# Building MEMS-Based Storage Systems for Streaming Media

RAJU RANGASWAMI
Florida International University
ZORAN DIMITRIJEVIĆ and EDWARD CHANG
Google, Inc.
and
KLAUS SCHAUSER
University of California, Santa Barbara

The performance of streaming media servers has been limited by the dual requirements of high disk throughput (to service more clients simultaneously) and low memory use (to decrease system cost). To achieve high disk throughput, disk drives must be accessed with large IOs to amortize disk access overhead. Large IOs imply an increased requirement of expensive DRAM, and, consequently, greater overall system cost. MEMS-based storage, an emerging storage technology, is predicted to offer a price-performance point between those of DRAM and disk drives. In this study, we propose storage architectures that use the relatively inexpensive MEMS-based storage devices as an intermediate layer (between DRAM and disk drives) for temporarily staging large disk IOs at a significantly lower cost. We present data layout mechanisms and synchronized IO scheduling algorithms for the real-time storage and retrieval of streaming data within such an augmented storage system. Analytical evaluation suggests that MEMS-augmented storage hierarchies can reduce the cost and improve the throughput of streaming servers significantly.

Categories and Subject Descriptors: C.0.a [**Computer System Organization**]: General—*System architectures*; C.3.r [**Special-Purpose and Application-Based Systems**]: *Real-time and embedded systems*; D.4.2 [**Operating Systems**]: Storage Management

General Terms: Design, Algorithms, Performance

This work is sponsored by NSF grants EIA-0080134 and IIS-0534530 and DoE grant ER25739.
This is an extended version of the paper titled MEMS-Based Disk Buffer for Streaming Media Servers, by Raju Rangaswami, Zoran Dimitrijević, Edward Chang, and Kalus Schauser, which appeared in Proceedings of the IEEE International Conferences on Data Engineering, March 2003, 619–630. ©2003 IEEE.
Author's addresses: R. Rangaswami, School of Computing and Information Sciences, Florida International University, 11200 S.W. 8th Street, Miami FL 33199; email: raju@cis.fiu.edu; Z. Dimitrijević and E. Chang, Google, Inc., 1600 Amphitheater Pkwy, Mountain View CA 94043; email: zorand@gmail.com, edchang@google.com; K. Schauser, Department of Computer Science, University of California, Santa Barbara CA 93106.

Additional Key Words and Phrases: Storage architecture, MEMS-based storage, multidisk storage, I/O scheduling, streaming media

**ACM Reference Format:**

Rangaswami, R., Dimitrijević, Z., Chang, E., and Schauser, K. 2007. Building MEMS-based storage systems for streaming media. ACM Trans. Storage 3, 2, Article 6 (June 2007), 31 pages. DOI = 10.1145/1242520.1242523 http://doi.acm.org/ 10.1145/1242520.1242523

## 1. INTRODUCTION

The performance gap in the storage hierarchy between disk drives and DRAM is increasing every year. Although disk capacities and data transfer rates are improving at 60% and 40% per year, respectively, disk access times are improving at less than 10% per year [Grochowski and Halem 2003; Thompson and Best 2000]. For continuous retrieval of data in streaming servers, the long access times necessitate accessing the disk drive in large chunks to utilize the available disk bandwidth. Larger IOs imply that larger (expensive) buffers are required to temporarily store data before it is consumed by a client. This requirement increases the overall cost of a streaming media server, making the storage system a key cost and performance bottleneck.

An alternative approach to improving storage system performance is incorporating new and emerging storage technologies into existing storage systems. MEMS-based storage is one such new technology that promises to bridge the performance gap between magnetic disk drives and DRAM. MEMS devices are predicted to be an order of magnitude cheaper than DRAM, while offering an order of magnitude faster access times than disk drives. These devices offer a unique low-cost solution for streaming applications. Recent research [Carley et al. 2000; Vettiger et al. 2000; Schlosser et al. 2000; Hong et al. 2006; Uysal et al. 2003; Rangaswami et al. 2003; Yu et al. 2003; Schlosser and Ganger 2004] has served as the initial demonstration of the potential IO performance improvement due to these devices for a varied workload.

In this study, we evaluate the integration of MEMS-based storage devices into existing disk-based storage systems for the class of streaming media server systems. This is enabled by introducing the MEMS storage as an additional persistent layer between the DRAM and the disk drive. Since DRAM buffering is a significant fraction of the total cost of a streaming server, such a bridge buffer has the potential to substantially reduce the per-stream cost of the streaming server. The key idea here is that by buffering active portions of disk-resident streaming data on MEMS storage, the subsequent movement of data from the intermediate MEMS storage into DRAM can be performed more efficiently, thereby incurring significantly lower DRAM requirement.

At first glance, placing MEMS storage between the DRAM and the disk-drive seem straightforward enough. However, the real-time IO requirement in streaming servers and the mismatch between the access times and the transfer rates of disk, MEMS storage, and DRAM make certain MEMS-augmented storage configurations counter-productive. Further, storage configurations involving multiple MEMS devices and multiple disk drives make architecture

decisions, resource allocation, and IO scheduling complex. We conduct an analytical study of such configurations by proposing architectures for single and multiple disk storage systems that also integrate one or more MEMS-based storage devices. To identify the potentially useful storage configurations, we develop new theoretical models for analyzing from a cost performance standpoint real-time data streaming using an augmented storage hierarchy.

The architectural abstractions as well as the cost performance analytical models we present are generic enough to be applied to other augmented storage hierarchies where devices have relatively similar characteristics. For instance, a hierarchy composed of DRAM, fast disk drives (high-end SCSI), and slow disk drives (low-end IDE) as suggested in Anderson et al. [2003] can be easily modeled with the framework presented here. Another example is a hierarchy of DRAM, compact flash storage, and disk drives. Although our abstractions, models, and analysis are generic, our analysis is limited in scope since it does not take into account additional issues that affect the total cost of ownership of such systems that also include an increased maintenance cost owing to the added complexity of the storage hierarchy.

We introduced the use of MEMS-storage for buffering real-time streaming data in single-drive systems in an earlier study [Rangaswami et al. 2003]. In this study, we extend our previous work to address the general problem of managing multiple disks and MEMS-based storage banks and address several issues in building a real-time streaming server system. In particular, we make the following contributions.

(1) We propose two possible extended storage architectures for integrating MEMS-based storage devices into existing disk-based storage systems.
(2) We design data placement and real-time IO scheduling policies for streaming media data based on logical abstractions that model augmented storage hierarchies.
(3) We propose analytical models that allow reasoning about the cost performance trade-offs of augmented storage hierarchies and demonstrate their effectiveness in increasing the performance and decreasing the cost of streaming media servers.

We organize the rest of this article as follows. In Section 2, we introduce key storage technologies and propose new storage architectures augmented with an additional MEMS storage layer. In Section 3, we address the problem of managing heterogeneous storage and data with a focus on time-sensitive real-time data. In Section 4, we present a quantitative analysis for systems supporting real-time applications using MEMS devices. In Section 5, we conduct an analytical evaluation of the MEMS-disk streaming storage system. Section 6 presents related research. In Section 7, we make concluding remarks and suggest directions for future work.

## 2. HETEROGENEOUS STREAMING STORAGE ARCHITECTURE

In this section, we first present an overview of the storage device hierarchy and introduce MEMS-based storage. We then present potential modifications to the existing storage architecture that can incorporate MEMS-based storage.

Table I. Storage Media Characteristics

| Year | Characteristic | DRAM | MEMS | Disk |
|---|---|---|---|---|
| 2006 | Capacity [GB] | 2 | n/a | 250 |
| | Access time [ms] | 0.00005 | n/a | 8 |
| | Bandwidth [MB/s] | 1600–10,750 | n/a | 100 |
| | Cost/GB | $100 | n/a | $0.5 |
| | Cost/device | $50–$200 | n/a | $100–$300 |
| 2008 | Capacity [GB] | 5 | 10 | 1,000 |
| | Access time [ms] | 0.00003 | 0.4–1 | 0.75–7 |
| | Bandwidth [MB/s] | 10,000–20,000 | 320 | 170–300 |
| | Cost/GB | $20 | $1 | $0.2 |
| | Cost/device | $50–$200 | $10 | $100–$300 |

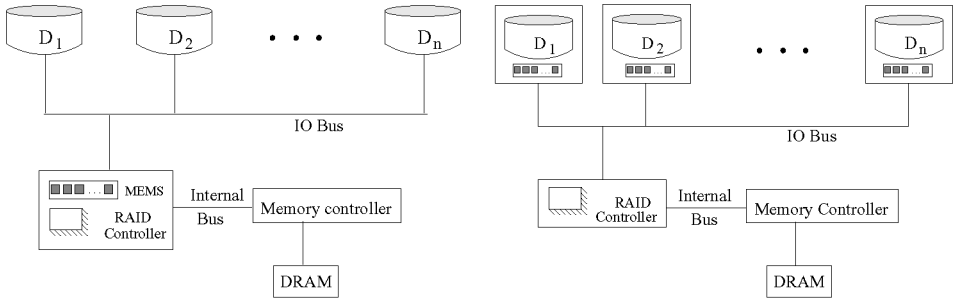
## 2.1 Overview of Storage Devices

Storage systems are organized in a hierarchy ranging from the on-board L1/L2/L3 caches, the main-memory, the storage controller (e.g. RAID controller) cache, the disk cache, to permanent disk storage. To the existing hierarchy, MEMS-based storage adds a new level, and based on its cost performance characteristics, places itself between DRAM and disk storage.

For simplicity, we focus on key storage levels in the hierarchy that gain significance when dealing with high-volume data. Table I summarizes the cost and performance characteristics of different storage media for the year 2006 and the predicted values for the year 2008. A recent press release from Nanochip Inc. [2006], one of the leading industry manufacturers of MEMS storage technology, reinforces the commercial viability of MEMS storage in the next few years. The DRAM predictions are based on Rambus Inc. [2006]; the MEMS devices are based on predictions in Schlosser et al. [2000] but include compensation to reflect recent industry progress on the MEMS storage technology; and the disk drive projections are based on Grochowski and Halem [2003], and Thompson and Best [2000].

*Disk Drives.* Typically, most storage media are optimized for sequential access. However, since magnetic disks have much longer access times than MEMS-based storage devices and DRAM, the effective throughput is governed by the amortized disk access overhead. These devices have to be accessed in large chunks to mask the access overheads. To deal with real-time streaming data, scheduling gains additional importance to ensure timeliness of data delivery.

*MEMS-Based Storage.* Researchers at the Carnegie Mellon University [Carley et al. 2000] and at IBM [Vettiger et al. 2000] have proposed similar architectures for a MEMS-based storage device. They both propose MEMS devices which would be fabricated on-chip but would use a spring-mounted media sled as a storage medium. The media sled is placed above a two-dimensional array of micro-electro-mechanical read/write heads (tips). Moving the sled along the Y dimension at a constant velocity enables the tips to concurrently access data stored on the media sled. Using a large number of tips (on the order of thousands) concurrently, such a device can deliver high data throughput. The




(a) Integrated Architecture (b) Partitioned Architecture

Fig. 1. Extended storage hardware architectures, integrated and partitioned.

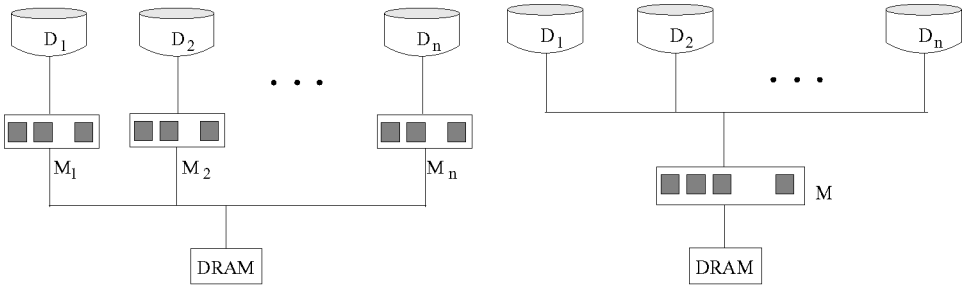
lightweight media sled of the MEMS device can be moved and positioned much faster than bulkier disk servo-mechanisms, thus cutting down access times by an order of magnitude.

Due to its their moderate access latency and moderate cost-per-byte, MEMS-based storage devices are both bandwidth-and space-constrained. While dealing with real-time streaming data, the buffering bandwidth requirement is twice the actual bit rate of each stream (as we shall see in Section 3). This additional bandwidth requirement as well as the real-time delivery constraints requires more complex IO scheduling algorithms at the MEMS devices.

*Main Memory.* The achievable throughput performance from a DRAM device is about two orders of magnitude greater than both disk drives and MEMS-based storage. The maximum DRAM throughput is achieved when data is accessed in sequential chunks, about the size of the largest cache block. These are typically tens to hundreds of bytes. In such a scenario, it is very unlikely that the DRAM bandwidth would become a bottleneck. Rather than its bandwidth, the DRAM is resource-constrained in space. The available DRAM space impacts the performance of the devices which are placed below in the memory hierarchy.

## 2.2 Storage Hardware Architecture

2.2.1 *Low-Level Architecture.* Commodity multidisk systems today typically consist of several disks connected to a RAID controller over an IO bus. The RAID controller issues both operating system generated IOs as well as control IOs required for RAID reconfiguration and management. Typically, the IO bus can support several disk drives without becoming a bottleneck itself. Data read or written on the storage system is transferred over the internal bus to the DRAM via the memory controller. Designing new storage architectures requires that none of the data-path components, including the data buses, become a performance bottleneck. Figure 1(a) presents one possible storage architecture which integrates MEMS devices. In this architecture, the RAID controller has an on-board MEMS bank which it can use to temporarily buffer or cache IO data. This architecture does not increase IO traffic on any of the buses on the




(a) Partitioned abstraction (b) Integrated abstraction

Fig. 2. Logical hardware abstractions corresponding to the hardware architectures in Figure 1.

data path, but may require additional on-board bus bandwidth on the controller to transfer data to/from the MEMS bank.

*Integrated Architecture.* Depending on how MEMS-based storage technology is packaged in the future, it may or may not be feasible to integrate it into the RAID controller. If MEMS-based storage devices are packaged with a SCSI or IDE-like interface, they may have to be independently connected to and accessed on the IO bus. Future RAID controllers may be equipped to manage this additional device or an independent MEMS controller may be required. Although such an architecture is possible (even possibly unavoidable), it generates additional traffic on the IO bus and hence is an undesirable configuration. Such an integrated hardware architecture is depicted in Figure 1(a).

*Partitioned Architecture.* Another future scenario arises if disk manufacturers immediately embrace MEMS storage devices, considering it as an assisting technology rather than a competing one. They could then integrate MEMS storage into disk-drive units to be used as a large disk cache or buffer. This can improve disk performance significantly by allowing large amounts of data to be prefetched or stored over long periods of time (akin to a persistent store) in the on-disk MEMS store. In addition, if an interface is provided to the operating system or the RAID controller to independently access the MEMS store, they could also be used to selectively buffer or cache file data. The partitioned hardware architecture is depicted in Figure 1(b).

2.2.2 *Logical Abstractions.* The hardware architecture configurations for integrating MEMS stores previously presented provide the basis for discussing resource allocation for streams both in terms of space and IO bandwidth. However, the architecture-level is too detailed to allow easy reasoning about performance in a complex time-shared, real-time system such as a streaming media server. We now present logical abstractions of the hardware architecture scenarios just outlined. These abstractions serve as simple models of the underlying storage architecture and simplify designing higher-level data placement and IO scheduling policies. Two natural logical abstractions are evident: the *integrated* and *partitioned* MEMS buffer-cache abstractions. Figure 2 presents these abstractions.

Although the partitioned hardware architecture can only allow the partitioned logical abstraction, the integrated hardware architecture allows for both logical abstractions and can be dynamically configured and tailored to the workload.

## 3. MEMS DISK-STREAMING STORAGE

Based on the characteristics of the streaming data, we can apply simple rules for placing them on storage devices. Streaming data files are usually large and hence must be placed permanently on disk drives for cost effective storage. Due to the superior access characteristics of MEMS storage in comparison to disk drives, accessing them as an intermediate layer (rather than disk drives directly) greatly reduces the DRAM requirement for buffering a large number of streams simultaneous serviced from the storage system. Further, the MEMS device, when used as an intermediate buffer, also provides low-cost buffering for disk IOs and consequently allow the disk drive to operate at higher throughput.

Caching streaming data on MEMS storage (over the medium-term), as opposed to buffering (over the short-term), has been explored in Rangaswami et al. [2003]. We do not discuss caching any further, but instead focus on using MEMS storage as an intermediate buffer when servicing real-time data streams. In this configuration, as data streams are retrieved from the disk drive to meet client requests, the streams go through an intermediate buffering stage within MEMS storage. Adding an additional layer of buffering for streams presents new challenges in real-time IO scheduling which is the subject of this section.

In the rest of this section, we first present our real-time IO scheduling model for meeting the timeliness guarantees required by streaming data. We then explore single-disk and multidisk storage systems that integrate a MEMS-based storage as a data buffering device.

### 3.1 Real-Time IO Scheduling for Streaming Data

Integrating multiple real-time streams in a common storage platform raises several issues. Streams compete for the same storage resources in terms of space as well as IO bandwidth. While serving multiple application-generated IOs simultaneously, a significant nonuniform switching overhead exists for nonuniform memory access (NUMA) devices like disk drives and MEMS-based storage devices. Allocation and reservation schemes must take this additional overhead into account when attempting to meet timeliness guarantees.

Streaming data are read or written on the storage device at a certain rate, thereby requiring real-time IO scheduling in the storage system. We adopt the time cycle-based scheduling model [Rangan et al. 1992] for scheduling continuous media streams. In this model, time is split into basic units called IO time cycles. In each IO cycle, the disk performs exactly one IO operation for each media stream. This is depicted in Figure 3, where $T_1$, $T_2$, and $T_3$ are three successive time cycles and the system in servicing three streams. The IO scheduler services the streams in the same order in each IO time cycle. Careful management of data buffers and precise scheduling [Chang and Garcia-Molina 1997]

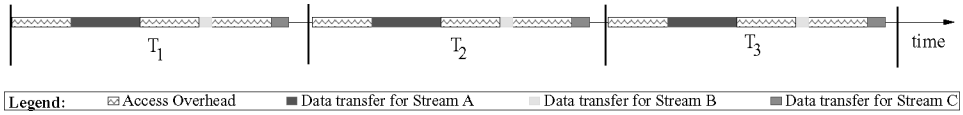



Fig. 3. Time cycle-based IO scheduling for streaming data.

can reduce the total amount of buffering required to the amount of data read in one time cycle.

### 3.2 Single-Disk Streaming

When serving a request for streaming data from a client, segments of the streaming data file are temporarily buffered in DRAM before they are consumed to minimize DRAM requirement and obtain acceptable performance from the storage device. Given the increasing gap between disk throughput and access times, filling this buffering requirement using DRAM is becoming increasingly expensive. Using MEMS storage as an intermediate buffering device can alleviate most of the buffering requirement for operating the disk drive at high throughput levels.

At a fraction of the cost of DRAM, MEMS storage can provide a large amount of buffering required for achieving high disk utilization. Although DRAM buffering cannot be completely eliminated, the low-access latency of MEMS storage provides high throughput with a significantly lower DRAM buffering requirement. The MEMS device can thus act as a speed-matching buffer between the disk drive and the system memory, in effect addressing the disk utilization and data buffering trade-off.

Using MEMS storage as an intermediate buffer implies that the MEMS-based device must handle both disk and DRAM data traffic simultaneously. To understand the service model, let us assume that the multimedia streams serviced are all read streams so that stream data read from the disk drive is buffered temporarily in the MEMS device before it is read into the DRAM. This model can be easily extended to address write streams.

To service buffered data from the MEMS device, we use the time-cycle IO service model. Data is retrieved in cycles into the DRAM such that no individual stream experiences data underflow at the DRAM. At the same time, the data read from the disk drive must be written to the MEMS device. The disk IO scheduler controls the read operations at the disk drive. The MEMS IO scheduler controls the write operations for data read from the disk as well as read operations into the DRAM. In the steady state, the amount of data written to the MEMS device is equal to the amount read from it. Thus, although the MEMS device can help improve disk utilization, we must realize that to do so, it must operate at twice the throughput of the disk drive. In order to minimize buffering requirements between the disk drive and the MEMS storage, the disk and the MEMS IO schedulers must therefore cooperate.

The MEMS buffer could consist of multiple physical MEMS devices to provide greater buffering capacity and throughput. As we shall see in Section 5, a bank of MEMS devices may be required to buffer IOs for a single disk. The (predicted) low entry cost of these devices makes such configurations practical.

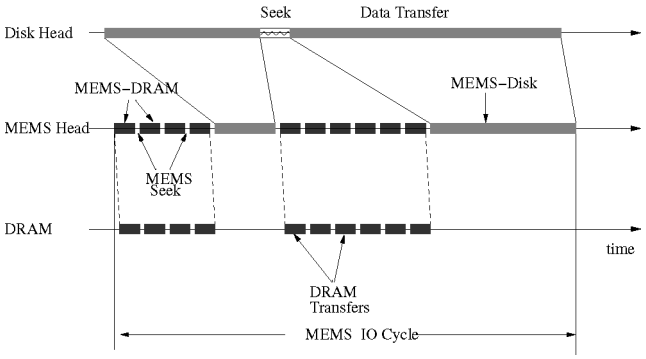



Fig. 4. Single MEMS IO scheduling.

3.2.1 *IO Scheduling: Single MEMS.* To guarantee real-time data retrieval from the disk using a single-device MEMS buffer, the MEMS IO scheduler services IOs on the MEMS device in rounds or *IO cycles*. Due to the extra layer of the MEMS buffer, there exist two distinct IO cycles, one for the disk (the *disk IO cycle*, during which $N$ IOs are performed on the disk drive) and the other for the MEMS buffer (the *MEMS IO cycle*, during which $N$ IO transfers occur from the MEMS buffer to the DRAM). The disk IO cycle operates as described earlier. In each MEMS IO cycle, the MEMS device services exactly one DRAM transfer for each of the $N$ streams serviced by the system. The amount of data read for each stream is sufficient to sustain playback before the next IO for the stream is performed, thereby meeting the real-time constraint. Further, the MEMS device also services $M$ transfers ($M < N$) from the disk in each IO cycle. In the steady state, the total amount of data transferred from the disk to the MEMS buffer is equal to that transferred from the MEMS buffer to DRAM.

Figure 4 describes the data transfer operations occurring during a single MEMS IO cycle. Note that the disk IO cycle is much larger than the MEMS IO cycle; it allows for $N$ IOs from the disk drive to the MEMS device. The X-axis is the time axis. The three horizontal lines depict the activity at the disk head, the MEMS tips, and the DRAM, respectively. In this example, the system services 10 streams ($N = 10$). The lightly-shaded regions depict data transfer from the disk drive into the MEMS device. The dark regions depict data transfer between the MEMS device and the DRAM. The MEMS device performs $N$ small IO transfers between MEMS and DRAM and $M$ large disk transfers in each MEMS IO cycle.

3.2.2 *IO Scheduling: Multiple MEMS.* According to certain predictions [Schlosser et al. 2000; Vettiger et al. 2000], a single MEMS device might not be able to support twice the bandwidth of future disk drives. In such cases, a bank of $k$ MEMS devices would provide a higher aggregate bandwidth. Using $k$ MEMS devices for buffering disk IOs raises interesting questions. How should stream data be split across these devices? What constitutes an IO cycle at the MEMS buffer? To what uses can we put any spare storage or bandwidth at the MEMS devices? We answer each question in turn.

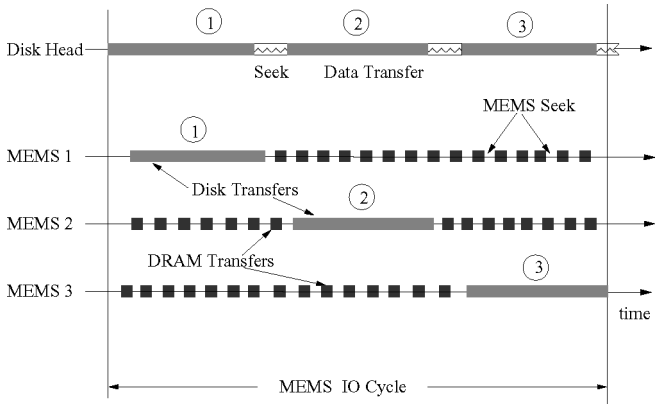
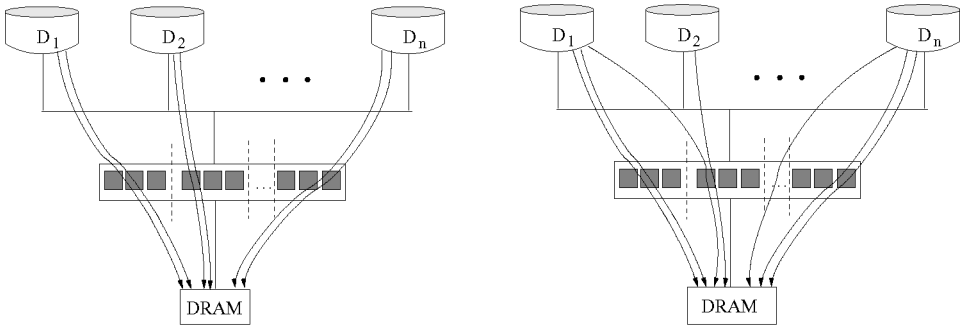



Fig. 5. IO Scheduling for a MEMS bank.

*Placing stream data.* Buffered data can either be striped across the MEMS bank or simply placed on a single MEMS device. Striping data for each stream across the $k$ MEMS devices can be accomplished by splitting each disk IO into $k$ parts and routing each part to a different MEMS device. The size of disk-side IOs performed on the MEMS device is reduced by a factor of $k$. Since a smaller average IO size decreases the MEMS device throughput, striping can be undesirable. Instead of striping the data, the set of streams could be split across the MEMS bank. Each stream would thus be buffered on a single MEMS device. This would preserve the size of disk-side IO transfers. To achieve such a split, the disk IOs are routed to the MEMS devices in a round-robin fashion. Every $k$th disk IO is routed to the same MEMS device. Routing each IO to a single MEMS device improves the MEMS throughput and is thus preferable to striping each disk IO across the MEMS bank.

*IO Cycle for a k-device MEMS bank.* Routing each disk IO to a single MEMS device splits the set of streams across the $k$ MEMS devices in the bank. The notion of *IO cycle* for the MEMS bank can be defined in the same manner as that for a single device MEMS buffer. It is the time required to perform exactly one DRAM transfer for each of the $N$ streams serviced by the system. For the sake of simplicity, let us assume that each MEMS device services $\frac{N}{k}$ streams. Figure 5 depicts the operations during an IO cycle at each MEMS device. The number of streams, $N$, is 45 and the number of MEMS devices in the bank is $k = 3$. For each disk IO, 15 DRAM transfers take place. The amount of data read from and written into the MEMS device is the same in the steady state.

*Using the spare MEMS storage and bandwidth.* Depending on the number and type of streams serviced and the capacity of the MEMS device bank, spare storage and/or bandwidth might be available at the MEMS device. If additional storage is available at the MEMS device, the operating system could use it for other non-real-time data: as a persistent write buffer, as a cache for read data with temporal or spatial locality, or as a disk prefetch buffer. The MEMS storage




(a) Partitioned streaming (b) Integrated streaming

Fig. 6. The partitioned and integrated MEMS buffer schemes.

can also be used to cache entire streams which are found to be popular among users. Spare bandwidth, if available, can be used for non-real-time traffic.

### 3.3 Multidisk Streaming

We now extend our techniques for data placement and IO scheduling previously presented for single disk with integrated MEMS store to a multidisk setting. We analyze how the two logical hardware abstractions, partitioned and integrated (introduced in Section 2) can be managed to serve streaming data. Each of these abstractions has its advantages and disadvantages when used to store streaming data. We propose two MEMS buffer management schemes that work with these abstractions.

3.3.1 *Partitioned MEMS Buffer Cache.* The partitioned MEMS buffer cache scheme can work with both logical hardware abstractions. It assumes that it is possible to allocate a fixed and exclusive MEMS buffer to each disk. Each disk only has access to its exclusive buffer and cannot access those of the other disks in the multidisk system. Such exclusiveness, naturally available in the partitioned abstraction, can be easily enforced for an integrated hardware abstraction.

In the partitioned configuration, streams stored on the diskdrive are cached and buffered on its exclusive MEMS buffer. Figure 6(a) depicts the stream flows in this configuration for the integrated hardware architecture. Strict partitioning of the space is enforced within the MEMS buffer. In this configuration, the solutions presented earlier for the single-disk case are directly applicable. The exclusive nature of the buffer cache makes the data placement and IO scheduling solutions for the partitioned configuration relatively easier to design and implement. The only difference from the single-disk case is that the amount of DRAM required is the sum of those required for the individual disks in the array.

Although the partitioned solution is easy to implement, it may not be the optimal under certain conditions.

—*Load imbalance.* We define *loadimbalance* in a multidisk system as the ratio of the number of streams serviced by the most-loaded disk to the number

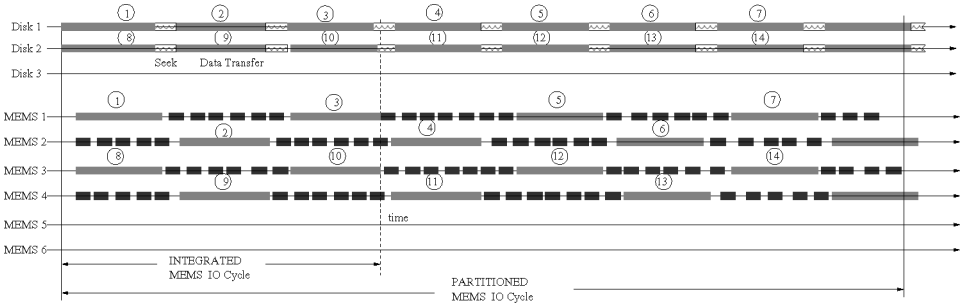



Fig. 7. Partitioned scheduling.

of streams served by the least-loaded disk, assuming that all streams share a common bit rate. In the case of load imbalance, some disk drives may be serving more data streams than others. In such a case, exclusive access to buffer or cache space may lead to underutilization and overload. Disk drives that serve more streams may underperform because the MEMS cache or buffer space is insufficient, while MEMS buffers exclusive to other disks may be underutilized.

—*Bit rate heterogeneity.* We define *bit rate heterogeneity* in a multidisk system as the ratio of the maximum average-bit-rate to the minimum average bit rate of the streams served by any single disk in the system, assuming that all the disks stream the same total bandwidth. Lower bit rate streams typically require more buffer space to sustain higher throughput levels. If a disk drive serves a large fraction of low bit rate streams, it may find buffer space to be inadequate to sustain high throughput levels. Another disk drive serving exclusively high bit rate streams may waste buffer space.

—*Underutilized MEMS bandwidth.* The portion of the MEMS buffer that is unused also implies unused MEMS buffering bandwidth. The unusable bandwidth on the MEMS buffer drives up the DRAM buffering requirement for streams by requiring larger accesses on the MEMS device to deliver the throughput required.

Figure 7 depicts the data transfer operations for a partitioned MEMS buffer for a three-disk system with two MEMS devices per disk. Of these, Disk 3 has zero load, and hence its corresponding MEMS devices (#5 and #6) are completely unutilized. In a real system, such load imbalance scenarios may arise when stream popularity changes or when disks are added or removed dynamically. Given a multidisk system where the load is imbalanced, the partitioned solution fractures the MEMS buffer space as well as bandwidth and is unsuitable under the conditions outlined. The *integrated* solution that we propose next overcomes these shortcomings.

3.3.2 *Integrated MEMS Buffer Cache.* The integrated MEMS buffer configuration applies only to the integrated hardware abstraction. It overcomes the shortcomings of the partitioned solution by sharing a common MEMS buffer pool between all the drives. However, single-disk solutions cannot be directly

applied to this configuration. Since the buffer resources are now common, rather than being completely exclusive, new sharing and protection schemes need to be introduced.

Figure 6(b) depicts the stream flows in this configuration for the integrated hardware architecture. Disk drives that are more loaded than others use up more of the MEMS buffer and cache space as well as bandwidth. Even in the case of load imbalance or bit rate heterogeneity, all MEMS resources are utilized. Since all the available MEMS buffer bandwidth is utilized, the DRAM requirement is also minimized.

In the integrated configuration, since all the drives may access any portion of the buffer, contention for the resources makes the solution more complex than that for the partitioned configuration. To resolve space and bandwidth contention, new data placement and IO scheduling strategies are required. Since these resources are now part of a common pool, they must be dynamically allocated based on the load on the individual disk drives. The resource allocation strategy must accommodate both overload and underload at the individual drives. At the same time, it is also important that bandwidth and space load on the individual devices in the MEMS device pool is balanced to minimize the DRAM requirement and improve the overall throughput of the storage system.

We now outline the resource allocation strategy for both space and bandwidth on the MEMS devices. To absorb overload at an individual drive, the resource allocator does not differentiate between disks and disk IOs. Its view of the multidisk store is equivalent to a single device from where all IOs are executed. Whenever a new stream is serviced on a disk drive, the MEMS space and bandwidth resources required for the stream is allocated based on the load on the individual MEMS devices.

*Resource Allocation and Load Balancing.* A purely space-based or purely bandwidth-based resource allocation and load balancing is unlikely to yield good overall performance. A bandwidth-loaded device may be the least space-constrained device. Similarly a space-constrained device may possess adequate spare bandwidth. Our resource allocation strategy combines space-based and bandwidth-based allocation.

The time-cycle utilization of a MEMS device gives a good approximation of how resource constrained the device is on both counts. A space-constrained device utilizes more of the time cycle for data transfers, whereas a bandwidth-constrained device utilizes more of the time cycle for both switching as well as data transfers. The MEMS devices in the pool are sorted according to their time cycle utilization. When a new stream must be served, the MEMS device with the least utilized time cycle is picked. If the buffer space available on the device is insufficient for the new stream, the next device in the sorted list is chosen. This procedure is followed until a device satisfying both space and bandwidth requirements is found.

Figure 8 shows the IO operations scheduled on the MEMS devices for the duration of a MEMS IO cycle. The number and nature of streams served ($N$) is the same as that in the partitioned case (shown in Figure 7). In the integrated case, all the available MEMS devices are utilized regardless of the load distribution

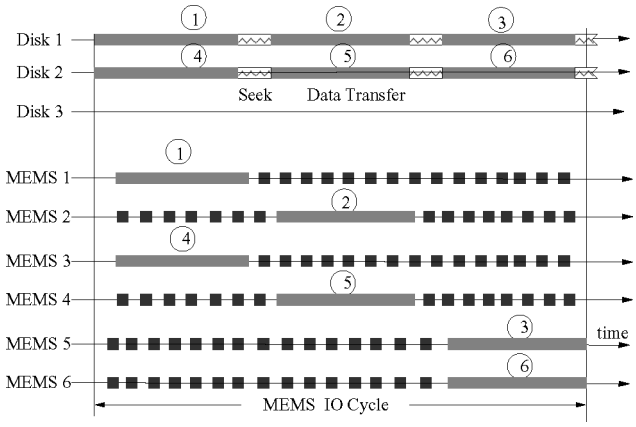



Fig. 8. Integrated scheduling.

on the disk drives. A simple round-robin strategy is used to balance the IOs on the MEMS buffer. Also notice that the MEMS IO cycle duration is considerably shortened. Since the DRAM buffering requirement is directly proportional to the length of the MEMS IO cycle, the integrated MEMS buffer configuration minimizes the DRAM buffering cost for a given IO workload.

If the system services multiple bit rate streams, it is necessary to balance both the number of streams buffered as well as the bandwidth requirement of each MEMS device. One possible heuristic is to assign each stream from a disk to a MEMS device in the bank in a round-robin fashion. This would distribute the number of streams equally over all the MEMS devices. If the variance in bit rate of streams from a single disk is not significant, this strategy would work well. Another heuristic first tries to balance the bandwidth requirement of the MEMS devices. In the case that the bandwidth requirements are balanced, the heuristic assigns any new stream to the MEMS device which serves the minimum number of streams.

In Section 5, we evaluate the performance of the integrated MEMS buffer using a combination of these approaches by varying load imbalance as well as bit rate heterogeneity.

## 4. QUANTITATIVE ANALYSIS

In the previous section, we generalized time cycle-based I/O scheduling for real-time streams to a three-layer storage hierarchy. Time cycle-based schedulers naturally allow for closed-form expressions for various performance metrics. In this section, we derive a quantitative model to analyze buffering requirements for systems supporting real-time streaming applications. The model is very general and allows us to argue about the performance of a streaming multimedia system that services multiple streams of varying and variable bit rates and employs a MEMS-augmented storage hierarchy consisting of one or more MEMS-based storage devices and one or more disk drives.

To evaluate the effectiveness of MEMS-based buffering, we compare the system cost with and without MEMS storage. Let $C_{dram}$ and $C_{mems}$ denote

the unit cost ($/B) of DRAM and MEMS buffer, respectively. Furthermore, we use a per-device cost model for MEMS storage. The $k$ MEMS devices cost $k \times C_{mems} \times Size_{mems}$ even if the system does not utilize all the available MEMS storage.

### 4.1 Single-Disk Analysis

Let $S_{disk\text{-}dram}$ denote the per-stream IO size from disk to DRAM, $S_{disk\text{-}mems}$ from disk to MEMS, and $S_{mems\text{-}dram}$ from MEMS to DRAM. Let $k$ denote the number of MEMS devices in the system. Let $N$ denote the number of streams in the system. The buffer cost with and without the MEMS buffer is

$$COST_{without\_mems} = N \times C_{dram} \times S_{disk\text{-}dram} \tag{1}$$

$$\begin{aligned} COST_{with\_mems} &= k \times C_{mems} \times Size_{mems} \\ &\quad + N \times C_{dram} \times S_{mems\text{-}dram}, \end{aligned} \tag{2}$$

where $k \times Size_{mems} \geq N \times S_{disk\text{-}mems}$. Using MEMS devices in a streaming system is cost effective only if $COST_{with\_mems} < COST_{without\_mems}$.

In order to calculate the system cost, we first calculate IO sizes that guarantee the real-time streaming requirements. We next compute disk and MEMS IO sizes given the following four input parameters.

—$N$: The number of streams that the server supports

—$\bar{B}$: The average bit rate of the $N$ streams

—$R_d$: The data transfer rate of device $d$ $R_d$ is substituted by $R_{disk}$ (disk transfer rate) or $R_{mems}$ (MEMS transfer rate) depending on where the IO takes place

—$\bar{L}_d$: The average latency of device $d$ in a time cycle. $\bar{L}_d$ is substituted by $\bar{L}_{disk}$ (disk latency) or $\bar{L}_{mems}$ (MEMS latency) depending on where the IO takes place. $\bar{L}_d$ also depends on the scheduling policy employed to manage device $d$.

In computing IO size, we assume a time cycle-based IO scheduler. Further, to simplify the analytical model, we assume all streams to be in constant bit rate (CBR).[1] We summarize the parameters used in this paper in Table II.

THEOREM 1. *For a system which streams directly from the disk to DRAM, the minimum size of the per-stream DRAM buffer required to satisfy real-time requirements is*

$$S_{disk\text{-}dram} = \frac{N \times \bar{L}_{disk} \times R_{disk} \times \bar{B}}{R_{disk} - N \times \bar{B}} \quad , \quad where \;\; R_{disk} > N \times \bar{B}. \tag{3}$$

PROOF. Given $N$ streams to support, in each IO cycle, the disk must perform $N$ IO operations, each consisting of a latency component and a data transfer component. Let $T_{disk}$ denote the disk cycle time. Let $L_{disk_i}$ denote the disk latency

[1]VBR can be modeled by CBR plus some memory cushion for handling bit rate variability [Makaroff et al. 1997].

Table II. Parameter Definitions

| Parameter | Description |
|---|---|
| $N$ | Number of continuous media streams |
| $\bar{B}$ | Average bit rate of the streams serviced $[B/s]$ |
| $k$ | Number of MEMS devices in system |
| $R_{disk}$ | Data transfer rate from disk media $[B/s]$ |
| $R_{mems}$ | Data transfer rate from MEMS media $[B/s]$ |
| $\bar{L}_{disk}$ | Average latency for disk IO operations $[s]$ |
| $\bar{L}_{mems}$ | Average latency for MEMS IO operations $[s]$ |
| $C_{dram}$ | Unit DRAM cost $[\$/B]$ |
| $C_{mems}$ | Unit MEMS cost $[\$/B]$ |
| $Size_{mems}$ | MEMS capacity per device $[B]$ |
| $Size_{disk}$ | Disk capacity $[B]$ |
| $S_{disk\text{-}dram}$ | Average IO size from disk to DRAM $[B]$ |
| $S_{disk\text{-}mems}$ | Average IO size from disk to MEMS $[B]$ |
| $S_{mems\text{-}dram}$ | Average IO size from MEMS to DRAM $[B]$ |
| $T_{disk}$ | Disk IO cycle $[s]$ |
| $T_{mems}$ | MEMS IO cycle $[s]$ |

to start IO transfer for stream $i$. Then $T_{disk}$ can be written as

$$T_{disk} \geq \sum_{i=1}^{N} L_{disk_i} + \frac{N \times T_{disk} \times \bar{B}}{R_{disk}}.$$

To fulfill the real-time data requirement, the amount of data transferred during $T_{disk}$ must be sufficient to sustain the next $T_{disk}$ time of playback for $N$ streams. Thus the average disk IO size $S_{disk\text{-}dram} = T_{disk} \times \bar{B}$. Now, we isolate $T_{disk}$ on the left-hand side of the equation and rewrite $T_{disk}$ as

$$T_{disk} \geq \frac{N \times \bar{L}_{disk} \times R_{disk}}{R_{disk} - N \times \bar{B}}.$$

Given $T_{disk}$ and $\bar{B}$, we can compute the average disk IO size as $S_{disk\text{-}dram} = T_{disk} \times \bar{B}$. □

COROLLARY 1. *To stream directly from the MEMS device to the DRAM, the minimum size of the per-stream DRAM buffer required to satisfy real-time requirements is*

$$S_{mems\text{-}dram} = \frac{N \times \bar{L}_{mems} \times R_{mems} \times \bar{B}}{R_{mems} - N \times \bar{B}} \quad , \quad where \;\; R_{mems} > N \times \bar{B}. \tag{4}$$

PROOF. The same as for Theorem 1. □

Although Theorem 1 is well established [Rangan et al. 1992], calculating IO sizes is more complex in a system that uses MEMS as an intermediate buffer between the disk and DRAM because we must consider the real-time requirements between the disk and MEMS as well as between the MEMS and DRAM.

THEOREM 2. *For a system which uses k MEMS devices as a disk buffer, the minimum size of the per-stream DRAM buffer required to satisfy real-time*

*requirements is*

$$S_{mems\text{-}dram} = \bar{B} \times \frac{C \times (1 + \frac{2k-2}{N}) \times T_{disk}}{T_{disk} - C}, \quad (5)$$
$$where\ C = \frac{N \times \bar{L}_{mems} \times R_{mems}}{k \times R_{mems} - 2 \times (N + k - 1) \times \bar{B}}.$$

*$T_{disk}$ is the largest value such that the following three conditions (real-time requirement, storage requirement, and scheduling requirement) are satisfied:*

$$T_{disk} \geq \frac{N \times \bar{L}_{disk} \times R_{disk}}{R_{disk} - N \times \bar{B}}, \ \ where \ \ R_{disk} > N \times \bar{B} \quad (6)$$

$$2 \times N \times T_{disk} \times \bar{B} \leq k \times Size_{mems} \quad (7)$$

$$\frac{T_{mems}}{T_{disk}} = \frac{M}{N}, \ \ M < N, \ \ M \in Integer. \quad (8)$$

PROOF. Let $M$ denote the number of disk IOs performed in one MEMS time cycle. For the sake of simplicity, we will assume that $M$ must be an integer. Let $k$ denote the number of MEMS devices acting as a buffer. The amount of data transferred between the disk drive and the MEMS buffer in one MEMS IO cycle must be equal to that transferred between the MEMS buffer and DRAM. This is to ensure that the MEMS buffer does not overflow or underflow in the steady state. Consequently,

$$N \times S_{mems\text{-}dram} = M \times S_{disk\text{-}mems}$$
$$\frac{N}{M} = \frac{S_{disk\text{-}mems}}{S_{mems\text{-}dram}} = \frac{\bar{B} \times T_{disk}}{\bar{B} \times T_{mems}} = \frac{T_{disk}}{T_{mems}}$$
$$M = N \times \frac{T_{mems}}{T_{disk}}, \ \ \{N, M\} \in Integer. \quad (9)$$

Equations (6) and (7) give a possible range for $T_{disk}$. However, the lower bound for $T_{mems}$ has to be derived. Let us first express $T_{mems}$ in a system with $k$ MEMS devices so that real-time streaming requirements are satisfied. Each MEMS device performs at most $\lceil \frac{M}{k} \rceil$ disk transfers and $\lceil \frac{N}{k} \rceil$ DRAM transfers. Thus, we can express $T_{mems}$ at each MEMS device as

$$T_{mems} \geq \left( \left\lceil \frac{N}{k} \right\rceil + \left\lceil \frac{M}{k} \right\rceil \right) \times \bar{L}_{mems} + \left( 2 \times \left\lceil \frac{N}{k} \right\rceil \times \bar{B} \times T_{mems} \right) \Big/ R_{mems}. \quad (10)$$

Using Equation (10), we derive minimum value for $T_{mems}$ as

$$T_{mems} \times \left( R_{mems} - 2 \times \left\lceil \frac{N}{k} \right\rceil \times \bar{B} \right) \geq \left( \left\lceil \frac{N}{k} \right\rceil + \left\lceil \frac{M}{k} \right\rceil \right) \times \bar{L}_{mems} \times R_{mems}$$
$$T_{mems} \geq \frac{(N + 2k - 2 + M) \times \frac{\bar{L}_{mems}}{k} \times k \times R_{mems}}{k \times R_{mems} - 2 \times (N + k - 1) \times \bar{B}}. \quad (11)$$

□

COROLLARY 2. *When $N$ and $M$ are divisible by $k$ (or are relatively large compared to $k$), $k$ MEMS devices behave as a single MEMS device with both $k$ times smaller average latency and $k$ times larger throughput.*

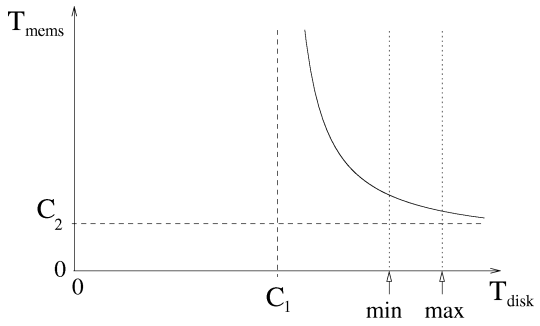



Fig. 9. $\mathbf{T}_{mems}$ as a function of $\mathbf{T}_{disk}$. DRAM buffering can be reduced by increasing $T_{disk}$.

PROOF. Since $S_{mems\text{-}dram} = \bar{B} \times T_{mems}$, the proof follows from a comparison between Equations (4) and (11). Specifically, when N and M are divisible by $k$, Equation (10) can be reduced to:

$$T_{mems} \geq \frac{(N+M) \times \frac{\bar{L}_{mems}}{k} \times k \times R_{mems}}{k \times R_{mems} - 2 \times N \times \bar{B}}. \tag{12}$$

We can see that when $k = 1$, the previously equation describes the MEMS time cycle as a function of the MEMS device parameters ($\bar{L}_{mems}$ and $R_{mems}$) and the application parameters ($N$ and $\bar{B}$). When $k > 1$, the time-cycle duration resembles the time-cycle duration with a single device with $k$ times larger throughput and $k$ times smaller average latency. When $N$ and $M$ are large relatively to $k$, Equation (11) approximates to Equation (12).

Let us now express minimum $T_{mems}$ as a function of $T_{disk}$ using Equations (6), (9), and (11) as

$$\begin{aligned} T_{mems} &= \frac{C \times (1 + \frac{2k-2}{N}) \times T_{disk}}{T_{disk} - C}, \\ where\ C &= \frac{N \times \bar{L}_{mems} \times R_{mems}}{k \times R_{mems} - 2 \times (N + k - 1) \times \bar{B}}. \end{aligned} \tag{13}$$

Figure 9 depicts the possible values of $T_{mems}$ as a function of $T_{disk}$ ($C_1 = C$ and $C_2 = C \times (1 + (2k - 2)/N)$). Equation (6) gives minimum value for $T_{disk}$ so that the system satisfies real-time requirements. Maximum $T_{disk}$ is bounded by the capacity of MEMS storage as in Equation (7). □

We can now find a minimum cost for the streaming buffer using the method depicted in Figure 10. This method finds iteratively the optimal value of $k$, the size of the MEMS bank, so that the total cost of the system, including DRAM and MEMS storage cost, is minimized.

### 4.2 Multidisk Analysis

Table III lists additional parameters that we will use for our multidisk analysis. Further, to refer to a parameter for a specific drive, we will use the parameter from Table II marked with the respective subscript; an example is shown in the Table III for $N_i$.

**Input:** $N$, $\bar{B}$, $\bar{L}_{disk}$, $R_{disk}$, $\bar{L}_{mems}$, $R_{mems}$.
**Output:** $k$, $T_{mems}$, and $T_{disk}$.

1. $MinCost = C_{dram} \times N \times S_{disk-dram}$
2. $k = 0$
3. **do**
4. $\quad k = k + 1$
5. $\quad$ Find $T_{mems}$ using Theorem 2.
6. $\quad Cost = k \times C_{mems} \times Size_{mems} + C_{dram} \times N \times T_{mems} \times \bar{B}$
7. $\quad$ **if** $Cost < MinCost$ **then** $MinCost = Cost$
8. **while** $MinCost > (k+1) \times C_{mems} \times Size_{mems}$

Fig. 10. Procedure for finding the minimum buffer cost.

Table III. Additional Parameter Definitions

| Parameter | Description |
|---|---|
| $d$ | Number of disk drives in system |
| $k$ | MEMS devices per disk in the partitioned configuration |
| $N_i$ | Number of streams serviced by disk $i$ |
| $B_{req}$ | Total bandwidth requested from a single disk in a multi-disk streaming server |
| $i$ | Load imbalance: Ratio of streams serviced by most-loaded to least-loaded disk |
| $N_{min}$ | Number of streams serviced by the least loaded disk |
| $N_{max}$ | Number of streams serviced by the most loaded disk |
| $h$ | Bit rate heterogeneity: Ratio of maximum to minimum average bit rates serviced |
| $\bar{B}_{min}$ | Average bit rate of streams at the disk with least average bit rate |
| $\bar{B}_{max}$ | Average bit rate of streams at the disk with greatest average bit rate disk |
| $T_{disk_i}$ | Disk IO cycle duration for disk $i$ |

4.2.1 *Load Imbalance Analysis.* Given the load imbalance factor, $i$ ($i = \frac{N_{max}}{N_{min}}$), and the total streams serviced by the entire system, $N$, the number of streams serviced by the least- and most-loaded disks, $N_{min}$ and $N_{max}$, can be calculated as:

$$N_{min} = \frac{2 \times N}{d \times (i+1)} \quad ; \quad N_{max} = i \times N_{min}. \tag{14}$$

The streams serviced by individual disks, $N_i$ can be obtained as:

$$N_i = N_{min} + (i-1) \times \frac{N_{max} - N_{min}}{d-1}. \tag{15}$$

*Partitioned configuration.* Using Theorem 1, the minimum size of the per-stream MEMS buffer required to satisfy real-time requirements for streams serviced by disk $i$ is:

$$S_{disk\text{-}mems_i} = \frac{N_i \times \bar{L}_{disk} \times R_{disk} \times \bar{B}}{R_{disk} - N_i \times \bar{B}} \quad , \quad where \;\; R_{disk} > N_i \times \bar{B}. \tag{16}$$

Since the disk drives have independent MEMS buffers in the partitioned configuration, the analysis for each disk in the multidisk setting is similar to that for the single disk starting from Equation (17). $S_{mems\text{-}dram_i}$, $T_{disk_i}$, and $T_{mems_i}$ can be calculated similarly.

*Integrated configuration.* For the integrated configuration, since the MEMS buffer is shared among all the drives, from the perspective of the MEMS storage, the multiple-disk configuration appears as a single drive serving all the $N = \sum N_i$ streams. Hence, we can use Equation (17) for the integrated configuration as follows and calculate the effective average IO size between the disk drive array and the MEMS bank, $S_{disk\text{-}mems}$, as:

$$S_{disk\text{-}mems} = \frac{1}{d} \sum_i S_{disk\text{-}dram_i}. \tag{17}$$

Further, $S_{mems\text{-}dram}$, $T_{disk}$, and $T_{mems}$ can be calculated independent of the load-imbalance factor. The integrated configuration effectively balances the load imbalance on the disk drives at the MEMS buffering layer by providing the flexibility of allocating variable per-disk MEMS buffer storage.

4.2.2 *Bit Rate Heterogeneity Analysis.* Given the bit rate heterogeneity, $h$ ($i = \frac{\bar{B}_{max}}{\bar{B}_{min}}$), and the system average bit rate, $\bar{B}$, the average bit rates at the individual drives, $\bar{B}_i$, can be calculated as:

$$\bar{B}_{min} = \frac{2 \times \bar{B}}{h+1} \quad ; \quad \bar{B}_{max} = h \times \bar{B}_{min}. \tag{18}$$

The average bit rate of individual disks, $\bar{B}_i$ can be obtained as:

$$\bar{B}_i = \bar{B}_{min} + (i-1) \times \frac{\bar{B}_{max} - \bar{B}_{min}}{d-1}. \tag{19}$$

To analyze system performance in the presence of bit rate heterogeneity, we assume that each disk is requested with the same total bandwidth and differs only in the average bit rate of streams serviced. To do so, we include an additional specification parameter $B_{req}$, the streaming bandwidth requested from each disk in the system. Given this requested bandwidth, we then calculate the number of streams serviced by each disk and the total MEMS buffering requirement for the partitioned and integrated configurations. We expect that the integrated configuration because of its ability to share MEMS buffering space and bandwidth between disk drives will incur a smaller MEMS buffering requirement. The number of streams serviced by disk $i$, $N_i$, is given by:

$$N_i = \frac{B_{req}}{\bar{B}_i}. \tag{20}$$

*Partitioned configuration.* For the partitioned configuration, the minimum size of the per-stream MEMS buffer at disk $i$, $S_{disk\text{-}mems_i}$, is given by replacing $\bar{B}$ with $\bar{B}_i$ in Equation (17). The total MEMS requirement can be obtained as $\sum N_i \times S_{disk\text{-}mems_i}$.

*Integrated configuration.* For the integrated configuration, since the MEMS buffer is shared among all the disks, the calculation of the minimum size of the per-stream MEMS buffer must take into account the average IO sizes at all the disks. To do so, we first calculate the IO time cycle for each disk, $T_{disk_i}$, using individual $N_i$ and $\bar{B}_i$ for each disk. The minimum size of the per-stream MEMS

Table IV. Expected Performance Characteristics of Storage Devices in the Year 2008

| Parameter | FutureDisk | G3 MEMS | DRAM |
|---|---|---|---|
| RPM | 20,000 | – | – |
| Max. bandwidth [MB/s] | 300 | 320 | 10,000 |
| Average seek [ms] | 2.8 | – | – |
| Full stroke [ms] | 7.0 | 0.45 | – |
| X settle time [ms] | – | 0.14 | – |
| Capacity per device [GB] | 1,000 | 10 | 5 |
| Cost/GB [$] | 0.2 | 1 | 20 |
| Cost/device [$] | 100–300 | 10 | 50–200 |

buffer can be calculated as:

$$S_{disk\text{-}mems} = \frac{\sum_i \frac{N_i \times S_{disk\text{-}mems_i}}{T_{disk_i}}}{\sum_i \frac{N_i}{T_{disk_i}}}. \tag{21}$$

The total MEMS requirement can then be obtained as $N \times S_{disk\text{-}mems}$.

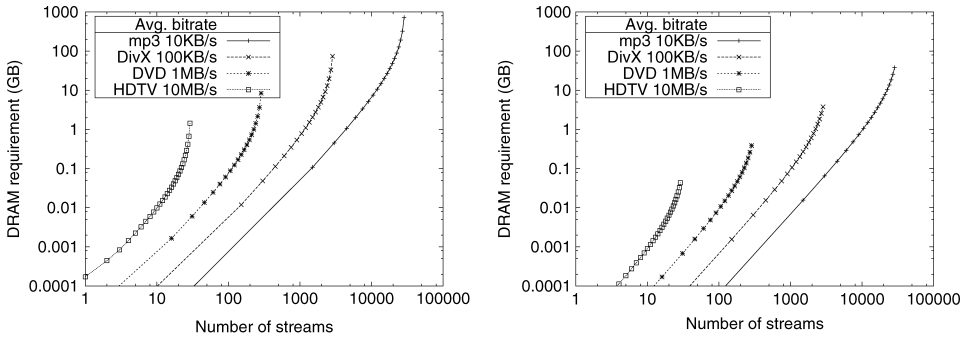
## 5. ANALYTICAL EVALUATION

This section presents an evaluation based on the quantitative model presented in the previous section. To model the performance of the MEMS device, we closely followed one such model proposed by researchers at Carnegie Mellon University. For our experiments, we use the 3rd Generation (G3) MEMS device model proposed in Schlosser et al. [2000]. Note that we defer the predicted availability of the G3 MEMS device by one year to reflect more recent industry progress on delivering MEMS storage technology commercially. We obtained predictions for disk drive and DRAM performance by using projections on current day devices produced by Maxtor Corporation [2002] and Rambus Inc. [2006], respectively. These are summarized in Table IV.

In our experiments, the average bit rate of streams, $\bar{B}$, was varied within the range of 10KB/s to 10MB/s. Since the maximum bandwidth of the FutureDisk, $R_{disk}$, is 300MB/s, it can support tens of high-definition streams at a few megabytes per-second each, more than a hundred compressed MPEG2 (DVD quality) streams at 1MB/s, or a thousand DivX (MPEG4) streams at 100KB/s, or even tens of thousands of MP3 audio at a bit rate of 10KB/s. To minimize the misprediction of seek-access characteristics for the MEMS device, we assume that MEMS accesses, $\bar{L}_{mems}$, always experience the maximum device latency. We use scheduler-determined latency values, $\bar{L}_{disk}$, for disk accesses. The disk IO scheduler uses elevator scheduling to optimize for disk utilization.

### 5.1 Single Disk

As mentioned in Section 3, using MEMS storage to buffer streaming data requires that it supports twice the streaming bandwidth of the disk drive.In our experiments, we used at least two G3 MEMS devices for buffering the streaming data, which provided a maximum aggregate MEMS throughput of




(a) Without MEMS buffer (b) With MEMS buffer

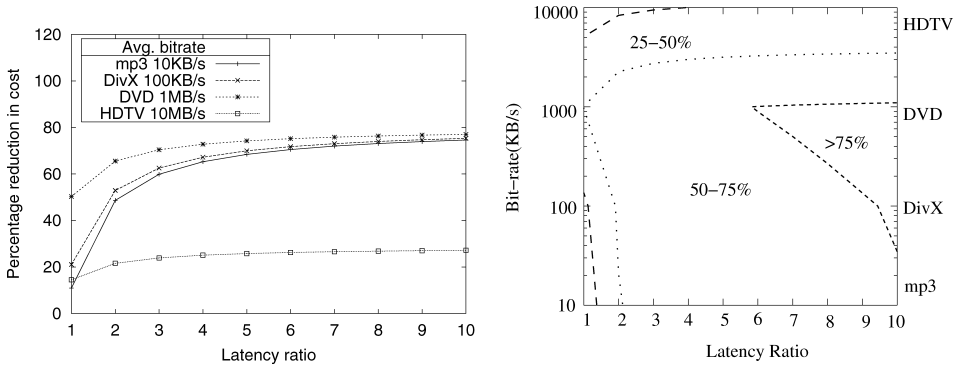
Fig. 11. DRAM requirement for various media types.

640MB/s. To evaluate the performance of the MEMS multimedia buffer, our experiments aimed to determine the reduction in DRAM requirement as well as overall system cost due to the addition of MEMS storage. In addition, we also determined the sensitivity of these metrics to variations in MEMS device characteristics.

The theorems presented in our earlier study [Rangaswami et al. 2003] describe the relationship between the system parameters. To study the sensitivity of our evaluation to MEMS device characteristics, we introduce the *latency ratio*, as a tunable parameter. We define the latency ratio as the ratio of the average disk access latency to the maximum MEMS access latency. We varied the latency ratio within the range of 1 to 10. The value for this parameter is around 5 for the FutureDisk and the G3 MEMS device listed in Table IV.

For performance evaluation, we conducted three experiments. In the first two experiments, we assumed that the maximum amount of DRAM and MEMS storage was unlimited. We also used a cost-per-byte price model for MEMS storage. These relaxations allowed us to observe the relationship between the system parameters. In the third experiment, we performed a case study using an off-the-shelf system which could be developed by the year 2008. The available buffering on this system is limited, and its size is based on current trends in server system configurations.

5.1.1 *Reduction in DRAM Requirement.* In Figure 11, we vary the number of streams, $N$, and the average stream bit rate, $\bar{B}$. We plot the DRAM requirement on the Y-axis. The X and Y-axes are drawn to logarithmic scale. The total buffering requirement increases rapidly with the number of streams (according to Equations 1 and 4). For a fixed system throughput, the buffering requirement is thus much larger for smaller bit rates than for larger ones. In the absence of a MEMS buffer, the DRAM requirement for a fully utilized disk ranges from 1GB for 10MB/s streams to 1TB for 10KB/s streams. With a MEMS buffer, the DRAM requirement is reduced by an order of magnitude to support a given system throughput.




(a) Cost-reduction curves (b) Cost-reduction regions

Fig. 12. Percentage cost reduction.

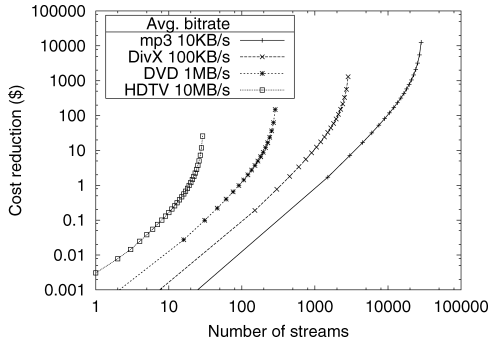
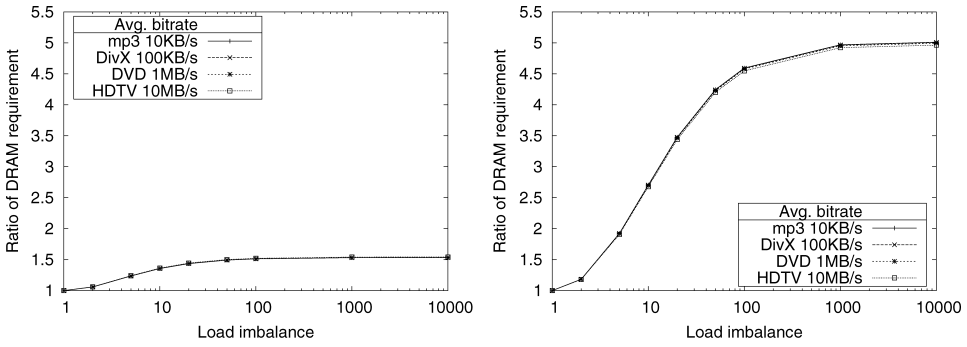



Fig. 13. Reduction in the total buffering cost.

5.1.2 *Reduction in Cost.* Addition of a MEMS buffer reduces DRAM requirement. However, we must take the total system cost into account before drawing any conclusion about the benefits. To calculate the cost of buffering, we use cost predictions as presented in Table IV. According to the predictions, MEMS buffering is 20 times cheaper than DRAM buffering per-byte (Figure 12).

Figure 13 shows the reduction in total buffering cost, including the additional cost of the MEMS buffer. In spite of the addition of the MEMS buffer, the total cost of buffering is reduced for all media types. Cost savings range from tens of dollars for high bit rate streams to tens of thousands of dollars for lower bit rates. These cost savings are almost directly proportional to the DRAM reductions presented in Figure 11, since the fractional cost addition due to the MEMS storage is negligible. This curve also shows that using a MEMS buffer for streaming large bit rates does not offer a significant reduction in the buffering cost. Indeed, for large bit rates, even DRAM buffering is sufficient to achieve high disk utilization. Upon calculation, we found that the cost reduction ranges between 80% and 90% depending on the number of streams, $N$, and their average bit rates, $\bar{B}$.




(a) Average disk utilization = 10% (b) Average disk utilization = 50%

Fig. 14. Improvement in DRAM requirement of the integrated over the partitioned configuration for different average disk utilization values.

## 5.2 Multiple Disks

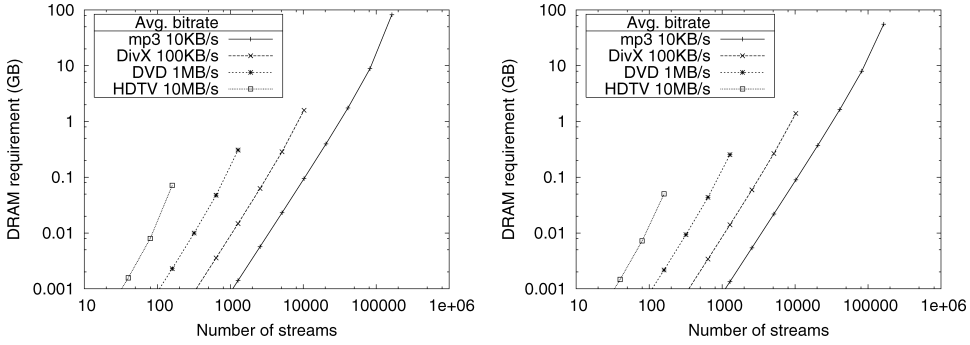
In a multidisk system supplied with a MEMS buffer bank, the MEMS buffer can be managed in either of the two configurations, partitioned or integrated. While the partitioned scheme offers the advantages of simplicity and potentially less IO bus traffic, it has performance drawbacks in certain settings. We alluded to these settings earlier in Section 3. Additional parameters in a streaming multidisk system may make a partitioned MEMS buffer a performance bottleneck and even make it unsuitable for a region of the parameter space. The additional parameters in a streaming multidisk system are:

—Load imbalance ($i$)
—Bit rate heterogeneity ($h$)

Although these definitions for these parameters (presented in Section 3) are restrictive, we use these parameter definitions to conduct an initial evaluation of the partitioned and integrated configurations. Varying other parameters apart from these two can create different workloads. However, we feel that these two parameters are sufficient to conduct an initial evaluation.

We now conduct an evaluation of the effect of these parameters on each configuration, namely the partitioned and integrated MEMS-buffer. For both parameters, we determine the MEMS buffer-space and the DRAM buffer required to support a given IO load.

5.2.1 *Effect of Load Imbalance.* We first explore the effect of an imbalanced load on the individual disk drives in each configuration. In the following experiment, we assume that all the streams serviced by the system have the same bit rate. The load distribution on the disk drives is set to follow an exponential distribution. Figure 14 depicts the improvement in DRAM requirement of the integrated configuration over the partitioned configuration. This improvement is the ratio of the DRAM requirement of the partitioned to the integrated configuration. When the disks are running at a low average utilization (Figure 14(a)),




(a) Partitioned MEMS buffer (b) Integrated MEMS buffer

Fig. 15. DRAM requirement for load imbalance ($i$) = 2.

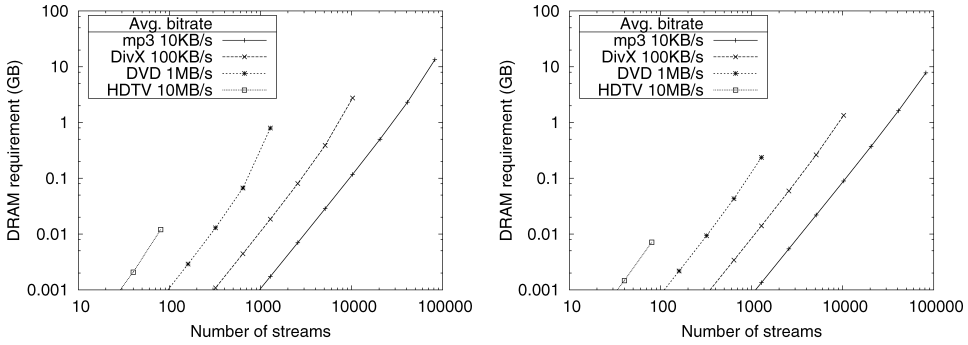
for most of the disks as well as the MEMS devices, the average IO size, and consequently the DRAM requirement, is small. The integrated configuration offers potential improvement by reducing the DRAM requirement even in such a case. This is simply because the uneven disk load is evenly distributed over the MEMS bank and no single MEMS device is overloaded. When the disks are made to operate at a higher average utilization (Figure 14(b)) by distributing the load evenly over all the MEMS devices, the integrated configuration offers significant savings in DRAM requirement of as much as 5X.

We also note that the reduction in DRAM requirement does not vary significantly for different bit rates. In fact, this improvement depends only on the product of the bit rate and the number of streams served by the system which is a constant for a given disk utilization. The slight difference we notice is due to the change in disk access overhead depending on the actual number of streams served.

Finally, we notice that the improvement levels off at higher values of load imbalance. This anomalous behavior is due to the definition of the load-imbalance metric. At higher values, although the load-imbalance value changes drastically, the actual load does not change significantly, and therefore the DRAM requirements for either configuration do not change significantly.

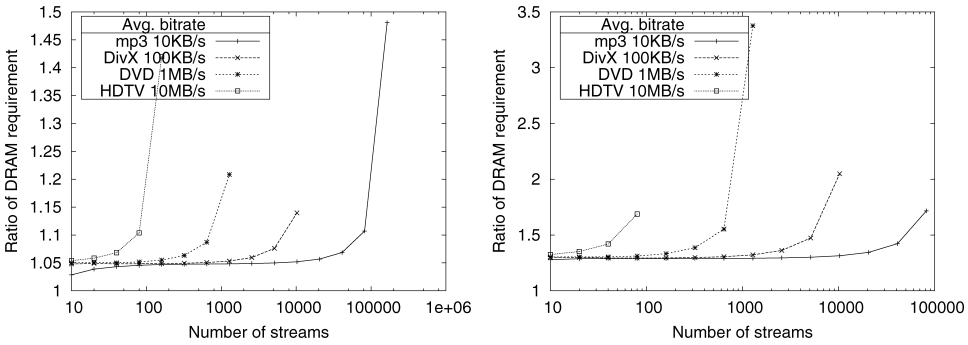
Figures 15 and 16 show the actual DRAM requirement for the partitioned and integrated configurations under different load conditions and for different average bit rates. For each curve, we also assume that all streams share the same bit rate. We vary the total number of streams serviced by the system and load the system to the point where the disks simply do not possess the raw throughput required to support all the streams. As observed in the single-disk case, streaming 10MBps HDTV streams requires far less DRAM than 10KBps MP3 streams. Due to a balanced MEMS buffer load in the integrated configuration, it can support a greater number of streams than the partitioned case given a fixed amount of DRAM.

Figure 17 shows the ratio of the DRAM requirements for the partitioned and integrated configurations for the same experiment. The trend shows an increase as the number of streams are increased for a given average bit rate. This is




(a) Partitioned MEMS buffer (b) Integrated MEMS buffer

Fig. 16. DRAM requirement for load imbalance ($i$) = 10.



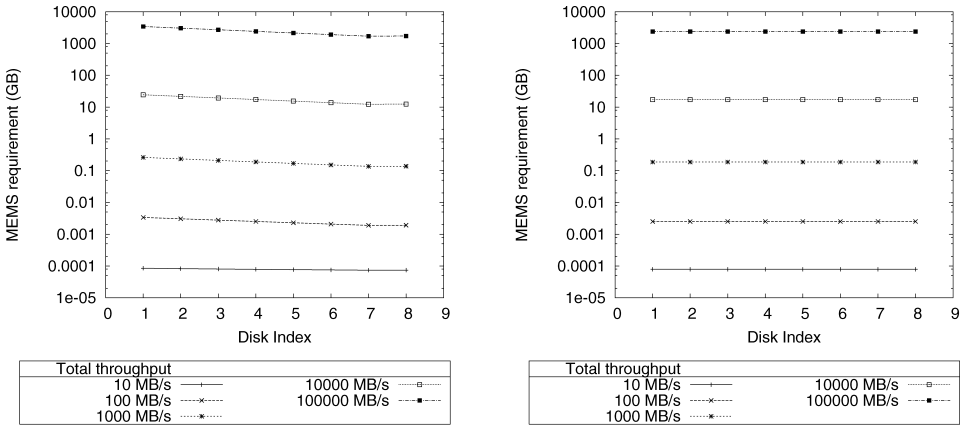

(a) Load imbalance $(i) = 2$ (b) Load imbalance $(i) = 10$

Fig. 17. Improvement in DRAM requirement of the integrated over the partitioned configuration for different load imbalance values.

expected because the load on the disks and MEMS devices increases, thereby also increasing the actual difference in load. The maximum improvement is different for different bit rates because, at these points, the total streamed bandwidths are also different. Savings in DRAM are as much as 1.47X for a load imbalance of 2 and as much as 3.3X for a load imbalance of 10. For a greater load imbalance, these values will be higher.

5.2.2 *Effect of Bit Rate Heterogeneity.* Apart from load imbalance, bit rate heterogeneity is another factor for an uneven space load on the MEMS buffer. In the case of bit rate heterogeneity, the MEMS buffering requirement for each disk is different. The integrated configuration can handle bit rate heterogeneity by distributing the MEMS buffering requirement for each disk over all the MEMS devices in the MEMS bank.

It is important to point out here that bit rate heterogeneity does not impact the total DRAM requirement. If all the disk throughputs are the same, the MEMS device throughputs are also the same. Since the total number of streams serviced by the entire system is also a constant, the product of the average IO




(a) Partitioned MEMS buffer (b) Integrated MEMS buffer

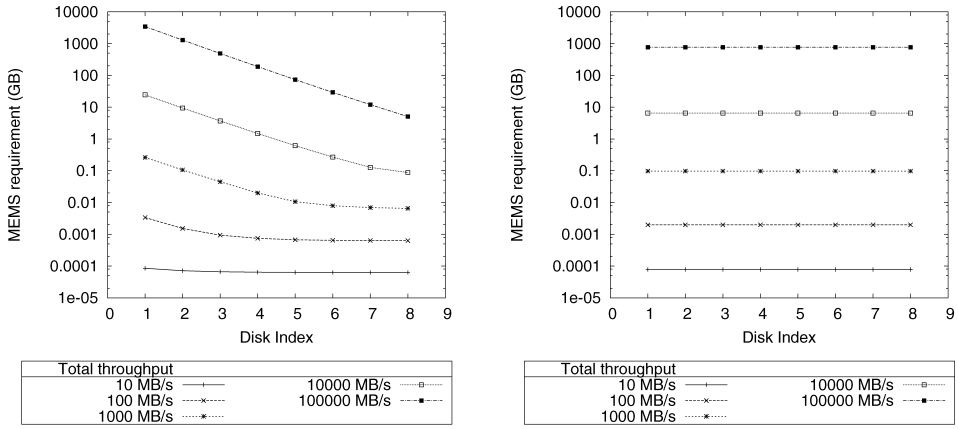
Fig. 18. MEMS requirement for individual disks for bit rate heterogeneity ($h$) = 2.

size for the entire MEMS bank and the total number of streams gives the DRAM requirement, a constant value. We now explore the effect of bit rate heterogeneity on the MEMS buffering requirement for both configurations.

To obtain Figure 18, we conduct the following experiment. Keeping the total bandwidth requirement of each disk constant, we vary the average bit rate streamed by the disks by employing bit rate heterogeneity ($h$). Disks with lower indices, stream a lower average bit rate than those with higher indices.

Figure 18 presents the MEMS buffering requirement for each disk for $h = 2$. This represents a factor of two difference between the lowest average bit rate to the highest average bit rate. In a system that serves a heterogeneous mix of streaming media, this is reasonable to expect. For the integrated case, the MEMS buffering requirement is uniform across all the MEMS devices and does not depend on individual disks. It is therefore a horizontal line for each value of the total system throughput. For the partitioned case, since the number of streams serviced by a disk with a lower average bit rate is greater, disks with lower average bit rate require more MEMS buffer space than ones which serve high bit rate streams. This variance in MEMS buffer space requirement, though small, can lead to inadequate MEMS buffer for certain disks and thereby reduce the overall throughput of the system.

Figure 19 presents the MEMS buffering requirement for each disk for $h = 1000$. If a system divides the streams over the disks based on their bit rates, some disks may service streams with significantly lower bit rate than others. There is a factor of 1000X in the bit rates of MP3s and HDTV streams. This being the case, the variance in the MEMS buffering requirement is a few orders of magnitude. With a 20GB partitioned MEMS buffer per-disk, only system throughputs lesser than 10, 000MB/s can be supported. For the integrated case, this value is higher and can be estimated in the range of approximately 25, 000MB/s.




(a) Partitioned MEMS buffer (b) Integrated MEMS buffer

Fig. 19. MEMS requirement for individual disks for $h = 1000$.

## 6. RELATED WORK

Several research efforts have focused on improving quality-of-service support for conventional disk-only storage systems. The earliest work [Rangan et al. 1992] proposed a time-sharing approach to service multiple IO streams. It proposes reservation schemes and admission control to ensure the timeliness of data delivery. The multiple MEMS and multiple-disk cooperative IO scheduling techniques we propose are based on the quality-proportional multisubscriber scheduling model presented in this study. Since then, there have been numerous efforts towards building storage systems that inherently support IO guarantees or which are highly engineered solutions for improving the storage system performance. Some of these include the work of Daigle and Strosnider [1994], Chervenak and Patterson [1995], Wolf et al. [1995], Molano et al. [1997], Chang and Garcia-Molina [1997], To and Hamidzadeh [1999], Santos et al. [2000], Shahabi et al. [2002], Dimitrijevic and Rangaswami [2003], Dimitrijevic et al. [2003], Denehy et al. [2002], Schindler et al. [2004], and Liu et al. [2006].

The augmented hierarchy we explore in this study is reminiscent of prior efforts that explored integration of DRAM, disk, and tape into an online storage hierarchy [Chervenak 1994; Hillyer and Silberschatz 1996]. The study of Chervenak [1994] specifically pointed out the unsuitability of such an augmented online storage hierarchy (with tapes at the tertiary level) for the class of streaming media applications. In contrast, in this study, we argue in favor of an intermediate storage layer between disk drives and DRAM and demonstrate the gains it affords when serving streaming media data. Our findings also echo the observations of Hsu and Smith [2004] who suggest that a reliable method for improving performance is to use larger caches up to and even beyond 1% of the storage used.

The approach of using assistive storage technology has been proposed before for non-real-time data by several researchers. Schlosser et. al. [2000] have proposed using MEMS-based storage devices as a disk cache and have reported a

3.5X improvement in IO response time for best-effort data. Uysal et al. [2003] report that replacing disk arrays partially or entirely with MEMS-based storage improves storage latency as well as throughput significantly for file system and database traces over conventional arrays. Hong et al. [2006] also report about lower power consumption of MEMS-based storage devices compared to disk drives. In Yu et al. [2003], the authors exploit the two-dimensional nature of MEMS-storage access and develop a tabular data placement scheme for relational data. They show that using MEMS storage can improve IO utilization and cache performance for relational data. In a recent study, the authors have also demonstrated data placement techniques on the MEMS device for the general problem of declustering two-dimensional data [Yu et al. 2004]. Hong et. al. [2006] recently proposed using MEMS storage as an aid to boost disk performance as a read cache as well as write buffer for disk IOs.

The above body of work on MEMS-based storage has focused on improving storage performance for non-real-time data. We introduced using MEMS-storage for buffering real-time streaming data in single-drive systems in an earlier study [Rangaswami et al. 2003]. In this study, we extend our previous work to address the problem of managing multiple disks and MEMS-based storage banks. We propose and evaluate data allocation and real-time IO scheduling strategies for a MEMS-disk heterogeneous storage system.

## 7. CONCLUSION

MEMS-based storage is an emerging technology which is predicted to be available in the near future. This study demonstrates the potential of using this new storage technology in building high-performance streaming storage systems.

To incorporate MEMS storage into the existing hierarchy, we proposed the partitioned and integrated MEMS-augmented storage architectures. We also proposed abstractions for these architectures and used these abstractions to design data buffering and real-time I/O scheduling policies in a multilayer storage hierarchy. We developed theoretical models for analyzing the performance of storage systems with single or multiple disks and MEMS-based storage banks. For both single and multidisk systems, using an intermediate MEMS storage buffering layer can significantly reduce the overall cost of buffering large IOs as well as improve overall streaming throughput of the server. We further demonstrated in a multidisk setting that the integrated MEMS buffer configuration can handle both load imbalance and bit rate heterogeneity remarkably well.

The proposed extended storage architectures, their corresponding abstractions, and the proposed model for analyzing real-time streaming cost performance characteristics can also apply to similar hierarchical storage architectures such as DRAM-disk-disk or DRAM-CF-disk hierarchies. We believe that the findings of this study can serve as guidelines for building next-generation streaming media servers.

## REFERENCES

ANDERSON, D., DYKES, J., AND RIEDEL, E. 2003. More than an interface—SCSI vs. ATA. *Proceedings of the 2nd USENIX Conference on File and Storage Technologies.* 245–257.

CARLEY, L. R., GANGER, G. R., AND NAGLE, D. 2000. MEMS-based integrated-circuit mass-storage systems. *Comm. ACM 43*, 11 (Nov), 73–80.

CHANG, E. AND GARCIA-MOLINA, H. 1997. Effective memory use in a media server. *Proceedings of the 23rd VLDB Conference*. 496–505.

CHERVENAK, A. L. 1994. Tertiary storage: An evaluation of new applications. Tech. rep. No. UCB/CSD-94-847. University of California, Berkeley.

CHERVENAK, A. L. AND PATTERSON, D. A. 1995. Choosing the best storage system for video service. *Proceedings of ACM Multimedia*. 109–118.

DAIGLE, S. J. AND STROSNIDER, J. K. 1994. Disk scheduling for multimedia data streams. *Proceedings of the IS&T/SPIE Conference*.

DENEHY, T. E., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. 2002. Bridging the information gap in storage protocol stacks. *Proceedings of the USENIX Annual Technical Conference*. 177–190.

DIMITRIJEVIC, Z. AND RANGASWAMI, R. 2003. Quality of service support for real-time storage systems. *Proceedings of International IPSI Conference*.

DIMITRIJEVIC, Z., RANGASWAMI, R., AND CHANG, E. 2003. Design and implementation of semi-preemptible IO. *Proceedings of Usenix File and Storage Technologies Conference*.

GROCHOWSKI, E. AND HALEM, R. D. 2003. Technological impact of magnetic hard disk drives on storage systems. *IBM Sys. J. 42*, 2 (April) 338–346.

HILLYER, B. K. AND SILBERSCHATZ, A. 1996. Random I/O scheduling in online tertiary storage systems. *Proceedings of the 1996 ACM SIGMOD*. 195–204.

HONG, B., BRANDT, S. A., LONG, D. D. E., MILLER, E., AND LIN, Y. 2006. Using MEMS-based storage in computer systems : Device modeling and measurement. *ACM Trans. Storage 2*, 2 (May) 139–160.

HONG, B., WANG, F., BRANDT, S. A., LONG, D. D. E., AND SCHWARZ, T. J. E. 2006. Using MEMS-based storage in computer systems : MEMS storage architectures. *ACM Trans. Storage 2*, 1 (Feb) 1–21.

HSU, W. AND SMITH, A. J. 2004. The performance impact of I/O optimizations and disk improvements. *IBM J. Resear. Develop. 48*, 2 (March) 255–289.

LIU, B., RANGASWAMI, R., AND DIMITRIJEVIC, Z. 2006. Stream combination: Adaptive IO scheduling for streaming servers. *ACM SigBED Rev. 3*, 1 (Jan).

MAKAROFF, D., NEUFELD, G., AND HUTCHINSON, N. 1997. An evaluation of VBR disk admission algorithms for continuous media file. *Proceedings of the 5th ACM Multimedia Conference*. 143–154.

MAXTOR CORPORATION. 2002. Atlas 10KIII-U320 product datasheet.

MOLANO, A., JUVVA, K., AND RAJKUMAR, R. 1997. Guaranteeing timing constraints for disk accesses in RT-mach. *Proceedings of the Real-Time Systems Symposium*.

NANOCHIP INC. 2006. Nanochip secures $10 million in series C funding led By Intel capital. Nanochip Media Release (http://www.nanochip.com/pr/pr20060418.htm.

RAMBUS INC. 2006. RDRAM. http://www.rambus.com/.

RANGAN, P. V., VIN, H. M., AND RAMANATHAN, S. 1992. Designing and on-demand multimedia service. *IEEE Comm. Mag. 30*, 7 (July) 56–65.

RANGASWAMI, R., DIMITRIJEVIC, Z., CHANG, E., AND SCHAUSER, K. E. 2003. MEMS-based disk buffer for streaming media servers. *Proceedings of IEEE International Conference on Data Engineering*. 619–630.

SANTOS, J. R., MUNTZ, R. R., AND RIBEIRO-NETO, B. A. 2000. Comparing random data allocation and data striping in multimedia servers. *Proceedings of the International Conference on Measurement and Modeling of Computer Systems*. 44–55.

SCHINDLER, J., SCHLOSSER, S. W., SHAO, M., AILAMAKI, A., AND GANGER, G. R. 2004. Atropos: A disk array volume manager for orchestrated use of disks. *Proceedings of the USENIX Conference on File and Storage Technologies*.

SCHLOSSER, S. W. AND GANGER, G. R. 2004. Mems-based storage devices and standard disk interfaces: A square peg in a round hole? *Proceedings of the USENIX Conference on File and Storage Technologies*.

SCHLOSSER, S. W., GRIFFIN, J. L., NAGLE, D., AND GANGER, G. R. 2000. Designing computer systems with MEMS-based storage. *Proceedings of Architectural Support for Programming Languages and Operating Systems*. 1–12.

SHAHABI, C., GHANDEHARIZADEH, S., AND CHAUDHURI, S. 2002. On scheduling atomic and composite multimedia objects. *IEEE Trans. Knowl. and Data Engin. 14*, 2, 447–455.

THOMPSON, D. A. AND BEST, J. S. 2000. The future of magnetic data storage technology. *IBM J. Resear. Develop. 44*, 3 (May).

TO, T.-P. J. AND HAMIDZADEH, B. 1999. Dynamic real-time scheduling strategies for interactive continuous media servers. *ACM/Springer Multimedia Sys. 7*, 2, 91–106.

UYSAL, M., MERCHANT, A., AND ALVEREZ, G. A. 2003. Using MEMS-based storage in disk arrays. *Proceedings of Usenix Conference on File and Storage Technologies*. 89–101.

VETTIGER, P., DESPONT, M., DRECHSLER, U., DURIG, U., HABERLE, W., LUTWYCHE, M. I., ROTHUIZEN, H. E., STUTZ, R., WIDMER, R., AND BINNING, G. K. 2000. The "Millipede"—More than one thousand tips for future AFM data storage. *IBM J. Resear. Develop. 44*, 3, 323–340.

WOLF, J. L., YU, P. S., AND SHACHNAI, H. 1995. DASD Dancing: A disk load balancing optimization scheme for video-on-demand computer systems. *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*. 157–166.

YU, H., AGRAWAL, D., AND ABBADI, A. E. 2003. Tabular placement of relational data on MEMS-based storage devices. *Proceedings of the International Conference on Very Large Data Bases*.

YU, H., AGRAWAL, D., AND ABBADI, A. E. 2004. Declustering two-dimensional datasets over MEMS-based storage. *Proceedings of the International Conference on Extending DataBase Technology*.

Received May 2006; revised March 2007; accepted April 2007