

Persistent memory view of in-system storage

Brian Van Essen Maya Gokhale
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA 94550
{vanessen1, gokhale2}@llnl.gov

Over the past few years of investigation into potential exascale systems, there is growing consensus that an exascale machine will have to incorporate in-system storage, a radical departure from current HPC architectures that relegate storage to a separate network with limited connectivity to compute resources. This architectural shift to include storage on or near compute nodes is driven by concerns of memory capacity and reliability at the exascale.

A many-core compute node will have significantly smaller memory per core since the order of magnitude increase in number of cores will not be accompanied by a commensurate increase in node memory due to constrained power budget. Separately, it is expected that the failure rate will increase with increased component count, making defensive storage of state even more critical than in present-day systems. However, the three orders of magnitude increase in number of nodes will tax the I/O network to the point of non-functionality if traditional checkpoint methods are followed.

In-system storage encompassing 10^{18} tightly coupled computing elements presents significant challenges to the OS and runtime stack. Even at the level of a single node, the OS file system management algorithms were designed when the average access time to storage was measured in milliseconds. High performance storage now has access time measured in microseconds, which necessitates a re-evaluation of the existing buffering and I/O coalescing schemes. From the exascale perspective, there are many additional unanswered questions regarding how the global in-system storage resource should be managed.

- On a single node (assuming node local storage), should the storage be organized as a POSIX file system, with full OS VFS support? As a collection of block devices managed by an application (library)? As a swap device? As a hybrid between a ram disk and a true external disk-like store?
- How should the OS/R present a node's storage to other nodes?
- Should node local storage be managed as a cache to an external file system? If so, does that mean that all node local data must be mirrored in the external file system, or is there a need for multiple runtime libraries

that manage application-specific caches, pushing only a subset of data externally?

- What policies should the job management system impose on in-system storage? Leave in place until explicitly deleted or for a maximum duration? Mark deleted in the directory, but leave the data on the device? Erase after every job chain?

Challenges addressed Among these many research issues concerning in-system storage, we focus on a node local view of the storage that extends the memory hierarchy to incorporate low latency, random access NVRAM to hold persistent data structures. We assume that in-system storage will be in the form of NAND Flash, PCM, or hybrid arrays attached to a low-latency network and will be used to store and load persistent objects as if in memory (load/store rather than read/write). Heroux [1] advocates this abstraction as particularly valuable for local checkpoint/restart. We believe the persistent memory will have many other use cases, such as local re-play for steering of simulations, and to store data structures used for in-situ data analysis. Efficient OS support for memory-mapped access to persistent regions will greatly ease the application development effort compared to explicitly managing out-of-core state.

We advocate an advanced memory management runtime within the node OS that optimizes access to the persistent memory that is mapped into the application's address space. The key components to a high performance memory management runtime are effective buffering of application data, performance under memory pressure, and efficient use of low latency, random access storage. The fundamental insight to realize these goals is that (persistent) memory management can no longer be limited to a single "generic" use pattern, but must be optimized differently for different applications' behaviors.

It is no longer sufficient to assume that all data is accessed uniformly and should be managed in a monolithic buffer that grinds the system to a halt when the physical memory fills up. It is necessary for the operating system to understand that there are different tiers of user data and that different buckets of data within these tiers should be buffered and evicted differently. In this scenario, the OS/R manages distinct persistent memory regions that have been allocated by applications, and caches pages of regions into DRAM caches that are distinct from the file system page

buffers. The alternative page caching mechanisms might be invoked through user directives (eg. different policies for different persistent regions) and/or automatically through online profiling and adaptation.

A high performance memory management runtime has the opportunity to address the following challenges: limited main memory, high performance in-system storage, and minimizing energy consumption. The combination of latency tolerant, throughput-driven, massively parallel application with efficient data movement in the operating system will enable algorithmic data to reside in high performance storage [2] with minimal loss in performance over all-memory.

Additionally, the storage stack in traditional operating systems is optimized for sequential access to slow, rotating media. NVRAM demands new algorithms that are tuned for random access and low latency. Enabling effective use of NVRAM allows systems to minimize energy consumed by DRAM main memory. Furthermore, reducing inefficient data movement between main memory and the storage hierarchy lowers energy consumption.

Maturity Research into power and energy management is by far the most mature although the use of persistent storage to reduce main memory costs is still being proposed and discussed.

Research into the effectiveness of NVRAM for data-intensive computing is active and in progress. In the past four years, there have been several efforts to study effective methods for integrating NVRAM into the memory and/or the storage hierarchy at a hardware level [3]. Flash-optimized file systems and flash as cache to disk storage have also been studied. Mnemosyne [4], Moneta [5], and PerMA [6], have explored the impact of the operating system on the efficacy of NVRAM.

One example of optimizing data-intensive applications for latency tolerance and out-of-core execution is Pearce's work [7] with semi-external and external graph algorithms.

Uniqueness The need for efficient memory management is pervasive throughout high-performance computing and enterprise computing. Adding persistent memory in the form of low latency, random access NVRAM into the memory hierarchy of HPC compute nodes is more unique.

Novelty During the 1980s and 1990s there was a great deal of research in the micro-kernel community about efficient memory management and mechanisms allowing application directed runtime tuning. While many of these ideas were successful very few of them found fruit in macro-kernel operating system, such as Linux. Furthermore the dramatic growth in DRAM capacity diminished the impact of many of these techniques. Now with the average DRAM capacity per core declining and the availability of high-performance NVRAM, there is a great opportunity to revisit many of these ideas and explore their impact for exascale HPC and data-intensive applications.

Applicability These techniques are broadly applicable

across high-performance computing and also to data-intensive enterprise computing.

Effort To bring a high performance memory management to fruition will require some in-depth research and a fair bit of engineering. The research component is required to identify what algorithms and adaptability will provide the best performance for different application use cases. The engineering component for integrating a high-performance memory management runtime into an exascale operating system will be fairly time consuming and intrusive. In particular, for most macro-kernel operating system such as Linux the memory management system is integrally tied into the OS kernel. Therefore providing a suite of memory management profiles, including reasonable default schemes for different device drivers, file systems, and between kernel/user space will require sustained effort.

REFERENCES

- [1] M. A. Heroux, "A local checkpoint/restart api based on persistent data storage," <http://exascaleresearch.labworks.org/apr2012/dataforms/view/ShowFile/sectionId/D7CEFB8720B4EB6A3200E90D0EA0661/file/performedId/B236696AD75F48CD80577A0D58CFE8FB>.
- [2] B. Van Essen, R. Pearce, S. Ames, and M. Gokhale, "On the role of NVRAM in data-intensive architectures: an evaluation," in *IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, Shanghai, China, May 21-25 2012, pp. 703–714.
- [3] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *Proceedings of the 36th annual international symposium on Computer architecture*, ser. ISCA '09. New York, NY, USA: ACM, Jun 2009, pp. 24–33.
- [4] H. Volos, A. J. Tack, and M. M. Swift, "Mnemosyne: lightweight persistent memory," in *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, ser. ASPLOS '11. New York, NY, USA: ACM, 2011, pp. 91–104.
- [5] A. M. Caulfield, A. De, J. Coburn, T. I. Mollow, R. K. Gupta, and S. Swanson, "Moneta: A high-performance storage array architecture for next-generation, non-volatile memories," in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 385–395.
- [6] B. Van Essen, R. Pearce, S. Ames, and M. Gokhale, "On the role of NVRAM in data intensive HPC architectures," in *Workshop on Emerging Supercomputing Technologies In Conjunction with the 25th International Conference on Supercomputing*, May 2011. [Online]. Available: <https://sites.google.com/site/ics2011west/>
- [7] R. Pearce, M. Gokhale, and N. M. Amato, "Multithreaded asynchronous graph traversal for in-memory and semi-external memory," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–11.