Using MEMS Device as Disk Write Buffer

Feng Wang* Scott Brandt†

Abstract

Disk is approaching its performance limits in next decade. The performance gap between the main memory and the secondary storage device tends to be large. Research for new storage device is underway, among which the probe-based(MEMS) storage device [1, 2] is promising. It provides much better response time and bandwidth comparing to the modern disk.

In this paper, we exploit the performance improvement using MEMS device as the disk write buffer. Our result shows that several hundred MBs MEMS device will eliminate almost all the raw write traffic to disk. The average response time will be improved by factor of 8 to 10 comparing to modern disk.

1 Introduction

In the computer storage hierarchy, performance has suffered from the significant access, bandwidth and cost gaps between processor, RAM and Disk. Especially the RAM/Disk gap, which was widening to 6 order of magnitude in 1999, continues to widen at about 50% per year [15]. The widening gap will face more worse situation when disk reach its performance limits due to the superparamagnetic effect.

To allivate the impact of the widen performance gap, people employ the memory buffer. As the main memory become cheaper and cheaper, a large memory buffer is possible. A significant percentage of the raw disk traffic has be absorbed by the buffer. But most of the writes, especially the small meta-data writes have to be sent to disk to guarantee the system consistence and get little benefit from the large memory buffer. Thus the traffic between the disk and main memory is dominated by the small writes [14]. To find a effcient way to reduce the write traffic is a key issue to improve the overall system performance.

Many techniques has been employed to reduce the small write problem. LFS [13] uses large main memory to collect small writes and write them back to disk in a long sequential write. Write performance will be improved signficantly. But the writen data has to be kept in volatile ram for a relative long time, the reliability of the LFS is a big issue. Another way to attack this problem is using the non-volatile ram(NVRAM) [14]. shows that only small amount of the non-volatile ram will greatly reduce the meta-data write traffic. But the high cost of NVRAM limits us to widely use NVRAM as write buffer. With the size of the workload expanding fast, small NVRAM can't exploit the locality of the workload, thus a quite amount of write requests are still exposed to

*Department of Computer Science, University of California Santa Cruz, 1156 High Street, Santa Cruz, CA 95064
†Department of Computer Science, University of California Santa Cruz, 1156 High Street, Santa Cruz, CA 95064

disk. Yiming Hu $et.al$ propose another solution: using a small log disk as the write buffer of the disk(DCD) [8]. Each incoming write request could be served at the small disk by writing down the data in current log. The write response time is greatly decreased because the small log disk will write down the request in current track instead of seeking and writing. During the interval of the coming requests, small disk will write its content back to data disk. Moreover, they use a small RAM to collect the writes in log and write it to log disk whenever the log disk is idle.

As a new emerging technology, MEMS(MicroElectroMechanical System) storage device gives us a promising method to solve the write problem. Generally estimated, mems-based storage device has cost significantly lower than NVRAM and its access times is an order of magnitude faster than conventional disk. Our results show that several hundred MBs MEMS write buffer will elminate almost all the raw write traffic (more than 96%) to the disk; at the same time, around 30% raw read traffic hits the MEMS write buffer. The average response time will drematicly decrease from around 64 ms to around 8 ms.

Section 2 gives brief introduction of MEMS storage device and its performance. Section 3 descripts our design and implementation of the MEMS write buffer for disk. The simulation results could be found in section 4. And the final section summarize the major results of our work as well as mention some of the future works.

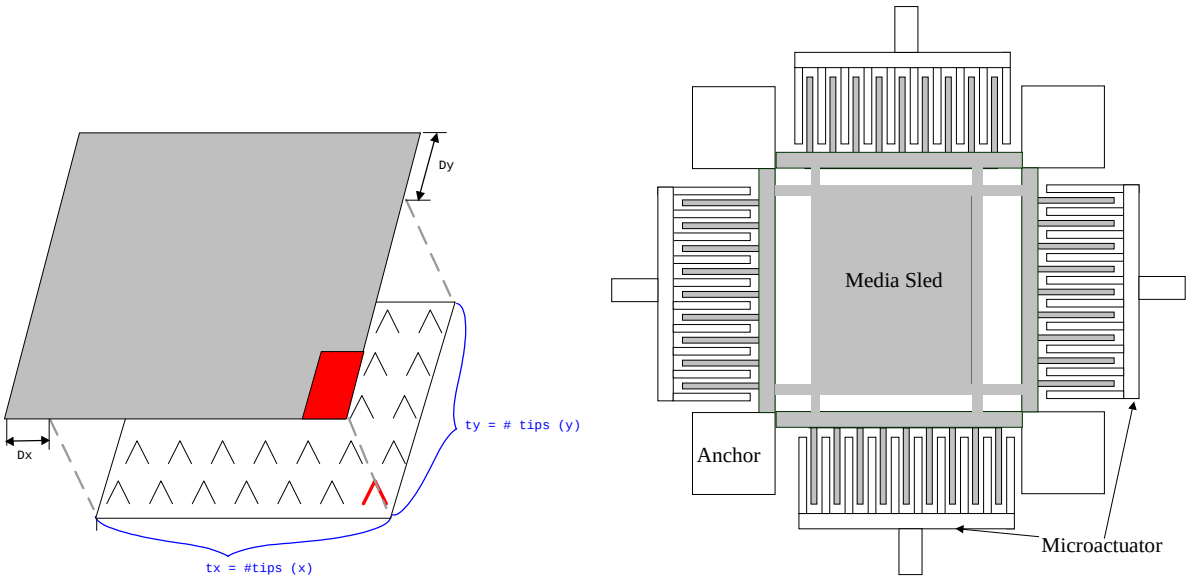
2 Background

For a long time, when we mention the storage device, most of the time we mean the disk. The mechanical parts of the disk limits it performance improvements which makes it hard to keep up with the processor and memory's performance. And in next several years, the superparamagnetic effect will make the slow improvement even impossible. It is time to exploit the new storage device.

Researchers are developing a variety of new devices based on holography [6, 11, 12], probe-based(AFM) storage [3, 9], and probe-based(MEMS) storage [1, 2]. Among those new possibility, probe-based(MEMS) storage is attractive. Because the core component of the probe-based(MEMS) storage device is the MicroElectroMechanical System which position the read/write head to get the data, we will reference it as MEMS device in following text.

MEMS device consists of a substrate of magnetic media suspended over an array of read/write tips, as shown in Fig. 1(a). These tips are etched on the silicon die using the silicon-wafer-fabrication processes. Each of the tips manipulate a small rectangle of the bits on the media sled moving over it. Depending on the design, the transfer rate of each tip could be 100Kb/s to 1Mb/s.

The media sled is suspended above the tip substrate by silicon beams that act as springs, and moved by forces generated by lateral resonant microactuators, as shown in Fig. 1(b). In this figure, the shaded parts move and the unshaded parts are stationary. Electric forces applied to the fingers of the microactuator combs exert electrostatic forces on the sled that cause it to move in the x and y direction, overcoming the forces exerted by the anchors and beams that keep it in place. To serve a request, the media sled first position itself so that the tips could start read/write. This period is what we call *seek time*. Once the sled finishs seeking, data access is accomplished by moving the media at a constant velocity in the Y direction while the data is read or written by the stationary probe tips.



(a) (b)

Figure 1: Probe-based(MEMS) storage device model

	RAM	Future Disk	G1 Model	G2 Model	G3 Model
Bandwidth	$3.2 - 6.4GB/s$	$0.10 - 0.15GB/s$	$0.23GB/s$	$0.8GB/s$	$1.14GB/s$
avg. access time	$2.5 - 5ns$	$3 - 4ms$	$1 - 2ms$	$0.8 - 1.0ms$	$0.4 - 0.7ms$
density	N/A	$0.1GB/cm^2$	$4GB/cm^2$	$6.25GB/cm^2$	$11.1GB/cm^2$

Table 1: MEMS device, disk and RAM performance comparision. G1, G2 and G3 Model are the CMU three generation model of the MEMS device. We predict the possible performance of disk in next five years and reference it as Future Disk.

In this research, we use the CMU three generation model of MEMS device [5]. The general comparision of these models with disk and RAM can be found in Table 1. This table esteem the possible performance of the disk and RAM in next five years and compares them with the CMU MEMS models. The MEMS device's performance is laying between the RAM and the disk. Its bandwidth outperform disk by an order of magnitude, while still an order of magnitude less than RAM. For the access time, MEMS device is still far slow than the RAM because it is bounded by the mechanical components. But it is much better than disk. The high density as well as the IC lithography product process make it possible to get small, high density, on-chip storage device.

3 Design and Implement MEMS Write Buffer

For our experiments, we implement our MEMS write buffer model based on the disksim [4]. Disksim is a well validated disk simulator which includes several fastest commerical disk models current we have. Ganger *et al.*, the author of the disksim, is working on MEMS device and implement the CMU MEMS model in disksim.

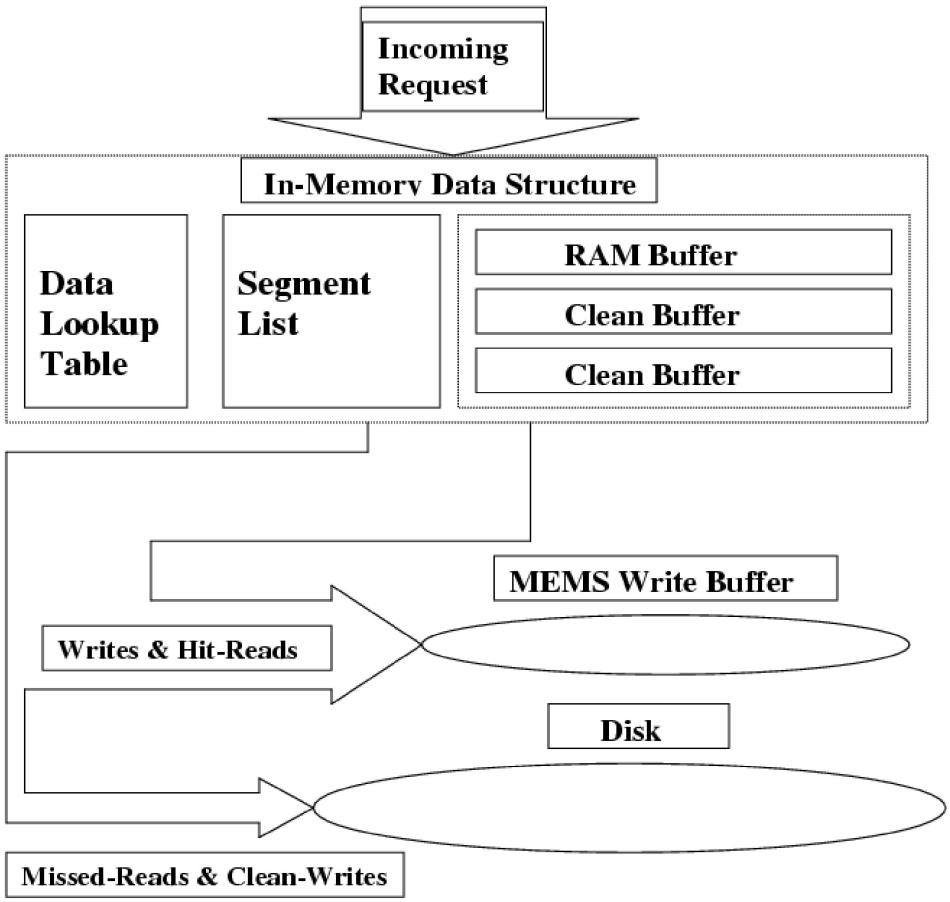


Figure 2: The Structure of the MEMS Write Buffer Driver

Our simulator drives two threads of disksim. One simulates the disk, which we use the Quantum Atlas 10K model. Another implements the MEMS device which is acting as the disk write buffer. To better simulate the write cache, we close the prefecting function in the MEMS device because we focus more on the write performance of the MEMS device and the read requests hits into write buffer is relative small. Thus the prefecting, especially deep prefecting in disksim, will have little benefit for write buffer. In some case, prefecting will have the adverse effect on write performance. For example, when some write requests comes after a read hit in write cache, those writes might not be served right away, which delays the response time. Another important reason to abandon the prefecting is that we use the log to collect the writes to MEMS buffer, which we will descript in following paragraph.

Fig. 2 shows the general structure of the MEMS write buffer dirver which is based on the DCD driver [10]. The core implementation is the In-memory data structure, which is composed of three parts: Data Lookup Table, Segment List, Main Memory Buffer. Because the size of the MEMS cache might scale from several MBs to several GBs, the corresponding data structures in main memory could be as large as several tens of MBs. Thus we need carefully design to make sure that our implementation doesn't use too much main memory.

Cache Management: data in cache is organized into logs. Each log fits into one segment of cache. The size of cache segment is 128 KB, 1 KB of which is used to keep the segment information, called segment head. The rest is the user data and divided into 1KB slots. Each requests will occupy one or several slots. The cache segment size is chosen randomly. Once the write request comes, we first check the cache usage. If it is beyond the high predefined water mark, the current write request will be suspended and the cache cleaning process will start until the cache usage reach the low water mark again. Currently, we clear one segment at one time. If the cache is ready for write, the incoming requests will be appanded to the end of the current log. By doing so, we hope to reduce the seek time because most of requests only need a relative small Y direction seek in MEMS device and avoid the time-comsuming X direction seek. Our simulation shows that under the write-intensive workload, the MEMS cache plus Disk is even working a little better than the pure MEMS device which totally replaces the disk as the secondary storage device.

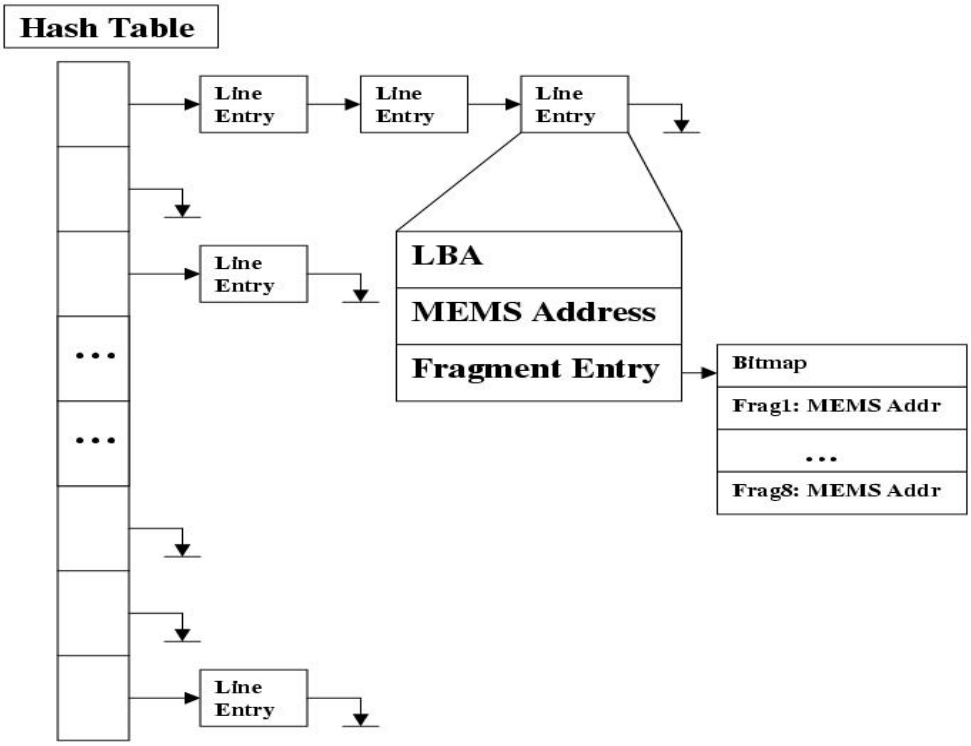


Figure 3: The Data Lookup Table

Data Lookup Table is used to located the data in MEMS cache. In above paragraph, we mention that we use log to collect the write requests. Thus the incoming request could be map to any block in MEMS device. In order to efficiently address those data, we implement the Data Lookup Table as hash table. Another complex factor is the size of the request. In many unix systems, the requests sent to the disk device have variable length and may be overlapped. Thus the entry key to the hash table should be carefully chosen. We can't simply use the request's LBA as the search key, because if this request is overlapping with other request, the searching result might be wrong. For example, if we use the request LBA as the search key, when a new read request comes we use its LBA, S, to index the hashtable; assume that another request which is overlapping

with current request is in cache and its LBA is S', since S is not equal S', we could not found the S in cache. Then we will send this request to disk. Obviously, this will result in wrong data. As shown in Fig. 3, each entry in the hash table contains the information of the blocks in cache. The entry is aligned to 8KB block boundary. All entries in hash table with the same hash value are linked together in a doubly-linked list. Based on the statistics of the disk trace data, most requests will fulfill one entry in hash table. Though still some requests are in small size, but at least they are all aligned to 1K boundary. Thus, for each entry in hash table, there is a fragment entry point. If the entry contains entire 8KB block, then the fragment entry point is NULL. If the entry contains different fragments, then we can find the associated information through the fragment entry point. For each incoming request, we round its LBA to the start LBA of 8K-aligned block, and then use it to index in hash table. The request could be found in MEMS cache in one 8K block, or in MEMS cache with several fragments or partial in MEMS and partial in disk. All these situation adds the complexity of implementation.

Segment List is the list of segment head in main memory. Segment head have two copies, one is linked in the Segment List; another is located on the MEMS cache. Segment head stores the mapping informantion as well as the statistics data of the segment. It is used when clean the segment. Each time we append new data or invaild old data, we should modify the segment head. But it is too costly to update the on-MEMS segment head. Thus, we decide only the in memory segment head needs to be update concurrently. The on-MEMS segment head will be update after the whole segment be writen or after certain among of time. Thus, we can recover the in-memory data structure from the on-MEMS segment heads by replaying them.

Main Memory Buffer has two types: one is the clean buffer; another is the log buffer. Clean buffer is use to clean the cache. During the cache cleaning, several segments will be read into clean buffer in one sequential read from the MEMS cache. Those blocks in those segements will be combined into requests as large as possible and write back to disk. Log buffer is used to collect incoming write request when the cache is busy. In our implement, we didn't make benefit from the log buffer. But we expect that clustering the requests in Log buffer into large request and write it to MEMS cache when it is available might greatly improvement the performance. During the simulation, we notice that a large amount of write requests arrive in very small period of time. If we serve those requests one by one, the service time of each request is almost same: around 1 ms. Thus the total response time of these requests will be large. If we could combine all those waiting requests into one large request, the average response time will be decreased significantly.

Cache Cleaning: currently we only implement the simplest method. When the used size of cache is beyond the high water mark, the clean process starts. The oldest segment will be read into clean buffer, then the valid blocks are combined and write back to disk. As soon as the corresponding data structure has been updated, MEMS cache could serve request again. This method has an obvious shortcome: the request will be blocked until the disk writes is finished and the data structure is updated even thought the MEMS cache is available during the disk write. [7] discuss the NVRAM cache management policies which could be applied to our research too. We are going to improve our model use different cache management policies.

4 Simulation Result

In this section, we will discuss the simulation results using our MEMS write buffer model. We use the two piece of traces to run our simulation. Both traces are got from HP lab. One is collected from Snake server disk 6 in 1992. The model of this disk is HP 97560 and its size is 1.3GB. Due to limited time, I use only one day trace to perform our experiment. During one day period, totally 77372 requests have been recorded, among which 43% is read. The detail description of this traces could be found in [14]. Another trace is collected recently from HP Lab's Cello server, a time sharing server, in 1999. The disk we focus is 17G seagate hard disk. During one day period, totally 1621165 requests have been generated and 30.1% of them are reads.

In our simulation study, we use the Quantum Atlas 10000RPM disk to act as our base disk, and use the CMU G2 model to imitate the MEMS write buffer. We first run the model under snake trace. Snake trace is relative old trace. The average request rate is a little samll than one request per second. In order to study the our model performance, we scale the trace from one request per second to sixteen requests per second.

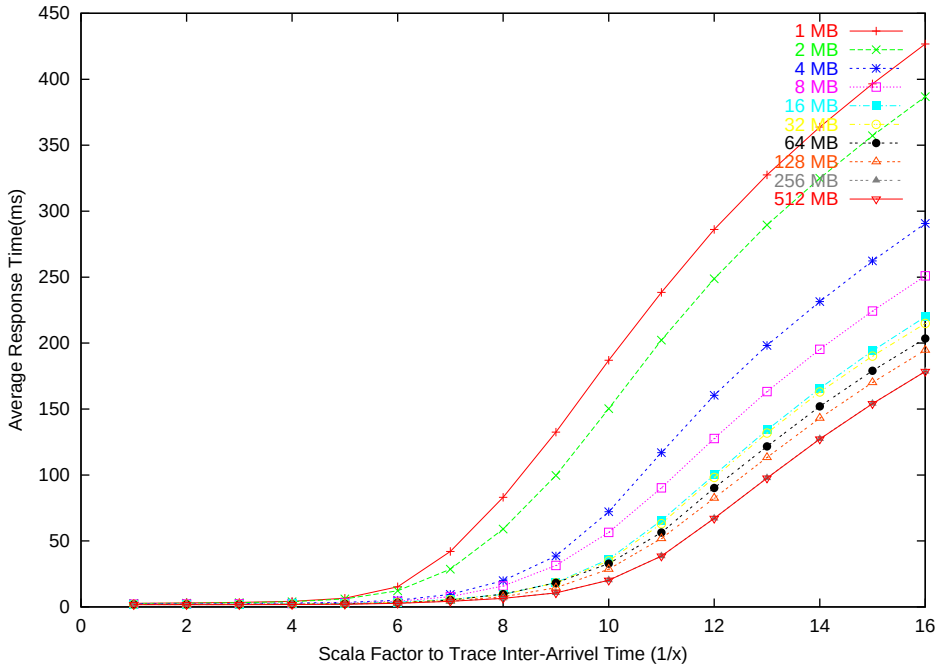


Figure 4: MEMS Write Buffer Model Avg. Response Time. X axis is the scaling factor, which means the trace interarrivel time are divide by the given factor. Y axis is the model average response time. Each curve in the figure represent one configeration of MEMS cache size.

Fig. 5 shows that when the requests of workload become more intensive, the disk fail to give good response time to each request: it even let each request wait for nearly one second when the scaling factor beyond seven. Employing the MEMS write buffer could prevent the fast increasing response time. Comparing to disk, MEMS write buffer model reduce the average response time by factor of nearly ten. The 'pure' MEMS device works even better. It doesn't be affected by the scaling factor. The average response time stay unchanged. The MEMS write buffer model

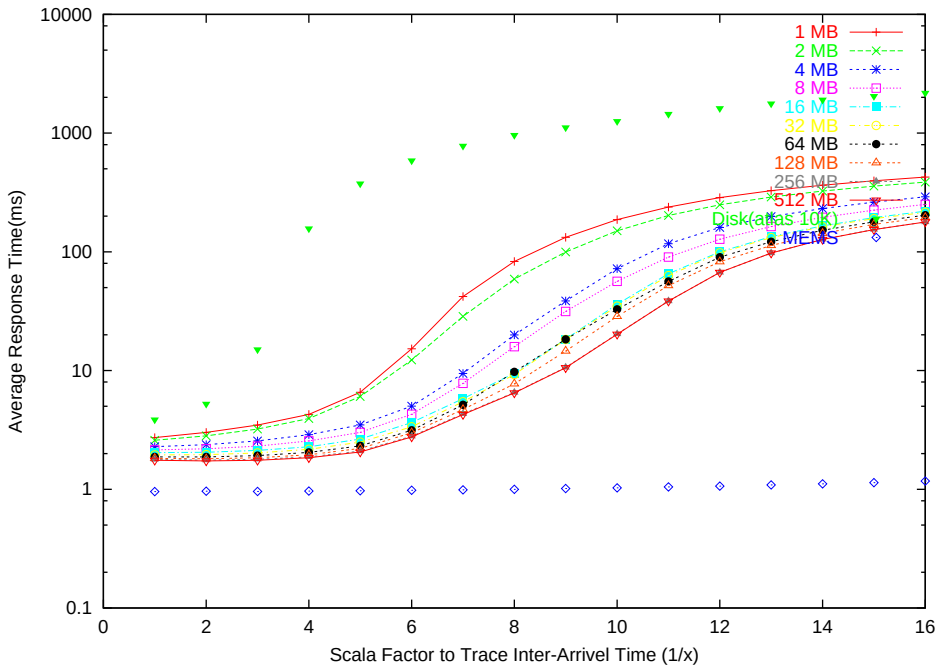


Figure 5: MEMS Write Buffer Model Avg. Response Time. Y axis is logical scale. In this figure, we compare the MEMS write buffer model with the disk and mems device performance. The uppest curve is the Quantum Atlas 10000RPM performance without the write buffer. The lowest curve is the 'pure' MEMS device performance which totally replace the disk.

seems performing not so good regarding the 'pure' MEMS's performance. The most important reason is the reads request which occupy 43% of all requests. Even the MEMS write buffer absorb almost all the write request, still a large portion of requests can't be served by the fast MEMS cache, which counts most of the delay time. Another reason is that the scaling method will make the trace more bursty. Especially we know that the Unix file system like to flush data from cache every 30 seconds. Scaling those traces will result in very bursty workload. Because MEMS device is ten time faster than disk, those scaling bursty has litte effect on it. While disk is suffer a lot from this bursty. For example, assuming the service time of MEMS device and disk are 0.6ms and 6ms, if the interarrive time of requests is 1ms, then the average response time for MEMS device is still 0.6ms while the average response time of disk becomes 21ms. The small data set is also a important reason to explain why MEMS device perform so good. The orginal disk of snake trace is only 1.3G. While our 'pure' MEMS device size is around 3.2GB. Thus only small portion of the MEMS device is exercised which result in relative small seek time.

Fig. 4 shows that increase the cache size will result in better performance. Most of the improvement will be credit to the more cache read hits and decreasing clean writes. But when the cache size reach 64MB, the improvement becomes slow. And the 512MB cache even have the same performance as the 256MB cache.

We use the cello trace to drive our simulation. Fig. 6 reflects the total decreased disk raw traffics. If we didn't use the MEMS write buffer, all the traffics will directly face the disk. By using 16MB MEMS write buffer, the total amount of requests to disk is sharply decreased to 30%

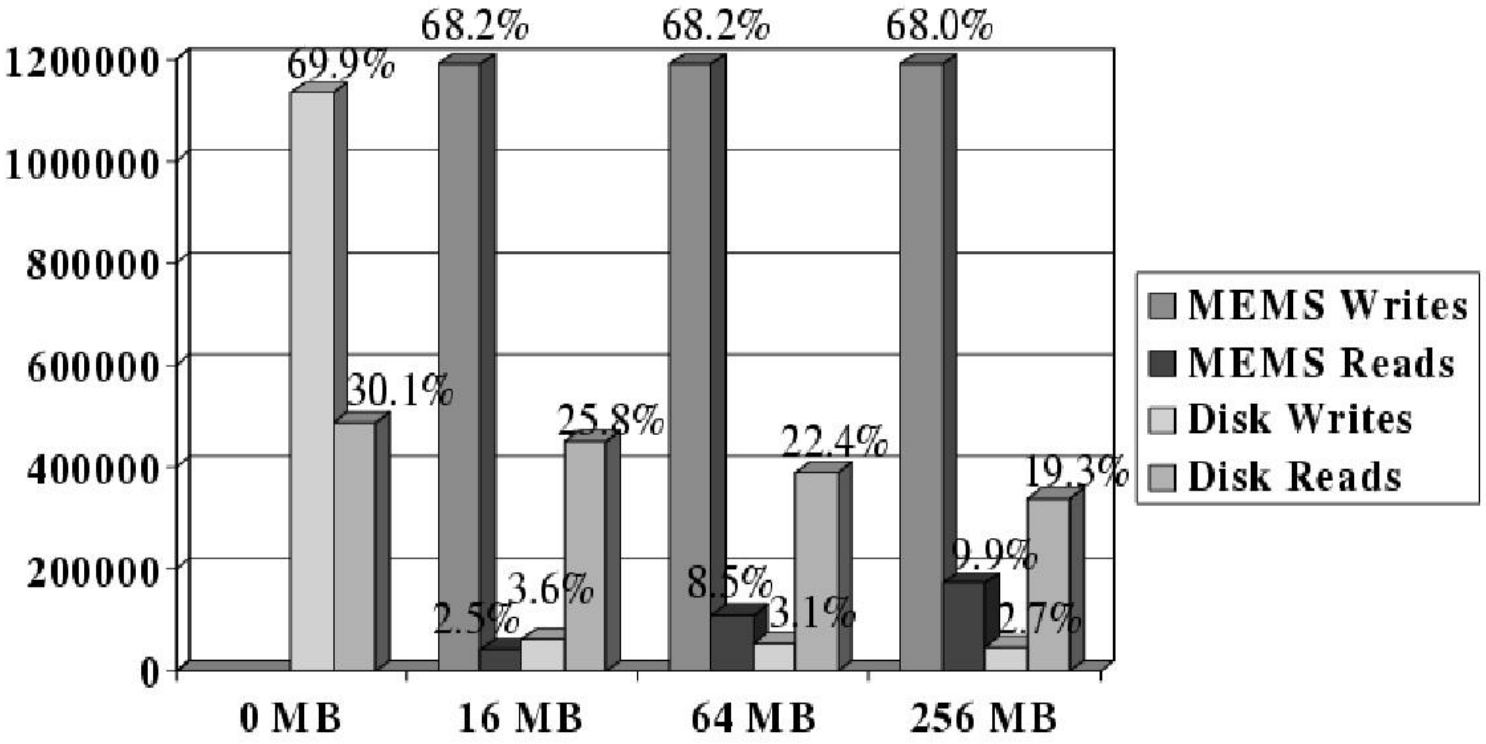


Figure 6: Number of Requests Hits in Each Device. X axis indicates the MEMS cache size. 0MB means we didn't use MEMS write buffer.

of orginal traffic. Larger write buffer, such as 256MB, will even reduce the disk traffic to 20% of orginal traffic. As we can see from the graph, no matter how large the write buffer is, all the write requests are absorb by the buffer and only very small amout of the clean write been generated. Most benefits gain from the large write buffer is the increasing read hit ratio. Larger write buffer makes it possible to exploit more on read locality. More reads could hit a copy in the MEMS write buffer.

Fig. 7 shows the average response time of the MEMS cache, disk and 'pure' MEMS device under the cello trace. Similar to the snake trace result, MEMS cache could improve disk average response time by factor from 6 to 9. One significant difference from the snake trace result is that the MEMS cache performance is close to the 'pure' MEMS device. This is because less bustiness of the cello trace comparing to the scaled snake trace. We also notice that only 30% of raw disk requests are read, which means the potential disk access is lowered. Actually, Fig 6 shows that only 20% of orginal requests still reach disk when we use 256MB MEMS cache. Due to the less busty of cello trace, the average response time of the disk and the MEMS cache are drematicly decreased comparing to the snake trace scaled by factor 16. But the 'pure' MEMS device is greatly increase. After examining the data we collect, we found this is most because the larger data set the cello trace has. Large data set requires some parameters of the MEMS device changed to incorporate

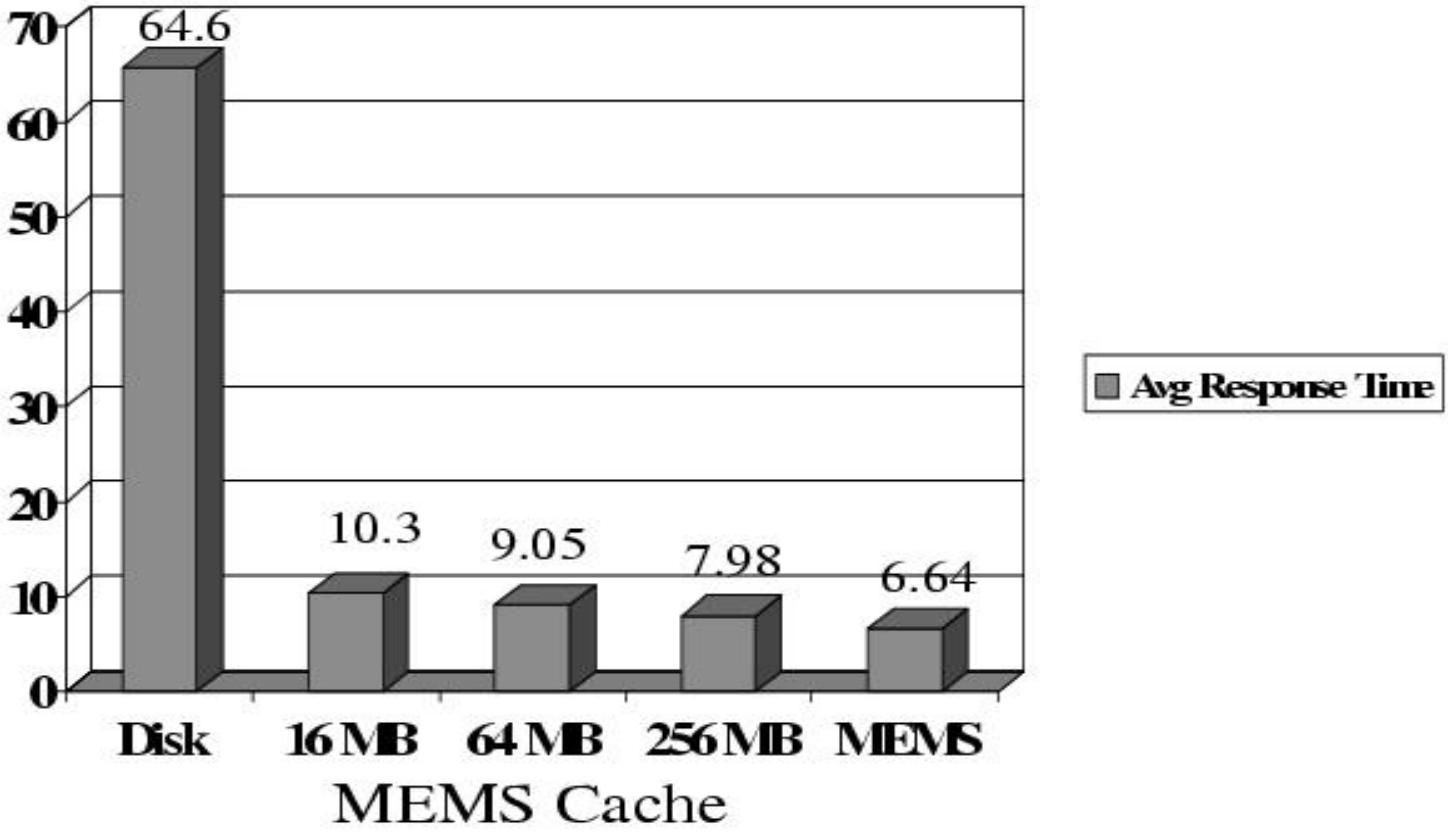


Figure 7: MEMS Write Buffer Model Performance using Cello Traces

with the capacity, which results in performance decrease.

5 Conclusion

From this simulation study, we learn that the MEMS write buffer could significantly improve the disk performance. Large write buffer could absorb nearly 80could be reduced by the factor of ten in the intensive workload.

There are still a lot of improvement could be applied to our model. We are going to implement the different cache management policies to exploit the best performance. And the ”clustering” waiting request is another interesting technique that might improve the performance.

References

[1] C. Brown. Microprobes promises a new memory option. *E.E. Times*, 1998.

[2] L. R. Carley. Center for highly integrated information processing and storage systems, carnegie mellon university. www.chips.ece.cmu.edu, 1999.

[3] M. Despont, J.Brugger, U. Drechsler, U.Durig, W. Haberle, M. Lutwyche, H. Rothuzen, R. Stutz, R. Widmer, H. Rohrer, G. Binnig, and P. Vettiger. VLSI-NEMS chips for AFM data storage. In *Technical Digest. IEEE International MEMS 99 Conference. Twelfth IEEE International Conference on Micro Electro Mechanical Systems*, 1999.

[4] G. Ganger, B. Worthington, Y. Patt, D. simulation, and e Report. Department of electrical engineering and computer science, 1998.

[5] 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*, pages 56–65, June 2000.

[6] G.T.Sincerbox and ed. Selected papers on holographic storage, vol.ms 95 of SPIE milestone series. In *Internal Society for Optical Engineering*, 1994.

[7] T. R. Haining and D. D. E. Long. Management policies for non-volatile write caches. In *Proceedings of the IEEE International Performance, Computing, and Communications Conference (IPCCC)*, Phoenix, Feb. 1999. IEEE.

[8] Y. Hu and Q. Yang. Dcd—disk caching disk: A new approach for boosting i/o performance, 1996.

[9] H. J. Mamin, B. D. Terris, L. S. Fan, S. Hoen, R. C. Barrett, and D. Rugar. High-density data storage using proximal probe techniques. In *IBM Journal of Research and Development*, 1995.

[10] T. Nightingale, Y. Hu, and Q. Yang. The design and implementation of a DCD device driver for Unix. pages 295–307.

[11] D. Psaltis and G.W.Burr. Holographic data storage. In *IEEE Computer*, 1998.

[12] S. Redfield and J. Willenbring. Holostore technology for higher levels of memory hierarchy. In *Eleventh IEEE Symposium on Mass Storage Systems*, 1991.

[13] 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.

[14] C. Ruemmler and J. Wilkes. Unix disk access patterns. In *Proc. of the Winter'93 USENIX Conference*, pages 405–420, 1993.

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