Using MEMS-based Storage to Boost Disk Performance

Feng Wang
cyclonew@cs.ucsc.edu

Bo Hong
hongbo@cs.ucsc.edu

Scott A. Brandt
sbrandt@cs.ucsc.edu

Darrell D.E. Long
darrell@cs.ucsc.edu

Storage Systems Research Center
Jack Baskin School of Engineering
University of California, Santa Cruz

Abstract

Non-volatile storage technologies such as flash memory, Magnetic RAM (MRAM), and MEMS-based storage are emerging as serious alternatives to disk drives. Among these, MEMS storage is predicted to be the least expensive and highest density, and at about 1 ms access times still considerably faster than hard disk drives. Like the other emerging non-volatile storage technologies, it will be highly suitable for small mobile devices but will, at least initially, be too expensive to replace hard drives entirely. Its non-volatility, dense storage, and high performance still makes it an ideal candidate for the secondary storage subsystem. We examine the use of MEMS storage in the storage hierarchy and show that using a technique called MEMS Caching Disk*, we can achieve 30–49% of the pure MEMS storage performance by using only a small amount (3% of the disk capacity) of MEMS storage in conjunction with a standard hard drive. The resulting system is ideally suited for commercial packaging with a small MEMS device included as part of a standard disk controller or paired with a disk.*

1. Introduction

Magnetic disks have dominated secondary storage for decades. A new class of secondary storage devices based on microelectromechanical systems (MEMS) is a promising non-volatile secondary storage technology currently being developed [3, 24, 26]. With fundamentally different underlying architectures, MEMS-based storage promises seek times ten times faster than hard disks, storage densities ten times greater, and power consumption one to two orders of magnitude lower. It is projected to provide several to tens of gigabytes of non-volatile storage in a single chip as small as a dime, with low entry cost, high resistance to shock, and potentially high embedded computing power. It is also expected to be more reliable than hard disks thanks to its architectures, miniature structures, and manufacture processes [4, 9, 21]. For all of these reasons, MEMS-based storage is an appealing next-generation storage technology.

Although the best system performance can be achieved by simply replacing disks with MEMS-based storage devices, the cost and capacity issues prevent it from being used in computer systems. It is predicted that MEMS is 5–10 times more expensive than disks. Its capacity is initially limited to 1–10 GB per media surface due to the performance and manufacture considerations. Therefore, disks will still be the dominant secondary storage in the foreseeable future thanks to their capacities and economy. Fortunately, MEMS-based storage shares enough similar characteristics with disks, such as non-volatility and block-level data accesses, while providing much better performance than disks. It is possible to incorporate MEMS into the current storage system hierarchy to make the system as fast as MEMS and as large and cheap as disks.

MEMS-based storage has been used to improve performance and cost/performance in the HP hybrid MEMS/disk RAID systems, proposed by Uysal *et al.* [25]. In this work, MEMS replaces half of the disks in RAID 1/0 and one copy of replicate data is stored in MEMS. Based on data access patterns, requests are serviced by the most suitable devices to leverage fast access of MEMS and high bandwidth of disk. Our work differs in two aspects. First of all, our goal on storage architectures is to provide a single "virtual" storage device with the performance of MEMS and the cost and capacity of disk, which can be easily adopted in every kind of storage systems, instead of just RAID 1/0. Secondly, our approach is using MEMS as another layer in the storage hierarchy, instead of disk replacement, so that we can mask the relatively large disk access latency by MEMS while still taking advantage of the low cost and high bandwidth of disk. Ultimately, we believe that our work complements theirs and could even be used in conjunction with their techniques.

We explore two alternative hybrid MEMS/disk subsystem architectures: *MEMS Write Buffer* (MWB) and *MEMS Caching Disk* (MCD). In MEMS Write Buffer, all written data are first staged on MEMS, organized as logs. Data logs are flushed to the disk during disk idle times or when necessary. The non-volatility of MEMS guarantees the consistency of file systems. Our simulations show that it can improve the performance of a disk-only system by up to 15 times under write dominant workloads.

MEMS Write Buffer is a workload-specific technique. It works well under write intensive workloads but obtains fewer benefits under read intensive workloads. MEMS Caching Disk addresses this problem and can achieve good system performance under various workloads. In MCD, MEMS acts as the front-end data store of the disk. All requests are ser-
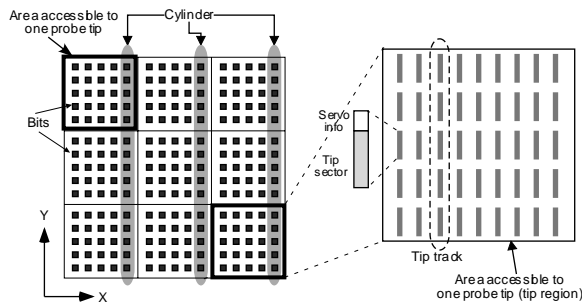
Figure 1. Data layout on a MEMS device.

Per-sled capacity	3.2 GB
Average seek time	0.55 ms
Maximum seek time	0.81 ms
Maximum concurrent tips	1280
Maximum throughput	89.6 MB/s

viced by MEMS. MEMS and disk exchange data in large data chunks (segments) when necessary. Our simulations show that it can significantly improve the performance of a disk-only system by 5.6–24 times under various workloads. MCD can provide 30–49% of the performance achieved by a MEMS-only system by using only a small fraction of MEMS (3% of the total storage capacity).

In this research, MEMS is treated as a small, fast, non-volatile storage device and shares the same interface with the disk. Our use of MEMS in the storage hierarchy is highly compatible with existing hardwares. We envision the MEMS device residing either on the disk controller or packaged with the disk itself. In either case, the relatively small amount of MEMS storage will have a relatively small impact on the system cost, but can provide significant improvements in storage subsystem performance.

2. MEMS-based Storage

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

Figure 1 illustrates the low level data layout of a MEMS storage device. The media sled is logically broken into non-overlapping *tip regions*, defined by the area that is accessible by a single tip, approximately 2500 by 2500 bits in size. It is limited by the maximum dimension of the sled movement. Each tip in the MEMS device can only read data in its own tip region. The smallest unit of data in a MEMS storage device is called a *tip sector*. Each tip sector, identified by the tuple $\langle x, y, tip \rangle$, has its own servo information for positioning. The set of bits accessible to simultaneously active tips with the same *x* coordinate is called a *tip track*, and the set of all bits (under all tips) with the same *x* coordinate is referred to as a *cylinder*. Also, a set of concurrently accessible tip sectors is

grouped as a *logical sector*. For faster access, logical blocks can be striped across logical sectors.

Table 1 summarizes the physical parameters of MEMS-based storage used in our research, based on the predicted characteristics of the second generation MEMS-based storage devices [21]. While the exact performance numbers depend upon the details of that specification, the techniques themselves do not.

3. Related Work

MEMS-based storage [3, 15, 24, 26] is an alternative secondary storage technology currently being developed. Recently, there has been interests in modeling the behavior of MEMS storage devices [8, 10, 13]. Parameterized MEMS performance prediction models [6, 22] were also proposed to narrow the design space of MEMS-based storage.

Using MEMS-based storage to improve storage system performance has been studied in several researches. Simply using MEMS storage as disk replacement can improve the overall application run-time by 1.8–4.8 and the I/O response time by 4–110; using MEMS as a non-volatile disk cache can improve I/O response times by 3.5 [9, 21]. Schlosser and Ganger [20] further concluded that today's storage interfaces and abstractions were also suitable for MEMS devices. The hybrid MEMS/disk array architecures [25], in which one copy of replicate data is stored in MEMS storage to leverage fast accesses of MEMS and high bandwidth of disks, can achieve most of the performance benefits of MEMS arrays and their cost/performance ratios are better than disk arrays by 4–26.

The small write problem has been intensively studied. Write cache, typically non-volatile RAM (NVRAM), can substantially reduce write traffic to disk and the perceived delays for writes [19, 23]. However, the high cost of NVRAM limits its amount, resulting in low hit ratios in high-end disk arrays [27]. Disk Caching Disk (DCD) [11], which is architecturally similar to our MEMS Write Buffer, uses a small log disk to cache writes. LFS [17] employs large memory buffers to collect small dirty data and write them to disk in large sequential requests.

MEMS Caching Disk (MCD) uses the MEMS device as a fully associative, write-back cache for the disk. Because the access times of disks are in milliseconds, MCD can take advantage of computationally-expensive but effective cache replacement algorithms, such as Least Frequent Used with Dynamic Aging (LFUDA) [2], Segment LRU (S-LRU) [12], and Adaptive Replacement Caching (ARC) [14].
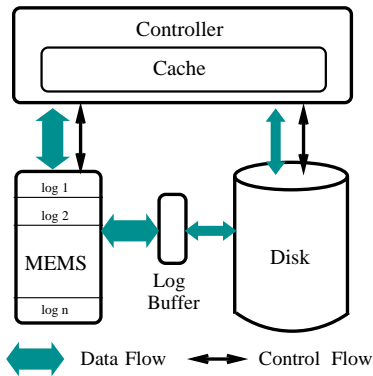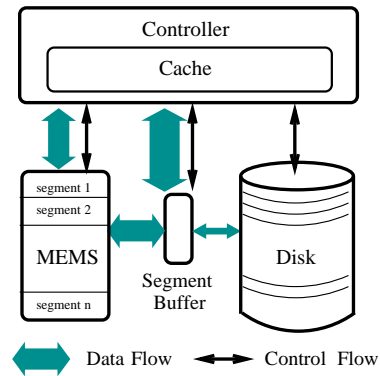
Figure 2. MEMS Write Buffer.

Figure 3. MEMS Caching Disk.

4. MEMS/Disk Storage Subsystem Architectures

The best system performance can be simply achieved by using MEMS-based storage as disk replacement thanks to its superior performance. On the other hand, disk has its strengths over MEMS in terms of capacity and cost per byte. Thus, we expect that disk will still play important roles in secondary storage in the foreseeable future. We propose to integrate MEMS-based storage into the current disk-based storage hierarchy to leverage the high bandwidth, high capacity, and low cost of disks and the high bandwidth and low latency of MEMS so that the hybrid systems can be roughly as fast as MEMS and as large and cheap as disk. The integration of MEMS and disk is done at or below the controller level. Physically, the MEMS device may reside on the controller or as part of the disk packaging. The implementation details are hidden by the controller or device interface and from the operating system point of view this integrated device appears to be nothing more than a fast disk.

4.1. Using MEMS as a Disk Write Buffer

The small write problem plagues storage system performance [5, 17, 19]. In *MEMS Write Buffer* (MWB), a fast MEMS device acts as a large non-volatile write buffer for the disk. All write requests are appended to MEMS as logs and reads to recently-written data can be also serviced by MEMS. A data lookup table maintains data mapping information from MEMS to disk, which is also duplicated in MEMS log headers to facilitate crash recovery. Figure 2 shows the MEMS Write Buffer architecture. A non-volatile log buffer stands between MEMS and disk to match transfer bandwidths of MEMS and disk.

MEMS logs are flushed to disk in background when the MEMS space is heavily utilized. MWB organizes MEMS logs as a circular FIFO queue so the earliest-written logs are cleaned first. During clean operations, MWB generates disk write requests, which can be further concatenated into larger ones if possible, according to the validity and disk locations of data in one or more MEMS logs.

MWB can significantly reduce disk traffic because MEMS is large enough to exploit spatial localities in data accesses and eliminate unnecessary overwrites. MWB stages bursty write activities and amortizes them to disk idle periods thus disk bandwidth can be better utilized.

4.2. MEMS Caching Disk

MEMS Write Buffer is optimized for write intensive workloads so it can only provide marginal performance improvement for reads. *MEMS Caching Disk* (MCD) addresses this problem by using MEMS as a fully associative, write-back cache for the disk. In MCD all requests are serviced by MEMS. The disk space is partitioned into *segments*, which are mapped to MEMS segments when necessary. Data exchanges between MEMS and disk are in segments. As in MWB, data mapping information from MEMS to disk is maintained by a table and is also duplicated in MEMS segment headers. Figure 3 shows the MEMS Caching Disk architecture. The non-volatile speeding-matching segment buffer provides optimization oppotunities for MCD, which will be described later.

Data accesses tend to have temporal and spatial localities [16, 19] and the amount of data accessed during a period tends to be relatively small compared to the underlying storage capacity [18]. MEMS Caching Disk can hold a significant portion of the working set thus effectively reduce disk traffic thanks to the relatively large capacity of MEMS. Exchanging data between MEMS and disk in large segments can better utilize disk bandwidth and implicitly prefetch sequentially accessed data. All of these improve system performance.

The performance of MCD can be improved by leveraging the non-volitile segment buffer between MEMS and disk to reduce unnecessary steps from the critical data paths and relaxing the data validity requirement on MEMS, which result in three techniques described below:

4.2.1. Shortcut MEMS Caching Disk buffers data in the segment buffer when it reads data from the disk. Pending requests can be serviced directly by the buffer without waiting for data being written to MEMS. This technique is called *Shortcut*. Physically, Shortcut adds a data path between the segment buffer and the controller. By removing an unnecessary step in the critical data path, Shortcut can improve response times for both reads and writes without extra overheads.

4.2.2. Immediate Report MCD writes evicted dirty MEMS segments to disks before it frees them to service new requests. Actually, MCD can free MEMS segments as soon as dirty data is safely destaged to the non-volatile segment buffer. This technique is called *Immediate Report*. It can

103

Table 2. Workload Statistics.

	TPCD	TPCC	Cello	Hplajw
Year	1997	2002	1999	1999
Duration (hours)	7	2.5	12	2
Num. of requests	238,971	1,246,763	1,088,407	308,788
Percentage of reads	100%	52.1%	28.7%	60.3%
Avg. request size (KB)	39.9	8	7.2	7.6
Working set size (MB)	1,242	1,006	775	376
Logical sequentiality	72.7%	0	7.4%	14.3%

improve system performance when there are free buffer segments available for holding dirty data; otherwise, buffer segment flushing will result in MEMS or disk writes anyway.

4.2.3. Partial Write MCD reads disk segments to MEMS before it can service write requests smaller than the MCD segment size. By carefully tracking the validity of each block in MEMS segments, MCD can write partial segments without reading them from disk first, leaving some MEMS blocks undefined. This technique is called *Partial Write*, which requires a block bit map in each MEMS segment header to keep track of block validity. It can achieve similar write performance as MEMS Write Buffer.

4.2.4. Segment Replacement Policy Two segment replacement policies are implemented in MCD: Least Recently Used (LRU) and Least Frequently Used with Dynamic Aging (LFUDA) [2]. LRU is a commonly-used replacement policy in computer systems. LFUDA considers both frequency and recency in data accesses by using a dynamic aging factor, which is initialized to be zero and updated to be the priority of the most recently evicted object. Whenever an object is accessed, its priority is increased by the current aging factor plus one. LFUDA evicts the object with the least priority when necessary.

5. Experimental Methodology

Because MEMS storage devices are not available today, we implement MEMS Write Buffer and MEMS Caching Disk in DiskSim [7] to evaluate their performance. The default MEMS physical parameters is shown in Table 1. We use the Quantum Atlas 10K disk drive (8.6 GB, 10,025 RPM) as the base disk model, whose average seek times of read/write are 5.7/6.19 ms. The disk model is relatively old and its maximal throughput is about 25 MB/s, which is far below what modern disks can provide. To better evaluate system performance under modern disks, we change its maximal throughput to 50 MB/s while keeping the rest of disk parameters unchanged.

We use workloads traced from different systems to exercise the simulator. Table 2 summarizes the statistics of the workloads. *TPCD* and *TPCC* are two *TPC* benchmark workloads, representing workloads for on-line decision support and on-line transaction processing applications, respectively. *Cello* and *hplajw* are a news server and a user workloads from

Hewlett-Packard Laboratories, respectively. The logical sequentiality is the percentage of requests that are at adjacent disk addresses or addresses spaced by the file system interleave factor. The working set size is the size of the uniquely accessed data during the tracing period.

6. Results

Because the capacity of Atlas 10K is 8.6 GB, we set the default MEMS size to be 256 MB, which corresponds to 3% of the disk capacity. The controller has 4 MB cache by default and employs the write-back policy. The non-volatile speed-matching buffer between MEMS and disk is 2 MB.

6.1. Comparison of Improvement Techniques in MEMS Caching Disk

We studied the performance impacts of different improvement techniques of MEMS Caching Disk (MCD) under various workloads, as shown in Figures 4(a)–4(d). The techniques are Shortcut (SHORTCUT), Immediate Report (IRE-PORT), and Partial Write (PWRITE) (described in Section 4.2). The performance of MCD with all three techniques disabled is labeled as NONE and with all three techniques enabled is labeled as ALL. For each of the experiments, we used a segment size that was the first power of two greater than or equal to the average request size. We discuss the segment size choice in more detail in Section 6.3.

Immediate Report can improve response times for write-dominant workloads, such as *TPCC* (Figure 4(b)) and *cello* (Figure 4(c)) by leveraging asynchronous disk writes. Partial Write can effectively reduce read traffic from disks to MEMS when the workloads are dominated by writes and their sizes are often smaller than the MEMS segment size, such as *cello*: 52% of its write requests are smaller than the 8 KB segment size. Performance improvement by Shortcut heavily depends on the amount of disk read traffic due to user read requests (*TPCD*, Figure 4(a)) and/or internal read requests generated by writing partial MEMS segments (*cello*, Figure 4(c)). The overall performance improvement by these techniques ranges from 14% to 45%.

6.2. Comparison of Segment Replacement Policy

We compared the performance of two segment replacement policies, LRU and LFUDA, in the MEMS Caching Disk architecture. We found that in general LRU performed as well as and, in many cases, better than LFUDA. Based on its simplicity and good performance, we chose LRU to be the default segment replacement policy in the future analysis.

6.3. Performance Impacts of MEMS Sizes and Segment Sizes

The performance of MCD can be affected by three factors: the segment size, the MEMS cache size, and the MEMS cache utilization. A large segment size increases disk bandwidth utilization and facilitates prefetching. However, it increases disk request service time and may reduce MEMS cache utilization if the prefetched data is useless. A large segment size also decreases the number of buffer segments given that the buffer size is fixed, which can have an impact
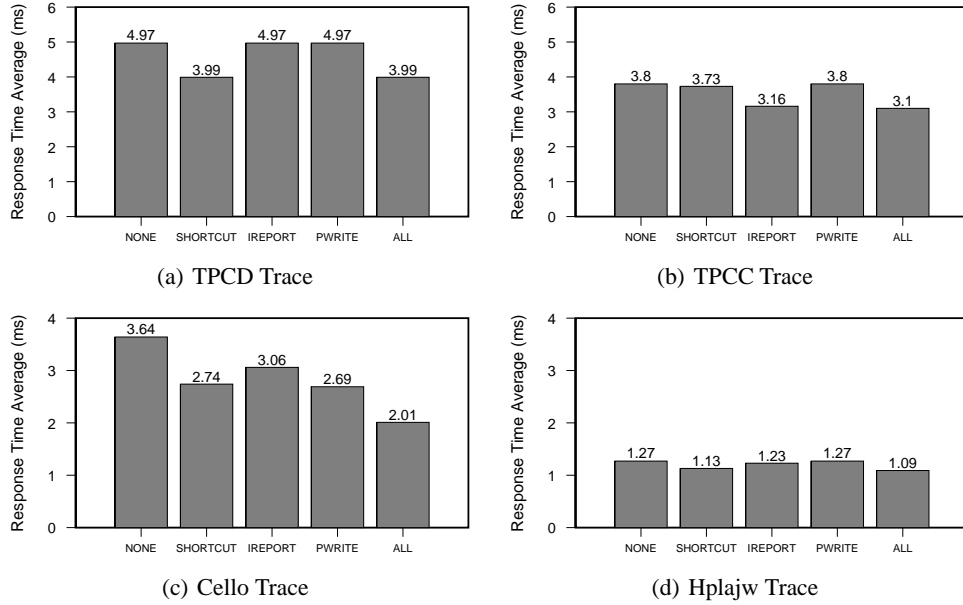
Figure 4. MCD Improvement Techniques with 256 MB MEMS

on the effectiveness of Shortcut and Immediate Report. A large MEMS size can hold a larger portion of the working set thus increases the cache hit ratio. We vary the MEMS cache size from 64 MB to 512 MB, which corresponds to about 0.75–6% of the disk capacity, to reflect the probable future capacity ratio of MEMS to disk. We vary the MCD segment size from 4 KB to 256 KB, doubling it each time. We also configure the MEMS size to be 3 GB to show the upper bound of the MCD performance, which can hold all of the data accessed in the workloads. We enable all improvement techniques.

The *TPCD* workload contains large sequential reads. Simply increasing the MEMS size does not significantly improve system performance unless MEMS can hold all accessed data, as shown in Figure 5(a). However, increasing segment sizes thus reducing commanding overheads and prefetching useful data can effectively utilize disk bandwidth and improve system performance. With 256 MB MEMS, the performance of MCD can be improved by a factor of 1.3 when the segment size is increased from 8 KB to 64 KB.

The *TPCC* and *cello* workloads are dominated by small and random requests. Using segment sizes larger than the average request size of 8 KB severely penalizes system performance because it increases disk request service times and decreases MEMS cache utilization by prefetching useless data. Also Shortcut and Immediate Report become less effective as the increase of the MCD segment size because the number of buffer segments decreases, results in less likelihood to find free buffer segments for optimization. MCD achieves its best performance at the segment size of 8 KB, as shown in Figures 5(b) and 5(c).

Figure 5(d) clearly shows the performance impacts of the MEMS size, the segment size, and their interactions on MCD. *Hplajw* is a workload with both sequential and random data accesses. The spatial locality in data accesses favors large segment sizes, which can prefetch useful data without be-

ing explicitly requested. For instance, at the MEMS size of 64 MB the average response time decreases with the increase of the segment size from 8 KB to 64 KB. However, *hplajw* is not as sequential as *TPCD*. Using even larger segment sizes thus prefetching more data, in turn, impairs system performance by reducing MEMS cache utilization and increasing disk request service times, which is indicated by the increased average response times at segment sizes larger than 64 KB. When the MEMS size is 3 GB (the infinite cache size for *hplajw*), the best segment size is 128 KB because MEMS cache utilization is not an issue any more.

6.4. Overall Performance Comparison

We evaluate the overall performance of MEMS Write Buffer (MWB) and MEMS Caching Disk (MCD). The default MEMS size is 256 MB. We either enable all MCD optimization techniques (MCD ALL) or not (MCD NONE). The segment sizes of MCD are 256 KB, 8 KB, 8 KB, and 64 KB for the *TPCD*, *TPCC*, *cello*, and *hplajw* workloads, respectively. The MWB log size is 128 KB.

We use three performance baselines to calibrate these architectures: the average response times by only using a disk (DISK), only using a large MEMS device (MEMS), and using a disk with 256 MB controller cache (DISK-RAM). By employing a controller cache with the same size of MEMS, we approximate the system performance of using the same amount of non-volatile RAM (NVRAM) instead of MEMS.

Figure 6(a)–6(d) show the average response times of different architectures and configurations under the *TPCD*, *TPCC*, *cello*, and *hplajw* workloads, respectively. In general, using MEMS only achieves the best performance in *TPCD*, *TPCC*, and *cello* thanks to its superior performance. DISK-RAM only performs slightly better than MEMS in *hplajw* because the majority of its working set can be held in nearly-zero-latency RAM. DISK-RAM performs better than MCD by various degrees (6–64%), dependent upon the workloads'
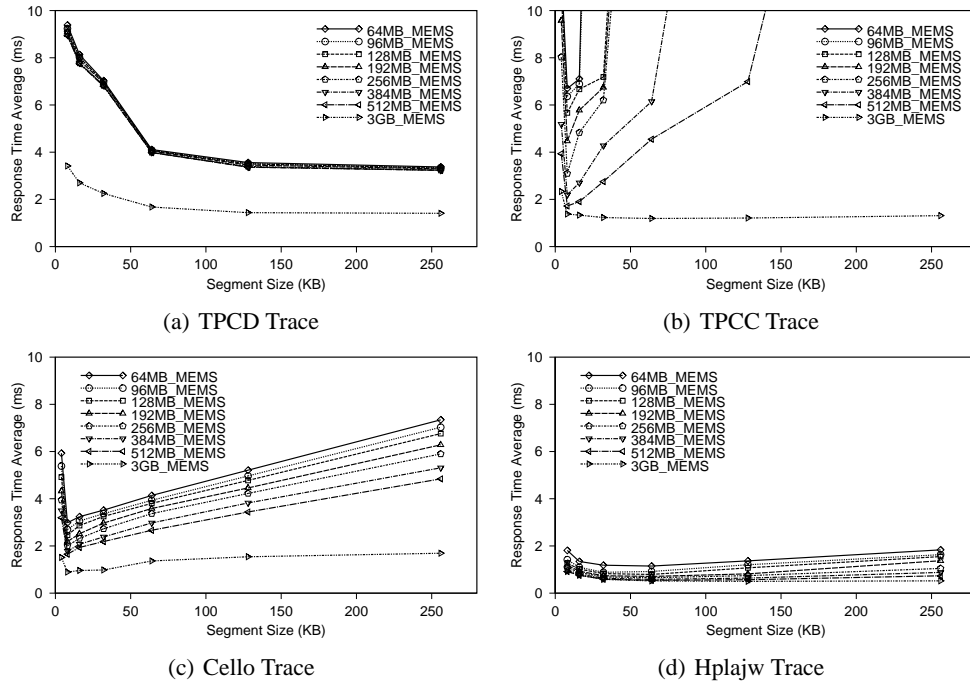
(a) TPCD Trace

(b) TPCC Trace

(c) Cello Trace

(d) Hplajw Trace

Figure 5. Performance of Different MEMS Sizes with Different Segment Sizes

characteristics and working set sizes. In both MCD and DISK-RAM, the large disk access latency is still a dominant factor.

DISK and MWB have the same performance in *TPCD* (Figure 6(a)) because *TPCD* has no writes. DISK has better performance than MCD. In such a highly sequential workload with large requests, the disk positioning time is not a dominant factor in disk service times and disk bandwidth can be fully utilized. The not-scan-resistant property of LRU makes MEMS cache very inefficient. Instead, MCD adds one extra step into the data path, which can decrease system performance by 25%. DISK-RAM only has moderate performance gain due to the same reason.

DISK cannot support the randomly-accessed *TPCC* workload, as shown in Figure 6(b), because the *TPCC* trace was gathered from an array with three disks. Although write activities are substantial (48%) in *TPCC*, MWB cannot support it either. Unlike MCD, which does update-in-place, MWB appends new dirty data to logs, which results in lower MEMS space utilization thus higher disk traffic in such a workload with frequent record updates. MCD significantly reduces disk traffic by holding a substantial fraction of the *TPCC* working set. Both MCD NONE and MCD All can achieve respectable response times of 3.79 ms and 3.09 ms, respectively.

Cello is a write intensive and non-sequential workload. MWB dramatically improves the average response time of DISK by a factor of 14 (Figure 6(c)) because it can significantly reduce disk traffic and increase disk bandwidth utilization by staging dirty data on MEMS and writing them back to disk in sizes as large as possible. In *cello*, 52% of the writes are smaller than the 8 KB MCD segment size. By data logging, MWB also avoids disk read traffic generated by MCD NONE fetching corresponding disk segments
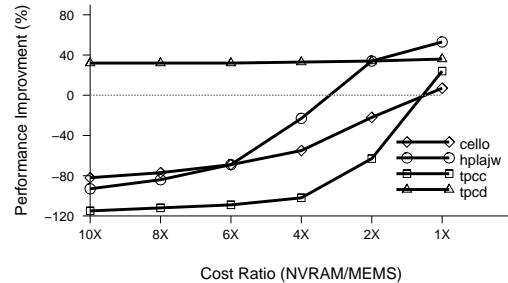

Figure 7. Cost/performance analyses

to MEMS, which is addressed by the technique of Partial Write. Generally MCD has better MEMS space utilization than MWB because it does update-in-place. Thus MCD can hold a larger portion of the workload working set, further reducing traffic to disk. For all these reasons, MEMS ALL performs better than MWB by 57%.

Hplajw is a read-intensive and sequential workload. Thus MWB can not improve system performance as much as MCD. The working set of *hplajw* is relatively small so MCD can hold a large portion of it and achieves response times less than 1 ms.

Although MCD degrades the system performance under the *TPCD* workload and its performance is sensitive to the segment size under *TPCD* and *TPCC*, system performance tuning under such specific workloads can be easy because the workload characteristics are typically known in advance. Controllers can also bypass MEMS under the *TPCD*-like workloads, in which the disk bandwidth is the dominant performance factor. In general purpose file system workloads, such as *cello* and *hplajw*, MCD performs well and robustly.
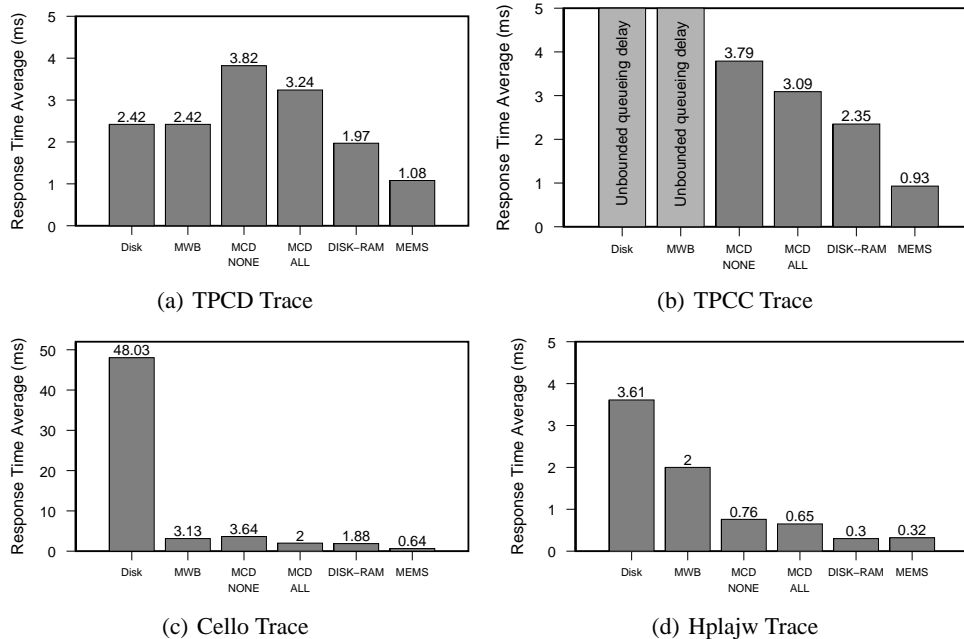
(a) TPCD Trace

(b) TPCC Trace

(c) Cello Trace

(d) Hplajw Trace

Figure 6. Overall performance comparison

6.5. Cost/Performance Analyses

DISK-RAM has better performance than MCD when the sizes of NVRAM and MEMS are the same. However, MEMS is expected to be much cheaper than NVRAM. Figure 7 shows the relative performance of using MEMS and NVRAM of the same cost as disk cache under different NVRAM/MEMS cost ratios, ranging from 1 to 10. The average response time of MCD is the baseline. Using MEMS, instead of NVRAM, as disk cache can achieve better performance in *TPCC*, *cello*, and *hplajw* unless NVRAM can be as cheap as MEMS. The disk access latency is 4–5 orders of magnitude and 10 times higher than the latencies of NVRAM and MEMS, respectively. Thus, the cache hit ratio, which determines the fraction of requests requiring disk accesses, is the dominant performance factor. With cheaper prices thus larger capacities, MEMS can hold a larger fraction of the workload working set than NVRAM, resulting in higher cache hit ratios and better performance. MCD does not work well under *TPCC* and has consistently worse performance than DISK-RAM.

7. Future Work

The performance of MEMS Caching Disk is sensitive, to various degrees, to its segment size and workload characteristics. MCD can maintain multiple "virtual" segment manager, each using a different segment size, to dynamically choose the best one. This technique is similar to Adaptive Caching Using Multiple Experts (ACME) [1]. MCD cannot improve system performance under highly-sequential streaming workloads, such as *TPCD*. We can identify streaming workloads at the controller level and bypass MEMS to minimize its impact on system performance. Techniques that can automatically identify workloads characteristics are desirable.

The performance and cost/performance implications of MEMS storage were studied in disk arrays [25]. The pro-

posed hybrid MEMS/disk arrays leverage MEMS fast accesses and high disk bandwidth by servicing requests from the most suitable devices. Our work shows that it is possible to integrate MEMS and disk into a single device that can approach the performance of MEMS as well as the economy of disk. It is probable that arrays built on such devices can achieve high system performance with relatively low costs. We will investigate its performance and cost/performance in the future.

8. Conclusion

Storage system performance can be significantly improved by incorporating MEMS-based storage into the current disk-based storage hierarchy. We evaluate two techniques, using MEMS as disk write buffer (MWB) or disk cache (MCD). Our simulations show that MCD usually has better performance than MWB. By using a small amount of MEMS (3% of the raw disk capacity) as disk cache, MCD improves system performance by 5.6 and 24 times in the user and news server workloads. It can even well support an on-line transaction workload traced from an array with three modern disks by using a combination of MEMS and a single relatively old disk. With well-configured parameters, MCD can achieve 30–49% of the pure MEMS performance. Thanks to its small physical size and low power consumption, it is suitable to integrate MEMS into controllers or with disks into commercial storage packages with relatively low costs.

References

[1] I. Ari, A. Amer, R. Gramacy, E. L. Miller, S. A. Brandt, and D. D. E. Long. ACME: adaptive caching using multiple experts. In *Proceedings in Informatics*, volume 14, pages 143–158, Paris, France, 2002. Carleton Scientific.

[2] M. Arlitt, L. Cherkasova, J. Dilley, R. Friedrich, and T. Jin. Evaluating content management techniques for web proxy caches. In *Proceedings of the 2nd Workshop on Internet Server Performance (WISP '99)*, Atlanta, Georgia, May 1999. ACM Press.

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

[4] L. R. Carley, G. R. Ganger, and D. F. Nagle. MEMS-based integrated-circuit mass-storage systems. *Communications of the ACM*, 43(11):72–80, Nov. 2000.

[5] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, June 1994.

[6] I. Dramaliev and T. Madhyastha. Optimizing probe-based storage. In *Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST)*, pages 103–114, San Francisco, CA, Mar. 2003. USENIX Association.

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

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

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

[10] 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 '03)*, pages 104–113, Orlando, FL, 2003.

[11] Y. Hu and Q. Yang. DCD—-Disk Caching Disk: A new approach for boosting I/O performance. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 169–178, Philadelphia, PA, May 1996. ACM Press.

[12] R. Karedla, J. Love, and B. G. Wherry. Caching strategies to improve disk performance. *IEEE Computer*, 27(3):38–46, Mar. 1994.

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

[14] N. Megiddo and D. S. Modha. ARC: A self-tuning, low overhead replacement cache. In *Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST)*, pages 115–130, San Francisco, CA, Mar. 2003. USENIX Association.

[15] Nanochip Inc. Preliminary specifications advance information (Document NCM2040699). Nanochip web site, at http://www.nanochip.com/nctmrw2.pdf, 1996–9.

[16] D. Roselli, J. Lorch, and T. Anderson. A comparison of file system workloads. In *Proceedings of the 2000 USENIX Annual Technical Conference*, pages 41–54, San Diego, CA, June 2000. USENIX Association.

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

[18] C. Ruemmler and J. Wilkes. A trace-driven analysis of disk working set sizes. Technical Report HPL-OSR-93-23, Hewlett-Packard Laboratories, Palo Alto, Apr. 1993.

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

[20] S. W. Schlosser and G. R. Ganger. MEMS-based storage devices and standard disk interfaces: A square peg in a round hole? In *Proceedings of the Third USENIX Conference on File and Storage Technologies (FAST)*, pages 87–100, San Francisco, CA, Apr. 2004. USENIX, USENIX Association.

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

[22] M. Sivan-Zimet and T. M. Madhyastha. Workload based modeling of probe-based storage. In *Proceedings of the 2002 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 256–257, Marina Del Rey, CA, June 2002. ACM Press.

[23] J. A. Solworth and C. U. Orji. Write-only disk caches. In H. Garcia-Molina and H. V. Jagadish, editors, *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 123–132, Atlantic City, NJ, May 1990. ACM Press.

[24] J. W. Toigo. Avoiding a data crunch – A decade away: Atomic resolution storage. *Scientific American*, 282(5):58–74, May 2000.

[25] M. Uysal, A. Merchant, and G. A. Alvarez. Using MEMS-based storage in disk arrays. In *Proceedings of the Second USENIX Conference on File and Storage Technologies (FAST)*, pages 89–101, San Francisco, CA, Mar. 2003. USENIX Association.

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

[27] T. M. Wong and J. Wilkes. My cache or yours? making storage more exclusive. In *Proceedings of the 2002 USENIX Annual Technical Conference*, pages 161–175, Monterey, CA, June 2002. USENIX Association.