Enabling NVM for Data-Intensive Scientific Services

Philip Carns, John Jenkins, Sangmin Seo, Shane Snyder, Robert B. Ross
Argonne National Laboratory, carns@mcs.anl.gov

Charles D. Cranor
Carnegie Mellon University

Scott Atchley
Oak Ridge National Laboratory

Torsten Hoefler
ETH Zürich

Abstract

Specialized, transient data services are playing an increasingly prominent role in data-intensive scientific computing. These services offer flexible, on-demand pairing of applications with storage hardware using semantics that are optimized for the problem domain. Concurrent with this trend, upcoming scientific computing and big data systems will be deployed with emerging non-volatile memory (NVM) technology to achieve the highest possible price/productivity ratio. Clearly, therefore, we must develop techniques to facilitate the confluence of specialized data services and NVM technology.

In this work we explore how to enable the composition of NVM resources within transient distributed services while still retaining their essential performance characteristics. Our approach involves eschewing the conventional shared file system model and instead projecting NVM devices as remote microservices that leverage user-level threads, remote procedure call (RPC) services, remote direct memory access (RDMA) enabled network transports, and persistent memory libraries in order to maximize performance. We describe a prototype system that incorporates these concepts, evaluate its performance for key workloads on an exemplar system, and discuss how the system can be leveraged as a component of future data-intensive architectures.

1 Introduction

Specialized, transient data services are playing an increasing role in data-intensive scientific computing. Examples include Dataspaces [4], DeltaFS [30], ADLB [26], and SDS [5]. These data services are optimized to facilitate analysis and computational applications that are not well served by a conventional global storage system. Specialized services can more easily match semantics, coherence, visibility scope, and hardware capacity with individual application requirements. Concurrent with this trend, future scientific computing [1] and big data systems are expected to be deployed with emerging NVM technology in order to achieve the highest possible price/productivity ratio [16]. In this work we use the term NVM to refer to any byte-addressible persistent memory technology.

Forward-looking services must be able to combine these two trends: specialized data services and emerging NVM technology. The most straightforward way to deploy NVM technology in this environment is by exposing private, node-local NVM devices for each task in an analysis or computation application. That deployment scenario is highly efficient for applications that can utilize it, but it does not address the full spectrum of multi-node parallel application use cases:

- sharing data across tasks or ensembles
- coherence for fine-grained parallel access
- support for imbalanced workloads
- access to NVM on dedicated storage nodes

We therefore focus our study on providing efficient *remote access to NVM devices distributed across nodes* for use in on-demand data services. The use of high-performance, byte-addressable storage devices exposes new bottlenecks in conventional shared storage systems, however. In this work we revisit the data models, concurrency mechanisms, network transports, and abstraction layers that are needed to make effective use of this technology. Section 2 describes the design of a prototype microservice for remote access to NVM devices, Section 3 evaluates the performance of the prototype for relevant workloads, Section 4 surveys related work, and Section 5 summarizes our conclusions.

2 Use case and design

Specialized data services are more likely to augment than supplant conventional file systems in large-scale big data or HPC systems. Figure 1 shows an example in which a resource manager has assigned compute nodes (with locally attached storage resources) to distinct, and in some cases overlapping, roles within the system. Each data

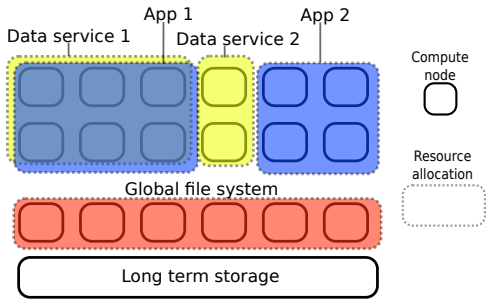

Figure 1: Example of resource allocations and deployment scenarios for specialized NVM services.

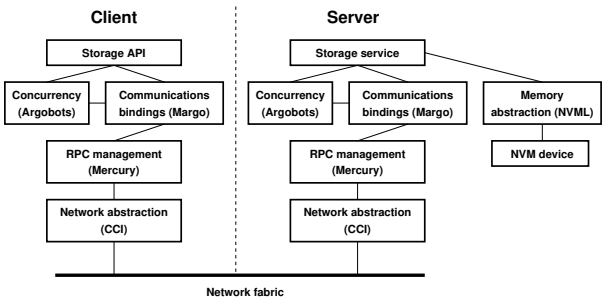

Figure 2: Prototype service software components.

service is scaled and scoped appropriately for the task at hand. Data services in this environment must be "good citizens" in terms of resource consumption, particularly when co-located with application tasks.

Because specialized data services do not replace conventional file systems or their administrative features, they need not conform to a conventional file system data model. Lower-level abstractions can be implemented by using a more direct mapping of data to hardware in order to avoid operating system block abstractions, global directory namespace updates, and other potential sources of overhead in conventional file system models. In this study we constructed a proof-of-concept object storage service to evaluate the effectiveness of deploying NVM technology for use in data-intensive scientific services. We envision object storage services as just one element of a suite of composable, on-demand microservices that include not only object storage but also functionality such as key/value storage, message buses, group membership, fault detection, and namespace management. Even conventional file system abstractions, if needed, can be implemented by composition of simple bulk storage and key/value services, as demonstrated by recent work in DeltaFS [30] and WarpFS [7].

2.1 Data transfer strategy

Our prototype explores a model in which clients interact with servers by exchanging RPC messages and data payloads are transferred using server-directed RDMA operations. By putting data transfer control in the hands of servers (and therefore co-locating transfer decision-making near critical shared resources and contention points), we greatly simplify the implementation of the following features for large-scale distributed services:

- flow control
- concurrency and coherency control
- access control (i.e., multiuser security)
- page-in or page-out from slower storage tiers

In contrast, a client-driven transfer model would use remote load/store primitives or client-initiated RDMA for remote access to byte-addressible NVM technology. The client-driven approach would grant application processes substantial control over server resources in order to minimize handshaking (i.e., control plane traffic) [11], but it provides no capability to coordinate concurrent access. Consider an example in a many-core system in which the ratio of application task to service daemons is 100 or more. It would be expensive for those tasks to make coherent distributed scheduling decisions to optimize (or avoid overwhelming) server-side resources. If transfers are server directed, on the other hand, then the server can provide flow control by limiting concurrent transfers, improve NUMA locality by re-ordering transfers, and constrain resource consumption by dividing large transfers into smaller RDMA operations.

2.2 Components

The low-latency properties of NVM storage are negated if the software stack introduces unecessary memory copies or context switches in the data path. Our prototype therefore combines user-space memory device abstractions, RDMA-capable network abstractions, and user-level threads to minimize overhead for remote NVM access while still maintaining portability. Figure 2 illustrates the major software components that make this possible. These components are described in greater detail in the following subsections, starting with the lowest-level server abstractions.

2.2.1 Local NVM access (NVM Library)

The NVM Library [13], or libpmem, is a collection of libraries that provide access to memory-mapped storage devices. These libraries translate persistent data structures into virtual memory mappings and facilitate durable and coherent updates to those data structures. The libpmemobj library and API, in particular, present a generalized transactional object store abstraction. The NVM Library bypasses the virtual file system and block device layers if it is used atop a memory-mapped device or a file system that supports DAX extensions.

2.2.2 Network transports (CCI)

The Common Communication Interface [2], or CCI, is a network abstraction library designed for use in large-scale high-performance systems. CCI presents a concise, RDMA-oriented API that focuses on lightweight expression of the most commonly used communication operations. The concise API allows CCI to be rapidly ported to a variety of network transports. In this study, we focus on CCI's InfiniBand Verbs transport implementation. Note that CCI supports both busy-polling and blocking modes of operations, but we use the latter exclusively in this study in order to minimize the impact of our service on co-located applications.

2.2.3 RPC services (Mercury)

Mercury is a user-space library for high-performance remote procedure calls [20]. It provides functionality for marshalling and unmarshalling RPC arguments, managing the state of concurrent operations, and transferring bulk data payloads using explicit RDMA operations. Mercury's modular architecture enables it to operate atop a variety of network transports (even those that do not natively support RDMA), but in this study we focus on the CCI module.

2.2.4 Concurrency (Argobots)

Our use case calls for a high degree of concurrency while coordinating access to multiple network and storage devices. However, conventional multithreaded concurrency can introduce significant context switching and contention overhead. We have therefore adopted user-level, cooperative threading as provided in Argobots [18] as our concurrency mechanism. Argobots threads are not preemptible. Instead, they yield control when blocked on I/O, when blocked on synchronization, or at thread completion time. User-level threads have been studied in previous work [24] as a means to retain the performance advantages of event-driven architectures while presenting simple sequential control paths to service developers.

2.2.5 Communication bindings (Margo)

The *Margo* component facilitates the mapping between Argobots threads and Mercury by way of Argobots-aware wrappers for the Mercury API and a transparent engine to drive communication progress. The goal of Margo is to retain the performance of the native event-driven Mercury API while presenting a conventional blocking interface and sequential control flow for library and service developers. Each blocking communication wrapper yields control to the Argobots scheduler when posting a Mercury operation and automatically resumes once that operation is complete. Margo also utilizes a custom thread scheduler for Argobots, called *abt-snoozer*, that gracefully idles processing cores if all user-level threads are waiting for I/O events.

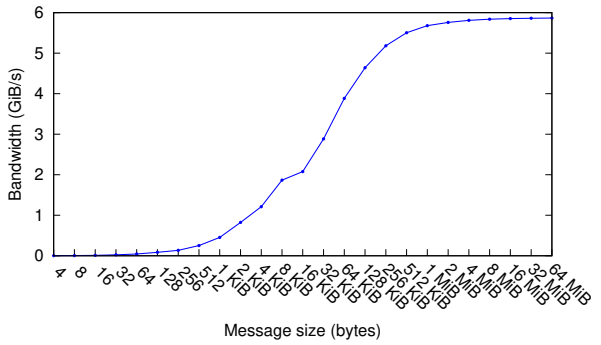

Figure 3: Baseline point-to-point network bandwidth.

The server daemon also uses Margo to spawn a new user-level thread for each incoming RPC request. User-level thread creation is exceptionally lightweight, and many threads may be instantiated simultaneously. The server daemon is a single-threaded application from the operating system's perspective, but Argobots could be trivially reconfigured to utilize additional OS threads if needed.

2.3 API semantics

The application-facing, client-side API presented by our service uses a data model similar to the libpmemobj [23] library, with one notable exception: it does not allow client-initiated load/store access, for the reasons discussed in Section 2.1. It uses explicit byte-granular read and write operations instead. Multiple clients may read and write the same object simultaneously, but the results of conflicting (i.e., overlapping at a byte level) write operations are undefined. Explicit persist operations are used to flush data and guarantee visibility to subsequent readers. Object creation and deletion are atomic operations.

The control flow during large I/O operations is as follows. The client library registers the application I/O buffer and sends an RPC request to the server. The server performs an RDMA operation to transfer data and then sends an RPC response. The client completes the I/O operation by deregistering the buffer and reporting a return code to the application. Server-side RDMA operations transfer data directly to and from memory-mapped storage devices when possible, though we will see in Section 3.2 that alternative protocol modes can be more effective in cases where RDMA registration accounts for a disproportionately large portion of the access latency.

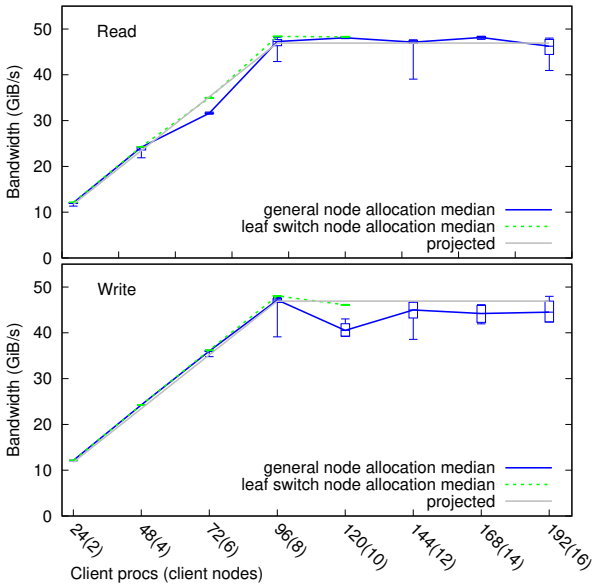

Figure 4: Median aggregate bandwidth with 8 servers.

3 Preliminary evaluation

All experiments presented in this paper were conducted on the Cooley Linux cluster operated by the Argonne Leadership Computing Facility. Each node contains two 2.4 GHz Intel Haswell E5-2620 v3 processors (12 cores total) and 384 GiB of RAM, and the nodes are connected via an FDR InfiniBand network fabric. All software was compiled with GCC 4.4.7 and O3 optimizations. The libpmem libraries were configured to use tmpfs volumes (i.e., conventional DRAM) as the backing store for experimental purposes in lieu of true NVM devices. Figure 3 shows the baseline asynchronous point-to-point network bandwidth for a logarithmic range of message sizes as measured using the mpptest benchmark [9] and the MVAPICH2 MPI implementation, version 2.1. This benchmark also exhibited a one-way latency of 1.3 microseconds for the smallest message sizes.

3.1 Aggregate concurrent bandwidth

We augmented the IOR benchmark [19] to use our prototype object storage API in order to evaluate aggregate I/O throughput. This action necessitated two key changes to IOR: adding an "aiori" module for our storage service and modifying the core benchmark to allow modules other than the POSIX module to issue fsync() operations.

Figure 4 shows the write and read bandwidth reported by IOR as we hold the number of server nodes (and thus the number of server daemons) fixed at 8 and vary the number of client nodes from 2 to 16. There are 12 processes per client node in all cases. Each experiment was repeated 30 times; box-and-whiskers plots show the minimum, maximum, median, first quartile, and third quartile for each set of measurements. IOR was configured with the following parameters: a block size (total data volume per process) of 6 GiB, a transfer size of 16 MiB, fsync enabled (to flush data at the conclusion of each write phase), data validation enabled, and file-per-process mode (which in our service equates to one object per process).

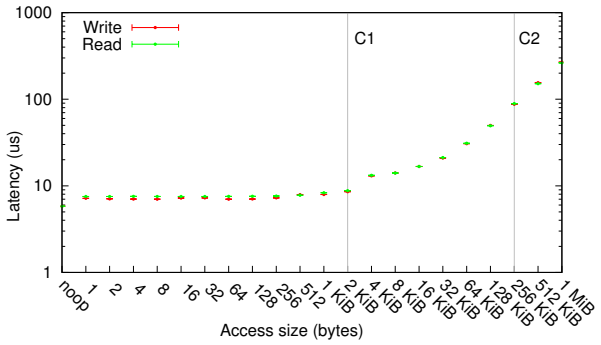

Figure 5: Median sequential access latency with one client and one server.

Our initial experiments, labeled "general node allocation," exhibited a high degree of variability. This phenomenon can be attributed to suboptimal routing within the Infiniband switch, which is a multistage switch rather than a true crossbar [12]. We repeated the experiments on a set of 18 nodes explicitly chosen to be co-located on a single leaf switch in order to confirm this behavior. These results, labeled "leaf switch node allocation," exhibit comparatively little variability, but the switch topology only allows us to scale up to 10 client nodes in this configuration. We also plot the projected aggregate bandwidth for comparison; this was calculated by multiplying the maximum baseline point-to-point bandwidth from Figure 3 by the minimum of the number of server or client nodes. Our prototype is capable of saturating the network bandwidth in each tested configuration.

3.2 Single-client latency

We constructed a microbenchmark that performs a series of sequential I/O operations from a single client to a single object to measure latency. It does not include data persistence or flush primitives, but each I/O access includes at least one round-trip network operation, at least one user-level thread creation and tear-down, and at least one libpmem memory access. The median access latency with a 95% confidence interval (calculated using the non-parametric method recommended in [10]) out of 10,000 samples for each access size is shown in Figure 5. We also plot the round-trip latency of a noop request on the left side of the x axis for comparison.

We also annotate two protocol crossover points in the

plot at 2 KiB (C1) and 256 KiB (C2). For message sizes less than C1, the data payload is eagerly copied into the RPC request or RPC response message with no explicit RDMA transfer. The eager mode size limit is determined by the maximum message size of the Mercury transport plugin. For message sizes between C1 and C2, the prototype uses RDMA to transfer data, but the data is copied to and from a pre-registered pool of memory on both the client and the server. For message sizes greater than C2, RDMA memory is registered on demand for each operation with no intermediate copies. The C2 crossover point was chosen empirically as the point at which memory registrations become more efficient than memory copies on our test platform.

The median noop latency is 5.8 microseconds, though the baseline measurements in Figure 3 indicate that 2.6 microseconds should be possible by projecting the 1.3 microsecond one-way latency to a round-trip cost. This performance gap can be attributed to a combination of signaling cost (recall from Section 2.2.2 that our prototype idles gracefully rather than busy-polling on the network) and software protocol overhead. We also observed significant tail latencies in some cases even though the median confidence intervals are narrow. Out of 10,000 noop measurements, for example, we observed 17 samples that exceeded 10 microseconds, including one that exceeded 70 microseconds. We will investigate the cause of these outliers in future work.

4 Related work

Adapting software and data abstractions to high-performance network and NVM architectures has been an active research topic in recent years. We aim to incorporate emerging best practices and design decisions presented in these works.

NVM-focused storage systems, such as Aerie [21], NOVA [28], SCMFS [27], BPFS [3], Mnemosyne [22], NV-Heaps [3], and NV-Tree [29], while differing significantly in design, nonetheless exhibit a number of key themes. NVM devices are amenable to log-structuring data and metadata because of their exceptional random-read performance. The comparatively high level of software overhead in traditional storage layers has led to explorations of low-level abstractions on which filesystems and other types of stores are built. These new abstractions have typically focused on more direct address space management (e.g., explicit allocation) and lightweight namespacing capabilities. Additionally, techniques such as copy-on-write and shadow paging are used for crash resiliency and as a wear-leveling mechanism, although these are by no means unique to NVM approaches.

Recent works in projecting data models across high-performance networks have approached maximization of current-gen hardware potential through careful mapping of service semantics to architectural features. MICA [17] focuses on concurrent tag matching, affinity/NUMA management, and kernel bypass for high-throughput UDP traffic serving lossy hash indices; and HERD [14] additionally performs a careful construction of InfiniBand verbs primitives to maximize concurrent throughput while minimizing trip counts. Follow-on work from the same group further investigates low-level architecture behaviors [15]. FaRM [6] applies similar design considerations to develop key-value and graph data models atop an RDMA-enabled shared memory address space. Additional recent works have explored different areas of the design space with similar themes [25].

Other related works have incorporated these lessons into the HPC domain. MDHIM projects a collection of LevelDB key-value store instances across nodes through MPI with user-defined deterministic partitioning functions [8]. DiDAFS [11] proposes extensions to the memory-management units of nodes to enable direct remote storage access as well as fine-grained, epoch-based caching and consistency semantics to manage distributed storage resources. DataSpaces [4] projects a key-value-like multidimensional address space with RDMA-capable access for code-coupling scientific workflows.

5 Conclusions and future work

Our proposed strategy to enable the use of NVM in data-intensive scientific services shows promise in terms of both software practice and empirical performance. The prototype makes use of existing runtime abstractions for NVM, network, RPC, and CPU resources to enable cross-platform portability with minimal effort. We have also demonstrated that an RPC-based service with server-directed RDMA transfers can achieve single-digit non-persistent microsecond remote access latencies and saturate the aggregate bandwidth of high-performance networks.

Our study also highlights a variety of opportunities for future work. We will continue to adopt best-in-class optimizations from previous studies and integrate performance metrics that facilitate root cause analysis of performance problems. We will also investigate methods to present the same API for both remote NVM access and local NVM access so that they can be used interchangeably depending on deployment scenarios and application requirements.

We will also extend our work by exploring more application-oriented use cases and contrasting our service implementation with other products. Moreover, we will investigate the behavior of our service in conjunction with true NVM devices in place of DRAM.

Acknowledgments

This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. This research also used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

[1] ARGONNE LEADERSHIP COMPUTING FACILITY. Aurora. `http://aurora.alcf.anl.gov/`.

[2] ATCHLEY, S., DILLOW, D., SHIPMAN, G., GEOFFRAY, P., SQUYRES, J. M., BOSILCA, G., AND MINNICH, R. The Common Communication Interface (CCI). In *19th Annual IEEE Symposium on High-Performance Interconnects* (August 2011).

[3] 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 *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles* (New York, NY, USA, 2009), SOSP '09, ACM, pp. 133–146.

[4] DOCAN, C., PARASHAR, M., AND KLASKY, S. DataSpaces: an interaction and coordination framework for coupled simulation workflows. *Cluster Computing 15*, 2 (June 2012), 163–181.

[5] DONG, B., BYNA, S., AND WU, K. SDS: A framework for scientific data services. In *Proceedings of the 8th Parallel Data Storage Workshop* (New York, NY, USA, 2013), PDSW '13, ACM, pp. 27–32.

[6] DRAGOJEVI, A., NARAYANAN, D., HODSON, O., AND CASTRO, M. FaRM: Fast remote memory. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2014), NSDI'14, USENIX Association, pp. 401–414.

[7] ESCRIVA, R., AND SIRER, E. G. The design and implementation of the Warp transactional filesystem. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)* (Santa Clara, CA, 2016), USENIX Association, pp. 469–483.

[8] GREENBERG, H., BENT, J., AND GRIDER, G. MDHIM: A Parallel Key/Value Framework for HPC. In *Proceedings of the 7th USENIX Workshop on Hot Topics in Storage and File Systems* (Santa Clara, CA, 2015), USENIX Association.

[9] GROPP, W., AND LUSK, E. Reproducible measurements of MPI performance characteristics. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer, 1999, pp. 11–18.

[10] HOEFLER, T., AND BELLI, R. Scientific Benchmarking of Parallel Computing Systems. ACM, pp. 73:1–73:12. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC15).

[11] HOEFLER, T., ROSS, R. B., AND ROSCOE, T. Distributing the data plane for remote storage access. In *15th Workshop on Hot Topics in Operating Systems (HotOS XV)* (Kartause Ittingen, Switzerland, 2015), USENIX Association.

[12] HOEFLER, T., SCHNEIDER, T., AND LUMSDAINE, A. Multistage Switches are not Crossbars: Effects of Static Routing in High-Performance Networks. In *Proceedings of the 2008 IEEE International Conference on Cluster Computing* (Oct. 2008), IEEE Computer Society.

[13] INTEL CORPORATION. NVM library. `http://pmem.io/nvml/`. Retrieved July 2016.

[14] KALIA, A., KAMINSKY, M., AND ANDERSEN, D. G. Using RDMA efficiently for key-value services. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (New York, NY, USA, 2014), SIGCOMM '14, ACM, pp. 295–306.

[15] KALIA, A., KAMINSKY, M., AND ANDERSEN, D. G. Design guidelines for high performance RDMA systems. In *Proceedings of the 2016 USENIX Annual Technical Conference (USENIX ATC 2016)* (Denver, CO, 2016), USENIX Association, pp. 437–450.

[16] KAMBATLA, K., KOLLIAS, G., KUMAR, V., AND GRAMA, A. Trends in big data analytics. *Journal of Parallel and Distributed Computing 74*, 7 (2014), 2561–2573. Special Issue on Perspectives on Parallel and Distributed Processing.

[17] LIM, H., HAN, D., ANDERSEN, D. G., AND KAMINSKY, M. MICA: A holistic approach to fast in-memory key-value storage. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2014), NSDI'14, USENIX Association, pp. 429–444.

[18] SEO, S., AMER, A., BALAJI, P., BORDAGE, C., BOSILCA, G., BROOKS, A., CASTELLO, A., GENET, D., HERAULT, T., JINDAL, P., KALE, L., KRISHNAMOORTHY, S., LIFFLANDER, J., LU, H., MENESES, E., SNIR, M., SUN, Y., AND BECKMAN, P. H. Argobots: A lightweight threading/tasking framework. Tech. Rep. ANL/MCS-P5515-0116, Argonne National Laboratory, 2016.

[19] SHAN, H., ANTYPAS, K., AND SHALF, J. Characterizing and predicting the I/O performance of HPC applications using a parameterized synthetic benchmark. In *Proceedings of Supercomputing* (November 2008).

[20] SOUMAGNE, J., KIMPE, D., ZOUNMEVO, J., CHAARAWI, M., KOZIOL, Q., AFSAHI, A., AND ROSS, R. Mercury: Enabling remote procedure call for high-performance computing. In *2013 IEEE International Conference on Cluster Computing (CLUSTER)* (Sept 2013), pp. 1–8.

[21] VOLOS, H., NALLI, S., PANNEERSELVAM, S., VARADARAJAN, V., SAXENA, P., AND SWIFT, M. M. Aerie: Flexible file-system interfaces to storage-class memory. In *Proceedings of the Ninth European Conference on Computer Systems* (New York, NY, USA, 2014), EuroSys '14, ACM, pp. 14:1–14:14.

[22] VOLOS, H., TACK, A. J., AND SWIFT, M. M. Mnemosyne: Lightweight persistent memory. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2011), ASPLOS XVI, ACM, pp. 91–104.

[23] VON BEHREN, P. NVML: implementing persistent memory applications. `http://www.snia.org/sites/default/files/Paul_von_behren_NVML_Implementing_Persistent_Memory.pdf`, 2015. Storage Networking Industry Association Tutorial.

[24] VON BEHREN, R., CONDIT, J., ZHOU, F., NECULA, G. C., AND BREWER, E. Capriccio: Scalable threads for internet services. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (October 2003), SOSP '03, pp. 268–281.

[25] WANG, Y., ZHANG, L., TAN, J., LI, M., GAO, Y., GUERIN, X., MENG, X., AND MENG, S. HydraDB: A resilient RDMA-driven

key-value middleware for in-memory cluster computing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (New York, NY, USA, 2015), SC '15, ACM, pp. 22:1–22:11.

[26] WOZNIAK, J. M., PETERKA, T., ARMSTRONG, T. G., DINAN, J., LUSK, E., WILDE, M., AND FOSTER, I. Dataflow coordination of data-parallel tasks via MPI 3.0. In *Proceedings of the 20th European MPI Users' Group Meeting* (New York, NY, USA, 2013), EuroMPI '13, ACM, pp. 1–6.

[27] WU, X., AND REDDY, A. L. N. SCMFS: A file system for storage class memory. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* (New York, NY, USA, 2011), SC '11, ACM, pp. 39:1–39:11.

[28] XU, J., AND SWANSON, S. NOVA: A log-structured file system for hybrid volatile/non-volatile main memories. In *Proceedings of the 14th Usenix Conference on File and Storage Technologies* (Berkeley, CA, USA, 2016), FAST'16, USENIX Association, pp. 323–338.

[29] YANG, J., WEI, Q., CHEN, C., WANG, C., YONG, K. L., AND HE, B. NV-Tree: Reducing consistency cost for NVM-based single level systems. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2015), FAST'15, USENIX Association, pp. 167–181.

[30] ZHENG, Q., REN, K., GIBSON, G., SETTLEMYER, B. W., AND GRIDER, G. DeltaFS: Exascale file systems scale better without dedicated servers. In *Proceedings of the 10th Parallel Data Storage Workshop* (New York, NY, USA, 2015), PDSW '15, ACM, pp. 1–6.