStore is exploiting something MEMStore-specific (e.g., because of a particularly well-matched access pattern) or just benefitting from its general properties (e.g., that they are faster than current disks). In any given role, external software uses various *policies*, such as data layout and request scheduling, for managing underlying storage. The sub-question here is whether MEMStore-specific policies are needed, or are those used for disk systems sufficient.

The contribution of this paper is to address this core question about the use of MEMStores in systems:

> *Do MEMStores have unique, device-specific characteristics that a computer system designer should care about, or can they just be viewed as small, low-power, fast disk drives?*

Of course, MEMStores may realize performance and power characteristics that are unachievable with real disk technologies. The question restated, then, is: would a hypothetical disk, scaled from existing technology to the same average performance as a MEMStore, look the same to the rest of the system as a MEMStore? If MEMStores have characteristics that are sufficiently different from disk drives, then systems should use a different abstraction to customize their accesses to take advantage of the differences. If MEMStores do not have sufficiently different characteristics, then systems can simply treat MEMStores as fast disks and use the same abstraction for both.

To help answer this question, we use two simple objective tests. The first test, called the *specificity test*, asks: Is the potential role or policy truly MEMStore-specific? To test this, we evaluate the potential role or policy for both a MEMStore and a (hypothetical) disk drive of equivalent performance. If the benefit is the same, then the potential role or policy (however effective) is not truly MEMStore-specific and could be just as beneficial to disk drives. The second test, called the *merit test*, asks: Given that a potential role or policy passes the specificity test, does it make enough of an impact in performance (e.g., access speed or energy consumption) to justify a new abstraction? The test here is a simple improvement comparison, e.g., if the system is less than 10% faster when using the new abstraction, then it's not worth the cost.

In most aspects, we find that viewing MEMStores as fast disks works well. Although faster than disks, MEMStores share many of their access characteristics. Signal processing and media access mechanisms strongly push for a multi-word storage unit, such as the ubiquitous 512 byte block used in disks. MEMStore seek times are strongly distance-dependent, correlated with a single dimension, and a dominant fraction of access time, motivating data layouts and scheduling algorithms that are similar to those used for disks. After positioning, sequential access is most efficient, just like in disks. The result is that most disk-based policies will work appropriately, without specialization for MEMStores, and that most roles could equally well be filled by hypothetical disks with equivalent average-case performance and power characteristics.

Our model of MEMStores is based on lengthy discussions with engineers who are designing the devices, and on an extensive study of the available literature. However, as MEMStores are not yet available to test and characterize, it is impossible to know for sure whether the model is entirely accurate. Therefore, the conclusions of this paper are subject to the assumptions that we, and others, have made. The theme of the paper remains, though, that researchers should apply the objective tests to determine whether a suggested role or policy is specific to MEMStores. As time goes on and our understanding of MEMStore performance becomes more detailed, or as alternative designs appear, we believe it is useful for these tests to be re-applied.

In a few aspects, MEMStore-specific features can provide substantial benefits for well-matched access patterns, beyond the performance and power levels that would be expected from hypothetical fast disks. This paper discusses three specific examples. First, tip-subset parallelism flexibility, created by expected power and component sharing limitations, can be exploited for two-dimensional data structures accessed in both row- and column-order. Second, lack of access-independent motion (e.g., continuous rotation) makes repeated access to the same location much more efficient than in disks, fitting read-modify-write access sequences well. Third, the ratio of access bandwidth to device capacity is almost two orders of magnitude higher than disk drives, making full device scans a more reasonable access pattern.

The rest of this paper is organized as follows. Section 2 overviews MEMS-based storage and related work. Section 3 describes the standard storage interface and how it works for disks. Section 4 explores how key aspects of this interface fit with MEMStore characteristics. Section 5 gives results applying the objective tests to several roles and policies for MEMStores. Section 6 identifies unique features of MEMStores and how they could be exploited for specific application access patterns. Section 7 discusses major assumptions and their impact on the conclusions. Section 8 summarizes the paper.

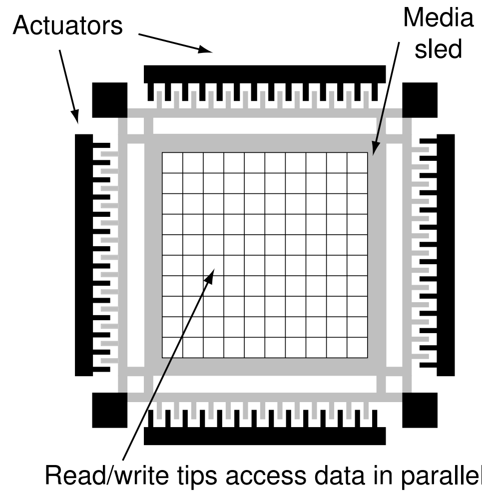


Figure 1: **High-level view of a MEMStore.** The major components of a MEMStore are the sled containing the recording media, MEMS actuators that position the media, and the read/write tips that access the media. The simplified device shown here has a ten by ten array of read/write tips, each of which accesses its own portion (or *square*) of the media. As the media is positioned, each tip accesses the same position within its square, thus providing parallel access to data.

2 Background

MEMStores store data in a very small physical medium that is coated on the surface of a silicon chip. This storage is non-volatile, just as in disk drives. Physically, the devices are much smaller than disks, on the order of a few square centimeters, they store several gigabytes of data, and they access data in a few milliseconds. This section describes in some detail how MEMStores are built, how various designs differ, and what they have in common.

Microelectromechanical systems (MEMS) are microscopic mechanical machines that are fabricated on the surface of silicon chips using techniques similar to those used to make integrated circuits [18]. MEMS devices are used in a wide range of applications, such as accelerometers for automotive airbag systems, high-quality projection systems, and medicine delivery systems. MEMStores use MEMS machinery to position a recording medium and access the data stored in it.

A high-level picture of a MEMStore appears in Figure 1. There are three main components: the media sled, the actuators, and the read/write tips. Data is recorded in a medium that is coated onto the media sled, so named because it is free to move in two dimensions. It is attached to the chip substrate by beam springs at each corner. The media sled is positioned by a set of actuators, each of which pulls the sled in one dimension. Data is accessed by a set of several thousand read/write tips, which are analogous to the heads of a disk drive.

Accessing data requires two steps. First, the media sled is positioned or "seeks" to the correct offset. Second, the sled moves at a constant rate as the read/write tips access the data stored in the medium. The appropriate subset of tips is engaged to access the desired data.

There are three important differences between the positioning of disk drives and MEMStores. First, the media in the MEMStore can be positioned much more quickly than in a disk because the size, mass, and range of motion of the components are significantly smaller. The seek time of a disk drive averages around 5 ms, while that of a MEMStore is expected to be less than 1 ms. Second, there is no access-independent motion in a MEMStore like the rotation of a disk drive's platters.[1] The rotating media of a disk drive adds, essentially, a random variable (uniform from zero to the full revolution time) that is independent of the access itself to positioning time. Third, positioning takes place in two dimensions.

The last of these differences, that positioning is two-dimensional in nature, is one of the most radical departures of MEMStores from disk drives. Positioning in each dimension takes place independently and in parallel, making the overall positioning time equal to the longer of the two. Once the sled arrives at its destination, there is expected to be a significant settle time while the actuators eliminate oscillations. Section 4.2 discusses the impact of this difference on systems.

2.1 Related work

Building practical MEMStores has been the goal of several major research labs, universities, and startup companies around the world for over a decade. The three most widely publicized are from IBM Research in Zurich, Carnegie Mellon University, and HP Labs. The three designs differ largely in the types of actuators used to position the media and the methods used to record data in the medium. IBM's Millipede designs use electromagnetic motors and a novel thermomechanical recording technique [19, 33]. The device being designed at Carnegie Mellon University uses electrostatic motors for positioning and standard magnetic recording [1, 2]. The Hewlett-Packard Atomic Resolution Storage project utilizes electrostatic stepper motors, phase-change media, and electron beams to record data [12]. Despite these differences, however, each shares the same basic design shown in Figure 1, utilizing a moving media sled and a large array of read/write tips. In the Millipede chip, the read/write tips are in constant physical contact with the media, raising some questions about wear. The others maintain a constant spacing between the tips and the media.

The performance of the various actuator types seems to be similar, but their energy consumption differs somewhat. The electromagnetic actuators of the IBM Millipede chip draw more current, and hence consume more energy, as the media sled is pulled further from its rest position [23, 33]. The electrostatic actuators require higher voltages as the sled is displaced further, but require little current, so the energy consumption is lower overall. This difference could lead to interesting tradeoffs between positioning distance and energy consumption for MEMStores with electromagnetic actuators.

Since MEMStores are still being developed, systems researchers with knowledge of how they may be used can influence their design. Researchers have studied the many physical parameters of MEMStores and how those parameters should be chosen to improve performance on various workloads [7, 17].

Several researchers have studied the various roles that MEMStores may take in computer systems. Schlosser et al. [28] simulated various application workloads on MEMStores, and found that runtime decreased by 1.9–4.4×. They also found that using MEMStores as a disk cache improved I/O response time by up to 3.5×. Hong [13] evaluated using MEMStores as a metadata cache, improving system performance by 28–46% for user workloads. Rangaswami et al. [22] proposed using MEMStores in streaming media servers as buffers between the disks and DRAM. Uysal et al. [32] evaluated the use of MEMStores as components in disk arrays. In evaluating the various roles that a MEMStore may take in a system, it is useful to apply the two objective tests described in Section 1. In this way, one can determine if benefits come from the fact that the workload is particularly well-matched to a MEMStore, or just the fact that a MEMStore is faster than current disks.

Various policies for tailoring access to MEMStores have been suggested. Griffin et al. [11] studied scheduling algorithms, layout schemes, and power management policies, using a disk-like interface. Lin et al. [15] also studied several power conservation strategies for MEMStores. Several groups have suggested MEMStore-specific request scheduling algorithms [14, 36]. These groups have not applied their scheduling algorithms to disk drives to see if they are MEMStore-

[1] Some MEMStore designers have discussed building devices that operate in a resonant mode, in which the media sled moves in resonance along the recording dimension. Such a design would change this assumption and there would be access-independent motion, just like the rotation of the platters in a disk drive.

specific, and we believe it is likely that their algorithms will apply equally well to disks. Lastly, two groups have proposed using tip-subset parallelism in MEMStores to efficiently access tabular data structures [29, 35, 37]. Again, in evaluating potential policies that will be used for MEMStores, one can use the two objective tests to decide whether the policy is MEMStore-specific, or if it can be applied to both MEMStores and disk systems.

3 Standard storage abstractions

High-level storage interfaces (e.g., SCSI and ATA) hide the complexities of mechanical storage devices from the systems that use them, allowing them to be used in a standard, straightforward fashion. Different devices with the same interface can be used without the system needing to change. Also, the system does not need to manage the low-level details of the storage device. Such interfaces are common across a wide variety of storage devices, including disk drives, disk arrays, and FLASH- and RAM-based devices.

Today's storage interface abstracts a storage device as a linear array of fixed-sized *logical blocks* (usually 512 bytes). Details of the mapping of logical blocks to physical media locations are hidden. The interface allows systems to READ and WRITE ranges of blocks by providing a starting logical block number (*LBN*) and a block count.

Unwritten contract: Although no performance specifications of particular access types are given, an unwritten contract exists between host systems and storage devices supporting these standard interfaces (e.g., disks). This unwritten contract has three terms:

- Sequential accesses are best, much better than non-sequential.
- An access to a block near the previous access in *LBN* space is usually considerably more efficient than an access to a block farther away.
- Ranges of the *LBN* space are interchangeable, such that bandwidth and positioning delays are affected by relative *LBN* addresses but not absolute *LBN* addresses.

Application writers and system designers assume the terms of this contract in trying to improve performance.

3.1 Disks and standard abstractions

Disk drives are multi-dimensional machines, with data laid out in concentric circles on one or more media platters that rotate continuously. Data is divided into fixed-sized units, called sectors (usually 512 bytes to match the *LBN* size). The sector (and, thereby, *LBN*) size was originally driven by a desire to amortize both positioning costs and the overhead of the powerful error-correcting codes (ECC) required for robust magnetic data storage. The densities and speeds of today's disk drives would be impossible without these codes, and many disk technologists would like the sector size (and, thus, the *LBN* size) to grow by an order of magnitude to support more powerful codes. Each sector is addressed by a tuple, denoting its cylinder, surface, and rotational position.

*LBN*s are mapped onto the physical sectors of the disk to take advantage of the disk's characteristics. Sequential *LBN*s are mapped to sequential rotational positions within a single track, which leads to the first point of the unwritten contract. Since the disk is continuously rotating, once the heads are positioned, sequential access is very efficient. Non-sequential access incurs large re-positioning delays. Successive tracks of *LBN*s are traditionally mapped to surfaces within cylinders, and then to successive cylinders. This leads to the second point of the unwritten contract: that distant *LBN*s map to distant cylinders, leading to longer seek times.

The linear abstraction works for disk drives, despite their clear three-dimensional nature, because two of the dimensions are largely uncorrelated with *LBN* addressing. Access time is the sum of the time to position the read/write heads to the destination cylinder (seek time), the time for the platters to reach the appropriate rotational offset (rotational latency), and the time to transfer the data to or from the media (transfer time). Seek time and rotational latency usually dominate transfer time. The heads are positioned as a unit by the seek arm, meaning that it usually doesn't matter which surface is being addressed. Unless the abstraction is stripped away, rotational latency is nearly impossible to predict because the platters are continuously rotating and so the starting position is essentially random. The only dimension that remains is that across cylinders, which determines the seek time.

Seek time is almost entirely dependent on the distance traversed, not on the absolute starting and ending points of the seek. This leads to the third point of the unwritten contract. Ten years ago, all disk tracks had the same number of sectors, meaning that streaming bandwidths (and, thus, transfer times) were uniform across the *LBN* space. Today's zoned disk geometries, however, violate the third term since streaming bandwidth varies between zones.

3.2 Holes in the abstraction boundary

Over its fifteen year lifespan, shortcomings of the interface and unwritten contract have been identified. Perhaps the most obvious violation has been the emergence of multi-zone disks, in which the streaming bandwidth varies by over 50% from one part of the disk to another. Some application writers exploit this difference by explicitly using the low-numbered LBNs, which are usually mapped to the outer tracks. Over time, this may become a fourth term in the unwritten contract.

Some have argued [4, 27] that the storage interface should be extended for disk arrays. Disk arrays contain several disks which are combined to form one or more logical volumes. Each volume can span multiple disks, and each disk may contain parts of multiple volumes. Hiding the boundaries, parallelism, and redundancy schemes prevents applications from exploiting them. Others have argued [8] that, even for disks, the current interface is not sufficient. For example, knowing track boundaries can improve performance for some applications [26].

The interface persists, however, because it greatly simplifies most aspects of incorporating storage components into systems. Before this interface became standard, systems used a variety of per-device interfaces. These were replaced because they complicated systems greatly and made components less interchangeable. This suggests that the bar should be quite high for a new storage component to induce the introduction of a new interface or abstraction.

It is worth noting that some systems usefully throw out abstraction boundaries entirely, and this is as true in storage as elsewhere. In particular, storage researchers have built tools [25, 30] for extracting detailed characteristics of storage devices. Such characteristics have been used for many ends: writing blocks near the disk head [39], reading a replica near the disk head [38], inserting background requests into foreground rotational latencies [16], and achieving semi-preemptible disk I/O [5]. Given their success, adding support for such ends into component implementations or even extending interfaces may be appropriate. But, they do not represent a case for removing the abstractions in general.

4 MEMStores and standard abstractions

Using a standard storage abstraction for MEMStores has the advantage of making them immediately usable by existing systems. Interoperability is important for getting MEMStores into the marketplace, but if the abstractions that are used make performance suffer, then there is reason to consider something different.

This section explains how the details of MEMStore operation make them naturally conform to the storage abstraction used for today's disks. Also, the unwritten contract that applications expect will remain largely intact.

4.1 Access method

The standard storage interface allows accesses (READs and WRITEs) to ranges of sizeable fixed-sized blocks. The question we ask first is whether such an access method is appropriate for a MEMStore.

Is a 512 byte block appropriate, or should the abstraction use something else? It is true that MEMStores can dynamically choose subsets of read/write tips to engage when accessing data, and that these subsets can, in theory, be arbitrarily-sized. However, enough data must be read or written for error-correcting codes (ECC) to be effective. The use of ECC enables high storage density by relaxing error-rate constraints. Since the density of a MEMStore is expected to equal or exceed that of disk drives, the ECC protections needed will be comparable. Therefore, block sizes of the same order of magnitude as disks should be expected. Also, any block's size must be fixed, since it must be read or written in its entirety, along with the associated ECC. Accessing less than a full block, e.g., to save energy [15], would not be possible. The flexibility of being able to engage arbitrary sets of read/write tips can still be used to selectively choose sets of these fixed-sized blocks.

Large block sizes are also motivated by embedded servo mechanisms, coding for signal processing, and the relatively low per-tip data rate of around 1 Mbit/s. The latter means that data will have to be spread across multiple parallel-operating read/write tips to achieve an aggregate bandwidth that is on-par with that of disk drives. Spreading data across multiple read/write tips also introduces physical redundancy that will allow for better tolerance of tip failures. MEMStores will use embedded servo [31], requiring that several bits containing position information be read before any access in order to ensure that the media sled is positioned correctly. Magnetic recording techniques commonly use transitions between bits rather than the bits themselves to represent data, meaning that a sequence of bits must be accessed together. Further, signal encodings use multi-bit codewords that map a sequence of bits to values with interpretable patterns (e.g., not all ones or all zeros). The result is that, in order to access any data after a seek, some amount of data (10 bits in our model) must be read for servo information, and then bits must be accessed sequentially with some coding overhead (10 bits per byte in our model). Given these overheads, a large block size should be used to amortize the costs. This block will be spread across multiple read/write tips to improve data rates and fault tolerance.

Using current storage interfaces, applications can only request ranges of sequential blocks. Such access is reasonable for MEMStores, since blocks are laid out sequentially, and their abstraction should support the same style of access. There may be utility in extending the abstraction to allow applications to request batches of non-contiguous *LBN*s that can be accessed by parallel read/write tips. An extension like this is discussed in Section 6.

4.2 Unwritten contract

Assuming that MEMStore access uses the standard storage interface, the next step is to see if the unwritten contract for disks still holds. If it does, then MEMStores can be used effectively by systems simply as fast disks.

The first term of the unwritten contract is that sequential access is more efficient than random access. This will continue to be the case for MEMStores because data still must be accessed in a linear fashion. The signal processing techniques that are commonly used in magnetic storage are based on transitions between bits, rather than the state of the bits in isolation. Moreover, they only work properly when state transitions come frequently enough to ensure clock synchronization so they encode multi-bit data sequences into alternate codewords. These characteristics dictate that the bits must be accessed sequentially. Designs based on recording techniques other than magnetic will, most likely, encode data similarly. Once the media sled is in motion, it is most efficient for it to stay in motion, so the most efficient thing to access is the next sequential data, just as it is for disks.

The second term of the unwritten contract is that the difference between two *LBN* numbers maps well to the physical distance between them. This is dependent on how *LBN*s are mapped to the physical media, and this mapping can easily be constructed in a MEMStore to make the second point of the unwritten contract be true. A MEMStore is a multi-dimensional machine, just like a disk, but the dimensions are correlated differently. Each media position is identified by a tuple of the X position, the Y position, and the set of read/write tips that are enabled, much like the cylinder/head/rotational position tuples in disks. There are thousands of read/write tips in a MEMStore, and each one accesses its own small portion of the media. Just as the heads in a disk drive are positioned as a unit to the same cylinder, the read/write tips in a MEMStore are always positioned to the same offset within their own portion of the media. The choice of which read/write tips to activate has no correlation with access time, since any set can be chosen for the same cost once the media is positioned.

As with disks, seek time for a MEMStore is a function of seek distance. Since the actuators on each axis are independent, the overall seek time is the maximum of the individual seek times in each dimension, X and Y. But, the X seek time almost always dominates the Y seek time because extra settle time must be included for X seeks, but not for Y seeks. The reason for this is that post-seek oscillations in the X dimension lead to off-track interference, while the same oscillations in the Y dimension affect only the bit rate of the data transfer. Since the overall seek time is the maximum of the two individual seek times, and the X seek time is almost always greater than the Y seek time, the overall seek distance is (almost) uncorrelated with the Y position, as seen in Figure 2. In the end, despite the fact that a MEMStore has multiple dimensions over which to position, the overall access time is (almost) only correlated with just a single dimension, which makes a linear abstraction sufficient.

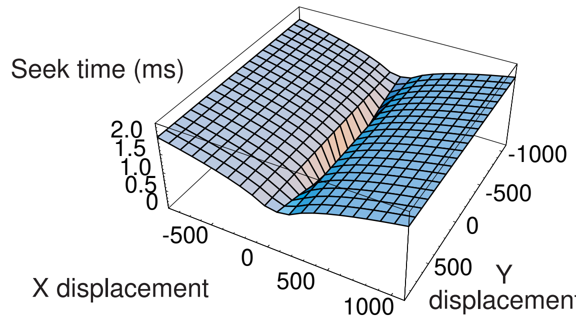


Figure 2: **MEMStore seek curve.** The seek time of a MEMStore is largely uncorrelated with the displacement in the Y dimension due to a large settling time required for the X dimension seek that is not required for the Y dimension seek [7, 10]. The overall seek time is the maximum of the two independent seek times.

The last term of the unwritten contract states that the *LBN* space is uniform, and that access time does not vary across the range of the *LBN*s. The springs that attach the media sled to the chip do affect seek times by applying a greater restoring force when they are displaced further. However, the effect is minimal, with seek times varying by at most 10–15%, meaning that overall access times at the application level would vary by far less. Also, MEMStores do not need zoned recording. It is safe to say that the last point of the unwritten contract still holds: ranges of the *LBN* space of a MEMStore are interchangeable.

4.3 Possible exceptions

This section has explained how MEMStores fit the same assumptions that make storage abstractions work for disks. There are a few aspects of MEMStores, discussed in Section 6, that set them apart from disks for specific access patterns. These exceptions can be exploited with little or no change to the existing storage interface. Of course, the discussion above is based on current MEMStore designs. Section 7 discusses the most significant design assumptions and what removing them would change.

5 Experiments

There are two objective tests that one should consider when evaluating whether a potential role or policy for MEMStores requires a new abstraction. The *specificity test* asks whether the role or policy is truly MEMStore-specific. The test here is to evaluate the role or policy

Capacity	3.46 GB
Average access time	0.88 ms
Streaming bandwidth	76 MB/s

Table 1: **G2 MEMStore parameters.** These parameters are for the G2 MEMStore design from [28].

Capacity	41.6 GB
Rotation speed	55,000 RPM
One-cylinder seek time	0.1 ms
Full-stroke seek time	2.0 ms
Head switch time	0.01 ms
Number of cylinders	39511
Number of surfaces	2
Average access time	0.88 ms
Streaming bandwidth	100 MB/s

Table 2: **Überdisk parameters.** The Überdisk is a hypothetical future disk drive. Its parameters are scaled from current disks, and are meant to represent those of a disk that matches the performance of a MEMStore. The average response time is for a random workload which exercised only the first 3.46 GB of the disk in order to match the capacity of the G2 MEMStore.

for both a MEMStore and a (hypothetical) disk drive of equivalent performance. If the benefit is the same, then the role or policy (however effective) is not truly MEMStore-specific. Given that the role or policy passes the specificity test, the *merit test* determines whether the difference makes a significant-enough impact in performance (or whatever metric) to justify customizing the system. This section examines a potential role and a potential MEMStore-specific policy, under the scrutiny of these two tests.

5.1 G2 MEMStore

The MEMStore that we use for evaluation is the G2 model from [28]. Its basic parameters are given in Table 1. We use DiskSim, a freely-available storage system simulator, to simulate the MEMStore [6].

5.2 Überdisk: A hypothetical fast disk

For comparison, we use a hypothetical disk design, which we call the Überdisk, that approximates the performance of a G2 MEMStore. Its parameters are based on extrapolating from today's disk characteristics, and are given in Table 2. The Überdisk is also modeled using DiskSim. In order to do a capacity-to-capacity comparison, we use only the first 3.46 GB of the Überdisk to match the capacity of the G2 MEMStore. The two devices have equivalent performance under a random workload of 4 KB requests that are uniformly distributed across the capacity (3.46 GB) and arrive one at a time. Since our model of MEMStores does not include a cache, we disabled the cache of the Überdisk.

We based the seek curve on the formula from [24], choosing specific values for the one-cylinder and full-stroke seeks. Head switch and one-cylinder seek times are expected to decrease in the future due to microactuators integrated into disk heads, leading to shorter settle times. With increasing track densities, the number of platters in disk drives is decreasing steadily, so the Überdisk has only two surfaces. The zoning geometry is based on simple extrapolation of current linear densities.

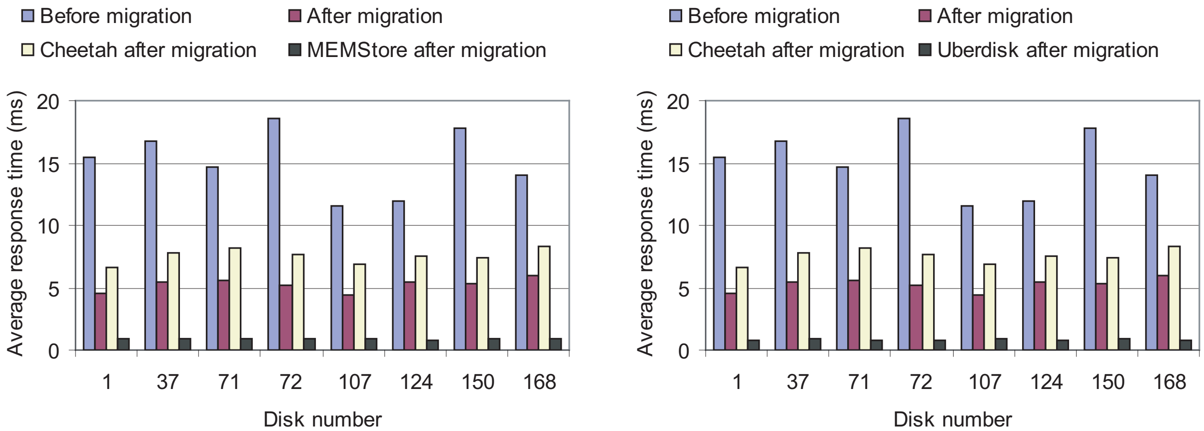
An Überdisk does not necessarily represent a realistic disk; for example, a rotation rate of 55,000 RPM (approximately twice the speed of a dental drill) may never be attainable in a reasonably-priced disk drive. However, this rate was necessary to achieve an average rotational latency that is small enough to match the average access time of the MEMStore. The Überdisk is meant to represent the parameters that would be required of a disk in order to match the performance of a MEMStore. If the performance of a workload running on a MEMStore is the same as it running on an Überdisk, then we can say that any performance increase is due only to the intrinsic speed of the device, and not due to the fact that it is a MEMStore or an Überdisk. If the performance of the workload differs on the two devices, then it must be especially well-matched to the characteristics of one device or the other.

5.3 Role: MEMStores in disk arrays

One of the roles that has been suggested for MEMStores in systems is that of augmenting or replacing some or all of the disks in a disk array to increase performance [28, 32]. However, the lower capacity and potentially higher cost of MEMStores suggest that it would be impractical to simply replace all of the disks. Therefore, they represent a new tier in the traditional storage hierarchy, and it will be important to choose which data in the array to place on the MEMStores and which to store on the disks. Uysal et al. evaluate several methods for partitioning data between the disks and the MEMStores in a disk array [32]. We describe a similar experiment below, in which a subset of the data stored on the back-end disks in a disk array is moved to a MEMStore.

We can expect some increase in performance from doing this, as Uysal et al. report. However, our question here is whether the benefits are from a MEMStore-specific attribute, or just from the fact that MEMStores are faster than the disks used in the disk array. To answer this question, we apply the specificity test by comparing the performance of a disk array back-end workload on three storage configurations. The first configuration uses just the disks that were originally in the disk array. The second configuration augments the overloaded disks with a MEMStore. The third does the same with an Überdisk.

The workload is a disk trace gathered from the disks



(a) MEMStore

(b) Überdisk

Figure 3: **Using MEMStores in a disk array.** These graphs show the result of augmenting overloaded disks in a disk array with faster storage components: a MEMStore (a) or an Überdisk (b). In both cases, the busiest logical volume on the original disk (a 73 GB Seagate Cheetah) is moved to the faster device. Requests to the busiest logical volume are serviced by the faster device, and the traffic to the Cheetah is reduced. The results for both experiments are nearly identical, leading to the conclusion that the MEMStore and the Überdisk are interchangeable in this role (e.g., it is not MEMStore-specific.)

in the back-end of an EMC Symmetrix disk array during the summer of 2001. The disk array contained 282 Seagate Cheetah 73 GB disk drives, model number ST173404. From those, we have chosen the eight busiest (disks 1, 37, 71, 72, 107, 124, 150, and 168), which have an average request arrival rate of over 69 requests per second for the duration of the trace, which was 12.5 minutes. Each disk is divided into 7 logical volumes, each of which is approximately 10 GB in size. For each "augmented" disk, we move the busiest logical volume to a faster device, either a MEMStore or an Überdisk. The benefit should be twofold: first, response times for the busiest logical volume will be improved, and second, traffic to the original disk will be reduced. Requests to the busiest logical volume are serviced by the faster device (either a MEMStore or an Überdisk), and all other requests are serviced by the original Cheetah disk.

Figure 3(a) shows the result of the experiment with the MEMStore. For each disk, the first bar shows the average response time of the trace running just on the Cheetah, which is 15.1 ms across all of the disks. The second bar shows the average response time of the same requests after the busiest logical volume has been moved to the MEMStore. Across all disks, the average is now 5.24 ms. The third and fourth bars show, respectively, the average response time of the Cheetah with the reduced traffic after augmentation, and the average response time of the busiest logical volume, which is now stored on the MEMStore. We indeed see the benefits anticipated — the average response time of requests to the busiest logical volume have been reduced to 0.86 ms, and the reduction of load on the Cheetah disk has resulted in a lower average response time of 7.56 ms.

Figure 3(b) shows the same experiment, but with the busy logical volume moved to an Überdisk rather than a MEMStore. The results are almost exactly the same, with the response time of the busiest logical volume migrated to the Überdisk being around 0.84 ms, and the overall response time reduced from 15.1 ms to 5.21 ms.

The fact that the MEMStore and the Überdisk provide the same benefit in this role means that it fails the specificity test. In this role, a MEMStore really can be considered to be just a fast disk. The workload is not specifically matched to the use of a MEMStore or an Überdisk, but can clearly be improved with the use of any faster device, regardless of its technology.

Although it is imperceptible in Figure 3, the Überdisk gives slightly better performance than the MEMStore because it benefits more from workload locality due to the profile of its seek curve. The settling time in the MEMStore model makes any seek expensive, with a gradual increase up to the full-stroke seek. The settling time of the Überdisk is somewhat less, leading to less expensive initial seek and a steeper slope in the seek curve up to the full-stroke seek. The random workload we used to compare devices has no locality, but the disk array trace does.

To explore this further, we re-ran the experiment with two other disk models, which we call Simpledisk-constant and Simpledisk-linear. Simpledisk-constant responds to requests in a fixed amount of time, equal to that of the response time of the G2 MEMStore under the random workload: 0.88 ms. The response time of Simpledisk-linear is a linear function of the distance from the last request in *LBN* space. The endpoints of the function are equal to the single-cylinder and full-stroke seek times of the Überdisk, which are 0.1 ms and 2.0 ms, respectively. Simpledisk-constant should not benefit from locality, and Simpledisk-linear should benefit from locality even more than either the MEMStore or the Überdisk. Augmenting the disk array with these devices gives response times to the busiest logical volume of 0.92 ms and 0.52 ms, respectively. As expected, Simpledisk-constant does not benefit from workload locality and Simpledisk-linear benefits more than a real disk.

Uysal proposed several other MEMStore/disk combinations in [32], including replacing all of the disks with MEMStores, replacing half of the mirrors in a mirrored configuration, and using the MEMStore as a replacement of the NVRAM cache. In all of these cases, and in most of the other roles outlined in Section 2.1, the MEMStore is used simply as a block store, with no tailoring of access to MEMStore-specific attributes. We believe that if the specificity test were applied, and an Überdisk was used in each of these roles, the same performance improvement would result. Thus, the results of prior research apply more generally to faster mechanical devices.

5.4 Policy: distance-based scheduler

Mechanical and structural differences between MEMStores and disks suggest that request scheduling policies that are tailored to MEMStores may provide better performance than ones that were designed for disks. Upon close examination, however, the physical and mechanical motions that dictate how a scheduler may perform on a given device continue to apply to MEMStores as they apply to disks. This may be surprising at first glance, since the devices are so different, but after examining the fundamental assumptions that make schedulers work for disks, it is clear that those assumptions are also true for MEMStores.

To illustrate, we give results for a MEMStore-specific scheduling algorithm called *shortest-distance-first*, or SDF. Given a queue of requests, the algorithm compares the Euclidean distance between the media sled's current position and the offset of each request and schedules the request that is closest. The goal is to exploit a clear difference between MEMStores and disks: that MEMStores position over two dimensions rather than only one. When considering the specificity test, it is not surprising that this qualifies as a MEMStore-specific policy. Disk drives do, in fact, position over multiple dimensions, but predicting the positioning time based on any dimension other than the cylinder distance is very difficult outside of disk firmware. SDF scheduling for MEMStores is easier and could be done outside of the device firmware, since it is only based on the logical-to-physical mapping of the device's sectors and any defect management that is used, assuming that the proper geometry information is exposed through the MEMStore's interface.

The experiment uses a random workload of 250,000 requests uniformly distributed across the capacity of the MEMStore. Each request had a size of 8 KB. This workload is the same as that used in [34] to compare request scheduling algorithms. The experiment tests the effectiveness of the various algorithms by increasing the arrival rate of requests until saturation — the point at which response time increases dramatically because the device can no longer service requests fast enough and the queue grows without bound.

The algorithms compared were first-come-first-served (FCFS), cyclic LOOK (CLOOK), shortest-seek-time-first (SSTF), shortest-positioning-time-first (SPTF), and shortest-distance-first (SDF). The first three are standard disk request schedulers for use in host operating systems. FCFS is the baseline for comparison, and is expected to have the worst performance. CLOOK and SSTF base their scheduling decisions purely on the *LBN* number of the requests, utilizing the unwritten assumption that *LBN* numbers roughly correspond to physical positions [34]. SPTF uses a model of the storage device to predict service times for each request, and can be expected to give the best performance. The use of the model by SPTF breaks the abstraction boundaries because it provides the application with complete details of the device parameters and operation. The SDF scheduler requires the capability to map *LBN* numbers to physical locations, which breaks the abstraction, but does not require detailed modeling, making it practical to implement in a host OS.

Figure 4 shows the results. As expected, FCFS and SPTF perform the worst and the best, respectively. CLOOK and SSTF don't perform as well as SPTF because they use only the *LBN* numbers to make scheduling decisions. The SDF scheduler performs slightly worse than CLOOK and SSTF. The reason is that positioning time is not as well correlated with two-dimensional position, as described in Section 4.2. As such, considering the two-dimensional seek distance does not provide any more utility than just considering the one-dimensional seek distance, as CLOOK and SSTF effectively do. Thus, the suggested policy fails the merit test: the same or greater benefit can be had with existing schedulers that don't need MEMStore-specific knowledge. This is based, of course, on the assumption that settling time is a significant component of position-

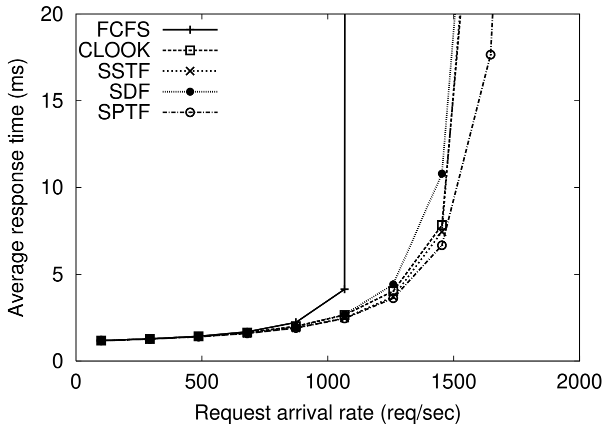


Figure 4: **Performance of shortest-distance-first scheduler.** A MEMStore-specific scheduler that accounts for two-dimensional position gives no benefit over simple schedulers that use a linear abstraction (CLOOK and SSTF). This is because seek time in a MEMStore is correlated most strongly with only distance in the X dimension.

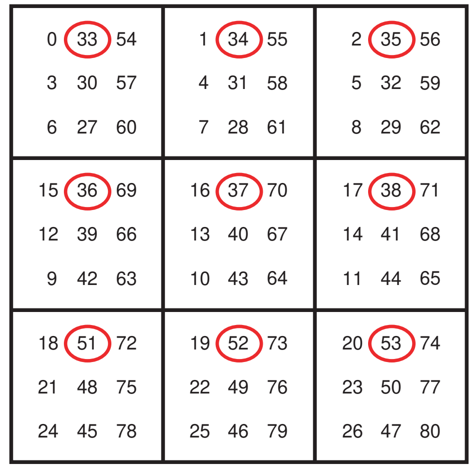


Figure 5: **Data layout with a set of equivalent *LBN*s highlighted.** The *LBN*s marked with ovals are at the same location within each square and, thus, are "equivalent". That is, they can potentially be accessed in parallel.

ing time. Section 7 discusses the effect of removing this assumption.

The fundamental reason that scheduling algorithms developed for disks work well for MEMStores are that seek time is strongly dependent on seek distance, but only the seek distance in a single dimension. The seek time is only correlated to a single dimension, which is exposed by the linear abstraction. The same is true for disks when one cannot predict the rotational latencies, in which only the distance that the heads must move across cylinders is relevant. Hence, a linear logical abstraction is as justified for MEMStores as it is for disks.

Of course, there may be yet-unknown policies that exploit features that are specific to MEMStores, and we expect research to continue in this area. When considering potential policies for MEMStores, it is important to keep the two objective tests in mind. In particular, these tests can expose a lack of need for a new policy or, better yet, the fact that the policy is equally applicable to disks and other mechanical devices.

6 MEMStore-specific features

This section describes three MEMStore-specific features that clearly set them apart from disks, offering significant performance improvements for well-matched workloads. Exploiting such features may require a new abstraction or, at least, changes in the unwritten contract between systems and storage.

6.1 Tip-subset parallelism

MEMStores have an interesting access parallelism feature that does not exist in modern disk drives. Specifically, subsets of a MEMStore's thousands of read/write tips can be used in parallel, and the particular subset can be dynamically chosen. This section briefly describes how such access parallelism can be exposed to system software, with minimal extensions to the storage interface, and utilized cleanly by applications. Interestingly, our recent work [27] has shown the value of the same interface extensions for disk arrays, suggesting that this is a generally useful storage interface change.

Figure 5 shows a simple MEMStore with nine read/write tips and nine sectors per tip. Each read/write tip addresses its own section of the media, denoted by the nine squares in the figure. Sectors that are at the same physical offset within each square, such as those indicated with ovals, are addressed simultaneously by the tip array. We call these sectors *equivalent*, because they can be accessed in parallel. But, in many designs, not all of the tips can be actively transferring data at the same time due to power consumption or component sharing constraints. Using a simple API, an application or OS module could identify sets of sectors that are equivalent, and then choose subsets to access together. Since the *LBN*s which will be accessed together will not fall into a contiguous range, the system will need to be able to request batches of non-contiguous *LBN*s, rather than ranges.

6.1.1 Efficient 2D data structure access

The standard interface forces applications to map their data into a linear address space. For most applications, this is fine. However, applications that use two-dimensional data structures, such as non-sparse matrices or relational database tables, are forced to serialize their storage in this linear address space, making efficient access possible along only one dimension of the data structure. For example, a database can choose to store its table in column-major order, making column accesses sequential and efficient [3]. Once this choice is made, however, accessing the table in row-major order is very expensive, requiring a full scan of the table to read a single row. One option for making operations in both dimensions efficient is to create two copies of a table; one copy is optimized for column-major access and the other is optimized for row-major access [21]. This scheme, however, doubles the capacity needed for the database and requires that updates propagate to both copies.

With proper allocation of data to a MEMStore *LBN* space, parallel read/write tips can be used to access a table in either row- or column-major order at full speed [29, 35]. The table is arranged such that the same attributes of successive records are stored in sequential *LBN*s. Then, the other attributes of those records are stored in *LBN*s that are equivalent to the original *LBN*s, as in Figure 5. This layout preserves the two-dimensionality of the original table on the physical media of the MEMStore. Then, when accessing the data, the media sled is positioned and the appropriate read/write tips are activated to read data either in row- or column-major order.

To quantify the advantages of such a MEMStore-specific scan operator, we compare the times required for different table accesses. We contrast their respective performance under two different layouts on a single G2 MEMStore device. The first layout, called *normal*, is the traditional row-major access optimized page layout used in almost all database systems [20].

The second layout, called *MEMStore*, uses the MEMStore-specific layout and access described above. The sample database table consists of 4 attributes a_1, a_2, a_3, and a_4 sized at 8, 32, 15, and 16 bytes respectively. The *normal* layout consists of 8 KB pages that hold 115 records. The table size is 10,000,000 records for a total of 694 MB of data.

Figure 6 compares the time of a full table scan for all attributes with four scans of the individual attributes. The total runtime of four single-attribute scans in the *MEMStore* case takes the same amount of time as the full table scan. In contrast, with the *normal* layout, the four successive scans take four times as long as the full table scan. Most importantly, a scan of a single attribute in the *MEMStore* case takes only the amount of time needed for a full-speed scan of the corresponding amount of data, since all of the available read/write tips read records of the one attribute. This result represents a compelling performance improvement over current database systems. This policy for MEMStores passes both the specificity test and the merit test.

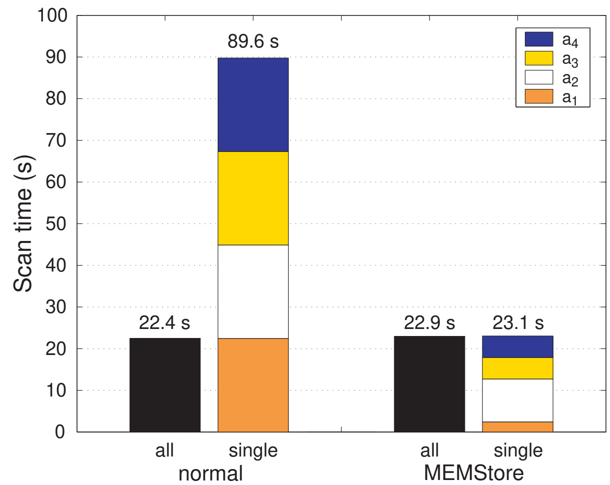


Figure 6: **Database table scan with different number of attributes.** This graph shows the runtime of scanning 10,000,000 records using a G2 MEMStore. For each of the two layouts, the left bar, labeled "all," shows the time to scan the entire table with four attributes. The right bar, labeled "single," is composed of four separate scans of each successive attribute, simulating the situation where multiple queries access different attributes. Since the *MEMStore* layout takes advantage of MEMStore's tip-subset parallelism, each attribute scan runtime is proportional to the amount of data occupied by that attribute. The *normal* layout, on the other hand, must read the entire table to fetch any one attribute.

6.2 Quick turnarounds

Another aspect of MEMStores that differs from disk drives is their ability to quickly access an *LBN* repeatedly. In a disk, repeated reads to an *LBN* may be serviced from the disk's buffer, but repeated synchronous writes or read/modify/write sequences will incur a full rotation, 4-8 ms on most disks, for each access. A MEMStore, however, can simply change the direction that the media sled is moving, which is predicted to take less than a tenth of a millisecond [11]. Read/modify/write sequences are prevalent in parity-based redundancy schemes, such as RAID-5, in which the old data and parity must each be read and then updated for each single block write. Repeated synchronous writes are common in database log files, where each commit entry must propagate to disk. Such operations are much more expensive in a disk drive.

6.3 Device scan time

Although the volumetric density of MEMStores is on-par with that of disk drives, the per-device capacity is much less. For example, imagine two 100 GB "storage bricks," one using disk storage and the other using MEMStores. Given that the volumetric densities are equal, the two bricks would consume about the same amount of physical volume. But, the MEMStore brick would require at least ten devices, while the disk-based brick could consist of just one device. This means that the MEMStore-based brick would have more independent actuators for accessing the data, leading to several interesting facts. First, the MEMStore-based brick could handle more concurrency, just as in a disk array. Second, MEMStores in the brick that are idle could be turned off while others in the brick are still servicing requests, reducing energy consumption. Third, the overall time to scan the entire brick could be reduced, since some (or all) of the devices could access data in parallel. This assumes that the bus connecting the brick to the system is not a bottleneck, or that the data being scanned is consumed within the brick itself. The lower device scan time is particularly interesting because disk storage is becoming less accessible as device capacities grow more quickly than access speeds [9].

Simply comparing the time to scan a device in its entirety, a MEMStore could scan its entire capacity in less time than a single disk drive. At 100 MB/s, a 10 GB MEMStore is scanned in only 100 s, while a 72 GB disk drive takes 720 s. As a result, strategies that require entire-device scans, such as scrubbing or virus scanning, become much more feasible.

7 Major assumptions

Unfortunately, MEMStores do not exist yet, so there are no prototypes that we can experiment with, and they are not expected to exist for several more years. As such, we must base all experiments on simulation and modeling. We have based our models on detailed discussions with researchers who are designing and building MEMStores, and on an extensive study of the literature. The work and the conclusions in this paper are based on this modeling effort, and is subject to its assumptions about the devices. This section outlines two of the major assumptions of the designers and how our conclusions would change given different assumptions.

Some of our conclusions are based on the assumption that post-seek settling time will affect one seek dimension more than the other. This effectively uncorrelates seek time with one of the two dimensions, as described in Section 4.2. The assumption is based on the observation that different mechanisms determine the settling time in each of the two axes, X and Y. Settling time is needed to damp oscillations enough for the read/write tips to reliably access data. In all published MEMStore designs, data is laid out linearly along the Y-axis, meaning that oscillations in Y will appear to the channel as minor variations in the data rate. Contrast this with oscillations in the X-axis, which pull the read/write tips off-track. Because one axis is more sensitive to oscillation than the other, its positioning delays will dominate the other's, unless the oscillations can be damped in near-zero time.

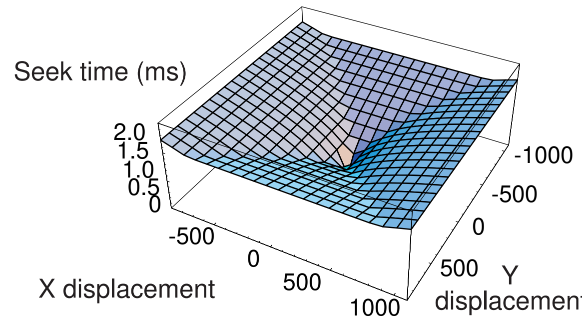


Figure 7: **MEMStore seek curve without settling time.** Without the settling time, the seek curve of a MEMStore is strongly correlated with displacement in both dimensions [10].

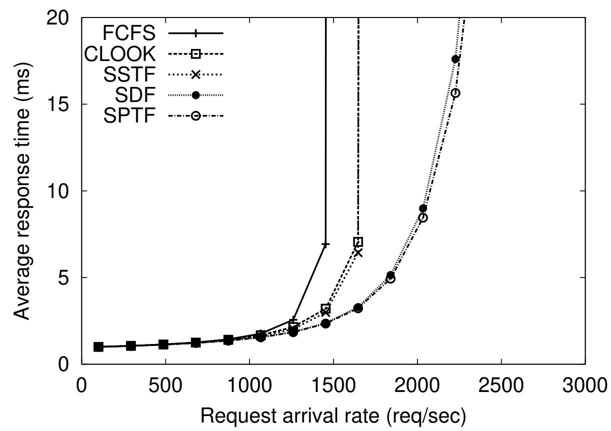


Figure 8: **Performance of shortest-distance-first scheduler without settle time.** If post-seek settle time is eliminated, then the seek time of a MEMStore becomes strongly correlated with both the X and Y positions. In this case, a scheduler that takes into account both dimensions provides much better performance than those that only consider a single dimension (CLOOK and SSTF).

If this assumption no longer held, and oscillations affected each axis equally, then MEMStore-specific policies that take into account the resulting two-dimensionality of the seek profile, as illustrated in Figure 7, would become more valuable. Now, for example, two-dimensional distance would be a much better predictor of overall positioning time. Figure 8 shows the result of repeating the experiment from Section 5.4, but with the post-seek settle time set to zero. In this case, the performance of the SDF scheduler very closely tracks shortest-positioning-time-first, SPTF, the scheduler based on full knowledge of positioning time. Further, the difference between SDF and the two algorithms based on single-dimension position (CLOOK and SSTF) is now very large. CLOOK and SSTF have worse performance because they ignore the second dimension that is now correlated strongly with positioning time.

Another closely-related assumption is that data in a MEMStore is accessed sequentially in a single dimension. One could imagine a MEMStore in which data is accessed one point at a time. As a simple example, imagine that the media sled would move to a single position and then engage 8×512 read/write probes (plus ECC tips) in parallel to read one 512 byte sector from the media at once. From that point, the media sled could then re-position in either the X or Y dimension and read another 512 byte sector. In fact, the device could stream sequentially along either dimension. Current designs envision using embedded servo to keep the read/write tips on track, just as in disks [31]. Both servo and codewords would have to be encoded along both dimensions somehow to allow streaming along either. The ability to read sequentially along either dimension at an equal rate could improve the performance of applications using two-dimensional data structures, as described in Section 6.1.1. Rather than using tip subset parallelism, data tables could be stored directly in their original format on the MEMStore, and then accessed in either direction efficiently. Note, however, that the added complexity of the coding and access mechanisms would be substantial, making this unlikely to occur.

8 Summary

One question that should be asked when considering how to use MEMStores in computer systems is whether they have unique characteristics that should be exploited by systems, or if they can be viewed as small, low-power, fast disk drives. This paper examines this question by establishing two objective tests that can be used to identify the existence and importance of relevant MEMStore-specific features. If an application utilizes a MEMStore-specific feature, then there may be reason to use something other than existing disk-based abstractions. After studying the fundamental reasons that the existing abstraction works for disks, we conclude that the same reasons hold true for MEMStores, and that a disk-like view is justified. Several case studies of potential roles that MEMStores may take in systems and policies for their use support this conclusion.

Acknowledgements

We thank the members of the MEMS Laboratory at CMU for helping us understand the technology of MEMStores. We would like to thank the anonymous reviewers and our shepherd, David Patterson, for helping to improve this paper. We thank the members and companies of the PDL Consortium (EMC, Hewlett-Packard, Hitachi, IBM, Intel, LSI Logic, Microsoft, Network Appliance, Oracle, Panasas, Seagate, Sun, and Veritas) for their interest, insights, feedback, and support. This work is funded in part by NSF grant CCR-0113660.

References

[1] L. R. Carley, J. A. Bain, G. K. Fedder, D. W. Greve, D. F. Guillou, M. S. C. 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, 1 May 2000.

[2] Center for Highly Integrated Information Processing and Storage Systems, Carnegie Mellon University. http://-www.ece.cmu.edu/research/chips/.

[3] G. P. Copeland and S. Khoshafian. A decomposition storage model. *ACM SIGMOD International Conference on Management of Data* (Austin, TX, 28–31 May 1985), pages 268–279. ACM Press, 1985.

[4] T. E. Denehy, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Bridging the information gap in storage protocol stacks. *Summer USENIX Technical Conference* (Monterey, CA, 10–15 June 2002), pages 177–190, 2002.

[5] Z. Dimitrijević, R. Rangaswami, and E. Chang. Design and implementation of semi-preemptible IO. *Conference on File and Storage Technologies* (San Francisco, CA, 31 March–02 April 2003), pages 145–158. USENIX Association, 2003.

[6] The DiskSim Simulation Environment (Version 3.0). http://www.pdl.cmu.edu/DiskSim/index.html.

[7] I. Dramaliev and T. M. Madhyastha. Optimizing probe-based storage. *Conference on File and Storage Technologies* (San Francisco, CA, 31–02 April 2003), pages 103–114. USENIX Association, 2003.

[8] G. R. Ganger. *Blurring the line between OSs and storage devices*. Technical report CMU–CS–01–166. Carnegie Mellon University, December 2001.

[9] J. Gray. A conversation with Jim Gray. *ACM Queue*, **1**(4). ACM, June 2003.

[10] J. L. Griffin, S. W. Schlosser, G. R. Ganger, and D. F. Nagle. Modeling and performance of MEMS-based storage devices. *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (Santa Clara, CA, 17–21 June 2000). Published as *Performance Evaluation Review*, **28**(1):56–65, 2000.

[11] J. L. Griffin, S. W. Schlosser, G. R. Ganger, and D. F. Nagle. Operating system management of MEMS-based storage devices. *Symposium on Operating Systems Design and Implementation* (San Diego, CA, 23–25 October 2000), pages 227–242. USENIX Association, 2000.

[12] Hewlett-Packard Laboratories Atomic Resolution Storage. http://www.hpl.hp.com/research/storage.html.

[13] B. Hong. Exploring the usage of MEMS-based storage as metadata storage and disk cache in storage hierarchy. http://www.cse.ucsc.edu/˜hongbo/publications/-mems-metadata.pdf.

[14] B. Hong, S. A. Brandt, D. D. E. Long, E. L. Miller, K. A. Glocer, and Z. N. J. Peterson. Zone-based shortest positioning time first scheduling for MEMS-based storage devices. *International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems* (Orlando, FL, 12–15 October 2003), 2003.

[15] Y. Lin, S. A. Brandt, D. D. E. Long, and E. L. Miller. Power conservation strategies for MEMS-based storage devices. *International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems* (Fort Worth, TX, October 2002), 2002.

[16] C. R. Lumb, J. Schindler, and G. R. Ganger. Freeblock scheduling outside of disk firmware. *Conference on File and Storage Technologies* (Monterey, CA, 28–30 January 2002), pages 275–288. USENIX Association, 2002.

[17] T. M. Madhyastha and K. P. Yang. Physical modeling of probe-based storage. *IEEE Symposium on Mass Storage Systems* (April 2001). IEEE, 2001.

[18] N. Maluf. *An introduction to microelectromechanical systems engineering*. Artech House, 2000.

[19] The Millipede: A future AFM-based data storage system. http://www.zurich.ibm.com/st/storage/millipede.html.

[20] R. Ramakrishnan and J. Gehrke. *Database management systems*, number 3rd edition. McGraw-Hill, 2003.

[21] R. Ramamurthy, D. J. DeWitt, and Q. Su. A case for fractured mirrors. *International Conference on Very Large Databases* (Hong Kong, China, 20–23 August 2002), pages 430–441. Morgan Kaufmann Publishers, Inc., 2002.

[22] R. Rangaswami, Z. Dimitrijević, E. Chang, and K. E. Schauser. MEMS-based disk buffer for streaming media servers. *International Conference on Data Engineering* (Bangalore, India, 05–08 March 2003), 2003.

[23] H. Rothuizen, U. Drechsler, G. Genolet, W. Häberle, M. Lutwyche, R. Stutz, R. Widmer, and P. Vettiger. Fabrication of a micromachined magnetic X/Y/Z scanner for parallel scanning probe applications. *Microelectronic Engineering*, **53**:509–512, June 2000.

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

[25] J. Schindler and G. R. Ganger. *Automated disk drive characterization*. Technical report CMU–CS–99–176. Carnegie-Mellon University, Pittsburgh, PA, December 1999.

[26] J. Schindler, J. L. Griffin, C. R. Lumb, and G. R. Ganger. Track-aligned extents: matching access patterns to disk drive characteristics. *Conference on File and Storage Technologies* (Monterey, CA, 28–30 January 2002), pages 259–274. USENIX Association, 2002.

[27] J. Schindler, S. W. Schlosser, M. Shao, A. Ailamaki, and G. R. Ganger. Atropos: A disk array volume manager for orchestrated use of disks. *Conference on File and Storage Technologies* (San Francisco, CA, 31 March–02 April 2004). USENIX Association, 2004.

[28] S. W. Schlosser, J. L. Griffin, D. F. Nagle, and G. R. Ganger. Designing computer systems with MEMS-based storage. *Architectural Support for Programming Languages and Operating Systems* (Cambridge, MA, 12–15 November 2000). Published as *Operating Systems Review*, **34**(5):1–12, 2000.

[29] S. W. Schlosser, J. Schindler, A. Ailamaki, and G. R. Ganger. *Exposing and exploiting internal parallelism in MEMS-based storage*. Technical Report CMU–CS–03–125. Carnegie-Mellon University, Pittsburgh, PA, March 2003.

[30] N. Talagala, R. H. Dusseau, and D. Patterson. *Microbenchmark-based extraction of local and global disk characteristics*. Technical report CSD–99–1063. University of California at Berkeley, 13 June 2000.

[31] B. D. Terris, S. A. Rishton, H. J. Mamin, R. P. Ried, and D. Rugar. Atomic force microscope-based data storage: track servo and wear study. *Applied Physics A*, **66**:S809 S813, 1998.

[32] M. Uysal, A. Merchant, and G. A. Alvarez. Using MEMS–based storage in disk arrays. *Conference on File and Storage Technologies* (San Francisco, 20, 31-02 April 2003), pages 89–101. USENIX Association, 2003.

[33] P. Vettiger, G. Cross, M. Despont, U. Drechsler, U. Dürig, B. Gotsmann, W. Häberle, M. A. Lantz, H. E. Rothuizen, R. Stutz, and G. K. Binnig. The "millipede": nanotechnology entering data storage. *IEEE Transactions on Nanotechnology*, **1**(1):39–55. IEEE, March 2002.

[34] B. L. Worthington, G. R. Ganger, and Y. N. Patt. Scheduling algorithms for modern disk drives. *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (Nashville, TN, 16–20 May 1994), pages 241–251. ACM Press, 1994.

[35] H. Yu, D. Agrawal, and A. E. Abbadi. Tabular placement of relational data on MEMS-based storage devices. *International Conference on Very Large Databases* (Berlin, Germany, 09–12 September 2003), pages 680–693, 2003.

[36] H. Yu, D. Agrawal, and A. E. Abbadi. Towards optimal I/O scheduling for MEMS-based storage. *IEEE Symposium on Mass Storage Systems* (San Diego, CA, 07–10 April 2003), 2003.

[37] H. Yu, D. Agrawal, and A. E. Abbadi. *Declustering two-dimensional datasets over MEMS-based storage*. UCSB Department of Computer Science Technical Report 2003-27. September 2003.

[38] 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. *Symposium on Operating Systems Design and Implementation* (San Diego, CA, 23–25 October 2000), pages 243–258. USENIX Association, 2000.

[39] C. Zhang, X. Yu, A. Krishnamurthy, and R. Y. Wang. Configuring and scheduling an eager-writing disk array for a transaction processing workload. *Conference on File and Storage Technologies* (Monterey, CA, 28–30 January 2002), pages 289–304. USENIX Association, 2002.