Operating System Implications of Fast, Cheap, Non-Volatile Memory

Katelin Bailey   Luis Ceze   Steven D. Gribble   Henry M. Levy
University of Washington
Department of Computer Science & Engineering
{*katelin, luisceze, gribble, levy*}*@cs.washington.edu*

Abstract. The existence of two basic levels of storage (fast/volatile and slow/non-volatile) has been a long-standing premise of most computer systems, influencing the design of OS components, including file systems, virtual memory, scheduling, execution models, and even their APIs. Emerging resistive memory technologies – such as phase-change memory (PCM) and memristors – have the potential to provide large, fast, non-volatile memory systems, changing the assumptions that motivated the design of current operating systems. This paper examines the implications of non-volatile memories on a number of OS mechanisms, functions, and properties.

1 Introduction

New memory technologies promise game-changing features whose impact felt broadly, from embedded computers to mobile devices to datacenters. For example, phase-change [14] and memristor [15] memories can provide fast, inexpensive, non-volatile, highly dense (denser than DRAM), byte-addressable storage systems. Think of the impact of a terabyte byte-addressable persistent storage chip on your mobile phone.

The architecture research community is actively exploring [9] the implications of fast, cheap non-volatile memory, including error correction mechanisms and memory hierarchy organization. Software research has focused primarily on NVRAM-based file systems that maintain current file system semantics [6] and on programming interfaces for persistent objects [3]. These efforts are evolutionary: they integrate NVRAM into existing architectures and programming structures.

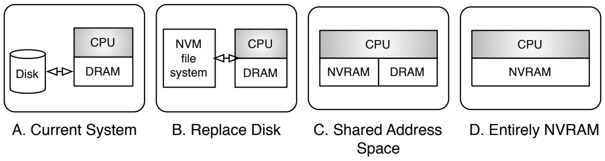
Instead, we believe that NVRAM could be revolutionary rather than evolutionary; this paper seeks to discuss the ways that cheap byte-addressable NVRAM could substantially affect OS design. For example, new NVRAM technology might change the basic premise of a two-level store (a fast primary memory and a slow secondary memory) that we have assumed for over 50 years. The structure of moving-head disks impacts the entire system structure, including the I/O system, virtual memory, the protection system, the scheduler, the way that processes are managed, and the way that programs are initiated. What if future systems contained only one level of memory that was persistent and uniform? What if there were no disk pages, no memory pages, no buffer cache, no page faults, no swapping, no booting on restart? How would we choose to organize the OS and persistent storage? How would we structure, share, and protect persistent storage? What parts of the OS could we simplify or remove, and what features or capabilities would be enabled by this technology?

The goal of this paper is not to propose a specific system, but simply to raise potential issues, questions, and opportunities that NVRAM technology presents for OS design. In the following sections we first discuss NVRAM technology and then examine some of the OS components and functions whose traditional designs might be reexamined for NVRAM-based systems.

2 Hardware alternatives

Cell Technology. Phase-change memory (PCM) and memristors are among the most viable of emerging cell technologies for cheap NVRAM. PCM cells are based on a chalcogenide material that, when heated, can be cooled into crystalline or amorphous states that have very different resistivity properties, hence encoding binary information. Memristors are passive two-terminal circuit elements whose resistance is a function of the history of current that flowed through the device. We focus on PCM in this section. Though not entirely commercially viable yet, there are several PCM prototypes available. One of the key challenges with current PCM technology is its relatively limited write lifetime (order of 10^8 writes).

Most of the work on PCM in the architecture community has focused on improving lifetime via wear-leveling [9] and exploring where PCM fits in the memory hierarchy [13]. Interestingly, several architectural al-



A. Current System B. Replace Disk C. Shared Address Space D. Entirely NVRAM

Figure 1: System architecture options for NVRAM.

ternatives focus on PCM mostly for its density, since it is projected to be denser than DRAM; in this context, non-volatility properties are largely unexploited. Current PCM cells have access delays of up to 150ns (writes are much slower than reads), which is several times slower than DRAM. However, some proposals mitigate this characteristic [9] with careful buffer organizations and show that most applications suffer only a slowdown of a few percent. Moreover, expected improvements in device technology are likely to further close this gap.

Architectural Alternatives. To set the stage for our discussion of operating system design in the next section, we first examine several architectural alternatives that are relevant to OS design, focusing on the hardware-software interface. To simplify the discussion, we ignore I/O. Figure 1 shows a progression of options, from a conventional system with CPU, DRAM and disk (option A) to a system with only CPU and NVRAM (option D).

The most direct and straightforward option (B) is to simply replace disks with NVRAM, since NVRAM could be seen as a dense and fast form of previous non-volatile storage technologies. This maintains many of the disk semantics that programmers are used to; it is essentially plug-and-play and changes very little other than secondary store access speed. Even the packaging and access interface (via an I/O bus) are the same as for normal disks. In essence there are two distinct address spaces, one for non-volatile data and one for volatile data. This is a logical evolution in solid-state disks – simply replacing current flash-based SSDs with PCM, which is two orders of magnitude faster.

The next option (C) is to move the NVRAM to the same physical address space as DRAM. Here the CPU sees two types of memory, one volatile and one non-volatile. This is more interesting (and general purpose) than the previous option, because NVRAM can be conveniently accessed with regular CPU load and store instructions. Previous systems have taken advantage of physical memory with heterogeneous characteristics, for example, non-uniform-memory-access (NUMA) machines offer uniform addressing but different access latencies depending on where data is located.

At the end of the spectrum is a system with only NVRAM in a single, flat, non-volatile physical memory space (D). This is the most advanced alternative, as it requires NVRAM to be as fast as DRAM so that the non-volatility of memory does not come at a performance cost. Such performance might be achieved by advances that make NVRAM cells as fast as DRAM cells or through the use of non-volatile caches and enough reserve energy (e.g., in large capacitors) to write caches back in case of a power failure. Relatedly, the previous alternative (C) is likely the most flexible from a hardware implementation perspective, because it allows NVRAM and DRAM to have different performance and power characteristics. Although the hybrid system (C) is the most flexible and potentially provides numerous areas of research into the interaction of DRAM and NVRAM, we chose to purse the more radical OS opportunities afforded by the all-NVRAM option (D).

3 OS implications

NVRAM has the potential to influence the design of major operating system components, program and OS execution models, and the performance and reliability characteristics of the overall system. We discuss each of these topics in turn.

3.1 OS Components

Virtual Memory. Today's computers have two levels of storage: slow, large capacity, durable secondary storage, and fast, smaller capacity, volatile primary memory. Many aspects of VM design are influenced by this.

- **Paging.** Virtual memory systems treat primary memory as a fast cache of secondary storage, paging data between these two levels. A computer with only NVRAM has one level of physical storage, not two, and no longer needs to page.[1]

- **Page granularity.** VM systems treat the page as the unit of allocation, hardware protection, and transfer between primary memory and disk. Page size is influenced by many factors, including minimizing fragmentation, amortizing the latency of disk seeks, and minimizing the overhead of page table structures. In an NVRAM-based computer, pages could still be useful for address translation, protection, and memory management (e.g., system-supported garbage collection); however it is no longer clear what page size is appropriate, as many of the factors noted above will change or be eliminated. More aggressively, one can imagine mechanisms for memory protection and allocation that do not use pages

[1] RAMCloud [12] uses large DRAMs to remove secondary store and paging as well. They focus on improving performance for datacenter applications, while we focus on the OS implications of non-volatility.

at all (e.g., [16, 10]), which might be especially interesting in an all-NVRAM system.

- **Separate protection systems for primary and secondary storage.** Virtual memory provides hardware-enforced page-granularity protection, whereas file systems have a richer but coarser protection structure. In an NVRAM-based computer, both protection systems might need to be reconsidered. Since there is only one level of storage, hardware-enforced protection and the protection abstractions the OS provides will likely need to reflect each other more closely, and perhaps they can even be unified.

- **Multiple address spaces.** Today's VM systems provide separate address spaces for each process, whereas file systems expose a single global namespace. Given that an NVRAM-based computer has a single, fast, large-capacity store, alternative structures such as single-address-space operating systems [5] might be more appropriate, especially if we simultaneously reconsider how protection works.

Overall, the existence of large-scale, dense, byte-addressable persistent memory calls into question many properties of traditional virtual memory systems. Virtual memory can be re-architected, simplified, or generalized for computers with a single level of durable, fast storage.

File systems. File systems for non-volatile memory have already received significant research attention. For example, Condit's work on BPFS [6] demonstrates that file systems that optimize for byte addressability can achieve significantly higher performance on non-volatile storage than existing sector-oriented file systems.

However, fundamental questions remain. BPFS is still predicated on a computer with two levels of physical storage: it redesigns the file system to reflect a different secondary storage technology, but it keeps the same API and does not consider how storage should be organized on a system with *only* non-volatile memory. A major question for an NVRAM-based computer is whether the operating system should expose a single logical storage system to programs and users, similar to Multics [1], or whether it is still better to provide separate interfaces and semantics for virtual memory and file systems.

3.2 Execution Models

In NVRAM-based systems, nearly all execution state is durable.[2] Even with unmodified operating systems and applications, after power is lost and restored, such a computer should have the ability to resume execution where it left off, with the exception of re-establishing any lost volatile device state. More broadly, non-volatile memory could influence our models of how programs are installed, launched, and executed, what it means to boot, and how systems handle software faults.

3.2.1 Applications and Processes

Non-volatile memory can affect several aspects of how users, programmers, and operating systems interact with applications and running processes.

Application installation and launch. Today, applications exist in three different forms: their packaged state prior to installation, their post-installed state in the file system, and their execution state when launched into processes. Non-volatile storage has the potential to blur the boundaries between these, similar to how cheap checkpointing and migration have blurred the same boundaries for virtual machines and appliances [4].

Instead of shipping in packaged form, applications could ship as checkpoints of an executing process, assuming that the checkpoint can be taken at a "convenient" moment where there are as few dependencies as possible in the underlying OS, such as outstanding system calls. Independently, instead of distinguishing between inactive applications and active processes, every application that is resident on a local computer could be perpetually "active," since its execution state is durable and, in effect, is a checkpoint for the process. The concept of launch would no longer need to be exposed to users, and instead applications are scheduled whenever some form of input (user, device, or other event) is directed towards them.

Faults. If an application experiences a fault, however, a persistent address space can become a liability: some mechanism needs to be in place to recover, rollback, or otherwise rejuvenate the application. One possibility is for the system software to automatically checkpoint program execution over time, using logs of non-deterministic inputs or copy-on-write chains of the program's execution state. Given this, transparent rollback recovery could be attempted, though there are known limits [11]; instead, operating systems and users might need to identify safe checkpoints to which programs can revert. As well, even though NVRAM is expected to be large and cheap, logs or checkpoint state will eventually need to be garbage collected.

[2] Processor registers and cache might still be volatile, but processors could use capacitance-based energy reserves to stash transient state in non-volatile memory when power is lost. Device state, such as in-flight packets or GPU state, is another concern.

Update. If applications never terminate, updating applications becomes more tricky. The simplest case is to treat an update as a separate version or checkpoint of the application, allowing users to choose when to switch between old and new, though this is undesirable in the case of urgent security patches. In the extreme, system software or users might be able to name any historical checkpoint of application state and switch execution to that checkpoint at any time; updates just become new branches in the historical timeline. Other alternatives include hot-patching a running program, which seems technically messy and complex, or re-introducing a notion of halting and restarting an application to update it.

More generally, as the differences between long-term durable application state and short-term transient execution state diminish, we need to reconsider who decides when and how to launch, quit, and reset an application, and what these concepts mean. Is it the user, the application developer, or the OS who makes these decisions?

3.2.2 Operating Systems

Many of the questions raised about execution models apply to operating systems as well. Today, OS boot (or reboot) is an all-too-frequent event. Given non-volatile memory, we can change the idea that a power cycle must trigger the loss of OS execution state and system re-initialization. Instead, power loss and software rejuvenation can be completely decoupled from each other.

Given this, we can imagine several alternatives. For simplicity, we could choose to couple power loss with "reboot." At the other extreme, the entire system could resume uninterrupted after power is restored. In between are designs that microreboot or selectively rejuvenate OS data structures, components, or applications [2], such as by rolling back to checkpoints or equilibrium states.

These questions extend to the kernel, devices, and power management. While some system structures could be durable after power loss, others—such as those dealing with network and I/O devices—may need to be reexamined and updated. Along these lines, NVRAM gives systems more flexibility to power down subsets of the computer. Some I/O devices may be able to continue DMA transfers, even when the CPU is powered off, as long as NVRAM receives power. Similarly, the OS could cause power to stop flowing to the CPU and memory completely when the system is completely idle, quickly resuming when work arrives via an external device without needing to reload memory context.

3.3 System Characteristics

Because NVRAM is non-volatile, it has the potential to influence reliability, security, and privacy. We believe it can be beneficial to systems and users, but some care must be taken, to avoid harmful consequences.

3.3.1 Reliability

Because all execution state is non-volatile and power failure is decoupled from system failure, NVRAM-based systems can be more reliable: data is less likely to be lost after a crash. However, two issues arise:

- **Data corruption.** If a hardware or software fault corrupts application state, then that corruption is non-volatile, particularly if the system has a single-level store. A benefit of having a two-level store is that data is (implicitly or explicitly) scrubbed as it is transferred between the levels. As well, with the exception of mmap'ed data, memory corruption is unlikely to propagate to the file system. An NVRAM-based system might need to give programmers abstractions to regain this kind of resilience.

- **Data portability.** Hard drives and SSDs can be physically removed from one computer and mounted on another, as file system formats are independent of specific computer configurations and system architecture. NVRAM-based systems might lose this property, both because NVRAM might not be easily removable by users, and because NVRAM will contain non-portable architecture-specific execution state as well as potentially portable application data. Either OSs need to provide applications with a way to separate out the two kinds of state, or perhaps it is finally time to assume that all migration of data between devices will happen over networks and through the cloud.

3.3.2 Security, Privacy, and Forensics

Users' computing devices are increasingly theft-prone, thanks to the proliferation of laptops, pads, and smartphones. As well, computers are still susceptible to attack by remote adversaries or local malware. In either case (theft or intrusion), critical data on the device is at risk. Since execution state is durable even after a device is powered down, NVRAM can exacerbate this risk [7]: NVRAM makes cold-boot attacks [8] trivial to perform.

Volatility is therefore beneficial in some cases. For example, applications should explicitly destroy encryption keys and decrypted data after finishing with them. If an application forgets to do this or is not given the opportunity because of a crash or malicious halt, then it would be helpful if the hardware or OS ensured the memory used to store them was volatile. This could be done physically, by including some volatile RAM in hardware, or virtually, by having the OS use some combination of encryption or post-deallocation scrubbing on top of NVRAM.

Non-volatility could also improve forensic capabilities. If the OS provides frequent checkpoints, durable logs, or uses COW to preserve historical execution state in NVRAM, more information is available for forensic tools after an incident. On the other side of this coin lies privacy and confidentiality: if these forensic logs are not adequately protected, or in the absence of adequate garbage collection policies, the risk of information disclosure after theft or break-in is amplified as data will endure indefinitely.

We suspect that operating systems and programming languages will need to let programmers and applications help manage issues of volatility and durability. Only applications are aware of the meaning of certain types of data, and in some cases, only users will be able to indicate the risks of losing (or retaining) different pieces of information. Devising appropriate abstractions and mechanisms for managing this at the OS, language, and application level is an open problem.

4 Conclusions

This paper discussed the OS implications of new NVRAM in future systems. While OS changes are not strictly required by NVRAM, we believe that it offers an opportunity or incentive to finally update the 40-year-old architecture that still underlies most of today's operating systems. We are currently working on a new operating system whose design is motivated by future NVRAMs.

Acknowledgments

We thank the anonymous reviewers for their helpful comments. We thank Wen-Hann Wang and Richard Uhlig for their insight during earlier discussions of this work. This work was supported by NSF grants CNS-0627367 and CNS-1016477, the Torode Family Career Development Professorship, the Wissner-Slivka Chair, and gifts from Intel Corporation and Nortel Networks.

References

[1] A. Bensoussan, C.T. Clingen, and R.C. Daley. The multics virtual memory: Concepts and design. *Communications of the ACM*, 15(5), May 1972.

[2] George Candea, Shinichi Kawamoto, Yuichi Fujiki, Greg Friedman, and Armando Fox. Microreboot – a technique for cheap recovery. In *OSDI '04*, December 2004.

[3] Adrian M Caulfield, Arup De, Joel Coburn, Todor I Mollov, Rajesh Gupta, and Steven Swanson. Moneta: A High-performance Storage Array Architecture for Next-generation, Non-volatile Memories. *MICRO*, 2010.

[4] Ramesh Chandra, Nickolai Zeldovich, Constantine Sapuntzakis, and Monica S. Lam. The Collective: A cache-based system management architecture. In *NSDI 2005*, May 2005.

[5] Jeffrey S. Chase, Henry M. Levy, Michael J. Feeley, and Edward D. Lazowska. Sharing and protection in a single-address-space operating system. *ACM Transactions on Computer Systems*, 12(4), 1994.

[6] Jeremy Condit, Edmund B. Nightingale, Christopher Frost, Engin Ipek, Benjamin C. Lee, Doug Burger, and Derrick Coetzee. Better I/O Through Byte-Addressable, Persistent Memory. *SOSP*, 2009.

[7] William Enck, Kevin Butler, Thomas Richardson, Patrick McDaniel, and Adam Smith. Defending Against Attacks on Main Memory Persistence. In *ACSAC 2008*.

[8] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In *USENIX Security*, July 2008.

[9] B.C. Lee, Ping Zhou, Jun Yang, Youtao Zhang, Bo Zhao, E. Ipek, O. Mutlu, and D. Burger. Phase-change technology and the future of main memory. *Micro, IEEE*, 30(1), 2010.

[10] Henry M. Levy. *Capability-Based Computer Systems*. Digital Press, 1984.

[11] David E. Lowell, Subhachandra Chandra, and Peter M. Chen. Exploring failure transparency and the limits of generic recovery. In *OSDI*, October 2000.

[12] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazieres, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum, S. Rumble, E. Stratmann, and R. Stutsman. The Case for RAMClouds: Scalable High-Performance Storage Entirely in DRAM. *SIGOPS Operating Systems Review*, 43(4), December 2009.

[13] Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers. Scalable High Performance Main Memory System Using Phase-change Memory Technology. In *ISCA '09*, 2009.

[14] 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. Res. Dev.*, July 2008.

[15] Dmitri B. Strukov, Gregory S. Snider, Duncan R. Stewart, and R. Stanley Williams. The Missing Memristor Found. *Nature*, March 2008.

[16] Emmett Witchel, Josh Cates, and Krste Asanović. Mondrian memory protection. In *ASPLOS*, 2002.