Operating System Support for NVM+DRAM Hybrid Main Memory

Jeffrey C. Mogul, Eduardo Argollo, Mehul Shah, Paolo Faraboschi

HP Laboratories
HPL-2009-256

Keyword(s):
Flash, Non-volatile RAM. Operating Systems, memory

Abstract:
Technology trends may soon favor building main memory as a hybrid between DRAM and non-volatile memory, such as flash or PC-RAM. We describe how the operating system might manage such hybrid memories, using semantic information not available in other layers. We describe preliminary experiments suggesting that this approach is viable.

External Posting Date: November 6, 2009 [Fulltext] Approved for External Publication
Internal Posting Date: November 6, 2009 [Fulltext]

Published in the 12th Workshop on Hot Topics in Operating Systems (HatOS XII), May 18-20, 2009

Operating System Support for NVM+DRAM Hybrid Main Memory

Jeffrey C. Mogul* Jeff.Mogul@hp.com
Eduardo Argollo† Eduardo.Argollo@hp.com
Mehul Shah* Mehul.Shah@hp.com
Paolo Faraboschi† Paolo.Faraboschi@hp.com
HP Labs, **Palo Alto, CA 94304* and †*Barcelona, Spain*

Abstract

Technology trends may soon favor building main memory as a hybrid between DRAM and non-volatile memory, such as flash or PC-RAM. We describe how the operating system might manage such hybrid memories, using semantic information not available in other layers. We describe preliminary experiments suggesting that this approach is viable.

1 Introduction

For several decades, general-purpose CPUs have used DRAM for main memory. DRAM has many good features, and has benefited from Moore's Law, but DRAM is not perfect: it is relatively expensive in power and cost, as a fraction of an entire computer, and it is hard to put enough of it near a CPU. These problems are especially pressing in "scale-out" server farms, where we want both increased server density and reduced heat density.

On the other hand, flash memory can be denser, cheaper, and more power-efficient than DRAM, but it has problems with access timing, access unit sizes, and endurance. Because of these problems, and because it is reasonably non-volatile, flash is typically used in the storage hierarchy, rather than as main memory.

The semiconductor industry has been speculating for several years about the prospects for a "universal memory" (UM) technology that would replace both DRAM and flash, providing the best characteristics of both (and working around some scaling issues that might soon limit further improvements in both DRAM and flash) [1]. Unfortunately, while there are many potential UM technologies, all have their problems, and mass adoption (hence reasonable prices) is still several years away [12].

In this paper, we argue that if it becomes practical and desirable to replace main-memory DRAM with UM, the characteristics of UM technologies (limited endurance; slow writes) will require explicit support from the operating system, and we describe aspects of that support.

To make things concrete, we describe a near-term design for main memory based on a hybrid of flash and DRAM, or *FLAM*[1]. Quite possibly FLAM could improve main memory density, power, and cost. It might seem odd to propose using flash for main memory, since it has very high write latencies, and wears out after relatively few writes. Our goal, therefore, is to present a feasible design for hardware and operating system changes that compensate for these weaknesses of flash, and also to illustrate how the OS might support other varieties of UM.

[1]Per Wikipedia, "A flam is a rudiment [of drumming] consisting of a quiet 'grace' note on one hand followed by a louder 'primary' stroke on the opposite hand. The two notes are played almost simultaneously, and are intended to sound like a single, 'broader' note." This idea of two distinct notes merging almost into one reminds us of the DRAM+flash hybrid.

Many computer systems now use flash memory for all or part of their storage hierarchy. Our design is different; flash is parallel, not subordinate, to DRAM.

We argue for approaches that exploit per-page knowledge, relatively simple for an OS to obtain, to inform the movement of pages between read-write DRAM and (effectively) read-only flash. Note that the flash pages are truly in the CPU's memory address space – memory-read instructions are satisfied directly from flash – but protected read-only, to allow the OS to intervene on memory writes.

The key to this approach is the use of flash only for pages with a relatively high *time-to-next-write* (TTNW), since the penalties (latency; wear-out) for flash writes are so high. Of course, the OS can only know a page's *estimated time-to-next-write* (ETTNW), and we identify OS-level information that can help make these estimates. We also present experimental data suggesting that there are enough high-TTNW pages to justify using FLAM.

The main point of this paper is not that flash is ideal for this application – it is not – but that hybrid main memories built from DRAM and Non-Volatile Memory (NVM) are a plausible solution to some pressing problems, that software will have to manage the way these memories are used, and that the operating system is the best software layer to do that. The focus of our work is to show how having the OS carefully manage what goes into NVM, and when, can hide the non-ideal characteristics of NVM while allowing us to exploit its useful attributes. This approach should apply, with some variations, to both flash and other kinds of NVM, and we specifically discuss Phase-Change RAM (PC-RAM).

1.1 Motivation

Data center compute farms (this paper focusses on server applications) are increasingly limited by physical density and power constraints, and are being built from large numbers of relatively cheap servers. These trends put pressure on the amount of main memory per physical server, since they limit the number of DRAM "DIMMs" that can be placed in proximity to a CPU socket. Adding more DRAM chips to a server also increases bus loading, leading to signal integrity problems.

High-density DIMMs impose an exponential cost penalty; for example, Oct. 2008 street prices for 8GB of DRAM are $212/GB using 1 8GB DIMM, $50/GB using 2 4GB DIMMS, or $15/GB using 4 2GB DIMMs. However, adding DIMM slots consumes space on increasingly small "blade" server boards, and can complicate electrical signal integrity.

DRAM also consumes non-negligible power – per one study, 19% to 31% of peak power for recent server designs [13, Table 3-3].

We would therefore like to find a way to create at least the illusion of larger main memories without the cost, power, and density drawbacks of DRAM.

1.2 Memory technologies

The insight (not ours; see [16]) that inspired this paper is that flash has many of the characteristics that we want for main memory: per bit, it costs less, consumes less power, and consumes less space than DRAM.

However, flash has several problems that complicate its use as a direct replacement for DRAM:

- **Erasing**: Flash must be erased before it can be written. Erasing tends to be slow, and one must erase a large block, rather than an individual word.
- **Endurance**: flash typical wears out after a relatively small number of erase+write cycles. For storage, write bandwidths are slow enough to avoid this problem, but not for main memory: at 5 GB/s, one can wear out 256GB of NOR in less than 60 days.

  In a storage application, access frequencies can be low enough to avoid this problem, but the frequencies with which CPUs write to main emory could cause wear-out way too soon.
- **Slow writes**: Flash writes take much longer than DRAM writes, so flash writes are not compatible with typical memory controllers.
- **Read timing**: NAND flash requires reading an entire page, which makes it difficult to use for main memory. NOR flash, however, can be read more or less like DRAM (albeit at lower bandwidths).

Table 1 shows values for various characteristics of several memory technologies, which we will assume for the purposes of this paper. It is difficult to get accurate values to make direct comparisons; for example, NOR flash is usually fabricated with an older technology generation than DRAM or NAND flash. We express density in terms of the feature size F (actually, in terms of the area F^2) rather than in absolute values; this allows density comparisons if one assumes a fixed feature size.

Similarly, we do not directly estimate cost/bit; generally, cost is directly proportional to density assuming both equal fabrication complexity and similar production volumes. These assumptions might be optimistic.

Table 1 shows that NAND flash is superior to NOR flash in both cost and density, but because NAND flash cannot satisfy cache-line reads at speeds anywhere near DRAM, it is a poor basis for a hybrid design.

Table 1 shows values for *single-level cell* (SLC) flash. *Multi-level cell* (MLC) flash can store 2 bits per cell, doubling its density but at the cost of a higher bit-error rate and lower endurance. MLC NAND flash is widely available, and Spansion has developed a cost-effective MLC NOR design [11, 16]. However, it is not clear that flash can be scaled much further into the future [10].

Proposed future memory technologies include Ferroelectric RAM, Magnetic RAM, Spin-Torque Transfer RAM, Resistive RAM, but these are still too exotic or are clearly bad candidates for main memory. (Burr *et al.* [5] give a nice, up-to-date summary of non-volatile memory technologies, but they focus on storage-hierarchy applications.) However, the table includes data on Phase-Change RAM (PC-RAM; confusingly sometimes called PRAM, PCM, OUM, or C-RAM), which is at least a plausible candidate. PC-RAM uses heat to change a chalcogenide glass between crystalline and amorphous states. PC-RAM has much better endurance than flash, but apparently it is still finite.

Therefore, in this paper we explore two options for software-managed hybrid main memories: NOR flash and PC-RAM.

1.3 Other alternatives for using NVM

We note that there are many other ways to use NVM in computer systems. There are obviously many uses of NVM in the storage system [5]. One might also consider designing hybrid main-memory hardware that is fully transparent to software (perhaps this is Spansion's approach [16]), which avoids the need for SW changes. However, the OS can make better decisions, using its semantic information about memory pages.

One could use NAND flash on the PCI bus as a very fast backing store for demand-paged DRAM, but PCI cannot not achieve the high memory bandwidth that a NOR+DRAM hybrid can support.

Recently, Condit *et al.* [8] have proposed using byte-addressable, persistent memory (such as PC-RAM) on the memory bus for the specific purpose of implementing a file system with improved atomicity and performance. Their design, while it eliminates the need for some DRAM buffering of file-system data, does not attempt in general to replace DRAM, and it requires some minor changes to the CPU cache architecture.

2 Design issues: NOR-flash hybrids

Here we sketch both hardware and software designs for SW-managed hybrids using NOR flash. (Sec. 3 will discuss how the designs might vary if using PC-RAM.)

Table 1: Characteristics assumed in this paper (sources include [4, 9, 10, 14, 15])

Technology	**Density**	**Endurance**	**Rand. read**	**Write**	**Erase**	**Erase**	**Idle “on”**
		(cycles)		**time**		**size**	**power**
DRAM	6-8 F^2	10^{15}	~ 40–60 ns	~ 40–60 ns	—	—	~4W–8W/DIMM
NAND flash	4-5 F^2	10^5–10^6	5–50 us	200 us/page	2 ms	e.g. 512KB	~0
NOR flash	10 F^2	10^5–10^6	70 ns	1 us	1 sec	e.g. 128KB	~0
PC-RAM	8-16 F^2	10^8–10^{11}	60 ns?	100–1000 ns	—	—	~0

Note: most of these values either vary with technology or are poorly defined

2.1 Hardware design

We start by assuming that new hardware designs ought to be pin-compatible with standard DRAM DIMMs and memory controllers. While it is intriguing to consider how one might change these standard interfaces to better exploit novel main-memory technologies, incremental deployment is easier if one sticks to standards.

We propose a FLAM DIMM that contains as many NOR flash chips as possible, a modest amount of DRAM for buffering writes, and a simple controller ASIC.

The address space of the DIMM would be divided into several regions:

- **Flash**: directly mapped for cache-line-wide reads. The CPU would not be able to write directly to this region.
- **DRAM copy buffer (CB)**: mapped for both reads and writes by the CPU.
- **Control registers**: accessed via the standard System Management Bus (SMBus), mapped into I/O space.

Basic migration mechanism: Since flash cannot be written directly by CPU store operations, we instead stage page-sized writes in the CB. That is, when the OS decides to move a page P_d from main-memory DRAM to FLAM, it allocates a pre-erased flash page P_f (see sec. 2.2 for more details), copies P_d into a free page in the CB P_b, and then signals the FLAM controller to copy from P_b to P_f. This copy can proceed at flash-write speeds (i.e., slowly) without stalling the CPU. (In order to sustain the necessary write bandwidth, the NOR flash will have to be banked 8–16 ways.) When that copy is done, the controller signals the OS, which can then remap the corresponding virtual page P_v from P_d to P_f, and invalidate the TLB entry. (Modern CPUs allow flushing one entry rather than the entire TLB.)

A small portion of the CB would be set aside to hold the specifics of commands from the OS to the controller: for a copy commands, the (P_b, P_f) pair; for an erase command, the offset and size of the sector(s) to erase.

This simple hardware design poses several challenges:

- **Communication with the controller**: We expect that a small portion of the CB will have to be set aside to hold the specifics of commands from the OS to the controller: for a copy commands, the (P_b, P_f) pair; for an erase command, the offset and size of the sector(s) to erase.
- **BIOS support**: BIOS software might be confused by FLAM hardware in the main-memory address space, and would probably have to be modified; for example, to avoid subjecting the flash to a memory test, and to notify the OS which parts of the hardware address space are FLAM and which are standard DRAM.

One possibly solution would be to make the first page of FLAM read-only, consisting of a magic number and a set of configuration values, followed by a cryptographic hash of that data. The kernel would read the page and validate this hash, indicating with very high probability that it is seeing a FLAM device, and also learning its characteristics.

2.2 Basic software design

Given our proposed hardware design, the basic software mechanism is simple and is useful for UM technologies with limited or slow writes. We defer several more interesting problems to sec. 2.3 (deciding what pages to put into flash), sec. 2.4 (garbage-collecting the flash), and sec. 2.5 (wear-leveling).

Initially upon booting, the OS allocates all memory pages from DRAM. Based on heuristics described in sec. 2.3, it starts migrating pages from DRAM to FLAM.

The basic migration mechanism described in sec. 2.1 needs some elaboration. For example, the OS must set the page-table entries for P_d (during migration) and P_f (after migration) to be either read-only or read+execute, since any writes during migration could lead to inconsistencies, and any writes after migration won’t work. If SW generates a store to one of these pages, it will fault. Before the write can proceed after the fault, the OS must either abort the migration (if it is still in progress), or copy the page back from FLAM to DRAM. In effect, a migrated page is handled similar to a copy-on-write page (and a shared migrated page might end up in both FLAM and DRAM after a true COW fault).

2.3 Deciding which pages to put in FLAM

Some relatively small fraction of memory is kernel-private address space that cannot easily be migrated. However, pages used for user address space and file system buffering, which consume most of the DRAM, should be candidates for migration.

Within the set of candidate pages, the OS must choose those that have the best “return on investment” for migration to FLAM. This policy can vary depending on

HW technology. We propose several heuristics, including some tests that use static information:

- **Page types**: Operating systems tend to associate pages with type information. Code pages, for example, are good candidates for migration to FLAM; stack pages are bad candidates. Our guess is that non-file pages shared between two processes might be bad candidates.
- **File types**: File types can be good indicators of ETTNW. Vogels reported [17] that the local-disk file size distribution "is dominated by executables, [DLLs], and fonts" – all read-only – and on network shares "the set of large files is augmented with development databases, archives and installation packages" – probably also read-only.
- **File reference modes**: Vogels pointed out [17] that Windows offers a *temporary file* attribute to optimize its buffer cache behavior, that up to 80% of files in a Windows file trace were deleted within 4 secs. of creation, and that "at least 25%-35%" of these deleted new files could benefit from that attribute. Clearly, pages from files marked temporary should not be migrated to FLAM; unfortunately, most applications do not mark their temporary files.
- **Application-supplied page attributes**: We speculate that certain large, memory-hungry applications that understand their workload, such as databases, could provide coarse ETTNW values for certain pages, such as index files.

and some dynamic tests:

- **File names**: The OS could record the historical TTNW distribution (or its smoothed mean) of otherwise hard-to-classify files; limiting this DB to relatively large, frequently accessed files would maximize the benefits. Pages from a file with a large observed TTNW could be migrated to FLAM.
- **Page history**: In theory, the OS could track the TTNW for each page, and migrate pages with high observed TTNW. In practice, the DRAM space overhead for this tracking could be excessive, as we discovered when trying to design Linux modifications for generating page lifetime traces (see sec. 4.2). (Remember, the whole point of FLAM is to save DRAM.)

Throughout this paper, we have argued for using NVM as main memory while ignoring its non-volatility benefits. Of course, several OS and application functions could benefit from non-volatile main memory, and the OS could use this information to drive FLAM migration. We should note, however, that some superficially suitable applications (e.g., RIO [6]) may not have sufficient low page-write rates to work with FLAM.

Migration decisions would, of course, also depend on other factors, such as how much free FLAM space is available, and tradeoffs between the extra CPU load for migration versus the inefficiency of running out of DRAM. This is likely to involve more heuristics than theory, just as with page-frame reclamation algorithms [3].

2.4 Garbage collection

Since flash erase blocks are larger than pages, the OS will have to garbage collect as FLAM pages migrate back to DRAM, so that it can erase blocks in order to maintain a large-enough pool of free FLAM pages. We assume typical "copying garbage collector" mechanisms will apply, such as allocating pages with similar expecting lifetimes to the same erase block. For example, pages from the same mapped file probably have similar lifetimes.

Determining how large a "large-enough" pool of free FLAM pages may require adaptation to the different write and erase delays of different NVM technologies. We believe that the kernel could parameterize its own queueing model based on on-line observations of these delays.

2.5 Wear leveling

Because flash has limited endurance, the OS needs to implement wear-leveling for the FLAM. Generally, this requires tracking the allocation status (allocated, free, erased) and erase-count for each FLAM block. The OS can then migrate pages to the erased block with the lowest erase-count. Since this metadata must persist across crashes and reboots, it should itself be stored in the FLAM, for non-volatility.

Flash endurance appears to be stated in terms of the number of erase cycles guaranteed by the vendor. We suspect that many flash blocks have longer endurance (and some will have shorter endurance) than this value; however, we have not been able to learn what endurance distributions look like.

As we understand it, flash writes basically work by damaging the dielectric, and wear-out occurs when this damage is permanent. Therefore, we believe that if the OS reads a FLAM page immediately after migration and compares it successfully to the source page, the OS can assume that the page has not yet worn out. If the comparison fails, the OS can abort the migration, and set the page's erase-count to infinity. This approach might squeeze out more lifetime from flash.

2.6 Support for virtual machines

Our design assumes that the operating system kernel both controls the hardware and has extensive information about the way that applications use memory pages. When the operating system runs on top of a virtual machine monitor (VMM), these two aspects are split: the VMM controls the hardware, and the operating system knows about page-type-specific memory uses. In order to exploit FLAM and virtual machines simultaneously, which may be a pre-requisite for most data-center applications of FLAM, we expect that it will be necessary to use paravirtualization – that is, provide an explicit interface between the OS and the VMM to support the use of FLAM.

3 Designs for hybrids using PC-RAM

PC-RAM differs from flash in several important characteristics, both in terms of scaling (i.e., higher densities and potentially lower costs/bit) and endurance. We have seen a variety of estimated endurances for PC-RAM, possibly because there is some uncertainty as to which chalcogenic material yields the best tradeoffs [15], and also because the endurance for a large array is expected to be lower than for a single bit [4]. It seems reasonable to assume an endurance of 10^8, which is 2 or 3 orders of magnitude better than flash, but still finite.

PC-RAM also eliminates the erase phase associated with flash, greatly simplifying the overall system design.

While PC-RAM parts are almost ready to appear (e.g., from Samsung in 2009), we have not been able to find data sheets that specify read and write timings. It appears that PC-RAM writes cannot be faster than about 100ns [15], which implies that it might still be necessary to buffer writes in DRAM (as in sec. 2.1). We expect reads to happen at DRAM-like speeds, however.

One aspect of PC-RAM is that there is a risk of thermal cross-talk: “repeated programming operations on a cell can induce an unwanted heating of the adjacent bits that can lose the data stored.” [14]. Although Pirano *et al.* suggest that this should not be a problem in practice down to a feature size (F) of 65 nm, there is some possibility that at finer scales, thermal cross-talk could be a problem. This suggests that the OS might need to control how frequently certain regions of PC-RAM are written, to avoid cross-talk.

Therefore, a hybrid using PC-RAM instead of NOR should allow us to simplify the design in several ways: **no garbage collection**: since there is no need to pre-erase large blocks, and **possibly simpler wear-leveling**: since PC-RAM endurance should allow fairly frequent writes (a mean write-spacing of 1.6 seconds for a 5-year lifetime, assuming 10^8 cycles allowed). However, PC-RAM might add some complexity; for example, wear-out might not be immediately detectable using read-after-write, if (for example) “resistance drift” is to blame [4].

4 Preliminary experiments

The experiments we report here are quite preliminary. Our goal was to understand whether enough memory pages are accessed read-only for long enough intervals to justify migrating them into FLAM. (Prior work by Clark *et al.* exploited this same phenomenon for VM migration [7].)

We define “long enough” as a mean allowable Time Between Writes to a Page (TBWP), based on the endurance of the NVM technology and a desirable lifetime for the hardware. We chose a 5-year lifetime (which is probably slightly longer than typical server replacement life cycles). 5 years is 1.58e8 seconds, and so for endurances of 10^6 (flash) and 10^8 (PC-RAM), our target TBWPs are 158 sec. and 1.58 sec., respectively.

We considered three experimental approaches: hardware tracing via simulation, page-level tracing via kernel modification, and simulation of a FLAM system. We have only tried the first approach, so far.

4.1 Simulator-based tracing

We can get precise traces of when pages are written using an architectural simulator. We used COTSon, a relatively fast simulator [2]. However, even with COTSon, tracing at this level is slow. which limits the amount of simulated execution we can achieve. For example, one of our runs, covering 213 traced seconds, took 10 days.

We ran simulations of a 1-core Opteron CPU (64KB L1 caches, 2 MB L2 write-back cache, 2 GB main memory, 4KB/page) and Linux 2.6.15 running one of two applications: Nutch, a web-search engine (1.2 GHZ CPU), and the SPECjbb benchmark (3 GHz). SPECjbb was slightly modified for simulation feasibility, and was configured for 28 warehouses.

We had COTSon trace all physical memory write-backs from the L2 cache, then computed a series of “write intervals” for each (physical) page; the length of an interval is the TBWP. We then calculated the median (50th percentile) TBWP for each page; fig. 1(b) shows the distribution of the fraction of pages as a function of their median TBWP.

Although these traces are too short to show definitive TBWP results for a target of 158 sec., fig. 1(a) shows that more than half of the pages in both benchmarks have median TBWP above 15.8 sec. Even if we want at least 75% of a page’s write intervals to be longer than the target (see fig. 1(c)), we could still put about half of the pages in PC-RAM.

We also simulated a 4-core Nutch trial (800GHz Opteron, 8 MB L2), with a different dataset and workload. This trial modifies pages more rapidly, but still leaves more than half of RAM entirely unused; we are not sure yet what accounts for the difference in behavior.

4.2 Linux-based page-lifetime tracing

To trace page references over much longer intervals than simulation can support, we tried to modify Linux to trace the dirty/clean state of pages. We augmented each page frame descriptor with two flags: HasBeenDirtied and PrevHasBeenDirtied. We modified the macro that sets the dirty flag (PG_DIRTY) so that it sets HasBeenDirtied, but the macro that clears PG_DIRTY does not clear HasBeenDirtied.

We modified the `/proc/kpageflags` interface to expose the new flags, and to provide a way to cause all valid page frames to be scanned, rippling HasBeenDirtied to PrevHasBeenDirtied and then clearing

HasBeenDirtied.

Every N seconds, a user-mode daemon uses the modified `/proc/kpageflags` interface to record these bit values for all valid pages, and then to ripple the bits.

Thus, the trace records accurately list all pages that have ever been dirtied during the N-sec. interval. The trace can also record information such as the page's process ID, type, and (if appropriate) the mapped file inode. While we get only N-second TBWP granularity from this method (vs. exact TBWP values from hardware simulation), the traces are a lot smaller and can cover a lot more time.

(Currently, we have not finished debugging this approach, so we have no valid results to report.)

5 Challenges and some next steps

Evaluation and commercialization of FLAM faces a number of challenges, including:

- **Need for better simulations**: Our simulations (Sec. 4) covered very brief execution periods for just a few applications. We also were unable to simulate the effect of incorporating page-type-specific and application-specific knowledge.
- **Trace-based evaluation**: Our trace-based evaluation approach (Sec. 4.2) could answer many of the questions about using page-type-specific knowledge, but it needs to be debugged first.
- **Prediction accuracy**: We are not sure whether the kernel could predict TTNW with sufficient accuracy to make FLAM viable.
- **Costs for FLAM hardware**: While the FLAM hardware design is conceptually rather simple, to actually implement it in a way that avoid adding significant costs (for engineering and manufacturing) might be difficult.
- **Design choices for FLAM hardware**: The best choice of NVM memory technology (Flash vs. PC-RAM vs. Resistive RAM, etc.) for FLAM is still unclear. We also do not know how much DRAM to include for buffering, and or what the appropriate page sizes should be.
- **Kernel support**: We believe that adding kernel support for FLAM is feasible, but our limited experience with the Linux virtual memory system suggests that this could be a tedious, error-prone task for a non-expert. There might also be some challenges with respect to software and space overheads.
- **VMM support**: As we note in Sec. 2.6, extending FLAM to virtual machines will require additional work.
- **Garbage-collection and wear-leveling**: These issues, while probably tractable, will require significant software effort and parameter tuning.

Therefore, we view our work on FLAM as very preliminary, and not a true proof of concept.

6 Summary

We have argued that it might be advantageous to build main memory out of NVM+DRAM hybrids. To do so,

- Some operations (e.g., writes for NOR flash and possibly for PC-RAM; reads and writes for NAND flash) will require DRAM buffering to match the timing constraints of a synchronous interface. This buffering will be visible to the OS, and so will affect the OS's page-migration implementation.
- Avoiding problems with endurance will require some intelligence to decide what pages should migrate to NVM, and when; the OS appears the best place to apply that intelligence.
- Similarly, slow write speeds (especially for NOR flash) will also require OS-based intelligence to manage page migration.

In short, the exploitation of future universal memory technologies is likely to create new opportunities for OS research and innovation.

References

[1] J. Åkerman. Towards a Universal Memory. *Science*, 308(5721):508–510, 22 Apr. 2005. Summary at `http://scienceweek.com/2005/sw050520-1.htm`.

[2] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, and D. Ortega. COTSon: Infrastructure For Full System Simulation. *Operating Systems Review*, 43(1), Jan. 2009.

[3] D. P. Bovet and M. Cesati. *Understanding the Linux Kernel, Third Edition*. O'Reilly, 2005.

[4] M. J. Breitwisch. Phase change memory. *Interconnect Technology Conf.*, pages 219–221, June 2008.

[5] G. W. Burr, B. N. Kurdi, J. C. Scott, C. H. Lam, K. Gopalakrishnan, and R. S. Shenoy. Overview of candidate device technologies for storage-class memory. *IBM J. Research and Development*, 52(4/5), Jul./Sep. 2008.

[6] P. M. Chen, W. T. Ng, S. Chandra, C. Aycock, G. Rajamani, and D. Lowell. The Rio File Cache: Surviving Operating System Crashes. In *Proc. ASPLOS*, pages 74–83, 1996.

[7] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of Virtual Machines. In *Proc. NSDI*, pages 273–286, 2005.

[8] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, D. Burger, B. C. Lee, and D. Coetzee. Better I/O Through Byte-Addressable, Persistent Memory. In *Proc. SOSP-22*, pages 133–146, Big Sky, MT, Oct 2009.

[9] X. Dong, N. Muralimanohar, N. Jouppi, and Y. Xie. Leveraging PCRAM Technology to Reduce Checkpointing Overhead in MPP Systems. Under review, 2009.

[10] S. K. Lai. Flash memories: Successes and challenges. *IBM J. Research and Development*, 52(4/5), Jul./Sep. 2008.

[11] D. Lammers. Spansion Seeks DRAM Replacement in Servers. `http://www.semiconductor.net/article/CA6571267.html`, 18 Jun. 2008.

[12] M. LaPedus and D. McGrath. 'Universal memory' race still on the starting block. *EE Times*,

19 Dec. 2008. `http://www.eetimes.com/showArticle.jhtml?articleID=212501437`.

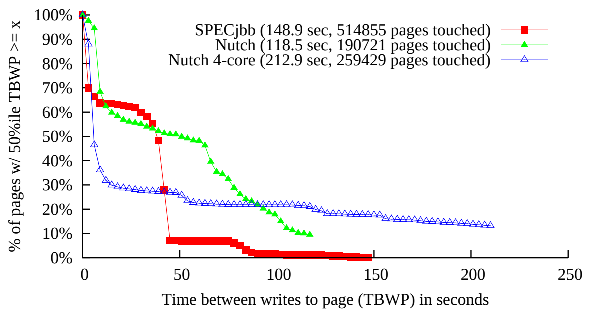
[13] K. Leigh. *Design and Analysis of Network and IO Consolidations in a General-Purpose Infrastructure*. PhD thesis, University of Houston, 2007.

[14] A. Pirovano, A. Redaelli, F. Pellizzer, F. Ottogalli, M. Tosi, D. Ielmini, A. Lacaita, and R. Bez. Reliability study of phase-change nonvolatile memories. *IEEE Trans. Device and Materials Reliability*, 4(3):422–427, Sept. 2004.

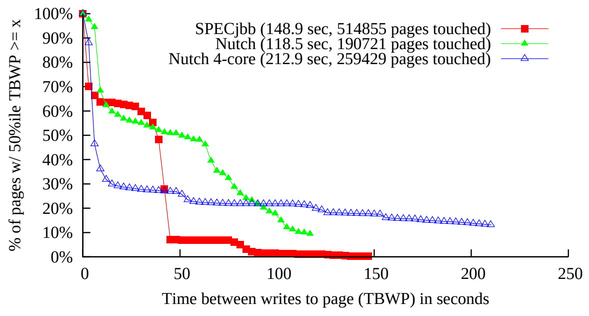
[15] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y.-C. Chen, R. M. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H.-L. Lung, and C. H. Lam. Phase-change random access memory: A scalable technology. *IBM J. Research and Development*, 52(4/5), Jul./Sep. 2008.

[16] Spansion, Inc. Using Spansion EcoRAM to Improve TCO and Power Consumption in Internet Data Centers. `http://www.spansion.com/about/news/events/spansion_ecoram_whitepaper_06%08.pdf`, 2008.

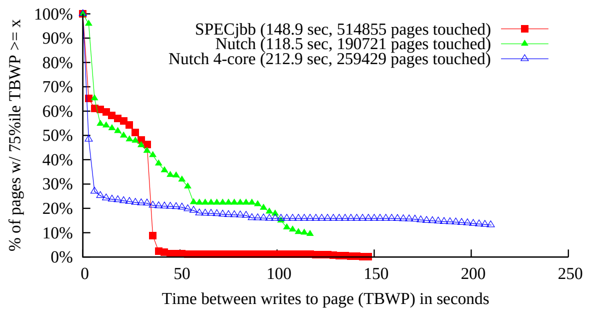
[17] W. Vogels. File system usage in Windows NT 4.0. In *Proc. SOSP*, pages 93–109, 1999.



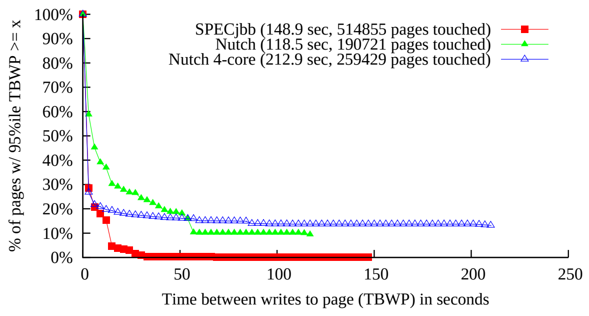
(a) 25th %ile TBWP



(b) Median (50th %ile) TBWP



(c) 75th %ile TBWP



(d) 95th %ile TBWP

Figure 1: Distributions of times between writes to pages