Policies for Probe-Wear Leveling in MEMS-Based Storage Devices

Mohammed G. Khatib and Pieter H. Hartel
Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
{m.g.khatib, p.h.hartel}@utwente.nl

***Abstract*—Probes (or read/write heads) in MEMS-based storage devices are susceptible to wear. We study probe wear, and analyze the causes of probe uneven wear. We show that under real-world traces some probes can wear one order of magnitude faster than other probes leading to premature expiry of some probes. Premature expiry has severe consequences for the reliability, timing performance, energy-efficiency, and the lifetime of MEMS-based storage devices. Therefore, wear-leveling is a must to preclude premature expiry.**

We discuss how probe wear in MEMS-based storage is different from medium wear in Flash, calling for a different treatment. We present three policies to level probe wear. By simulation against three real-world traces, our work shows that an inevitable trade-off exists between lifetime, timing performance, and energy efficiency. The policies differ in the size of the trade-off. One of the policies maximizes the lifetime, so that it is optimal; and the other two are less optimal, and are used based on the configuration of the device.

***Index Terms*—Probe wear, Wear leveling, MEMS-based storage, Probe storage**

I. Introduction

MEMS-based storage is an emerging technology that leverages the well-established MEMS fabrication techniques to offer inexpensive storage solutions. Using probe recording techniques, MEMS-based storage achieves high storage densities ($> 1\,\text{Tb/in}^2$) [1]. Combined with its energy efficiency, MEMS-based storage devices can be used as: a disk cache [2], a streaming buffer and cache [3], a replacement for disk drives [4], and a replacement for flash in mobile systems [5].

Problem: Like other technologies, MEMS-based storage faces some challenges, such as the wear challenge [6], [7], [8]. A MEMS-based storage device is susceptible to probe wear. Probe expiry results in a storage field fault that typically spans thousands of sectors. Maintaining an even level of wear across all probes prevents premature expiry of probes. Wear leveling provides a fully functional MEMS-based storage device, and potentially extends its lifetime.

Contribution: In this work, we devise three wear-leveling policies, and show that:

- An uneven distribution of the number and size of requests across probes causes probe uneven wear.
- Leveling probe wear is crucially important to increase the device lifetime.
- Extended lifetime has to be traded off for performance and energy-efficiency.

The remainder of this paper is organized as follows. Section II introduces MEMS-based storage and sketches its basics. Section III explains the types and causes of wear. Section IV offers our experimental methodology. Section V investigates workload characteristics that cause probe uneven wear. Sections VI–VIII detail our wear-leveling policies. Section IX compares the policies from different design perspectives. We contrast probe and medium wear leveling in Section X. Section XI discusses the related work and Section XII concludes.

II. MEMS-Based Storage

Several design models for MEMS-based storage have been proposed [1], [9], [10], [11]. Although these models adopt different storage and actuation techniques, they have a common architecture. A MEMS-based storage device consists of two distinct physical layers, one above the other, as shown in Figure 1a. The top layer, called the *media sled*, is suspended by springs across the bottom layer, where the Z distance is maintained by nanopositioners. The bottom layer is a two-dimensional array of read/write probes or heads, called the *probe array*. For example, an IBM MEMS prototype [1] has a 64×64 probe array.

Bits can be recorded on a magnetic patterned medium as in μSPAM [10] and the CMU MEMStore [9]; a polymer medium as in the IBM MEMS device [1]; or a phase-change medium as in the Nanochip MEMS device [11]. The sled moves independently in the X, Y, and Z directions relative to the probe array. In all design models, each probe sweeps over a bounded area of the media sled, called the *probe (storage) field* as sketched in Figure 1b. Consequently, seek times shorten. Further, a relatively high (aggregate) data rate is attained by striping a sector across a probe set of several probes. Schlosser *et al.* [12] study the data layout in detail.

III. Wear in MEMS-Based Storage

MEMS-based storage devices have two main types of wear: (1) probe wear and (2) medium wear [6], [7], [8]. Probe wear inhibits the tip of the probe to write or read nanometer-sharp bits. Medium wear inhibits the individual location on the medium to store or to retain a bit for a certain amount of time. Medium wear affects the device on a sector basis, whereas probe wear affects a probe field, which spans thousands of sectors.

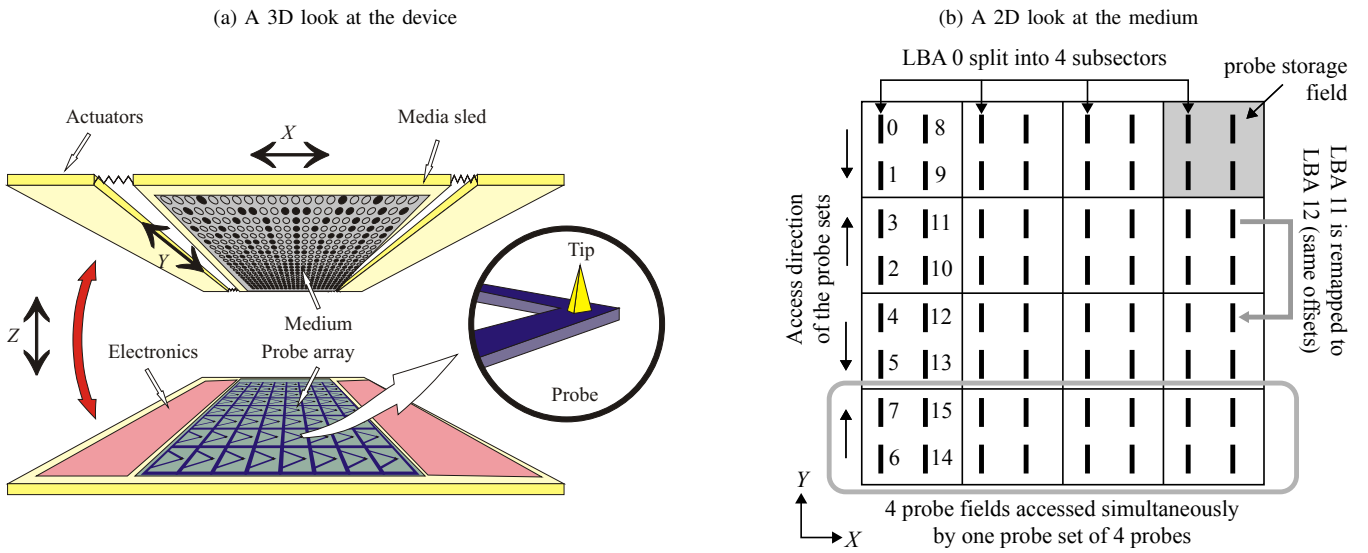

Fig. 1: Three- and two-dimensional views of a MEMS-based storage device. *(a) Two layers facing each other where the media sled is attached to springs that suspend it across the probe array. (b) The storage area of a simplified MEMS-based storage device consisting of* 4×4 *probe storage fields.*

The cause of probe wear varies depending on the recording technology. Usually, wear is caused by friction at high load on the probe tip, high temperature, and high velocity. In addition, Bhushan *et al.* [7] indicate that tribochemical reactions can take place. Likewise, the medium wears due to several factors including high temperature, and contact with the probe.

Based on the literature [6], [7], [8] and discussions with physicists, we assume the following in our work:

- The write operation is the main cause of probe wear and medium wear. We use *the number of written bits per probe as a metric of wear*. That is, the larger the number of written bits, the more significant the wear. We assume the effects of reading on probe wear to be small enough to be negligible.
- A probe wears a few orders of magnitude faster than an individual bit location on the medium. This work focuses on probe wear.
- A probe can write at least 100 times the capacity of its storage field (i.e., 10^9 bits in our model) before it starts to function unreliably. Physicists are enhancing the endurance of the probes and the medium [13].

A. Effects of Probe Wear

The wear phenomenon at the probe level manifests itself in a problem of *uneven wear* of probe sets at the device level. Since all probes in a probe set write exactly the same number of bits (Section II), probes within a probe set wear evenly. Individual probe sets, however, can write a different number of bits depending on the workload (Section V). Uneven wear can be significant as Figure 3 shows. Uneven wear of probe sets influences a MEMS-based storage device as follows:

Reliability: If some probe sets wear out before others, their respective storage fields become inaccessible. As a result, user data located in these fields are lost.

Performance: If some probe sets wear before others, the number of probes that can operate in parallel decreases. As a result, the data transfer rate of the device decreases.

Energy: Wear of probe sets reduces the probe parallelism and increases energy consumption: reducing probe parallelism reduces the number of bits accessed in parallel. Consequently, the sled moves longer distances along Y and stays still for a longer time along X, increasing the actuation energy [5].

Capacity: A loss of just one probe results in a loss of its storage field which is typically several megabytes in capacity. A loss of a probe set reduces the device capacity by several hundreds of megabytes.

B. Device Life

Our objective is to preclude premature expiry of probes in a MEMS-based storage device in order to prevent the effects of uneven wear. Rephrasing, our objective is *maximizing the lifetime of the individual probes, so that they live throughout the entire lifetime of the device,* and thus expire more or less simultaneously. The **device lifetime** is the total (aggregated) number of bits written by all probes of the device before it expires.

Let us assume an example MEMS-based storage device of 10 probes, each can write a maximum number of 10^9 bits before it expires. Figure 2 shows the two extreme cases of the life of this device. The best-case life happens when each probe lives the entire lifetime (i.e., 10×10^9 bits) of the device (the green dashed line). The worst-case life happens when probes expire one after another (the red dotted line). The best-case life and the worst-case life demonstrate how the maximum lifetime of the device can be achieved with a different lifetime of the individual probes. Therefore, probe lifetime is the main

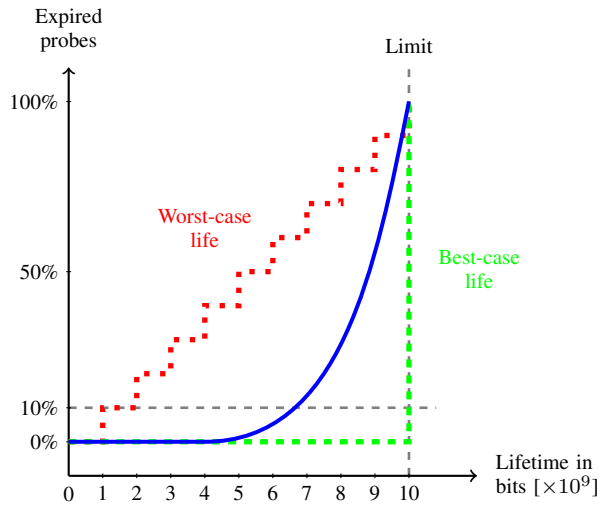

Fig. 2: Illustration of the best-case, worst-case, and typical life of a MEMS-based storage device

concern. Our objective life is the best-case one. An example of the life of a device with no wear leveling is shown by the blue curve.

Suppose that, in practice, a device is assumed expired if 10% of the probes expire. Figure 2 shows that the device writes 1×10^9, 7×10^9, 10×10^9 bits in the worst-case, typical, and best-case life, respectively, before it expires. Thus, we can say that wear leveling increases the device lifetime. Observe that if the threshold is 100% the device lifetime is achieved by any (even no) policy, so that it is not a concern. At 100%, the only difference between wear-leveling policies is the probe lifetime to prevent the four effects discussed previously.

Summarizing, wear leveling must maximizes the lifetime of the individual probes to preclude the effects on the reliability, timing performance, energy-efficiency, and the capacity of a MEMS-based storage device. In addition, the device lifetime as a fifth target is also equally important. This is because in practice a device is abandoned, if admitting a write request would expire one of the probe sets. In other words, the practical threshold is in fact 0% (and not 100% or even 10%), since user data must not be lost.

IV. Experimental Methodology

MEMS-based storage devices are not publicly available at present. Therefore, we use trace-driven simulations.

a) MEMS Model: We use the DiskSim simulator [14]; a validated modular simulator for simulating various types and architectures of storage subsystems. We refine the performance and energy models of the CMU MEMS model [12] to model the IBM MEMS with better accuracy. IBM prototyped a MEMS-based storage device of 64×64 probes [1]. All the model parameters including, the bit dimensions and the per-probe data rate of the model are set to those of the IBM MEMS device [1]. Table I summarizes the key settings of our MEMS model.

TABLE I: Settings of our MEMS model

total # of probes	64×64	probes
bit/track pitch	40	nm
probe field area	100×100	μm^2
per-probe data rate	40	Kbps
probe-set size	256	probes
sector parallelism	1	sector
sector size	4	KB

b) MEMS Translation Layer (MTL): We envisage a MEMS Translation Layer (MTL) between the interface of a MEMS-based storage device and the physical driver. We implemented an MTL and coupled it with DiskSim. The MTL receives I/O requests, processes them, and then forwards them to DiskSim for servicing. Implementations of the wear-leveling policies devised in this work took place in the MTL. The MTL enforces the wear-leveling policy, monitors wear of probes and hotness of data, remaps logical block addresses to physical block addresses (LBA to PBA), and gathers statistics about probe wear and the LBA-to-PBA mapping.

The map size can be of concern, since it can be relatively large. For example, a MEMS-based storage device of 64 GB capacity with a sector size of 4 KB requires a map size of 64 MB, if the address is represented in 32 bits. Flash Translation Layer solves this issue by storing the frequently accessed blocks in a fast memory, while keeping the rest on the Flash itself. Doing so, lookup operations are carried out quickly, while the demand for fast memory is reduced. Nonetheless, keeping parts in Flash itself increases the wear, since Flash cannot update data in place. Flash wear-leveling strive to minimizes this effect. Since a MEMS-based storage device has no erase-before-write constraint, it can update the parts of the map on the MEMS in place, so that it does not increase the probe wear due to migration. Further, these parts can be distributed throughout the device to maintain fair wear across probes.

In MEMS-based storage devices, updating incurs no data migration, since data can be updated in place, unlike in Flash. But updating itself causes wear. However, the influence of updating the index on probe wear is very marginal compared to wear due to writing a sector. For example, to write a sector $4 \times 8 \times 1024 = 32768$ bits are written, whereas updating its address requires writing just 32 bits. In other words, probe wear due to map updates is 1024 times smaller than that due to writing sectors. Still, if we factor in the buffering in the fast memory, then this factor is likely to increase further. Therefore, this work factors out the wear due to map updates.

c) Traces: We captured and simulated against three traces: (1) `iozone`, (2) `multimedia`, and (3) usage `scenarios`. We ported the IOzone benchmark [15] to our ARM-based PDA and tested with several access patterns, including sequential, random and stride reads and writes with various record and stride sizes. The wide coverage of access patterns of IOzone benchmark enables us to test the wear-leveling policies for almost every access pattern a MEMS device can encounter in real world, which increases the

TABLE II: Statistics of the three traces used in this research

metric	iozone	multimedia	scenarios
I/Os	410,627	21,867	2679
seq. %	21.2	82.7	44.1
write %	43.1	40.4	56.3
request size statistics [0.5 KB sector]			
min	2.0	8.0	8.0
median	8.0	8.0	8.0
max	256	256.0	256.0
mean	31.0	17.3	43.3
std. dev.	54.2	27.8	69.3

comprehensiveness of our study.

We also captured a `multimedia` trace that included photo taking, single and dual streaming from and to the storage device. Dual streaming represents a scenario where the user is playing back a stream and downloading another at the same time. All streaming scenarios were captured for audio ($16 - 128$ Kbps) and video qualities (64 Kbps–2 Mbps) with various chunk sizes ($4 - 256$ KB).

We captured a third `scenarios` trace that logs different system and application activities. System activities included booting and starting the Graphical User Interface, whereas application activities included: firing applications, such as the text editor and web browser; and creating, copying and deleting files. Table II summarizes the statistics of the three traces.

d) Evaluation: In the following, we devise three wear-leveling policies and evaluate them. The difference between these policies boils down to two choices: (1) the selection of a request for remapping (**candidate request**), and (2) the selection of a probe set to remap the candidate request to it (**victim probe set**). These selections affect the probe lifetime and thus the device lifetime, resulting in a different lifetime of the device with each policy. In all policies, if a victim probe set cannot be found, the default set is selected. The **default victim probe set** is determined according to the LBA to PBA mapping of the data layout as illustrated in Figure 1b.

For all policies, we preserve the X and Y offsets of a sector in its default storage field when remapping to another storage field (see Figure 1b). As a result, seeks due to unavailability in storage space are not included. However, seeks to reposition the medium after the distance it moved while remapping are included. Although seeks due to space unavailability are likely to be incurred in practice, excluding them allows us to single out the direct influences of remapping on the response time and energy-efficiency of MEMS-based storage devices. The direct influences are: (1) processing needed for remapping, (2) lowering the sequentiality, and (3) remapping of subsequent read requests.

e) Standard Deviation: As the green dashed line indicates in Figure 2, our objective is to maximize the lifetime of the individual probes. To this end, a wear-leveling policy must maintain simultaneous growth of wear across probe sets. The simultaneous growth corresponds to a constant probability distribution, where all probes write the same number of bits. The constant probability distribution has a standard deviation of zero. Uneven wear causes a deviation from the constant distribution, so that the standard deviation becomes larger than zero. We use the standard deviation as a metric of imbalance, and pursue its evolution over time to monitor the simultaneous wear growth.

Note that the policies, offered in this work, strive to maintain the wear across all probe sets at more or less the same level all the time. As a result, the policies prevent *by construction* the creation of tailed probability distributions. This prevention is achieved in all policies by (1) triggering the remapping upon every write request, and (2) cycling through the probe sets. The policies compete to transform the resulting (semi-) uniform probability distribution into an ideal constant probability distribution. Therefore, the standard deviation is a sufficient metric for the decrease in the width of the semi-uniform probability distribution to reach the constant distribution.

V. Causes of Uneven Wear

Uneven wear of probe sets in a MEMS-based storage device is caused by unevenly distributed accesses to areas on the storage medium by I/O requests. An I/O request r is represented as a tuple: $(t_\mathrm{r}, A_\mathrm{r}, S_\mathrm{r}, O_\mathrm{r})$, where t_r is the arrival time of the request, A_r is the logical address of the starting block, S_r is the size of the request, and O_r is the operation of the request: read or write.

The properties of a request r that affect the mapping to the physical space are the address (A_r) and the size (S_r). The address determines the starting probe set. The size determines the consecutive probe sets as well as the load on each set per request. These properties determine the wear of the probe set, if the operation (O_r) is a write operation. We quantify the influence of request address and size on uneven wear by testing two hypotheses using our traces. The hypotheses are:
Hypothesis 1: An uneven distribution of the number of requests across probe sets causes uneven wear.
Hypothesis 2: An uneven distribution of the size of requests across probe sets causes uneven wear.

We use the captured traces to test our hypotheses as follows. For hypothesis 1, we resize all requests in a trace to the average request size. We leave the address unchanged, thus any uneven wear of probe sets observed is attributed to the request address (A_r) only, and not to the request size. We take the average request size to quantify the influence of the request address versus the request size on uneven wear. For hypothesis 2, we map all requests in a round-robin fashion across all probe sets to distribute requests evenly over probe sets. We keep the size of individual requests unchanged, so that any uneven wear observed is attributed to the request size (S_r) only.

Figure 3 shows the distribution of written bits when testing hypothesis 1 for a full run of the `scenarios` trace. We observe the same when testing hypothesis 2, but at a smaller degree of imbalance. The results confirm the deviation from the ideal equal distribution of writes. We observe that wear can be larger by an order of magnitude for some probe sets than others; for example, compare probe set 35 to 16 in Figure 3.

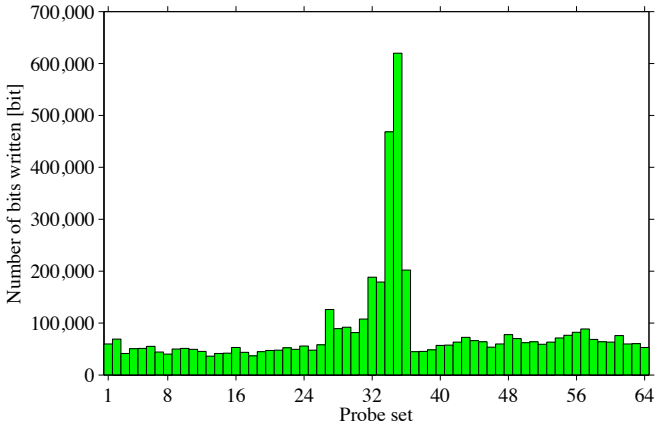

Fig. 3: Wear distribution across probe sets when simulating against the `scenarios` trace. The figure confirms Hypothesis 1 that the number of requests causes uneven wear.

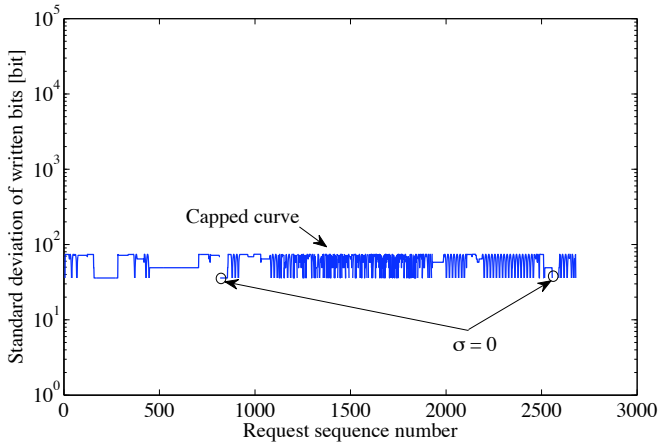

Fig. 4: Standard deviation of written bits across the probe sets for the `scenarios` trace when employing the sector-based round-robin policy

Note that each probe set is responsible for several noncontiguous physical areas on the medium, because of the logical data layout depicted in Figure 1b. As a result, the peaks in Figure 3 are not due to just one hot area, but actually several noncontiguous hot areas [1].

In addition, we test the hypotheses against the `multimedia` and `iozone` traces, and arrive at conclusions in conformity to those for the `scenarios` trace. Thus, we confirm the influence of the number of requests and the size of the request. Based on both hypotheses, we can construct a policy that optimally levels the wear. The policy should write an equal number of sectors to each probe set, while cycling through the probe sets in a round-robin fashion. We call this policy the sector-based round-robin policy and detail it next.

VI. Sector-Based Round-Robin Policy

The sector-based round-robin policy levels wear by writing to probe sets in a round-robin fashion.

Selection of the candidate request: `rrSector` considers each arriving I/O request as a candidate, and remaps its sectors.

Selection of the victim probe set: `rrSector` cycles in a round-robin fashion through probe sets. It maintains an index for cycling.

The `rrSector` policy remaps every sector to a subsequent probe set in a round-robin fashion, regardless of the request it belongs to. Consequently, sequential sectors of the same request are mapped to different probe sets, cutting the sequentiality.

The round-robin mapping scheme of `rrSector` guarantees simultaneous growth of wear across all probes. It further guarantees that the difference in wear between any two probe sets is limited to the subsector size at any time instance.

[1]We investigated identifying requests of hot sectors, and remapped them to the coldest probe set in an attempt to level wear. We concluded that hot data as well as cold data wear probe sets unevenly. Hot data are frequently accessed, so that their respective probe sets are heated. But also cold data arrive in large amounts that map to the same sets, so that they heat probe sets too.

Figure 4 confirms the bound and shows a capped standard deviation. The policy achieves an equal distribution of wear across probe sets every full round-robin cycle, reducing the standard deviation to zero at several points, such as the 800-th request. These points are shown as a discontinuity, since zero corresponds to minus infinity on a logarithmic scale.

f) Optimality: The `rrSector` policy represents the optimal wear-leveling policy in practice, since it bounds the difference in wear to the subsector size. And the subsector is the smallest atomic unit a probe (in a probe set) writes per sector. It, thus, maximizes the probe and the device lifetime.

The maximization of the probes and the device lifetime comes at a cost however. That is, `rrSector` compromises on the timing performance and energy-efficiency, because it remaps the sectors of a request to noncontiguous locations. As a result, the applicability of `rrSector` in practice reduces particularly for mobile battery-powered devices. In the following, we present alternative policies that favor timing performance and energy-efficiency. We take `rrSector` as a reference point with respect to lifetime.

VII. Coldest-Probe Policy

This section presents a policy that preserves the request in its entirety. For the sake of brevity, we represent the degree of wear (i.e., the number of written bits) by temperature. That is, the more worn a probe, the higher its temperature.

Selection of the candidate request: the policy marks an incoming request as a candidate, if its default probe set is hotter (more worn) than at least one other set.

Selection of the victim probe set: the policy remaps candidate requests to the coldest probe set at the arrival time of the request. The policy keeps track of the number of written bits of each probe set.

The policy assumes that remapping a request to the coldest probe set contributes to minimizing the variance in wear across probe sets. The efficacy of the `coldestProbe` policy depends on the size of the arriving request. That is, a large

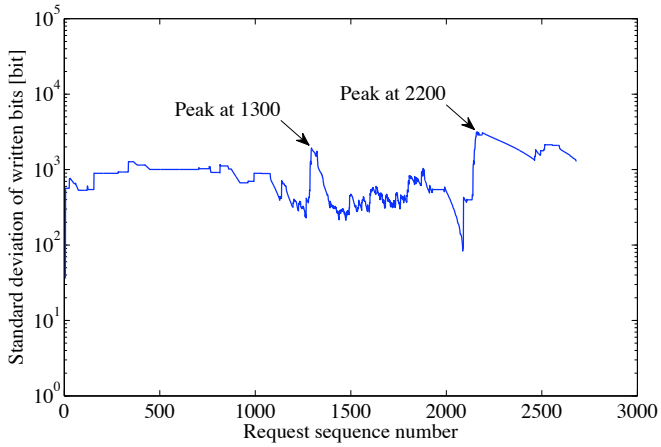

Fig. 5: Standard deviation of written bits across the probe sets for the `scenarios` trace when employing the coldest-probe policy. Peaks exist due to large sequential requests

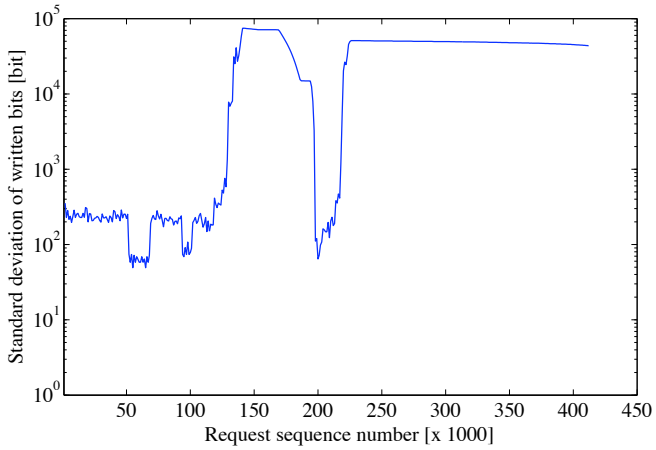

Fig. 6: Standard deviation of written bits across the probe sets for the `iozone` trace when employing the coldest-probe policy. Large peaks exist due to large sequential requests

request size increases the imbalance, whereas a small one has a little influence. We observed this in the difference of the maintained levels of the standard deviation for the three traces, since they have different dynamics.

Figure 5 shows two peaks in the standard deviation at the 1300-th and 2200-th request. The peaks are approximately one order of magnitude larger than the maintained level of the standard deviation. Similar peaks were observed with the other two traces, particularly for the `iozone` trace such peaks are two orders of magnitude larger than the maintained level as shown in Figure 6. Analyzing the mapping, we found that these peaks are caused by a flurry of large requests that always map unevenly to two or three consecutive probe sets. These flurries write not only with cold probe sets, but also with hot probe sets too, which increases the standard deviation. Worse, the cold probe set remains relatively cold to its hot neighbors, so that successive large requests to the same probe sets still map the same way, further increasing the deviation. Small requests arrive later, which map entirely to cold sets, leveling the wear again.

Such flurries can be potentially harmful for the device lifetime, since they can expire some probe sets before others; recall that we want to avoid the red dotted line in Figure 2. On the other hand, the flurries certainly harm the performance and energy-efficiency of MEMS-based storage devices. If a probe set gets overly heated by a flurry, `coldestProbe` remaps all future requests, whose default probe set is the heated one. For the ensuing remappings, a MEMS-based storage device has to lookup an alternative storage location and seek to it, incurring performance and energy costs[2]. Since the flurry is large (one to two orders of magnitude), the costs are likely to be incurred for a large number of requests, hurting the response time and energy-efficiency. The incurred costs can be particularly large, if they are caused by streaming (large) requests and afflicting best-effort (small) requests.

The overheating problem can be solved by cutting the request and remapping to cold probe sets. We rule out this choice, since it compromises on the performance and energy-efficiency. In a further study for large probe sets (of 1024 probes or more), we found that overheating disappears and thus `coldestProbe` remains a viable design choice. We leave out the results due to space limitations. Next, we present a variant policy.

VIII. Barrier-Based Policy

The barrier-based policy (`barrier`) is inspired by parallel computing [16]. A barrier, in parallel computing, is a synchronization technique that halts a process at a certain point in its execution from proceeding until all other processes reach the point. In the `barrier` policy, the barrier represents the number of written bits.

Our goal is to maintain simultaneous growth of wear across probe sets, so that, optimally, they reach their maximum lifetime simultaneously. Toward that ultimate goal, we set incremental subgoals for the probe sets, so that all probe sets reach each subgoal simultaneously. These subgoals make up our barriers down the operation time of the probe sets. The idea is that lining up probe sets at every barrier avoids racing between them toward the ultimate goal or barrier.

Selection of the candidate request: The barrier-based policy maps a request to its default probe set unless the probe set has already crossed the barrier. Then, the request becomes a candidate one and gets remapped. The policy maps a large request to its default probe sets as long as one of them has not crossed the barrier.

Selection of the victim probe set: The barrier-based policy selects for a candidate request a sufficiently cold probe set. That is, it chooses a probe set, so that if the request is remapped to this probe set, the probe set reaches or crosses the barrier with the minimum distance (i.e., number of written

[2]This resembles an inefficient usage of a RAID-0 system. Instead of dispatching requests to all available disks to elevate the throughput and reduce seeks, requests are dispatched to one disk at a time.

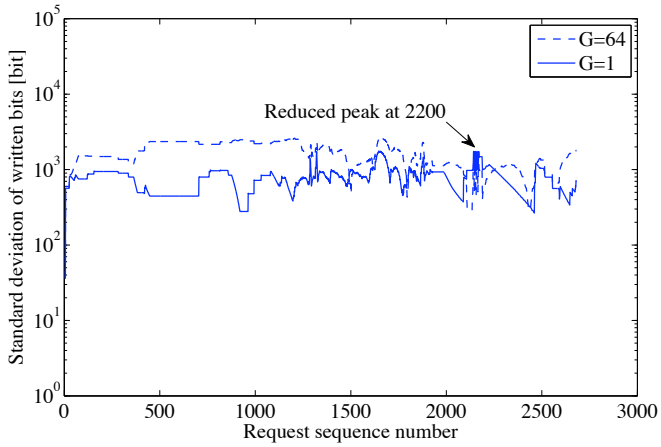

Fig. 7: Standard deviation of written bits across the probe sets for the `scenarios` trace when employing the `barrier` policy with $G = 1$, and 64 sectors. Increasing G increases the deviation from leveled wear.

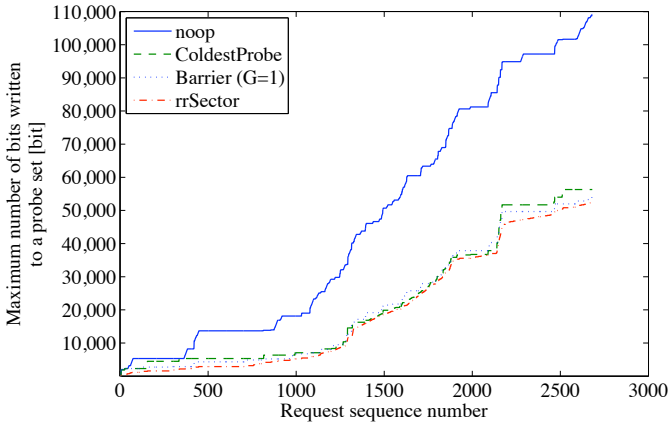

Fig. 8: Evolution of the maximum number of bits written per probe set for the `scenarios` trace. The figure has scale granularity of 10,000 bits, which amounts to 625 sectors.

bits). Ideally, this distance should be zero. Remapping with the minimum distance allows the barrier-based policy to minimize the difference between probe sets. If no victim probe set is found, the default probe set is selected.

Updating the barrier: If all probe sets have crossed the current barrier (or subgoal), the barrier is incremented to the next multiple of a granularity G that is a parameter of the policy. The granularity G represents a trade-off between uneven wear and the number of remappings. Setting the granularity G is particularly important, since very small G leaves no room for remappings, and large G defers remapping, resulting in either case in bad wear leveling. Our experiments suggest that the granularity should be larger than or equal to one sector and smaller than twice the maximum request size.

Figure 7 confirms the policy's capability of dealing with the causes of uneven wear distribution. The peak at 1300 shown in Figure 5 has disappeared, thanks to the barrier. The second at 2200 still appears but at a smaller magnitude. The figure also shows that uneven wear increases as the barrier granularity increases. The results for the `multimedia` and `iozone` traces agree with those for `scenarios`.

IX. Comparison

This section compares the three wear-leveling policies, `rrSector`; `coldestProbe`; and `barrier`, with respect to lifetime, response time, and energy-efficiency. We study three settings of the barrier for the `barrier` policy with $G = 1, 8$, and 32 sectors to extract possible trends. For comparison, we also show the figures when deploying no wear-leveling policy (called `noop`). We present the results for the `scenarios` trace and discuss for the other traces.

A. Device Lifetime

This section evaluates the device lifetime with each of the three policies by tracking the proliferation of the wear across the probe set with the maximum wear. The maximum wear is an indication of the reduction in the device lifetime (in bits) as the most worn probe set expires. Graphically, we are measuring the point at which the blue curve in Figure 2 leaves the 0% threshold.

At design time, the designer translates the number of bits into years depending on the expected application. For example, assume an application that sends 100 requests to the device on average per day, and the device expiry limit is 50,000 bits. Taking the `scenarios` trace as an example of a usage pattern, the 50,000 limit intersects with the curve of `noop` at the 1500-th request and intersects with the `barrier` at the 2200-th request in Figure 8. This boils down to a lifetime of 15 respectively 22 days, increasing the lifetime by a factor of 1.46.

Figure 8 plots the development of the maximum number of bits of the most worn probe set for the `scenarios` trace. The figure illustrates the benefit of implementing wear leveling. That is, any of the three policies results in an increased lifetime compared to the `noop` policy. Further, the `coldestProbe` and `barrier` polices are relatively much closer to the optimal `rrSector` policy than to the `noop` policy.

Figure 8 shows that `barrier` achieves better leveling than the `coldestProbe`, since it can handle flurries of large requests. The figure reveals that `barrier` exhibits occasionally larger wear than `coldestProbe`. This is because barrier maps a request to its default probe set as long as the probe set has not crossed the barrier, leading to temporary overheating. On the other hand, `coldestProbe` maps always to the coldest, and, therefore, avoids temporary overheating due to small requests.

B. Performance and Energy

This section studies the influence of each wear-leveling policy on the timing performance and the energy consumption of our simulated MEMS-based storage device.

Figure 9 shows that the `rrSector` policy exhibits longer response time than the other policies. The policy incurs

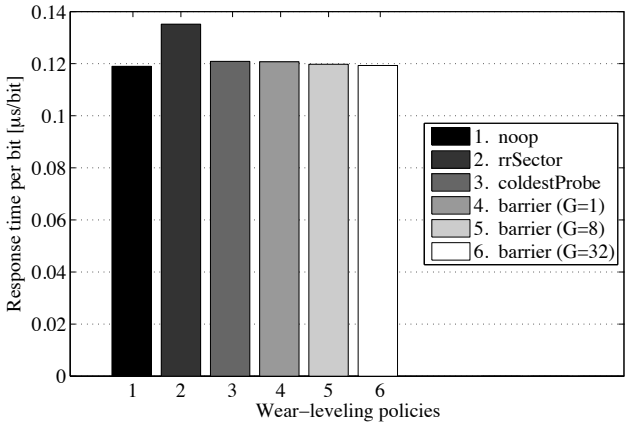

Fig. 9: Per-bit response time of our simulated MEMS-based storage device for the wear-leveling policies when simulating against the `scenarios` trace.

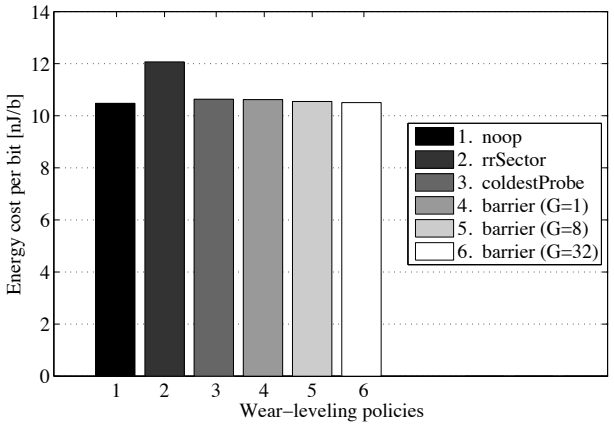

Fig. 10: Per-bit response time of our simulated MEMS-based storage device for the wear-leveling policies when simulating against the `scenarios` trace.

(1) larger request processing overhead, and (2) reduce the sequentiality of requests more than the other policies, since it remaps on a sector basis. As a consequence, the response time per bit of the MEMS-based storage device increases by about 20% for the `rrSector` policy compared to `noop`.

The influence of the other two policies is smaller than that of `rrSector`, since they preserve the sequentiality and reduce the remapping overhead significantly. Figure 9 shows that these policies exhibit a similar performance. This is because request processing overhead is incurred regardless if remapping is triggered or not. In practice, the difference for the favor of `barrier` is more pronounced, since additional seeks are incurred due to space unavailability when remapping.

Similar to performance, Figure 10 shows that the energy consumption increases as the sequentiality decreases. Further, an additional indirect influence on energy exists: since remapping causes more mechanical activities, the media sled spends more time in seeks and turnarounds. Consequently, a MEMS-based storage device has less opportunity to shut down for energy saving.

C. MEMS Design Trade-offs

We summarize our findings in Table III. It ranks the policies from five design targets: (1) device lifetime, (2) response time, (3) energy consumption, (4) the LBA-to-PBA map size of a MEMS-based storage device, and (5) the implementation cost of the policy. Two policies receive the same ranking when their influence on the respective target is comparable. A difference in ranking of n ($n \geq 1$) between two policies denotes a significant difference, in the range of n orders of magnitude.

From a lifetime perspective, `rrSector` and `barrier` rank first, since they maximize the utilization and the lifetime of the individual probes with a little difference in favor of the former. For a probe set larger than 1024 probes, `coldestProbe` ranks first as well and outperforms `barrier`. But for a probe set smaller than 1024, `coldestProbe` gives no guarantee for even wear.

From performance and energy-efficiency perspectives, `barrier` ranks second, since remapping is triggered every few requests (i.e., the barrier). The second last is `coldestProbe`, which remaps every request, whereas `rrSector` ranks last which remaps every sector. Unlike seek time, seek energy is small relative to the total energy of a MEMS-based storage device. As a result, `coldestProbe` and `barrier` ranks comparable.

For the map size, `rrSector` and `coldestProbe` rank comparable, since they remap almost every request. In contrast, `barrier` reduces the size of the map by an order of magnitude.

Another dimension that we add to the ranking study, is the cost or the effort needed to implement each of the policies, which can be measured by the lines of code written or amount of time needed to come up with an appropriate implementation. This factor influences the amount of time to implement a certain policy, which in turn reflects in the device cost. Here, `noop` incurs no implementation costs, whereas `rrSector` and `coldestProbe` incurs more cost to assign the probe for remapping and allocate an available sector. The most expensive policy is `barrier`, which requires implementing mechanisms to update the barrier. Further, if the barrier requires dynamic updating, then the implementation cost is likely to increase further.

X. Wear-Leveling in Flash Memory

This section discusses the differences between probe wear leveling in MEMS-based storage and medium wear leveling in Flash. To ease the discussion, we use the analogy of temperature: hot data are frequently written data, hot (physical) sectors are worn sectors, and hot probes are worn probes. Note that the temperature is a relative metric; for example a probe cools down if the temperature of the others increases relative to it and vice versa.

A probe is a means to read and write data, whereas the medium is a container that stores data. Henceforth, we discuss

TABLE III: Ranking from 1 (best) to 4 (worst) of the policies from different design perspectives

Wear-leveling policy	Device lifetime		Response time	Energy usage	Map size	Implementation cost
probe-set size	< 1024	≥ 1024				
`rrSector`	1	1	4	4	4	2
`coldestProbe`	3	1	3	2	4	2
`barrier`	1	1	2	2	3	3
`noop`	4	4	1	1	1	1

medium wear in terms of sector wear, since it is the basic unit of storage. It is essential to observe the difference in the cause of uneven wear between probes and sectors. Probe uneven wear is due to uneven number and size of writes dispatched to probes (Section V). On the other hand, sector uneven wear is due to the coexistence of hot and cold data. For example, assume that we write a sector ten times with one probe and we write ten other sectors one time with another probe. Both probes are worn equally by writing ten times the size of a sector. In contrast, the sectors have a different degree of wear; one is written nine times more often than the others. This example demonstrates the essential difference in the cause of uneven wear of probes and sectors, which stems from the difference in their function.

The difference in function necessities a different approach in cooling down probes and sectors. That is, a hot probe cools down by avoid using it temporarily. On the contrary, a hot sector cools down by writing cold data in it. This is, because writing data in a sector makes it unavailable, and thus cannot be used anymore unless data are migrated out. On the other hand, a probe does not contain data, and is thus available all the time. A probe becomes unavailable, if its storage field is entirely filled. In our case, all fields get entirely filled at the same time, because the data layout fills the fields simultaneously (Section II).

The difference in function has a third consequence. Recall that a sector is a container, so that it can be used only if it is available (i.e., contains no useful data). A probe can be used almost all the time, since it is merely a loader and not a container. Data should be migrated from a sector to make it available, so that the sector can be used to store new data for wear leveling. In other words, data migration is an intrinsic part of medium wear leveling to guarantee the circulation of hot data through all sectors in order to level the wear. Data migration causes wear itself since it incurs writing, therefore it should be minimized. Worse yet, cold data make up the majority of data in real-world workloads, so that minimizing the migration of a huge amount of data is essentially necessary for performance reasons. In contrast to medium wear leveling, probe wear leveling requires virtually no data migration.

In summary, probe wear leveling is essentially simpler than medium wear leveling. A probe wear leveling policy does not need to establish the temperature of data, it must avoid writing with hot probes, and it rarely involves data migration. On the contrary, a medium wear leveling policy must establish the temperature of data for proper mapping, must write cold data to hot sectors for cooling down, and must migrate data to keep the circulation of hot data through sectors flowing.

XI. Related Work

We can classify the work available in the literature on MEMS-based storage into two main categories: (1) deployment and (2) enhancement of MEMS-based storage devices. The deployment part investigates roles of MEMS-based storage devices to enhance the timing performance of computer systems. Previous work proposes to use MEMS-based storage devices as (1) a disk cache [2], (2) a streaming buffer and cache [3], (3) a replacement for disk drives [4], and a replacement for flash in mobile systems [5]. The enhancement part, on the other hand, investigates enhancing the performance and reducing the energy consumption of MEMS-based storage devices. Techniques for the data layout [12], [17], [5], scheduling policies [18], and techniques to estimate the physical dimensions [19] have been proposed.

Since only recent development reveals the wear challenge in MEMS-based storage, wear is a new research topic in MEMS-based storage. Experiments with MEMS-based storage prototypes reveal two types of wear: (1) medium wear and (2) probe wear [6], [7], [8]. Like MEMS-based storage, flash memory is susceptible to medium wear. The majority of work on wear in flash tackles data migration. Several techniques have been proposed to limit data migration, and thus to reduce its overheads. Chang [20] compares several of the proposed wear-leveling algorithms. Gal and Toledo [21] give a detailed survey of the algorithms and their data structures.

XII. Summary

This paper addresses the uneven wear problem of probes in MEMS-based storage devices. We devised three wear-leveling policies that result in a different influence on the lifetime, timing performance, energy-efficiency of a MEMS-based storage device. Also, the policies differ in their respective implementation cost. One of the policies represents the optimal policy in practice that maximizes the lifetime.

We evaluate the policies by simulating against real-world traces. We find in a case study that wear leveling increases the device lifetime by a factor of 1.46. Our simulation results show that designing a wear-leveling policy involves trade-offs between the device lifetime, the required storage resource, and the resultant performance and energy-efficiency.

We present the sector-based round-robin policy, which achieves the best wear leveling, resulting in the maximum device lifetime. The policy maximizes the device the lifetime. However, because it breaks requests into sectors and remaps

every sector, this policy increases the response time and the energy consumption by approximately 20% compared to the other two policies.

The second policy is called the coldest-probe policy, which remaps a request in its entirety to the probe with the least wear at the remapping time. This policy results in better performance and energy-efficiency than the previous policy for a comparable lifetime for large probe sets. In contrast, for small probe sets, the policy cannot cope with flurries of large requests, and can thus reduce the lifetime significantly.

The barrier-based policy sets barriers down the operation time of the device, where a barrier is a certain number of bits. The policy remaps requests in their entirety, so that all probe sets reach a barrier simultaneously. The policy has a smaller influence on the performance and energy-efficiency than the second policy. It incurs less remapping, and therefore requires less storage.

The barrier policy is capable of coping with flurries of large requests, and thus ranks second in lifetime after the optimal policy. The coldest-probe policy remains, however, a viable design choice for large probe sets, since it incurs less remapping overhead compared to the barrier policy. Unlike the other two policies, the barrier policy requires more implementation effort, adding to the cost of the device.

XIII. Acknowledgments

We thank Haris Pozidis from IBM Zürich Research Laboratory for providing insight into wear in MEMS-based storage. We also thank Ethan L. Miller at the University of California at Santa Cruz, Hylke W. van Dijk at the University of Twente, and the anonymous MASCOTS reviewers for their useful comments. This research is supported by the Technology Foundation STW, applied science division of NWO and the technology programme of the Ministry of Economic Affairs under project number TES.06369.

References

[1] M. A. Lantz, H. E. Rothuizen, U. Drechsler, W. Häberle, and M. Despont, "A Vibration Resistant Nanopositioner for Mobile Parallel-Probe Storage Applications," *Journal of Microelectromechanical Systems*, vol. 16, no. 1, pp. 130–139, February 2007.

[2] F. Wang, B. Hong, S. A. Brandt, and D. D. E. Long, "Using MEMS-based storage to boost disk performance." in *MSST'05: 22nd IEEE Conference on Mass Storage Systems and Technologies*, April 2005, pp. 202–209.

[3] R. Rangaswami, Z. Dimitrijevic, E. Chang, and K. E. Schauser, "MEMS-based disk buffer for streaming media servers," *Proceedings of 19th International Conference on Data Engineering*, pp. 619–630, March 2003.

[4] 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*, October 2000, pp. 1–12. [Online]. Available: http://citeseer.ist.psu.edu/schlosser00designing.html

[5] M. G. Khatib, E. L. Miller, and P. H. Hartel, "Workload-based Configuration of MEMS-Based Storage Devices for Mobile Systems," in *Proceedings of the 8th ACM & IEEE International Conference on Embedded Software (EMSOFT '08), Atlanta, USA*. USA: ACM, October 2008, pp. 41–50.

[6] M. Hinz, O. Marti, B. Gotsmann, M. A. Lantz, and U. Durig, "High resolution vacuum scanning thermal microscopy of HfO_2 and SiO_2," *Applied Physics Letters*, vol. 92, no. 4, p. 043122, August 2008. [Online]. Available: http://link.aip.org/link/?APL/92/043122/1

[7] B. Bhushan, K. J. Kwak, and M. Palacio, "Nanotribology and nanomechanics of AFM probe-based data recording technology," *Journal of Physics: Condensed Matter*, vol. 20, no. 36, p. 365207 (34pp), August 2008. [Online]. Available: http://stacks.iop.org/0953-8984/20/365207

[8] R. Berger, Y. Cheng, R. Forch, B. Gotsmann, J. Gutmann, T. Pakula, U. Rietzler, W. Schartl, M. Schmidt, A. Strack, J. Windeln, and H.-J. Butt, "Nanowear on polymer films of different architecture," *Langmuir*, vol. 23, no. 6, pp. 3150–3156, March 2007. [Online]. Available: http://pubs3.acs.org/acs/journals/doilookup?in_doi=10.1021/la0620399

[9] 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 (invited)," *Journal of Applied Physics*, vol. 87, no. 9 III, pp. 6680–6685, 2000. [Online]. Available: www.scopus.com

[10] L. Abelmann, T. Bolhuis, A. M. Hoexum, G. J. M. Krijnen, and J. C. Lodder, "Large capacity probe recording using storage robots," *IEE Proceedings: Science, Measurement and Technology*, vol. 150, no. 5, pp. 218–221, 2003. [Online]. Available: www.scopus.com

[11] "Nanochip develops MEMS-based storage." http://nanochipinc.com/tech.htm, accessed in November 2007.

[12] J. L. Griffin, S. W. Schlosser, G. R. Ganger, and D. F. Nagle, "Modeling and performance of MEMS-based storage devices," in *Proceedings of ACM SIGMETRICS 2000*, Santa Clara, California, 17-21 June, 2000, pp. 56–65. [Online]. Available: http://www.pdl.cmu.edu/PDL-FTP/Storage/sigmetrics2000{_}abs.html

[13] H. Bhaskaran, A. Sebastian, and M. Despont, "Nanoscale PtSi tips for conducting probe technologies," *Nanotechnology, IEEE Transactions on*, vol. 8, no. 1, pp. 128–131, January 2009.

[14] H. S. Bucy, G. R. Ganger, and Contributors, "The DiskSim simulation environment version 3.0," School of Computer Science, Carnegie Mellon University, Reference Manual, January 2003.

[15] "The IOzone Filesystem Benchmark," http://www.iozone.org/.

[16] "Barrier - a synchronization mechanism in parallel computing," http://en.wikipedia.org/wiki/Barrier_(computer_science), accessed in February 2009.

[17] H. Yu, D. Agrawal, and A. El Abbadi, "Declustering two-dimensional datasets over MEMS-based storage," in *Advances in Database Technology – Proceedings of the 9th International Conference on Extending Database Technology (EDBT 2004)*, ser. Lecture Notes in Computer Science, no. 2992. Heraklion, Crete, Greece, 14–18 March: Springer, 2004, pp. 495–512.

[18] 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," in *Proceedings of the 11th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2003)*, Orlando, FL, October 2003, pp. 104–113. [Online]. Available: http://citeseer.ist.psu.edu/hong03zonebased.html

[19] I. Dramaliev and T. Madhyastha, "Optimizing Probe-Based Storage," in *Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST'02)*, San Francisco, CA, USA, 31 March – 2 April 2003, pp. 103–114. [Online]. Available: http://hdl.handle.net/1882/81

[20] L.-P. Chang, "On efficient wear leveling for large-scale flash-memory storage systems," in *SAC'07: Proceedings of the 2007 ACM symposium on Applied computing*, New York, NY, USA, March 2007, pp. 1126–1130.

[21] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," *ACM Computing Survey*, vol. 37, no. 2, pp. 138–163, June 2005.