MEMS-based Disk Buffer for Streaming Media Servers*

Raju Rangaswami
raju@cs.ucsb.edu

Zoran Dimitrijević
zoran@cs.ucsb.edu

Edward Chang
echang@ece.ucsb.edu

Klaus E. Schauser
schauser@cs.ucsb.edu

University of California, Santa Barbara

Abstract

The performance of streaming media servers has been limited due to the dual requirements of high throughput and low memory use. Although disk throughput has been enjoying a 40% annual increase, slower improvements in disk access times necessitate the use of large DRAM buffers to improve the overall streaming throughput. MEMS-based storage is an exciting new technology that promises to bridge the widening performance gap between DRAM and disk-drives in the memory hierarchy. This paper explores the impact of integrating these devices into the memory hierarchy on the class of streaming media applications. We evaluate the use of MEMS-based storage for buffering and caching streaming data. We also show how a bank of k MEMS devices can be managed in either configuration and that they can provide a k-fold improvement in both throughput and access latency. An extensive analytical study shows that using MEMS storage can reduce the buffering cost and improve the throughput of streaming servers significantly.

1 Introduction

Applications such as news or video on demand, distance learning, scientific visualization, and immersive virtual reality must store, maintain, and retrieve large volumes of real-time data. These data are collectively referred to as *continuous* or *streaming* media. They require storage architectures that can accommodate their real-time delivery constraints as well as their large sizes. Economical delivery of these data requires that such architectures also provide high disk throughput and minimize memory usage.

Even after the long reign of Moore's Law, the basic memory hierarchy in computer systems has not changed significantly. At the non-volatile end, magnetic disks have managed to survive as the most cost-effective mass storage medium, and there are no alternative technologies which show promise for replacing them in the next decade [20]. Disk access times, however, are improving at the rate of only 10% per year. For more than a decade they have continued to lag behind the annual disk throughput increase of 40% and capacity increase of 60% [20]. Due to the increasing gap between the improvements in disk bandwidth and disk access times (both seek time and rotational delay), achieving high disk throughput necessitates accessing the disk drive in larger chunks. This translates to a rapidly-increasing DRAM buffering cost. A large DRAM buffer is especially necessary for servers which stream to a large number of clients. Multimedia server architects have tried to cope with this performance gap by proposing solutions ranging from simple resource trade-offs [13, 25] to more complex ones that require substantial engineering effort [2, 7, 10].

Micro-electro-mechanical-systems (MEMS) based storage is an emerging technology that promises to bridge the performance gap between magnetic disks and DRAM [1]. 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 [16]. These devices offer a unique low-cost solution for streaming applications. In this study, we propose an analytical framework to evaluate the effective use of MEMS devices in a streaming media server. Specifically, we derive analytical models for studying two MEMS configurations, *using MEMS storage as a buffer* between DRAM and disk, and *using MEMS storage as a cache*.

- *MEMS buffer*. When MEMS storage is used as a speed-matching buffer between the disk drive and DRAM, all data retrieved from the disk to DRAM are first retrieved into the MEMS buffer and then transferred to DRAM.
- *MEMS cache*. When MEMS storage is used as a cache, it stores popular multimedia streams in their entirety.

MEMS-based storage provides access characteristics superior to those of disk drives, thus reducing the DRAM

*This research was supported by SONY/UC DiMI and the NSF CISE Infrastructure grant EIA-0080134.

buffering requirement. In addition, being a magnitude cheaper than DRAM, MEMS devices can improve the disk throughput by providing low-cost buffering for large disk IOs. Caching popular content on MEMS storage can also reduce the DRAM buffer requirement and improve the total streaming throughput of the media server. At first glance, the issues involved in placing MEMS storage between the DRAM and the disk-drive, either as a buffer or as a cache, seem straightforward enough. However, the real-time IO requirement of the media data and the mismatch between the transfer rates of disk and DRAM make some MEMS storage configurations unfeasible or counter-productive. Our extensive analytical and empirical studies reveal the following design principles: (i) when used to buffer streaming data, MEMS storage must be used to buffer only low and medium bit-rate streams, (ii) When the streaming content has a non-uniform popularity distribution, MEMS-based disk caching can improve the server throughput regardless of the bit-rate of the streams serviced. We shall examine these principles further in the experimental section.

This work has led to several research contributions. Primarily, it takes the first step toward understanding the role of MEMS-based storage devices in next-generation streaming multimedia servers. In particular, we make the following contributions:

1. We propose using MEMS devices in two possible configurations: *MEMS as a buffer* and *MEMS as a cache*. We also show how a bank of k MEMS devices can be managed in either configuration and that they can provide a k-fold improvement in both throughput and access latency.

2. We develop an analytical framework for guaranteeing real-time constraints in the presence of the additional MEMS-based storage layer in the memory hierarchy.

3. Based on our evaluation, we provide guidelines for designing the next generation of streaming media servers using MEMS devices.

The rest of this paper is organized as follows: Section 2 briefly explains one possible architecture for the MEMS storage and presents current predictions for future DRAM, disk, and MEMS storage characteristics. Section 3 introduces MEMS storage as an intermediate buffer as well as a cache for streaming data. In Section 4 we present a quantitative model for analyzing the two MEMS storage configurations. Section 5 presents a performance evaluation of the MEMS-based streaming server architecture based on our analytical model. Section 6 presents related research. In Section 7, we suggest directions for future work.

2 MEMS-based Storage

Researchers at the Carnegie Mellon University have proposed one possible architecture for a MEMS-based storage device [5, 16] depicted in Figure 1. They propose MEMS devices which would be fabricated on-chip, but would use a spring-mounted magnetic media sled as a storage medium. The media sled is placed above a two-dimensional array of micro-electro-mechanical read/write heads (tips). Actuators move the media sled above the array of fixed tips along both the X and Y dimensions. Moving 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 (of the order of thousands) concurrently, such a device can deliver high data throughput. The light-weight 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.

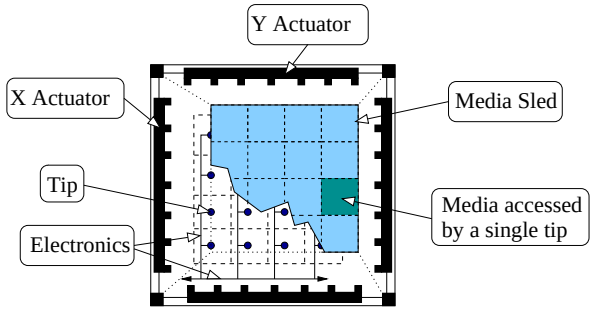


Figure 1. MEMS-based storage architecture.

Year		DRAM	MEMS	Disk
2002	Capacity [GB]	0.5	n/a	100
	Access time [ms]	0.05	n/a	$1 - 11$
	Bandwidth [MB/s]	$2,000$	n/a	$30 - 55$
	Cost/GB	$200	n/a	$2
	Cost/device	$50- $200	n/a	$100- $300
2007	Capacity [GB]	5	10	$1,000$
	Access time [ms]	0.03	$0.4 - 1$	$0.75 - 7$
	Bandwidth [MB/s]	$10,000$	320	$170 - 300$
	Cost/GB	$20	$1	$0.2
	Cost/device	$50- $200	$10	$100- $300

Table 1. Storage media characteristics.

Table 1 summarizes important characteristics of different storage media for the year 2002 and the predicted values for the year 2007. The MEMS device projections are borrowed from [16]; the disk drive projections are based on [20]; and DRAM predictions are based on [12].

Typically, most storage media are optimized for sequential access. For instance, 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. However, both magnetic disks and MEMS-based storage devices (MEMS) have much longer access times than DRAM. These devices have to be accessed in much larger chunks to mask the access overheads, the MEMS device being the faster of the two by an order of

magnitude. Figure 2 shows the effective throughput of the disk drive and the MEMS device depending on the average IO sizes on these devices. In Figure 2 we use the maximum access times for servicing MEMS IO requests, and the average access times for disk IO requests.

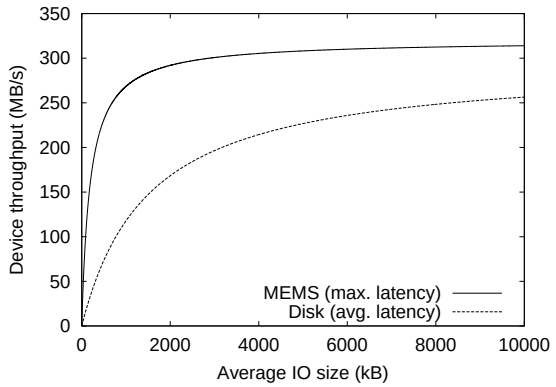


Figure 2. Effective device throughputs.

3 MEMS-based Streaming

Streaming data is characterized by the dual requirements of guaranteed-rate IO and high throughput. To service multiple streams, the disk-drive bandwidth is time-shared among the streams. However, this time-sharing degrades disk throughput. Any system designed for servicing streaming data must address the inherent trade-off between disk throughput and data buffering requirements.

In this paper, we adopt the time-cycle based scheduling model [13] for scheduling continuous media streams. In this model, time is split into basic units called IO cycles. In each IO cycle, the disk performs exactly one IO operation for each media stream. Given the stream bit-rates, the *IO cycle time* is the amount of time required to transfer sufficient amount of data for each stream so as to sustain jitter-free playback. The IO cycle time depends on the system configuration as well as the number and type of streams requested. To service multiple streams, the IO scheduler services the streams in the same order in each time-cycle. Careful management of data buffers and precise scheduling [2] can reduce the total amount of buffering required to the amount of data read in one time-cycle.

In traditional multimedia servers, the buffering requirement is addressed using the system memory (DRAM). MEMS-based storage devices can be used to offload part of the DRAM buffering requirement. They can also be used as caches to provide faster access to multimedia content. Figure 3 illustrates the new system architecture that includes the MEMS device in the storage hierarchy. The MEMS storage module can consist of multiple MEMS devices to provide greater storage capacity and throughput. The MEMS device

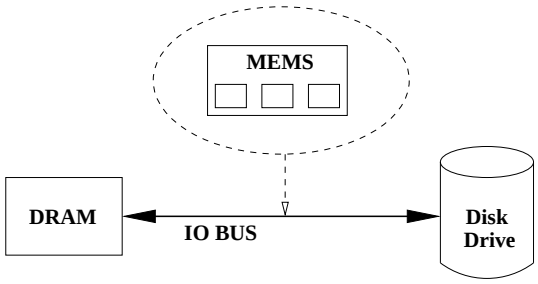


Figure 3. The MEMS Architecture

is accessed through the IO bus. It can be envisioned as part of the disk drive or as an independent device. In either case, IOs can be scheduled on the MEMS device as well as on the disk drive independently. Similar to disk caches found on current-day disk drives, we assume that MEMS storage devices would also include on-device caches. In what follows, we present two possible scenarios in which such an architecture can be used to improve the performance of a multimedia system.

3.1 MEMS Multimedia Buffer

Using MEMS storage as an intermediate buffer between the disk and DRAM enables the disk drive to be better utilized. At a fraction of the cost of DRAM, MEMS storage can provide a large amount of buffering required for achieving high disk utilization (see Figure 2). Although DRAM buffering cannot be completely eliminated, the low access latency of MEMS storage provides high throughput with significantly lesser 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 being 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-based service model previously proposed for disk drives. 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 being written to the MEMS device is equal to the amount read from it. The MEMS band-

width is thus split equally among read and write operations. 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 co-operate.

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. We now present a feasible IO schedule that maintains real-time guarantees, when a single MEMS device is used for buffering. We then extend our methodology to work with a bank of MEMS devices.

3.1.1 IO Scheduling: Single MEMS

We now present one possible IO scheduler which guarantees 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*. In each 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. 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. Thus, 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).

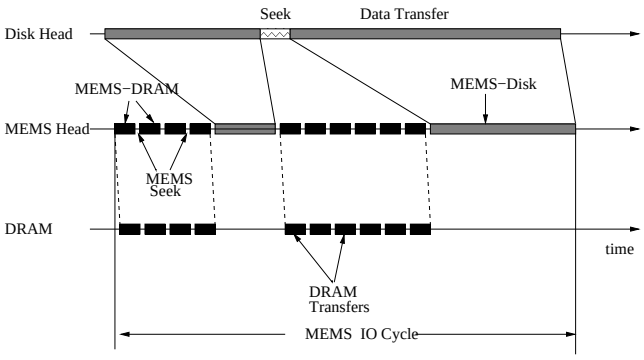


Figure 4. MEMS IO Scheduling.

Figure 4 describes the data transfer operations occurring during a single MEMS IO cycle. 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.

3.1.2 IO Scheduling: Multiple MEMS

According to certain predictions [16, 22], 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.

Placing stream data: Buffered data can be placed in one of two ways: stripe the buffered data for each stream across the MEMS bank or buffer each stream 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.

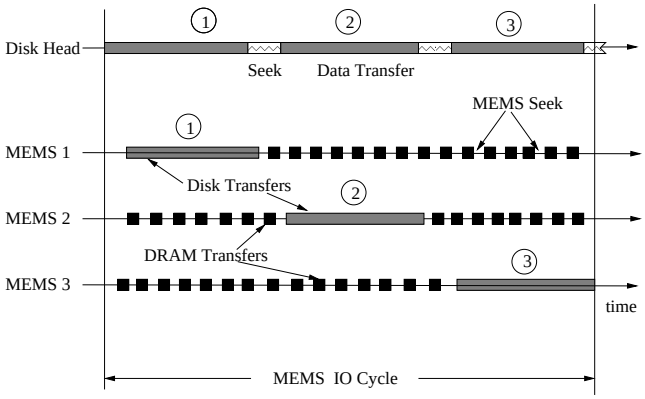


Figure 5. IO Scheduling for a MEMS bank.

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 can also be used to cache entire streams, as we shall explore next. Spare bandwidth, if available, can be used for non-real-time traffic.

3.2 MEMS Multimedia Cache

Multimedia content usually has a well-defined popularity distribution, and some content are accessed more frequently than others. Besides using MEMS storage as a buffer for streaming multimedia data from the disk drive, we can also use a MEMS storage device as a cache for popular multimedia content. Since the MEMS device offers low latency data access at throughput levels similar to those of the disk-drive, storing popular content on a MEMS cache reduces the buffering requirement for streaming data and hence DRAM cost. By using k MEMS devices, we can also use the aggregate bandwidth of the MEMS cache for improving the server throughput.

One significant difference between using a MEMS device as a cache and using it as a buffer is that with caching, the MEMS cache behaves primarily as a read-only device. The MEMS cache is updated only to account for changes in stream popularity. This can be accomplished off-line, during service down-time. To service streams from the cache, we use time-cycle-based IO scheduling. Again, there exist two distinct IO cycles, one for the streams serviced from the disk-drive and the other for those serviced from the MEMS cache. The performance of either device depends on the available DRAM buffering and the number of streams serviced from it.

When more than one MEMS device is used for caching, the performance of the MEMS cache depends on the data management policy used for the MEMS bank. With multiple devices, we must ensure that the load on the MEMS devices is balanced. In this regard, we can draw on research from data management policies for disk arrays [3]. We now investigate two cache-management policies, representing two classes of load balancing strategies, which ensure total load-balance across the MEMS bank. These approaches make different trade-offs to optimize for a sub-set of system configurations. More sophisticated load-balancing strategies, including hybrid approaches of the above, have been proposed in literature [3, 15, 23, 24]. We investigate two simple, representative approaches as a first step.

3.2.1 Striped Cache-management

Using *striped cache-management*, each stream is bit- or byte-striped across all the k MEMS devices. There is no redundancy, and data for each stream is distributed in round-robin fashion across the MEMS devices. To perform an IO on the MEMS cache, all the devices access exactly the same relative data location, in a lock-step fashion. The load is thus perfectly balanced across the MEMS cache. The effective data transfer rate of the MEMS cache is k times that of a single device. The effective access latency of the MEMS cache is the same as that of a single device. If N_m streams are serviced from the MEMS cache, the total number of seek operations in an IO cycle is $k \cdot N_m$. Using striped cache-management, perfect load-balancing is achieved at the cost of reduced access parallelism of the devices.

3.2.2 Replicated Cache-management

Using *replicated cache-management*, the cached streams are replicated on the k MEMS devices. All the devices store exactly the same content. To perform an IO on the MEMS cache, any of the k devices can be accessed. If the number of streams serviced from the MEMS cache is N_m, then each MEMS device services exactly $\frac{N_m}{k}$ of the streams. Since each MEMS device stores all the cached streams, such a division is possible. The effective data transfer rate of the MEMS cache is k times that of a single device. Operating the k devices independently improves parallelism of access. The total number of seek operations in an IO cycle is only N_m as opposed to $k \cdot N_m$ in striped cache-management. Using replicated cache-management, perfect load balancing is achieved at the cost of reduced cache size due to data redundancy. In Section 4.2, we further analyze the trade-offs involved in each policy.

4 Quantitative Analysis

In this section we present a quantitative model to analyze buffering requirements for systems supporting real-time streaming applications using MEMS devices. We discuss two system configurations:

- *MEMS buffer*. All data retrieved from the disk to DRAM are first retrieved into the MEMS buffer and then transferred to DRAM.
- *MEMS cache*. Selected data are cached on the MEMS device. Requested data can be serviced either from the disk-drive or the MEMS cache.

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 $[B]$
$S_{disk-dram}$	Average IO size from disk to DRAM $[B]$
$S_{disk-mems}$	Average IO size from disk to MEMS $[B]$
$S_{mems-dram}$	Average IO size from MEMS to DRAM $[B]$
T_{disk}	Disk IO cycle $[s]$
T_{mems}	MEMS IO cycle $[s]$

Table 2. Parameter definitions.

To evaluate the effectiveness of MEMS-based buffering and caching, 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. The proofs for the results presented in this section can be found in [14].

4.1 MEMS Multimedia Buffer

Let $S_{disk-dram}$ denote the per-stream IO size from disk to DRAM, $S_{disk-mems}$ from disk to MEMS, and $S_{mems-dram}$ 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

$$COST_{without\_mems} = N \times C_{dram} \times S_{disk-dram} \quad (1)$$

$$COST_{with\_mems} = k \times C_{mems} \times Size_{mems} + N \times C_{dram} \times S_{mems-dram} \quad (2)$$

where $k \times Size_{mems} \geq N \times S_{disk-mems}$. 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 make two assumptions that are commonly used in modeling a media server. First, we use time-cycle-based IO scheduling (Section 3). Second, 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 2.

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

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

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

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

Although Theorem 1 is well established [13], 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 per-stream DRAM buffer required to satisfy real-time requirements is

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

$$where\ C = \frac{N \times \bar{L}_{mems} \times R_{mems}}{k \times R_{mems} - 2 \times (N + k - 1) \times \bar{B}}.$$

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

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

[1] VBR can be modeled by CBR plus some memory cushion for handling bit-rate variability [8].

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

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

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.

4.2 MEMS Multimedia Cache

Although streaming data do not have temporal locality, they are often characterized by a non-uniform popularity distribution. Caching popular content in MEMS storage can decrease the DRAM buffering requirement so that the system can support more streams. Let h denote the hit-rate for the MEMS cache. Given N streams to service, $n = N \times h$ of them are serviced from the MEMS cache, and $N \times (1-h)$ streams are serviced from the disk. We can express the cost of DRAM buffer and MEMS cache as

$$\begin{aligned} COST_{with\_mems-cache} = k \times C_{mems} \times Size_{mems} \ + \\ h \times N \times C_{dram} \times S_{mems-dram} \ + \\ (1-h) \times N \times C_{dram} \times S_{disk-dram}. \end{aligned} \quad (9)$$

Using Theorem 1 we can calculate $S_{disk-dram}$ as

$$S_{disk-dram} = \frac{(1-h) \times N \times \bar{L}_{disk} \times R_{disk} \times \bar{B}}{R_{disk} - (1-h) \times N \times \bar{B}}, \quad (10)$$
$$where \ R_{disk} > (1-h) \times N \times \bar{B}.$$

In order to calculate h and $S_{mems-dram}$, let us assume that the popularity distribution of content is specified by $X : Y$, where $X\%$ of the streams are accessed $Y\%$ of the time. Let us assume that both popular ($X\%$) and non-popular streams ($100\% - X$) are accessed uniformly in their class. The capacity of the disk, $Size_{disk}$, is the total storage required for all the streams serviced by the system. Let the capacity of a single MEMS device be denoted by $Size_{mems}$. Let the percentage of movies cached be denoted by p. Then the cache hit ratio, h, can be expressed as

$$h = \begin{cases} \frac{p}{X} \times \frac{Y}{100\%} & \text{if } X \geq p, \\ \frac{Y}{100\%} + \frac{p-X}{100\%-X} \times \frac{100\%-Y}{100\%} & \text{otherwise.} \end{cases} \quad (11)$$

If the MEMS cache contains a single MEMS device, then p and $S_{mems-dram}$ (using Equation 4) are

$$p = \frac{SIZE_{mems}}{SIZE_{disk}} \ ;$$

$$S_{mems-dram} = \frac{n \times \bar{L}_{mems} \times R_{mems} \times \bar{B}}{R_{mems} - n \times \bar{B}}$$

However, if the cache consists of more than one MEMS device, both p and $S_{mems-dram}$ depend on the cache management policy used for accessing the MEMS cache. We explore the two policies, namely *striped* and *replicated*, introduced in Section 3.2.

4.2.1 Striped Cache Management

Theorem 3. For a server that employs the striped cache-management policy across an array of k MEMS devices for serving n streams, the minimum size of per-stream DRAM buffer required to satisfy real-time streaming requirements is

$$S_{mems-dram} = \frac{n \times \bar{L}_{mems} \times (k \times R_{mems}) \times \bar{B}}{(k \times R_{mems}) - n \times \bar{B}}, \quad (12)$$
$$where \ k \times R_{mems} > n \times \bar{B}.$$

Corollary 3. A striped cache, consisting of k MEMS devices, behaves as a single MEMS cache with k times larger throughput and unchanged access latency.

4.2.2 Replicated Cache Management

Theorem 4. For a server that employs the replicated cache-management policy across an array of k MEMS devices for serving n streams, the minimum size of per-stream DRAM buffer required to satisfy real-time streaming requirements is

$$S_{mems-dram} = \frac{(n+k-1)\frac{\bar{L}_{mems}}{k}(k \cdot R_{mems}) \times \bar{B}}{(k \cdot R_{mems}) - (n+k-1) \times \bar{B}}$$
$$where \ k \cdot R_{mems} > (n+k-1) \times \bar{B}. \quad (13)$$

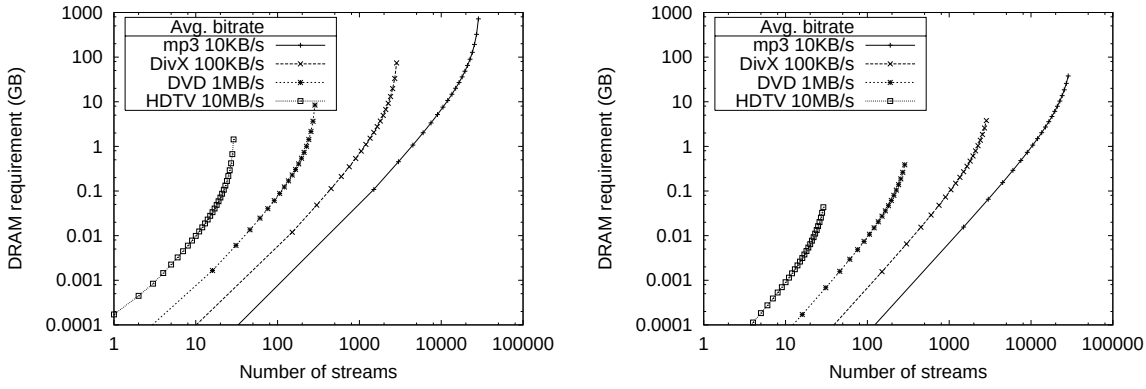
Corollary 4. When N is divisible by k (or is relatively large compared to k), k replicated cache behaves as a single MEMS device with k times larger throughput as well as k times smaller average latency.

5 Experimental Evaluation

This section presents an experimental evaluation of a streaming multimedia system equipped with MEMS storage. We compare its performance to that of a system without MEMS storage. We evaluate separately the performance of the MEMS device when it is used to buffer streaming data stored on the disk drive, and when it is used to cache popular streams. The evaluation presented in this section is based on the analytical model presented in Section 4.

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

Table 3. Performance characteristics of storage devices in the year 2007.



(a) Without MEMS buffer

(b) With MEMS buffer

Figure 6. DRAM requirement for various media types.

To model the performance of the MEMS device, we closely followed one such model proposed by researchers at Carnegie Mellon University. The work in [16] describes their model comprehensively. For our experiments, we use the "3rd Generation" (G3) MEMS device model proposed in [16]. We obtained predictions for disk-drive and DRAM performance by using projections on current-day devices produced by Maxtor [9] and Rambus [12], respectively. These are summarized in Table 3.

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 mis-prediction 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 MEMS Multimedia Buffer

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 640 MB/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 the above metrics to variations in MEMS device characteristics.

Theorems 1 and 2 presented in Section 4 define the relationship between the system parameters. To study the sensitivity of our evaluation to MEMS device characteristics, we introduce the *latency ratio*, $\frac{\bar{L}_{disk}}{\bar{L}_{mems}}$, as a tunable parameter.

Latency ratio: 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 1 to 10. The value for this parameter is around 5 for the FutureDisk and the G3 MEMS device listed in Table 3.

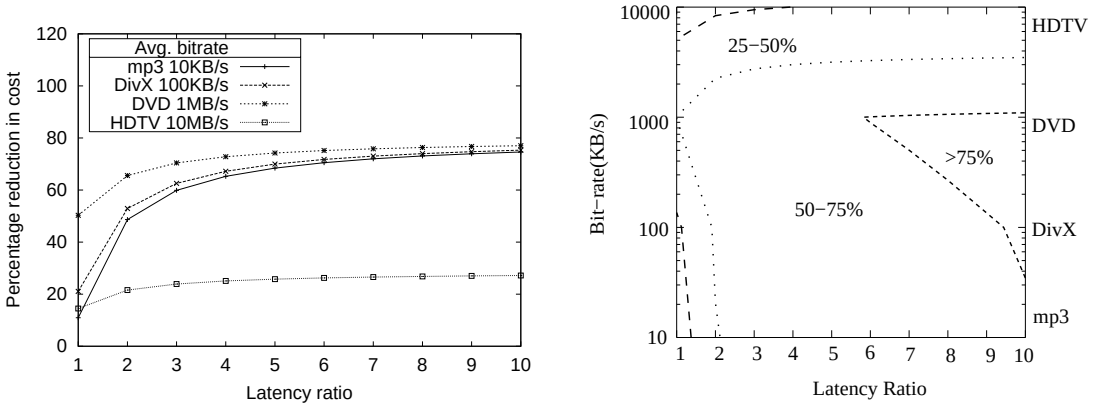
For performance evaluation, we conducted three experiments. In the first two experiments, we assumed that the maximum amount of DRAM and MEMS storage was unlimited. We also used a cost-per-byte price model for MEMS storage. These relaxations allowed us to observe the relationship between the system parameters. In the third experiment, we performed a case-study using an "off-the-shelf" system which could be developed by the year 2007. 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 6, 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.

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 reaching a conclusion about the benefits. To calculate



(a) Cost-reduction curves

(b) Cost-reduction regions

Figure 7. Percentage cost reduction.

the cost of buffering, we use cost predictions as presented in Table 3. According to the predictions, MEMS buffering is 20 times cheaper than DRAM buffering per-byte.

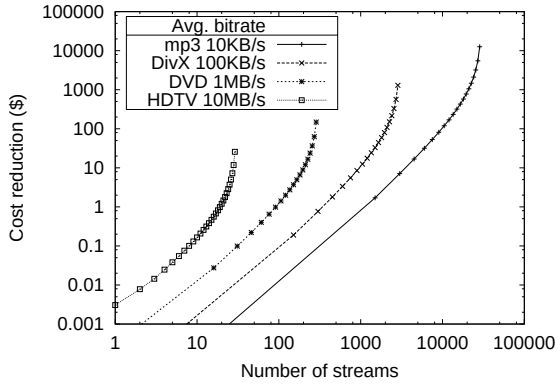


Figure 8. Reduction in the total buffering cost.

Figure 8 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 6, 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, $\bar{B}$.

5.1.3 A Parameter Sensitivity Study

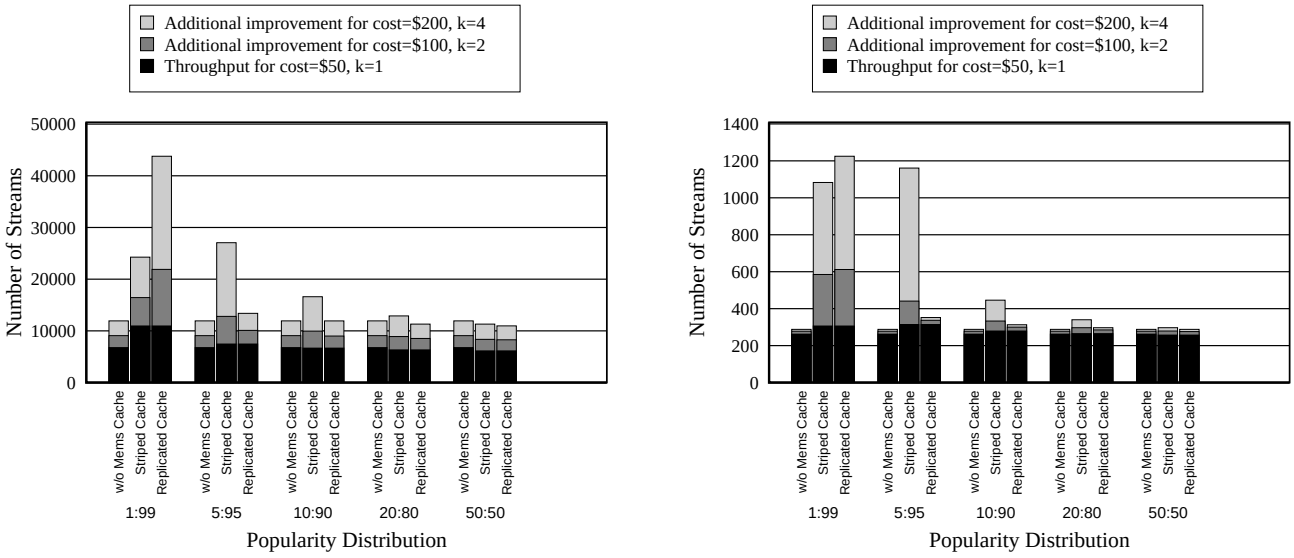
We now present a hypothetical off-the-shelf system which could be developed in the future. We determine the sensitivity of cost-reductions to unpredictable trend changes in device characteristics by varying the *latency-ratio*.[2]

In our earlier experiments, we assumed that the system could use an unlimited amount of DRAM and/or MEMS storage. However, a system with terabytes of DRAM and/or buffering would be prohibitively expensive. In the following experiment, we restricted the system configuration to an off-the-shelf system projected for the year 2007 (Table 3). The maximum DRAM size is restricted to $5GB$ and $20GB$ of MEMS buffering is used by adding two MEMS devices. The cost of the MEMS buffer is $\$20$, thereby restricting the cost-savings to a maximum of 80%.

Figure 7(a) depicts the percentage reduction in the buffering cost for different average stream bit-rates when the latency ratio is varied. As the latency-ratio is increased, the MEMS device can deliver higher throughputs with less buffering. As a result, the buffering cost is reduced. When the average bit-rate is increased, the performance of both the disk drive and the MEMS device improves. The differential improvement in throughput for the MEMS device is greater. However, beyond a certain average bit-rate (1MB/s in this case), the DRAM requirement in the absence of the MEMS buffer is small, and the available 5GB DRAM buffer is under-utilized even without the MEMS buffer. In the absence of a MEMS buffer, the DRAM requirement for the 10MB/s bit-rate range is approximately 1.5GB, thereby limiting the cost-reduction to only 30%.

In Figure 7(b), contour lines on the XY-plane depict the regions in which the percentages of reduction in total buffer-

[2]We also investigated the sensitivity of the MEMS buffer performance to the cost and bandwidth values. Our conclusion (that MEMS buffering is effective for low and medium bit-rate traffic) holds true as long as the MEMS device is an order of magnitude cheaper than DRAM and provides streaming bandwidths comparable to or greater than those of disk-drives.



(a) Average bit-rate 10KB/s

(b) Average bit-rate 1MB/s

Figure 9. MEMS cache performance.

ing costs are 25%, 50%, and 75%. The MEMS buffer is cost-effective almost over the entire parameter space, with at least a 50% to 75% reduction in the total buffering cost.

5.2 MEMS Multimedia Cache

When streaming content has a non-uniform access probability, then a MEMS cache for popular streams can improve the performance of a multimedia server as analyzed in Section 4. To measure the performance of the MEMS cache, we calculated the improvement in the server throughput (number of additional streams serviced) using a MEMS cache by fixing the total cost of the system. The additional cost of using a MEMS cache is reflected by a reduced amount of available DRAM buffering. We evaluated the performance of the MEMS cache using the two cache-management policies described in Section 3: *striped* and *replicated*.

The number of streams serviced from the MEMS device depends on the number of cache hits (it follows Equation 9). In our experiments, we assumed that if a stream was found in the cache, it would be serviced only from the cache[3].

The performance of the MEMS cache depends on four parameters: the popularity distribution, the total system cost, the average stream bit-rate, and the size of the MEMS bank. To evaluate the impact of these parameters, we conducted the following experiment. We fixed the cost of the system and varied the popularity distribution to determine the server throughput with and without a MEMS cache. Each MEMS caching device that we added reduced the amount of DRAM buffering available by 500MB in order to keep the cost invariant. The popularity distribution follows X:Y, where $X\%$ of the streams are accessed Y$\%$ of the time. We also assumed that all $X\%$ of the streams were equally popular. Similarly, we assumed that the remaining (100-X)$\%$ of the streams were equally unpopular. We conducted the same experiment for three different values of the total buffering cost: $\$50$, $\$100$, and $\$200$. At these costs, the numbers of MEMS devices, k, used for caching are chosen as 1, 2, and 4 respectively. The entire experiment was performed for two different average stream bit-rates: 10KB/s, and 1MB/s.

5.2.1 Effect of popularity distribution

Figure 9(a) compares the performance of the two cache-management policies: *striped* and *replicated*, against a system without a MEMS cache. The X-axis represents the popularity distribution and the Y-axis represents the server throughput in terms of the number of streams serviced. The average stream bit-rate is fixed at 10KB/s. A uniform popularity distribution is denoted by the point 50:50 along the X-axis. When $k = 1$, the replicated and striped caching is equivalent. For skewed popularity distributions of 1:99, 5:95, and 10:90, both cache management policies outperform the system without the MEMS cache. However, when the popularity distribution leans toward the uniform side, the MEMS caching is not cost-effective. The MEMS cache is able to store only a small fraction of the popular content, thereby failing to offset its cost.

We can also compare the relative performance of the two cache-management policies. When the popularity distribution is greatly skewed (1:99), the replicated cache-

[3]It can be shown that the performance of the cache can be further enhanced by off-loading the servicing of some popular streams to the disk drive if the disk is under-utilized. We leave this study for future work.

management policy supports the maximum number of streams. Since all the popular streams are available on the cache regardless of which policy is used, replication outperforms striping by offering much lower average access latencies to the MEMS cache. This is not the case for more uniform popularity distributions, wherein striping can cache more popular content than replication can.

5.2.2 Effect of varying the total cost

If the cost of the system is flexible, a system designer must know the size of the MEMS cache and DRAM buffer which provides the best server performance. Caching improves system performance when the number of MEMS devices is large enough to cache a majority of the popular movies. More MEMS devices can be used for caching when the available budget for buffering and caching is sufficient. The greater the available budget, the bigger the range of popularity distributions wherein the MEMS cache is cost-effective. For instance, in Figure 9(a), with a $\$50$ budget, the MEMS cache (using one MEMS device) is cost-effective only within the popularity range of 1:99 to 5:95. With a higher budget ($\$200$), the MEMS cache with four devices is cost-effective over a greater popularity range, 1:99 to 20:80.

5.2.3 Effect of varying the average stream bit-rate

Unlike the MEMS buffer, the performance improvement due to the MEMS cache is almost independent of the bit-rate of the streams serviced. Figures 9(a) and (b) show this behavior. Regardless of the bit-rate, addition of one or more MEMS devices increases the total streaming capacity of the server. An interesting behavior that depends on the bit-rate occurs in the case without the MEMS cache. In Figure 9(b), the additional improvement for buffering costs of $\$100$ and $\$200$ are negligible. For large bit-rates, even a small amount of buffering is sufficient for utilizing the disk throughput.

5.2.4 Effect of varying the number of MEMS devices

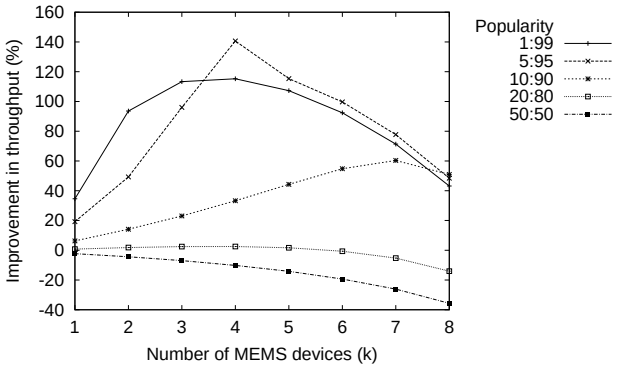


Figure 10. Varying the size of the MEMS cache.

Figure 10 shows the effect of varying the size of the MEMS cache, k, on the improvement in the server throughput. Striped cache-management policy is used for this experiment. The total cost of buffering and caching with and without the MEMS cache is fixed at $\$100$ and the system serves an average bit-rate of 100KB/s. Each MEMS device can cache 1% of the content. As k is increased, the size and throughput of the MEMS cache also increase. However, to keep the total cost invariant, the available DRAM buffer size decreases. Hence, for each popularity distribution, there exists a unique optimal size for the MEMS bank. When the popularity distribution is uniform, represented by 50:50, the MEMS cache always degrades performance. For skewed popularity distributions (1:99, 5:95, and 10:90), the MEMS cache improves the server throughput by as much as $2.4X$.

6 Related Work

Multimedia workloads fall into two main categories: (1) *best-effort* and (2) *real-time*. The *best-effort* class of multimedia data usually displays spatial and/or temporal locality attributes. Traditional caching policies [18] are well suited for such applications. The work of [16] shows that including a MEMS-based cache in the storage hierarchy can improve I/O response time by up to 3.5X for best-effort data. The focus of this work is to examine if such an addition can also improve the performance of the real-time class of multimedia data.

To service real-time disk IOs, two classes of scheduling algorithms have been proposed. *Quality Proportion Multi-Subscriber Servicing* (QPMS or *time-cycle* based scheduling) [13] and the *Earliest Deadline First (EDF)* [4] are representative scheduling algorithms. Several improvements to these algorithms have been since investigated [2, 21, 10, 17]. In this paper, we built upon the time-cycle model [13] for scheduling continuous media on MEMS devices.

Scheduling multimedia streams on a k-device MEMS bank is similar to scheduling streams on a disk array. However, using a MEMS bank as a disk buffer must consider the real-time requirements between the disk and MEMS storage as well as between MEMS storage and DRAM. On the other hand, disk load balancing policies [3, 15, 23, 24] are directly applicable to a MEMS bank when it is used as a cache. We investigated two representative policies from previous work on disk arrays for a MEMS bank.

7 Conclusions and Future Work

We have investigated the possibility of using MEMS storage for buffering and caching streaming multimedia content. Summarizing our findings, MEMS storage can improve the performance of streaming media servers by providing low-access latency and high throughput for accessing streaming data. We propose using MEMS storage in two possible con-

figurations: *MEMS as a multimedia buffer* and *MEMS as a multimedia cache*. We have developed an analytical framework for evaluating each of these configurations while using either a single MEMS device or a k-device MEMS bank. Based on our analytical study we have provided the following design guidelines: (i) for low and medium bit-rate streams, using MEMS storage as buffer reduces the cost of designing server systems by as much as an order of magnitude, (ii) when the popularity distribution is non-uniform, the server throughput can be improved by caching data in the MEMS device. The MEMS device improves access to popular content and also provides additional streaming bandwidth, regardless of the stream bit-rates.

Although this is a somewhat speculative study, several research and industry efforts [1, 6, 11, 19, 22] suggest that commercially mass-produced MEMS storage devices are only a few years away. Thus, it is necessary to start thinking now about the architectural impact of these devices on current applications. We hope that this study can provide guidelines for designing next-generation media servers.

Our work can be further explored in at least two directions. First, we realize that the MEMS storage could be simultaneously used for buffering and for caching popular streams. For instance, when the popularity distribution is not skewed sufficiently to efficiently utilize the MEMS storage as a cache, part of the MEMS storage can be used as a buffering medium to increase the disk throughput. Second, this work can be extended to include formulating intelligent placement policies for data on the MEMS device so as to improve the access characteristics of these devices for multimedia data.

References

[1] L. R. Carley, G. R. Ganger, and D. Nagle. MEMS-based Integrated-Circuit Mass-Storage systems. *Communications of the ACM*, 43(11):73–80, November 2000.

[2] E. Chang and H. Garcia-Molina. Effective memory use in a media server. *Proceedings of the 23rd VLDB Conference*, pages 496–505, August 1997.

[3] A. L. Chervenak and D. A. Patterson. Choosing the best storage system for video service. *Proceedings of ACM Multimedia 95*, pages 109–118, November 1995.

[4] S. J. Daigle and J. K. Strosnider. Disk scheduling for multimedia data streams. *Proceedings of the IS&T/SPIE*, February 1994.

[5] J. Grin, S. Schlosser, G. Ganger, and D. Nagle. Operating systems management of mems-based storage devices. *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, 2000.

[6] Kionix Inc. *http://www.kionix.com/*, June 2002.

[7] S.-H. Lee, K.-Y. Whang, Y.-S. Moon, and I.-Y. Song. Dynamic buffer allocation in Video-on-Demand systems. *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 343–354, May 2001.

[8] D. Makaroff, G. Neufeld, and N. Hutchinson. An evaluation of VBR disk admission algorithms for continuous media file. *Proceedings of the 5th ACM Multimedia Conference*, pages 143–154, 1997.

[9] Maxtor Corporation. Atlas 10KIII-U320 Product Datasheet.

[10] A. Molano, K. Juvva, and R. Rajkumar. Guaranteeing timing constraints for disk accesses in RT-Mach. *Real Time Systems Symposium*, 1997.

[11] Nanochip Inc. *http://www.nanochip.com/*, June 2002.

[12] Rambus Inc. RDRAM. *http://www.rambus.com/*, June 2002.

[13] P. V. Rangan, H. M. Vin, and S. Ramanathan. Designing and on-demand multimedia service. *IEEE Communications Magazine*, 30(7):56–65, July 1992.

[14] R. Rangaswami, Z. Dimitrijevic, E. Chang, and K. E. Schauser. MEMS-based disk buffer for streaming media servers (extended version). *http://www.cs.ucsb.edu/∼raju/mems-x.pdf*, October 2002.

[15] J. R. Santos, R. R. Muntz, and B. A. Ribeiro-Neto. Comparing random data allocation and data striping in multimedia servers. *Measurement and Modeling of Computer Systems*, pages 44–55, 2000.

[16] S. W. Schlosser, J. L. Griffin, D. Nagle, and G. R. Ganger. Designing computer systems with MEMS-based storage. *Architectural Support for Programming Languages and Operating Systems*, pages 1–12, 2000.

[17] C. Shahabi, S. Ghandeharizadeh, and S. Chaudhuri. On scheduling atomic and composite multimedia objects. *IEEE Transactions on Knowledge and Data Engineering*, 14(2):447–455, 2002.

[18] A. J. Smith. Cache memories. *ACM Computing Surveys*, pages 473–530, September 1982.

[19] The New York Times. A new system for storing data. *http://www.nytimes.com/2002/06/11/science/physical/11DATA.html*, June 11 2002.

[20] D. A. Thompson and J. S. Best. The future of magnetic data storage technology. *IBM Journal of Research and Development*, 44(3), May 2000.

[21] T.-P. J. To and B. Hamidzadeh. Dynamic real-time scheduling strategies for interactive continuous media servers. *ACM/Springer Multimedia Systems*, 7(2):91–106, 1999.

[22] P. Vettiger, M. Despont, U. Drechsler, U. Durig, W. Haberle, M. I. Lutwyche, H. E. Rothuizen, R. Stutz, R. Widmer, and G. K. Binning. The "Millipede" - More than one thousand tips for future AFM data storage. *IBM Journal of Research and Development*, 44(3):323–340, 2000.

[23] J. Wolf, P. Yu, and H. Shachnai. Disk load balancing for video-on-demand systems. *ACM Multimedia Systems*, December 1997.

[24] J. L. Wolf, P. S. Yu, and H. Shachnai. DASD Dancing: A disk load balancing optimization scheme for video-on-demand computer systems. *Proceedings of the 1995 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 157–166, 1995.

[25] P. Yu, M.-S. Chen, and D. Kandlur. Grouped sweeping scheduling for DASD-based multimedia storage management. *Multimedia Systems*, 1(1):99–109, January 1993.