Mnemosyne: Lightweight Persistent Memory

Haris Volos *, Michael Swift
University of Wisconsin–Madison
{hvolos, swift}@cs.wisc.edu

Andres Jaan Tack
Skype Limited
andres.jaan.tack@skype.net

Fast, cheap, and persistent memory has long been a dream for computer designers. Until recently, non-volatile storage was either slow (*e.g.*, disks) or expensive (*e.g.*, NVRAM). However, several new technologies promise cheap and fast storage that survives across system boot. For example, phase-change memory (PCM) provides near-DRAM speeds and is currently available in sizes up to 64 MB, and memristors may enable multiple terabytes of non-volatile memory to be placed on-chip . These devices are termed *storage-class memory* (SCM) as they provide the interface of memory (load and store instructions) but the persistence of disks.

Existing operating systems are designed for a strict bifurcation of devices into *memory*, fast random-access volatile structures erased on reboot, and *storage*, persistent, slow block-based devices. Thus, research on SCM tends to follow the same path. For example, recent work investigates use of PCM within file systems [1], or as a low-power volatile DRAM replacement [2]. Neither of these approaches exposes the full power of SCM to programmers.

We propose that operating systems should expose a new abstraction, *persistent memory*, to provide user-mode applications direct access to durable storage. This abstraction enables programmers to make in-memory data structures persistent without first converting them to a serialized format.

We see persistent memory not as a replacement for files, but as a fast mechanism to store moderate amounts of data. For example, Firefox 3 had a problem with calling `fsync` too frequently , bringing the system to a crawl by frequent flushes to disk. Persistent memory could address this problem by providing low-latency storage of program state. Other uses could be configuration changes, snapshots of in-progress edits, and logs in distributed agreement protocols . Applications can still use files for interchanging data.

We have three goals for our system to expose persistent memory. First, it must be simple for a programmer to declare data as persistent, and persistence must fit naturally into existing programming models for volatile data structures. Second, and more important, the system must support *consistent modifications* of data structures. The system must enable programmers to move data structures between consistent states, automatically recovering to such a state after a failure. Finally, we seek a design that is compatible with existing commodity processors.

We are building *Mnemosyne*[1], a lightweight system for exposing persistent memory to user-mode programs. Mnemosyne provides three key services that simplify programmer use of persistence. First, it provides *persistent memory regions*, segments of virtual memory stored in SCM rather than volatile memory. Regions can be created automatically to hold variables labeled with the keyword `persistent` or allocated dynamically. Mnemosyne virtualizes persistent regions by swapping SCM pages to a backing file. Second, Mnemosyne provides *persistence primitives*, low-level operations that support consistently updating data such as logging, shadow paging, and single variable atomic writes. Finally, Mnemosyne provides a *durable memory transaction* mechanism that enables consistent in-place updates of arbitrary data structures. Thus, Mnemosyne provides a low-level programming interface, similar to C, for accessing persistent memory. Upon this base, higher-level services such as garbage collection and safe references that ensure persistent data does not point to volatile data, can be provided by language frameworks.

Compared to past work on persistent memory and persistent object stores such as ObjectStore, Thor, Texas, LRVM, and QuickStore , Mnemosyne provides a low-level interface allowing both high-level transactions as well as low-level consistent updates. Furthermore, it operates at a much finer granularity than the virtual memory pages used by these systems. Most importantly, SCM enables the implementation to be much simpler, as data can be made persistent without writing it out through the file system.

We have implemented a prototype as a pair of libraries and a small set of modifications to the Linux kernel for allocating and virtualizing SCM pages. We designed Mnemosyne to run on conventional processors, requiring no special support beyond the necessary memory controller for SCM, and implement it using regular x86 instructions with a performance emulator for SCM accesses. Initial experiments show that Mnemosyne provides a simple abstraction for programmers to make data structures persistent. We compare Mnemosyne performance against Berkeley DB running on a RAM disk with the performance of PCM. For small data sizes, Mnemosyne transactions perform 50–250 percent better than Berkeley DB. We also convert two applications, OpenLDAP and Tokyo Cabinet, to use persistent memory and find the performance of moving existing in-memory data structures to persistent memory is 20–280 percent faster than Berkeley DB's optimized storage or flushing the whole structure to a file.

References

[1] CONDIT, J., NIGHTINGALE, E. B., FROST, C., IPEK, E., LEE, B., BURGER, D., AND COETZEE, D. Better i/o through byte-addressable, persistent memory. In *SOSP 22* (2009).

[2] LEE, B. C., IPEK, E., MUTLU, O., AND BURGER, D. Architecting phase change memory as a scalable dram alternative. In *ISCA 36* (2009).

* Student

[1] Mnemosyne is the personification of memory in Greek mythology, and is pronounced *nee–moss–see–nee*.