

An Empirical Evaluation of Semiconductor File Memory as a Disk Cache

John C. Koob, Duncan G. Elliott, Bruce F. Cockburn
Department of Electrical and Computer Engineering
ECERF, University of Alberta, Edmonton, Alberta, T6G 2V4, Canada
Email: {jkoob|elliott|cockburn}@ece.ualberta.ca

Abstract—This work investigates the application of semiconductor file memory as a disk cache to fill part of the access time gap between disk and main memory. Leveraging DRAM cell technology, block-addressable file memory can lower the overall cost per bit of conventional DRAM due to relaxed design constraints and bad block management. File memory could function in a distinct memory hierarchy stage known as extended storage, which was common in legacy mainframes but should be revisited as a cost-competitive means of improving the performance of modern systems. We have modified the Linux 2.4.18 operating system kernel to emulate file memory in a reserved region of system memory. This experimental model evaluates file memory as an extended storage hierarchy stage that is independent of the effects of the CPU caches and the variable-sized page cache. In an analysis of chip cost and performance, if file memory can be at least 27% less expensive than DRAM and is up to three times slower, it can be used as extended storage to improve performance for typical server loads.

I. INTRODUCTION

In spite of impressive improvements in hard disk and semiconductor technology, an access time gap of five orders of magnitude has persisted between disk and DRAM in the memory hierarchies of modern computer systems [8]. For decades, most memory hierarchy research has focused on the processor-memory performance gap, where the access time gaps between the different stages rarely exceed an order of magnitude [8]. However, the larger access time gap between memory and disk has received much less attention because I/O is typically buffered by using portions of main memory as a disk cache [23]. System performance improvements would be possible with a cost-effective solid-state technology between main memory and disk, effectively recreating a memory hierarchy stage known as *extended storage* (ES) [21], [28]. A distinct ES stage appeared in a number of older systems [21], [34]. Interestingly, some architectures used DRAM ES to back a fast SRAM-based main memory [1], [13]. The development of a storage technology with appropriate cost and performance characteristics could justify a return of ES into modern systems. One can consider experimental memories such as multilevel DRAMs that feature multiple signal levels per storage cell [15], [19]. By adding bad block marking, it has been shown that semiconductor file memories can be more cost-effective than conventional DRAMs even if they both use the same contemporary DRAM storage cell design [32].

To be successful, semiconductor file memories should leverage ongoing improvements in DRAM fabrication. Our pro-

posed *file memory* is based on DRAM technology with several relaxed design constraints that can reduce the overall cost per bit [32]. First, file memory is accessed in 512-byte or larger blocks instead of providing fast random-access capability to words within a contiguous address space. Block-oriented access allows bad block management methods to hide defective memory cells, thus increasing the manufacturing yield. Bad block management is currently necessary for NAND flash memory devices that are shipped with bad blocks [6], [25], and we anticipate that it could be just as effective for DRAM-based file memory. This could create a market for existing partially-good memory chips [24], which have failures in particular sections even after redundancy mechanisms have been applied. Finally, since file memory devices can be sold with less than 100% of the nominal capacity, file memory would be less expensive than DRAM. With lower costs and moderately longer access times relative to DRAM, file memory could be targeted for use as ES and would not need to compete directly with DRAM-based main memory.

We use conventional DRAM to model file memory to quantify the chip cost and capacity characteristics required to improve the performance of low-cost servers. Instead of using trace-driven simulations, the memory hierarchy of the Linux 2.4.18 operating system (OS) kernel was modified to serve as an experimental platform [14]. We will present a preliminary performance study where the dynamically-allocated page cache is replaced with slower, more economical file memory that functions as an *extended storage disk cache* (ESDC). The changes made to the hierarchy shown in Figure 1(a) permit an analysis of ESDC performance that is independent of the nondeterministic effects of an adaptive page cache in main memory. While the absence of the page cache has an expected negative impact on performance, we have full control over the sizes of main memory and ESDC. Our simplified experimental model clearly shows how slower file memory remains effective in hiding the access time latency of disk media. Synthetic benchmarks and workloads are used to identify which file memory configurations offer the most significant benefits. These results will aid in more general analyses of systems that contain both ESDC *and* a page cache, as illustrated in Figure 1(b).

The remainder of the paper is organized as follows. Section II reviews the use of ES in historical systems and summarizes recent research in the context of modern systems. Section III

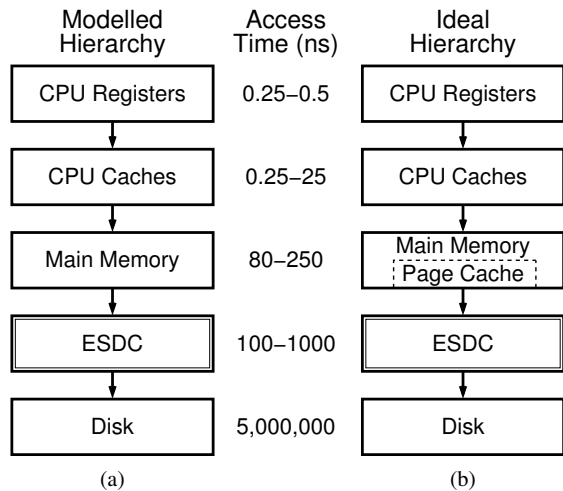


Fig. 1. A simplified ES hierarchy (a) is more suitable for an initial evaluation than the ideal hierarchy (b) due to the absence of the page cache. Access times of common hierarchy stages are from [8].

discusses file memory economics and Section IV presents the required OS modifications and methodology. Section V analyzes the experimental results and Section VI makes some concluding remarks.

II. BACKGROUND AND RECENT WORK

Extended storage has been employed primarily as page-addressable cache to improve I/O performance and to function as a backing store for demand paging.¹ Note that ES is not just a method of increasing storage capacity; the addition of general-purpose system memory will improve I/O performance provided that this memory is used for functions related to I/O [23]. As well, the memory used for ES should be more cost-effective than standard DRAM, and should tolerate slower access times [21]. Since it is more economical per bit than main memory, using ES as a cache to improve I/O performance can be more cost-effective than increasing the main memory size. A virtual memory manager can use ES to cache pages for faster demand paging. That is, a page fault can be serviced from ES rather than from backing store on disk [16]. ES could serve only as a fast paging device, but our work will examine the uses of ES as a general-purpose hierarchy stage.

Technology suitable for extended storage may be present at multiple stages of a memory hierarchy. Expensive disk caches using conventional memory technology are common in disk devices and disk controllers [11], [23]. Nonvolatile disk caches might become cheaper using microelectromechanical systems (MEMS) technology for probe-based storage [29], [31]. With 5 times the speed and 10 times the cost of disk [31], MEMS storage is ideal for the portion of the access time gap closest to disk. Similarly, a technology such as file memory could fill part of the performance gap closest to main memory. Thus, while caches that are managed by disk controllers can be categorized as ES [23], we focus on disk caches that are visible to the OS.

¹With minor architectural variations, the term “extended storage” is synonymous with *expanded storage*, *solid-state disk (SSD)*, *external memory* and *external storage* [1], [21], [28].

Although ES appeared early in the history of computing [26], the IBM 3090 featured a form of page-addressable ES known as expanded storage [21], [28]. Expanded storage was implemented using slower semiconductor memory that was less expensive than main memory [20], [34]. In the 3090, expanded storage was used as paging space, virtual I/O and other types of data spaces [34]. Pages must be transferred over dedicated channels from expanded storage to main memory to be accessed by the processor or migrated to disk [16], [21]. In the hierarchy, expanded storage used the least-recently used (LRU) replacement policy [20]. Another historical example of ES is the solid-state disk (SSD) found in Cray supercomputers, such as the Y-MP. Backing SRAM main memory with up to 4 GB of DRAM, Cray SSDs were used as ES in a manner similar to IBM’s expanded storage [1], [13]. The SSD could also serve as a distinct memory hierarchy stage that resides between a file cache in main memory and disk [13], [18]. This alternative was shown to improve workload performance by a factor of four on the Y-MP [18]. While an SSD was also used as a logical disk for frequently accessed files [18], this is normally not the best use of semiconductor devices because operating systems frequently optimize block operations for disks rather than for semiconductor storage [2].

Recent work involving disk caches appears to support the reintroduction of ES in modern systems. As processor speeds increase faster than disk speeds, a compressed main memory disk cache is becoming increasingly attractive [33]. In Kaplan’s research, the main memory disk cache is divided into uncompressed and compressed partitions [10]. The size of each cache partition is dynamically adjusted using an adaptive technique that monitors recent program activity to determine which pages should be compressed. The processing overhead incurred when compressing pages is offset by the 20% to 80% reduction in paging costs [33]. In related work, a Linux kernel patch is available that offers compressed caching with only 0.39% performance overhead [4]. A compressed hierarchy stage below main memory is the basis for a proposed multi-level main memory architecture that addresses DRAM cost and power issues and the increasing number of DRAM chips per CPU [5]. Although file memory is not used, these studies demonstrate the potential benefits of expanding the number of cached pages in spite of longer access times.

III. FILE MEMORY ECONOMICS

The introduction of a technology suitable for ES into low-cost servers must address a number of economic issues. Initially, file memory can exploit available partially-good DRAM memories. Such memories are typically not suitable for main memory applications due to the lack of a contiguous physical address space. If file memory used as ES gains acceptance, a market might develop for experimental storage technologies such as multilevel DRAM [19]. Targeting emerging storage technologies toward a new hierarchy stage avoids the challenges involved with competing directly with established technologies in the conventional markets of nonvolatile storage or general-purpose main memory.



Fig. 2. Linux virtual address space.

We propose that the bad block management techniques used for nonvolatile memories be adapted for DRAM-based file memories. In flash memory technology, blocks with bad cells are identified and remapped to spare blocks, which is analogous to DRAM redundancy schemes [6]. Additionally, some types of flash memories use an approach similar to disk fault tolerance to overcome low production yields. Specifically, NAND flash chips are shipped with bad blocks that need to be scanned and marked as defective by software [22], [25]. The manufacturing economics improve even though the functional cells within the defective blocks are not accessible. For instance, the number of functional 16-KB blocks in a commercial 1-Gbit NAND flash can range from 8032 to 8192 [27]. That is, this flash device is economical even though bad blocks can consume up to 1.95% of the nominal capacity. Just as bad block management is able to improve flash memory yields, similar techniques can be employed to increase the cost-effectiveness of block-addressable file memories that are based on conventional DRAM storage cells.

For ES to become economically feasible, its benefits must overcome the drawbacks of caching, implementation overhead and additional design effort. Various file system operations cause cache pollution—e.g., reading large files, system backups, file system searches and disk maintenance utilities [30]. As well, transferring data between disk and main memory through ES complicates the OS mechanisms that schedule I/O operations, manage metadata and reduce I/O latency [30]. The reliability of caches in hard disks is often addressed by using some form of non-volatile memory [17], but this substantially increases costs. A more economical solution adapts the approach used for the page cache, where dirty ES pages are periodically flushed to disk after a similar delay [2], [17]. Even if file memory is cheaper than conventional DRAM, the cost savings must be large enough to cover design issues, including a more complex memory controller and additional design and test effort. File memory would occupy similar physical real estate as additional conventional memory. However, file memory need not utilize valuable space on the main system board because a file memory interface is not subject to the same performance constraints as a main memory interface. This suggests that file memory could also consume less power than DRAM. In our preliminary work, the cost/performance analysis will focus exclusively on the costs related to the memory chips.

IV. METHODOLOGY

To implement our ESDC model, we modified the Linux 2.4.18 operating system kernel to study the impact of ES on system performance. Although file memory is currently unavailable, a portion of main memory can be reserved for

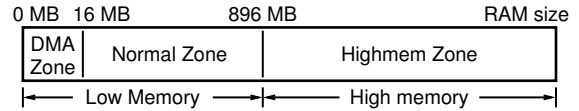


Fig. 3. Zoning of physical memory.

use as emulated file memory. For this preliminary performance evaluation, the page cache was replaced with ESDC to allow for more control over empirical parameters. The converted page cache functions as a distinct hierarchy stage with configurable size and performance parameters, control of caching properties and accurate performance metrics.

A. High Memory Zones

To distinguish conventional main memory from our emulated file memory in our ESDC model, we utilized a zoned memory architecture. In Linux, the first three gigabytes of the 4-GB virtual address space on 32-bit Intel architectures (IA-32) are available for user processes, while the upper fourth gigabyte is always mapped by the kernel, as shown in Figure 2 [2]. The IA-32 architecture has addressing constraints that preclude a consistent physical memory address space. Therefore, Linux partitions physical memory into the zones shown in Figure 3 [7]. Located in the first 16 MB of physical memory, the *DMA zone* is needed for compatibility with particular types of hardware. The *normal zone* contains physical page frames between 16 MB and 896 MB that can be directly accessed by the kernel, while page frames in the *highmem zone* cannot be directly accessed. The page cache pages can be consolidated in high memory by modifying these zone definitions. The modified kernel also permits the specification of a high memory zone even if the total amount of system memory is less than 896 MB.

B. Page Cache Containment

A *page cache* is a main memory disk cache of pages that are backed by files, devices or swap [7]. While most pages in the unmodified page cache are not contiguous, ESDC restricts page cache pages to a contiguous physical memory region.² This approach uses the existing zoned memory architecture to simplify the introduction of performance penalties and the adjustment of ESDC caching properties. ESDC inherits the page cache’s LRU replacement policy and adopts an existing hash table for efficient searches [2].

When the unmodified Linux kernel is configured to support high memory, only the page cache and pages belonging to user processes can be allocated in the high memory zone by default. These pages become interleaved with one another and may also appear in the low memory zone (see Figure 4). To evaluate file memory, kernel modifications were made so that the page cache is contiguous and contained entirely within high memory, as shown in Figure 5. Therefore, all user pages are relocated to the normal memory zone so that only ESDC pages remain in the high memory zone.

²Filesystem metadata, such as superblocks, make use of the buffer cache instead of the page cache and are not present in ESDC.

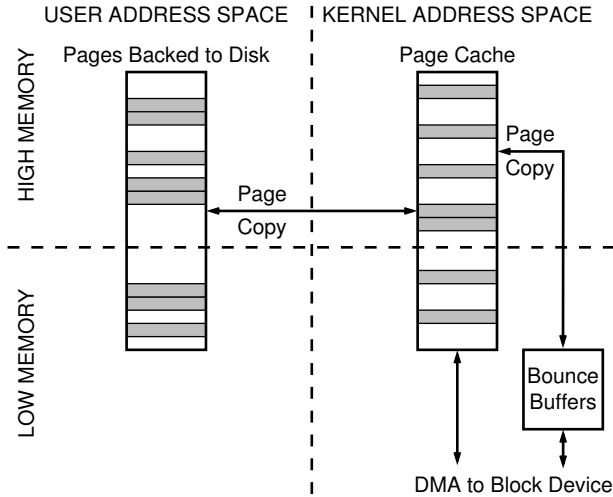


Fig. 4. Original page cache architecture: Page cache pages are interleaved and reside in both high and low memory. This memory organization is not suitable for an experimental model.

ESDC pages are not directly accessible by user processes since ESDC is located in the kernel address space. For example, a page backed by a file on disk must first be copied from kernel space to user space before a user process can access the data (see Figures 4 and 5). Pages in high memory must also be copied to bounce buffers, which are used for devices that are unable to access high memory addresses [7]. Note that for larger high memory capacities, the original and modified kernels require the same number of page copies. However, the original kernel copies fewer pages from the page cache since some pages reside in low memory. Thus, ESDC exploits existing kernel features rather than adding overhead by copying pages to new data structures. As well, the isolation of ESDC pages in a restricted memory zone offers improved immunity to software errors, which is evident in the Rio (RAM I/O) file cache [17].

C. Configurable Performance

The use of a contiguous page cache in the high memory zone simplifies the introduction of performance penalties to the transfer of pages between low memory and ESDC in high memory. The performance penalties are readily modeled as repeated calls to existing macros that copy pages between address spaces (e.g. `_copy_user`) or to existing functions that copy pages to and from the bounce buffers. The repeated replication of pages between low memory and ESDC is represented by the delay elements in Figure 5. However, multiple copies of a single page from the same source to the same destination will likely involve the CPU caches. Therefore, the modified kernel must prevent the CPU caches from caching the high memory zone to ensure penalty metric accuracy.

D. CPU Caching

Since the ESDC area is contiguous, it is possible to control how this memory is cached by the CPU caches. The IA-32 architecture provides a mechanism for specifying the caching parameters (such as write-back or write-through) of aligned

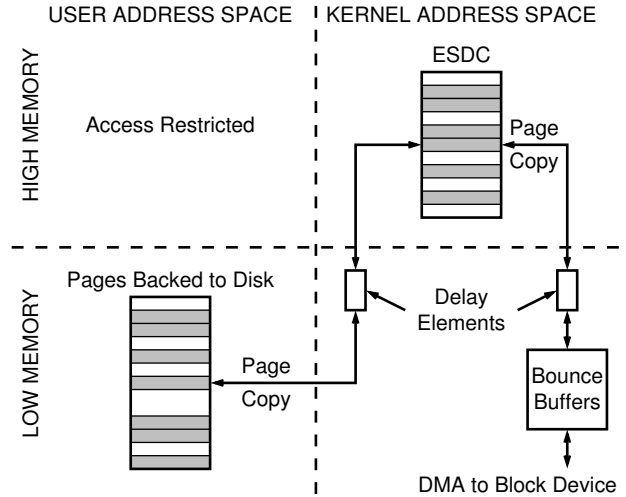


Fig. 5. ESDC experimental model: ESDC pages are contained within high memory, which permits configurable ESDC performance and control over the caching properties.

memory regions [9]. That is, a limited number of *memory type range registers* (MTRRs) are available, which are often used to optimize the caching of frame-buffer memory. To ensure that the ESDC model uses accurate performance metrics, MTRRs must set the ESDC memory type to *uncachable*. Using a minimum number of MTRR registers, we developed a recursive algorithm to partition a given contiguous ESDC region into aligned ranges.

E. Performance Metrics and Verification

We obtained performance metrics for the new hierarchy stage by instrumenting the modified kernel. To control and monitor ESDC activity, two new files were added to the `/proc` pseudo file system. This enables user processes to adjust ESDC parameters or access ESDC performance metrics at runtime. Important metrics include the number of pages written to or read from ESDC by user processes, the number of pages transferred between ESDC and disk, and ESDC miss rate metrics. The experimental error introduced by instrumenting the kernel, which affects benchmark performance results by at most 2%, is negligible.

Several tests confirmed the validity and reproducibility of the experiments. ESDC metrics were added to monitor the dynamic memory allocator to verify that the number of pages allocated to or freed from ESDC is consistent with other kernel metrics. This effort actually revealed some record-keeping errors in the official Linux kernel.³

V. EXPERIMENTAL RESULTS

A. Experimental Setup and Methodology

ESDC was evaluated with an experimental suite consisting of two synthetic benchmarks and an application workload. Measurements were obtained from benchmark output and ESDC metrics were acquired from the modified Linux 2.4.18

³For instance, the number of high memory page allocations was successfully correlated with the size of ESDC once an incomplete implementation of a function (`balance_classzones`) was excised from the kernel.

TABLE I
EXPERIMENTAL SETUP

Component	Specification
Processor	2.4-GHz Intel Pentium 4
CPU Cache	L1: 12-KB instr., 8-KB data; L2: 512-KB
Memory	2-GB PC2100 266-MHz DDR SDRAM
Hard Disk	18-GB SEAGATE Model ST318405LW
Controller	Adaptec 29160N Ultra160 SCSI adapter
OS Kernels	Linux 2.4.18 (original and modified)
Software	Slackware 8.1 (minimalist package list)

kernel running on the experimental platform specified in Table I. The 4-MB SCSI disk buffer is insignificant relative to ESDC capacities (up to 1.4 GB) and the 1-GB working sets. Empirical parameters have similar effects on benchmark write and read rates, so the analysis will focus on write performance. Six sequential runs of a benchmark were executed for each memory configuration, but since the first cold-start run is omitted, each plotted data point represents the mean of five measurements. Error bars represent the 95% confidence intervals calculated from data for five consecutive benchmark invocations. The system was automatically rebooted between each measured data point. Each set of benchmark data files was created in an empty file system. By periodically dedicating a session for file system scanning, unintentional scan operations, such as `fsck`, were avoided during the experiments.

B. PostMark Benchmark Performance Results

The PostMark benchmark models the heavy loads on Internet service provider systems, such as e-mail servers that make frequent accesses to short-lived small files [12]. Each execution of the benchmark involves 100,000 transactions on 100,000 files with sizes ranging from 500 B to 15,000 B, which gives a working set size of just over 1 GB. We considered four experimental system configurations:

Unmodified kernel configuration: The write throughput of PostMark running on an unmodified Linux kernel is shown in Figure 6. The total system memory is specified using a kernel boot parameter and high memory support is not enabled. Although increases in main memory capacity improve performance, the dynamic page cache is not able to grow large enough to accommodate the entire 1-GB working set. This unmodified kernel is an unsuitable reference configuration for several reasons: It is difficult to establish a fixed page cache size, the page cache is cached by the CPU caches and high memory overhead is not present.

Configuration with original page cache: Another experimental configuration to consider is an instrumented kernel with configurable low and high memory sizes. In this kernel, pages belonging to a page cache are distributed between both high memory and low memory (see Figure 4). As shown in Figure 7, the peak performance of 1000 KB/s is achieved when the combined capacity of low and high memory is at least 1280 MB, which is adequate for both the kernel and

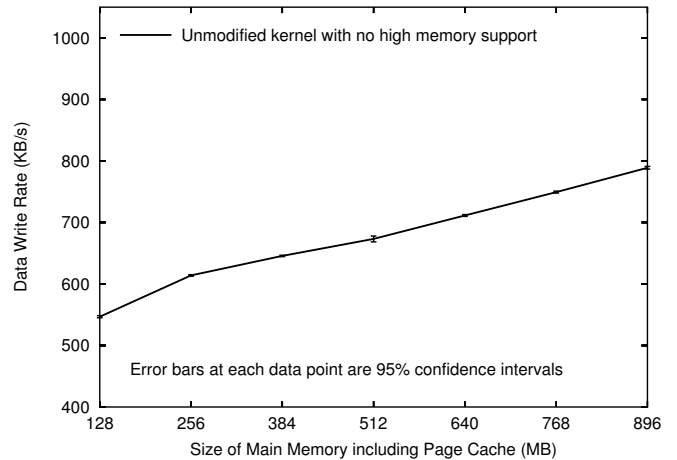


Fig. 6. PostMark write throughput on an unmodified kernel with default CPU caching properties.

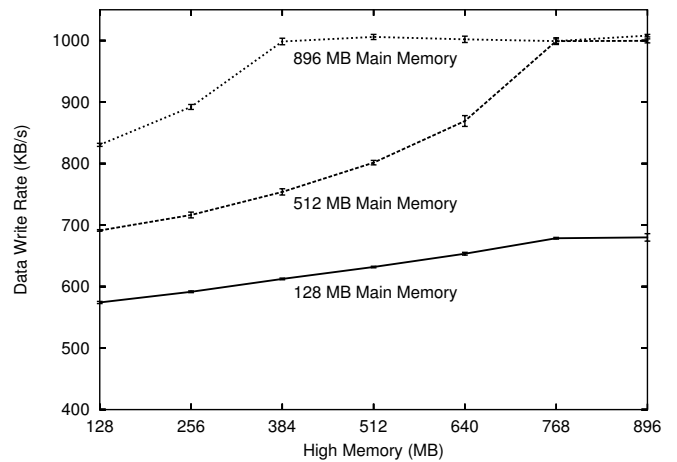


Fig. 7. PostMark write throughput for the original memory hierarchy of kernel with a variably-sized page cache in both main and high memory and default high memory caching properties.

the entire working set. When only 128 MB of main memory is available, the working set resides primarily in available high memory. That is, compared with an unmodified kernel, benchmark performance is degraded due to the use of high memory with the necessary bounce buffer overhead. Note that any 32-bit Linux system with more than 896 MB of DRAM would suffer from the same source of overhead due to the high memory implementation.

Reference configuration with ESDC as DRAM: A reference configuration must provide page containment and uncached high memory for a fair comparison with the ESDC experimental model. That is, page cache pages must reside entirely within high memory to allow for control over ESDC and main memory sizes (see Figure 5). Regardless of whether 128 MB or 896 MB of main memory is available, the data write rate is independent of main memory size. Thus, main memory is fixed at a representative size of 512 MB. The performance of PostMark for the reference configuration is shown by the top curve in Figure 8. In essence, the high memory zone models ESDC where file memory has the same access time as DRAM.

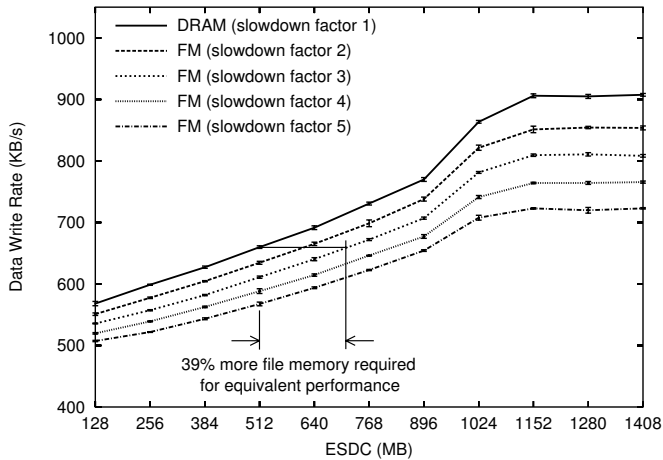


Fig. 8. PostMark write throughput: A comparison of ESDC performance for various file memory slowdown factors. A slowdown of 1 represents conventional DRAM.

TABLE II
EQUIVALENT PERFORMANCE COMPARISON

File Memory Slowdown Factor	Ratio of File Memory Capacity to DRAM Capacity	
	PostMark	Bonnie
2	1.19	1.13
3	1.37	1.32
4	1.63	1.66
5	2.03	2.33

Model with ESDC as file memory: Based on the memory hierarchy shown in Figure 5, the ES stage emulates file memory by using uncached memory that is slower than the conventional DRAM of the reference configuration. The access time of file memory relative to DRAM can be specified with a slowdown factor. For example, for the middle curve of Figure 8, ESDC is configured to model file memory with a slowdown factor of 3. Again, main memory is 512 MB.

We now compare the use file memory as ESDC against DRAM as ESDC. For example, if money were available to add 512 MB of DRAM to a system, only 39% more file memory (712 MB) with a slowdown factor of 3 would be required for equivalent performance (see the annotation in Figure 8). If ESDC sizes are varied over the range of the 1-GB working set, an average of 37% more file memory than DRAM is necessary for equivalent performance. Therefore, if the above file memory is at least 27% cheaper than DRAM, it can improve system performance. That is, in spite of the reduction in peak performance, file memory can cost-effectively improve performance within the working set range. The slowdown factor of 3 accommodates the potential overhead of a file memory implementation versus DRAM. For example, the required 37% file memory density increase would be achievable even if a 4-level multilevel DRAM could not double DRAM capacity at the same cost due to circuit overhead [15]. For equivalent performance, the required file memory densities relative to DRAM for different slowdown factors are shown in Table II.

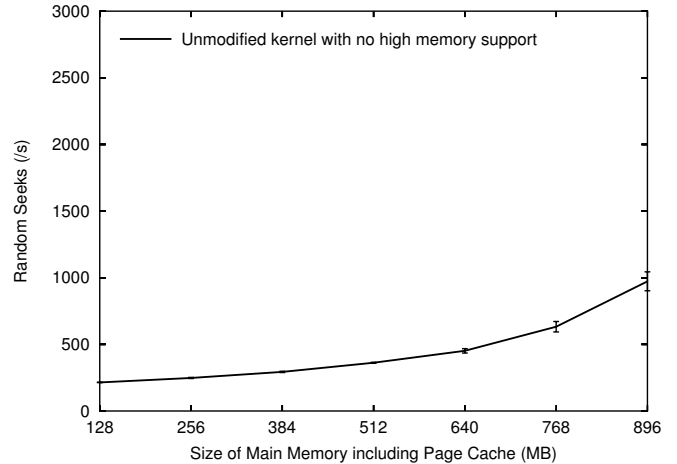


Fig. 9. Bonnie benchmark executing random seeks for an unmodified kernel with default CPU caching properties.

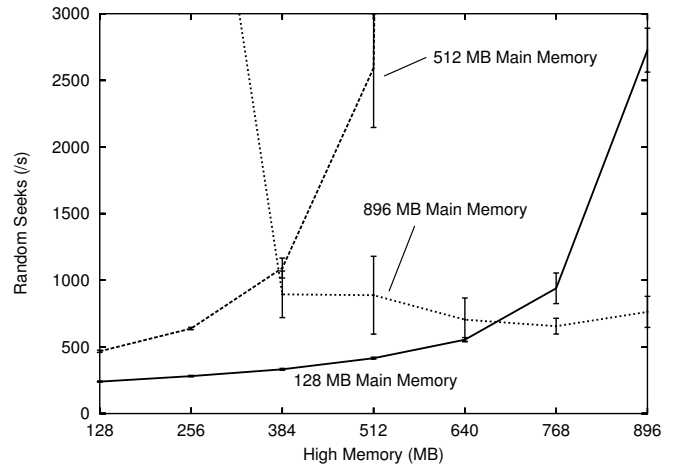


Fig. 10. Bonnie random seek performance for the original memory hierarchy of Figure 4: The page cache in the instrumented kernel can reside in both main and high memory.

C. Bonnie Benchmark Performance Results

The Bonnie benchmark measures the performance of Unix file system operations [3]. While PostMark involves many small files, Bonnie is configured to perform random block operations on a single file of known size. We investigated Bonnie's random seek rate on a 1-GB file for the same set of system configurations that was used for PostMark.

Unmodified kernel configuration: The performance of Bonnie for an unmodified Linux kernel is shown in Figure 9. The OS attempts to accommodate the 1-GB working set, but benchmark performance only reaches 1000 random seeks per second. As discussed in Section V-B, the unmodified kernel is inappropriate as a reference configuration.

Configuration with original page cache: Bonnie benchmark performance is shown in Figure 10 for an instrumented kernel with the original page cache architecture (see Figure 4). Pages from Bonnie's 1-GB working set are distributed in the page cache in both high memory and low memory. Figure 10 illustrates how the performance can rapidly increase past 2000 random seeks per second due to the CPU caching effect. When

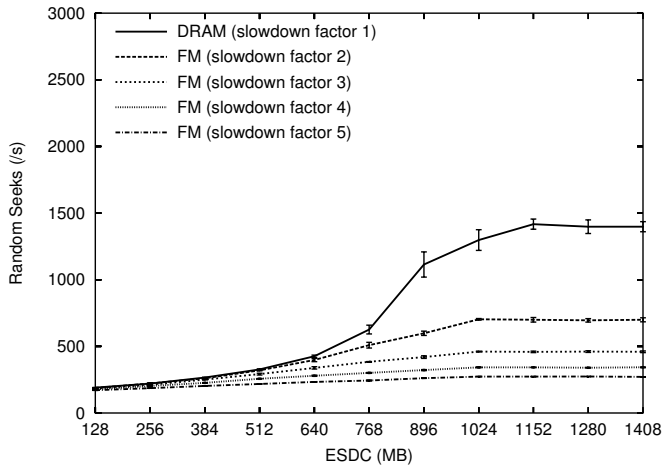


Fig. 11. Bonnie random seeks: A comparison of ESDC performance for various file memory slowdown factors, including a slowdown of 1 representing conventional DRAM.

the size of low memory is 128 MB or 512 MB, performance increases dramatically when at least 896 MB or 512 MB of high memory is available, respectively. That is, when the working set fits entirely within low and high memory, the CPU caches can boost the speed to about 51000 random seeks per second. However, for 896 MB of low memory and large high memory sizes, the kernel’s memory management mechanism and bounce buffer overhead actually hinder CPU caching. This is an unsuitable reference configuration due to the CPU caching effects and the variable page cache size.

Reference configuration with ESDC as DRAM: To permit fair comparisons, page cache pages must be contained entirely within uncached high memory (see Figure 5 and Section V-B). The top curve in Figure 11 illustrates how disabling the CPU caches for ESDC in high memory substantially reduces benchmark performance relative to the data in Figure 10. This curve is the reference configuration where file memory has the same access time as DRAM.

Model with ESDC as file memory: The experimental configuration emulates uncached file memory with different slowdown factors and reserves 512 MB for main memory. Figure 11 shows how ESDC affects the peak performance in the absence of the page cache. Even though a slower ES stage has replaced the page cache, performance improvements are still possible within the 1-GB working set range. For example, suppose file memory is three times slower than DRAM. On average, 32% more file memory than DRAM is necessary for equivalent performance. That is, if file memory can be at least 24% less expensive than DRAM but is no more than three times slower, it can be used as ES to improve performance under particular system load patterns. The required file memory capacity for other slowdown factors can be found in Table II.

D. Other Workload Results

The compilation of an unpatched 2.4.18 Linux kernel is suitable for reproducible experiments. Invoking multiple compiler processes, a kernel build causes memory pressure for main memory sizes less than 48 MB [4]. Unfortunately,

disabling the CPU caches for high memory increases workload execution times by an order of magnitude. This is likely due to the large number of hits on ESDC pages (about 250,000), which is an order of magnitude more than PostMark (32,000). When ES is under pressure, the kernel build miss rate is much larger than the case when the working set fits within ES. In the best case, miss rates approach 9% for PostMark, 9% for Bonnie and 6% for a kernel build.

VI. CONCLUSION AND FUTURE WORK

By leveraging DRAM technology, file memory could be developed to re-introduce a distinct hierarchy stage between main memory and disk. Industry and academia have demonstrated that the trade-offs in dynamic memory design can be adjusted to create parts with higher latency, higher density and lower costs per bit [19], [32]. As well, existing commodity DRAMs that contain errors even after redundancy mechanisms are exhausted can be sold as partially-good devices with less than 100% of the nominal capacity [24]. File memory can salvage more of the functional bits of such downgraded DRAMs due to relaxed design constraints and bad block management. Non-contiguous, block-addressable file memory is better suited for ES rather than for main memory applications. Although file memory technology incurs additional sources of overhead, changes to the conventional DRAM storage cell design are unnecessary. Thus, further improvements in DRAM technology would also benefit file memory.

In this paper, we quantified the file memory density improvements that are necessary to cost-effectively improve the performance of low-budget servers. We have demonstrated how our empirical model of ES offers excellent control over experimental parameters because it is independent of the Linux page cache and CPU caches. We have identified the most promising configurations that cost-effectively improve performance by adding file memory as ES instead of simply increasing main memory capacity. For example, we have shown that file memory can be at most 4 times slower than conventional DRAM before twice as much file memory is needed to maintain equivalent performance.

Ongoing research aims to evaluate the effect of the combination of both ES and the page cache in the memory hierarchy. The kernel’s page cache implementation will require additional modifications because the memory management algorithms assume that the page cache is backed by slow disk media. For example, some optimizations in the page fault mechanism become inappropriate if ES is added below the page cache in the hierarchy. While the experimental model and analysis will be more complex, this approach would address the peak performance overhead that is inevitable when the page cache is absent. If file memory technology that is slower but sufficiently more economical than DRAM becomes feasible, our work suggests that ES should be revisited for use in modern systems.

ACKNOWLEDGEMENTS

This research has been supported by the Alberta Ingenuity Fund and the Informatics Circle of Research Excellence.

REFERENCES

- [1] D. H. Bailey. FFTs in external or hierarchical memory. In *Proc. of the ACM/IEEE Conf. on Supercomputing*, pages 234–242, 1989.
- [2] D. P. Bovet and M. Cesati. *Understanding the Linux Kernel*. O’Reilly and Associates, 2nd edition, 2003.
- [3] T. Bray. Bonnie file system benchmark. www.textuality.com/bonnie/, 1996.
- [4] R. S. Castro, A. Pereira do Lago, and D. Da Silva. Adaptive compressed caching: Design and implementation. In *The 15th Symp. on Computer Architecture and High Performance Computing*, November 2003.
- [5] M. Ekman and P. Stenstrom. A case for multi-level main memory. In *Proc. of the ACM Workshop on Memory Performance Issues*, pages 1–8, 2004.
- [6] P. K. Garvin. Method and system for managing bad areas in flash memory. United States Patent 6260156, July 2001.
- [7] M. Gorman. *Understanding the Linux virtual memory manager*. Prentice Hall, 2004.
- [8] J. Hennessy and D. Patterson. *Computer architecture: A quantitative approach*. Morgan Kaufmann, 3rd edition, 2003.
- [9] Intel Corporation. *Intel Architecture Software Developer’s Manual: Volume 3: System Programming*, 1999.
- [10] S. F. Kaplan. *Compressed Caching and Modern Virtual Memory Simulation*. PhD thesis, University of Texas at Austin, 1999.
- [11] R. Karedla, J. S. Love, and B. G. Wherry. Caching strategies to improve disk system performance. *IEEE Computer*, 27(3):38–46, March 1994.
- [12] J. Katcher. PostMark: A new filesystem benchmark. Technical Report TR3022, Network Appliance, Inc., October 1997.
- [13] L. I. Kontothanassis, R. A. Sugumar, G. J. Faanes, J. E. Smith, and M. L. Scott. Cache performance in vector supercomputers. In *Proc. of the Conf. on Supercomputing*, pages 255–264, November 1994.
- [14] J. C. Koob. File memory for extended storage disk caches. Master’s thesis, University of Alberta, 2004.
- [15] J. C. Koob, S. A. Ung, A. Rao, D. Leder, C. Joly, K. Breen, T. Brandon, M. Hume, B. Cockburn, and D. Elliott. Test and characterization of a variable-capacity multilevel DRAM. In *Proc. VLSI Test Symp.*, pages 189–197, May 2005.
- [16] K. Li and K. Petersen. Evaluation of memory system extensions. In *Proc. of the Int. Symp. on Computer Arch.*, pages 84–93, May 1991.
- [17] W. T. Ng and P. M. Chen. The design and verification of the Rio file cache. *IEEE Trans. on Computers*, 50(4):322–337, April 2001.
- [18] V. Oklobdzija, editor. *The Computer Engineering Handbook*, chapter 80. CRC Press, 2000.
- [19] T. Okuda and T. Murotani. Four-level storage 4-Gb DRAM. *IEEE J. of Solid-States Circuits*, 32(11):1743–1747, November 1997.
- [20] N. Prasad and J. Savit. *IBM mainframes: Architecture and design*. McGraw-Hill, 2nd edition, 1994.
- [21] E. Rahm. Performance evaluation of extended storage architectures for transaction processing. In *Proc. of the 1992 ACM SIGMOD Int. Conf. on Management of Data*, pages 308–317, June 1992.
- [22] Reactive Computer Services Data Sheet. *K3M FFD 2.5” IDE Plus*, 2003.
- [23] A. J. Smith. Disk cache–miss ratio analysis and design considerations. *ACM Trans. on Computer Sys.*, 3(3):161–203, August 1985.
- [24] C. H. Stapper et al. Yield model for productivity optimization of VLSI memory chips with redundancy and partially good product. *IBM J. Res. Develop.*, 24(3):398–409, May 1980.
- [25] A. Tal. *Two technologies compared: NOR vs. NAND White Paper*. M-Systems, July 2003.
- [26] J. Thornton. *Design of a computer: the Control Data 6600*. Scott Foresman and Co., Glenview, Ill., 1970.
- [27] Toshiba Corp. *TC58V100 1-Gbit CMOS NAND E2PROM Datasheet*, March 2001.
- [28] S. G. Tucker. The IBM 3090 system: An overview. *IBM Systems J.*, 25(1):4–19, 1986.
- [29] M. Uysal, A. Merchant, and G. Alvarez. Using MEMS-based storage in disk arrays. In *Proc. of the USENIX Conference on File and Storage Technologies*, pages 89–101, March 2003.
- [30] A. Wang et al. Conquest: Better performance through a disk/ persistent-RAM hybrid file system. In *Proc. of the USENIX Annual Technical Conf.*, pages 15–28, June 2002.
- [31] F. Wang, B. Hong, S. A. Brandt, and D. D. E. Long. Using MEMS-based storage to boost disk performance. In *Proc. of the 2005 IEEE Conf. on Mass Storage Systems and Tech.*, pages 202–209, April 2005.
- [32] C. Wickman, D. Elliott, and B. Cockburn. Cost models for large file memory DRAMs with ECC and bad block marking. In *Int. Symp. Defect and Fault Tolerance in VLSI Systems*, pages 319–327, November 1999.
- [33] P. R. Wilson, S. F. Kaplan, and Y. Smaragdakis. The case for compressed caching in virtual memory systems. In *Proc. of the USENIX Annual Technical Conf.*, pages 101–116, 1999.
- [34] J. Young. *Exploring IBM’s New Age Mainframes*. Maximum Press, 1996.