Distributing the Data Plane for Remote Storage Access

Torsten Hoefler
ETH Zurich

Robert B. Ross
Argonne National Laboratory

Timothy Roscoe
ETH Zurich

Abstract

Sub-microsecond network and memory latencies require fast user-level access to local and remote storage. While user-level access to local storage has been demonstrated recently, it does currently not extend to serverless parallel systems in datacenter environments. We propose direct user-level access to remote storage in a distributed setting, unifying fast data access and high-performance remote memory access programming. We discuss a minimal hardware extension of the IOMMU to enable direct remote storage access. In order to maintain optimal performance in the system, we use epoch-based accesses to allow fine-tuning of atomicity, consistency, isolation, and durability semantics. We also address the problem of user-managed coherent caching. Finally, we briefly discuss the design of DiDAFS, a Distributed Direct Access File System that enables efficient data analytics use-cases such as buffered producer-consumer synchronization and key-value stores as well as deeper integration of storage into high performance computing applications.

1 Introduction

While both datacenter networks and storage devices in the form of both solid-state disks and persistent main memory are increasing in performance, single-core computing speed has essentially stalled. This means that software is increasingly the bottleneck for many I/O-bound tasks. In response to these hardware trends, recent proposals have revived the idea of separating the data and control planes and allowing applications to directly access hardware for network and storage I/O [3, 12, 38]. Such *software-defined I/O* (SDIO) designs promise superior local I/O performance, scalability, and consistency management by removing OS mediation from the data path, which can also benefit storage devices such as solid state drives (SSD) [1] or high-capacity shingled write disks [4] by reducing CPU overheads.

In this paper, we propose extending SDIO to distributed memory systems from rack-scale up to complete datacenters. The key challenge here is distributed access to storage: while Arrakis [38], for example, provides safe user-space access to local storage devices, it is less clear how to build the kind of distributed storage system common in datacenters without compromising this principle.

Our high-level goal is to build a high-performance scalable distributed file system that can integrate multiple storage technologies under one umbrella and derive the full benefits of control/dataplane separation.

We make two contributions in this paper. At the data access level, we envision extending the facilities of RDMA to block devices (e.g., SSDs). This in practice requires changes to disk controllers so they can respond directly to network requests (and verify appropriate authorization credentials in doing so), and we outline a potential minimal hardware facility to do this. A more traditional, if slower, data-path can be implemented using main memory as a cache and invoking the local operating system for misses. We will discuss both options.

Above this, a caching layer is required that can maintain consistency without incurring undue software overheads when data chunks are read and written, and we present a preliminary design inspired by processor cache coherence and classical distributed system protocols. A related challenge is to manage appropriate metadata for a distributed parallel storage system without this aspect of the system becoming the bottleneck. An open question is whether the solutions adopted in current distributed file systems suffice for a fully dataplane-based approach, or whether a different scheme is required.

2 Challenges and Opportunities

In the rest of this paper, we discuss the challenges of exploiting direct hardware access and SDIO in a distributed file system, and our early work designing such a system.

Our focus is on parallel or distributed jobs that consist of multiple processes cooperating to solve a task; we

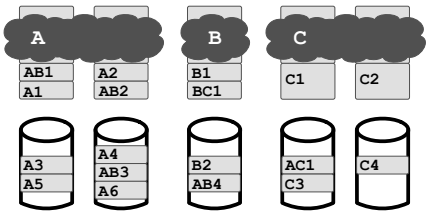

Figure 1: System overview with three jobs (A-C) on five machines and various shared (e.g., AB3) and private (e.g., A1) allocations in volatile and persistent storage.

show a simplified usage scenario in Figure 1. Here, three jobs ($A-C$) execute, with A and C using two nodes in the compute system and job B just one. Jobs consist of one or more processes, each with a local virtual address space containing a set of threads.

These types of jobs exhibit complex data sharing patterns that create a challenging environment for distributed storage systems generally, but their properties also create opportunities that are not well exploited by existing distributed storage solutions. We target a number of storage models and applications; we give two examples here.

Firstly, in advanced parallel programming schemes, remote memory is often exposed for direct access [18,26] by creating a *partitioned global address space* across multiple processes. New high-performance languages [8, 20] allow programmers to develop application-specific memory management schemes to allow algorithms to exploit locality in these cases. These advanced parallel programming models already explicitly manage data regions and exploit remote memory access mechanisms, creating an opportunity to extend these models to further exploit SDIO-based distributed storage.

Secondly, distributed data-analytics processing pipelines need to *share data efficiently* between programs, regardless of the technology on which the data is stored. For example DataPath [6], Naiad [35], Spark [51], and MillWheel [2] all establish fast data pipelines with buffered producer/consumer access. An SDIO-based distributed storage system has the potential to bring the performance of RDMA to these applications.

In both these use-cases, explicit data placement and cache management provides the control [20] needed to co-locate computation and data where possible.

Accordingly, inspired by Exokernels [22], we argue that using a client-side library to manage metadata operations and distributed coordination without kernel mediation has the additional advantage of allowing such policies to be implemented in the application. However, managing caching and consistency in an SDIO-based distributed system, at large scale, is a complex problem which, to date, is not well-understood.

For that reason, we must look for opportunities to simplify the problem. One opportunity lies in recognizing that not all processes need to coordinate.

We note that many independent, distributed applications will run concurrently in the datacenter as a whole. These applications have no requirement to share data among them, and it is therefore desirable to scope mechanisms for cache coherence or other synchronization to a single application rather than system-wide. At the same time, there will be cases where data may also need to be shared across jobs, serially (only one job at a time accesses a file) or concurrently (multiple jobs access a file at the same time). Any scoping mechanism, as well as access control system, must thus support its consistency model across jobs.

We also observe that our distributed applications are coordinating already, either via their programming model (in the case of parallel programs) or via coordination between tasks in workflows. This coordination provides natural points for managing consistency, and at a low level in the storage stack, this motivates a choice of explicit consistency operations and consistency scoping.

Many of the challenges we face are not radically new, but take on new characteristics when coupled with SDIO and RDMA. User-level file systems have existed since the Network-Attached Secure Disks (NASD) [25] and SHRIMP [9] projects, and have recently been revived in Aerie [44]. In this work we build on these to design a user-level distributed file system. In addition to providing low-level remote SDIO, a challenge we address in the next section, the use of such direct hardware access has design implications for other essential components of a distributed file system and has caused us to rethink several traditional functions. For example, we need to be able to *synchronize data and metadata operations* in an environment where locks might be prohibitively expensive. This is closely related to the problem of *caching*, inherent in any distributed storage system, and we address these issues in Section 4.

We note also that individual processes, or entire jobs, may fail at any time. A storage system should provide durability of data (under the control of the user), and therefore be *recoverable* to a consistent state.

3 Remote direct storage access

We now discuss the basic low-level facilities needed to extend user storage device access to the distributed case.

While direct *network* access has a nearly 30-year history [5, 10, 41, 45], direct storage access has received less attention, in part due to the storage itself being slow. This has all changed with the advent of SSDs, and direct access will become a necessity for fast byte-addressable persistent storage class memories

(SCM) [21] such as phase-change memories [49], spin-transfer torque RAM [13], or flash-backed DRAM.

Bailey et al. [7] discuss the general influence of these storage technologies on operating systems. In particular, a traditional OS-based file system quickly becomes a bottleneck [11, 12], and direct access to both the data as well as the metadata is required. Our aim is to provide a high-performance object store [36,48], integrating storage in memory and on persistent devices across a fast network, over which a variety of storage applications can be efficiently implemented.

Allocations: When discussing our design, we use the term *allocation* to refer to an area in main memory *or* on a storage device which can be used to store data, and which is accessed by jobs as an object with a contiguous address space. Block translations and free-space management on each device are either implemented by the operating system using standard techniques such as buddy allocation, or by the hardware offering a virtualized interface [15, 29, 30].

Allocation operations such as `open` or `create` can be either performed by a centralized control plane or in a distributed manner using capabilities and delegation. Once an allocation is created, it can be linked to an object in a global name space (e.g., a kernel-level parallel file system [31, 39, 40]) for sharing and access control.

Basic read and write access: Allocations in node-local main memory can be accessed using MMU mappings, while allocations on local storage devices can use existing SDIO features through the MMU as in Aerie [44].

Remote access to allocations poses more of a challenge, and distinguishes our work from prior systems. Allocations in remote SCM or main memory can use existing RDMA features. Since RDMA allows access to remote user virtual memory, and the RDMA NIC's IOMMU operates on physical memory, we require the IOMMU and destination address space to synchronize mappings for each allocation. For example, the OS can manage an RDMA communication endpoint for each allocation.

Today, RDMA NICs such as InfiniBand use static translation tables (STL) that are initialized by the operating system, and accesses causing translation misses are discarded. Both STLs and IOMMU page tables are set up by the OS (acting as a control plane). At the node hosting the in-memory allocation, this scheme can also support lazy setup of the tables if misses can be handled dynamically by the OS [47].

Remote allocations on disks or SSDs cannot be directly accessed through RDMA to main memory, and so a clean dataplane requires either an extension to current hardware or a software solution.

One option is to extend the IOMMU to log accesses, perform direct device I/O using DMA, and cache blocks in main memory for remote access. IOMMUs today already maintain an event log of failed accesses in main memory [28, §7.3] so such an extension seems plausible. If the log overflows, the IOMMU could drop the request and propagate an exception to the sender via the NIC's RDMA protocol. It could also exert back-pressure through the network (with associated head-of-line blocking) for reliable transports.

In a software solution, the OS would monitor the failed access log of the IOMMU and fetch and map the pages that were not resident when accessed remotely. The source of the access could either retry the operation upon failure or use a "ready-for-access" notification protocol. We discuss in Section 4 weakly consistent access semantics that facilitate such implementations.

Both these mechanisms imply using main memory on the storage device's node as a cache, but still remove any software involvement on the fast path when accessing remote storage. As it happens, this integrates well with a more general caching model we discuss in Section 4. However, a significant drawback of the software approach is that workloads with no locality will still involve the OS on every data access, whereas the hardware-based proposal does not.

This approach to direct remote access contrasts with the Direct Access File System (DAFS [16]), which extends the Network File System Version 4 protocol to allow direct RDMA access. DAFS enables a user-level file system client typically running as a library above the operating system. However, RDMA operations are generally initiated at the server-side and metadata operations are handled through standard RPC mechanisms. Thus, the expected speedup for frequent small-file accesses is limited [34]. The optimistic direct access file system proposes direct user-level access to a DAFS server [33]. We want to support high-speed user-level metadata access as well as scalable concurrent access. Trivedi et al. [43] propose using high-performance networking abstractions for storage access but do not consider caching, persistence, file systems, or shared access consistency.

Since remote storage transfers go through main memory on the server, the MMU and IOMMU on this node can provide a basic protection mechanism. The control plane (OS) at the device-owning node configures the mappings and protection bits to expose the allocation to both the NIC and local applications, and can implement inside it an Aerie-style user-level file system.

Usage: By way of summary, we will now briefly explain how a process could create an allocation, a file in it, write data into the file, and share it with another process.

Assume a process in job A (Figure 1) wants to create allocation AB1 (1 MiB) in its local DRAM. For this, it could request an allocation of size 1 MiB from its local

control plane (OS). The OS would identify free pages and install a mapping into the address space of all processes local to the allocating process. It would expose this allocation to processes of job A (and other jobs) on remote nodes by opening an RDMA communication endpoint (e.g., a queue pair) for direct access and installing translation tables on the NIC and the IOMMU. The RDMA connection information could be communicated through capabilities, or it can be stored in the global namespace. Allocations on remote devices can be performed similarly through RPCs to the remote OS.

Remote nodes connect and access the allocation directly via RDMA, for example, through a file system that manages file metadata in the allocation. File system operations are now performed through user-level file system calls operating directly on the remote memory.

Overall, the scheme we have outlined provides a foundation of read/write functionality on which to build a complete distributed storage system. We now go on to describe how allocations can be chained to provide the important function of caching.

4 Caching and Consistency

Having established the basic mechanisms for distributing dataplane storage, we now turn our attention to how flexible caching, consistency, and metadata management can be efficiently performed above this layer.

Some form of caching is intrinsic in any distributed storage system. Retrieving data from a remote node inherently caches it locally, and in the previous section we combine caching with RDMA to enable direct remote access to disks and SSDs.

However, given the importance of performance under a variety of different workloads, and the use of library implementations of storage functionality implied by the dataplane approach rather than OS services, we argue for maximal policy freedom in caching decisions.

We are exploring the idea of closely coupling caching with the concept of allocations: a user can allocate space on a device and link the resulting allocation as a (user-managed) cache for another allocation. A caching allocation typically resides in a faster (e.g., local) device than the origin and may be smaller.

Software caching can be offered by the user file system library, similar to local caching in RMA programming [52], and prefetching and replacement policies can be specialized for a specific application while avoiding the problems of double caching [42].

Hardware caching could be performed by an extended IOMMU using a user-specified cache allocations and generic replacement strategies.

Consistency and coherence: Any distributed caching strategy raises the question of maintaining some form of consistency and coherence. Parallel caching strategies are discussed in the context of Panache, a clustered file system cache [19]. In our case, the design space of consistency mechanisms is constrained by the need to support direct access to remote storage, bypassing system (and user) software on the storage node itself: RDMA makes it difficult to implement strongly consistent atomic accesses and still achieve high performance.

Instead, we propose a weak consistency model known from multiprocessor systems [17, 24] and remote memory access programming languages [18, 26], extended with light-weight transactions implemented in user caches, similar to transactional memory [27]. The ideal memory model for RMA caching is not yet clear, but we hope to gain insight into this question in the course of our implementation.

We arrange all accesses into epochs that are separated by explicit synchronization fence operations. We distinguish different epoch types: *shared*, *exclusive*, *persistent*, and *optimistic*. The different types provide different guarantees of the state of the storage after the epoch is ended, and correspond with the concept of isolation levels in transaction processing systems. Modified data is generally not valid during an epoch.

Shared epochs provide only consistency across epoch boundaries. Exclusive epochs guarantee consistency as well as atomicity. The atomicity is guaranteed by the system, which can use either buffering or locking to provide this guarantee. Persistent epochs provide durability, i.e. data is both consistent for all potential readers and also committed to persistent storage (in case the accessed allocation is cached in non-persistent storage). Optimistic epochs provide isolation, similar to transactional memory, and can thus fail to commit if conflicts occurred.

Epoch types can be freely combined, for example, exclusive persistent epochs. These combinations cover the whole spectrum from unprotected Rio Vista-style transactions [32] to fully isolated ACID (Atomicity, Consistency, Isolation, Durability) transactions, in a similar way to the Salt system for databases [50].

We expect an allocation to be opened multiple times with different epoch styles. Metadata can also stored in the allocation, and its consistency managed similarly – indeed, with different epoch modes if the consistency requirements between data and metadata differ.

Implementing epochs: We propose implementing access management in the user file system library, using RDMA-based distributed algorithms [23] to coordinate all processes that have an allocation open. We use existing work on remote memory access memory models to ensure consistency semantics [26]. However, this still presents interesting challenges.

Ensuring consistency, isolation, and durability is ideally a data plane operation and directly supported by the

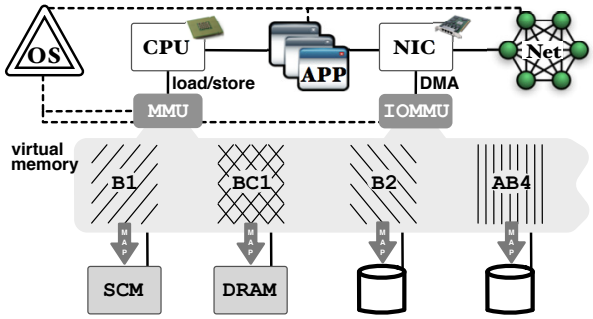

Figure 2: DiDAFS overview of a process in job B with allocations B1, BC1, B2, AB4.

storage device (e.g., RDMA remote completion operations or x86 `mfence`). However, not all devices support all synchronizations in hardware. For example, ending a persistent epoch with an allocation on an SSD or hard drive requires OS involvement to access the block device. If OS assistance is necessary, the file system library can contact the target OS using RPC after all accesses are committed to the volatile RDMA-accessible cache.

Crash recovery and integrity: As applications can exit in any state, metadata operations need to be managed carefully. One option is to only use transactional (optimistic, exclusive, and persistent) epochs for metadata updates. Another is to use journaling in user-space, but overheads must be kept in check at large scale [37].

A user-level file system must ensure that exclusive epochs that use locking cannot cause deadlocks if processes disappear. This can be done using generational locks (requiring additional coordination by the OS during allocation open) or lock timeouts. The impact of misbehaving clients (e.g., memory corruptions due to a bug) can be limited using memory protection mechanisms but cannot be avoided in general.

5 Current work

We are using ideas in the previous sections to design and build DiDAFS, a Distributed Direct Access File System. DiDAFS performs remote control plane operations such as `open` or `close` of allocations using light-weight RPC mechanisms, but also offers *collective* interfaces for allocation operations to scale jobs to thousands of processes by reducing offset translation storage [23].

Figure 2 shows an overview of a process in job B in DiDAFS. Solid connectors illustrate data plane accesses, and dashed connectors illustrate the control plane. The shaded area visualizes the process' address space that can be accessed by the local or remote processes through the NIC/IOMMU.

Allocations in DiDAFS can contain user-level file systems that are accessed through a library offering a file system interface. Once an allocation is opened, a user-level file system implemented purely through data-plane calls (RDMA) can be used.

For example, an application can use a set of POSIX-compatible access and metadata functions (e.g., `read` or `stat`). A POSIX-like file system would use exclusive persistent epochs for each function.

However, the real power of the approach is realized with application-specific naming schemes. For example, an eventually consistent key/value store can be implemented on top of DiDAFS without additional overheads and further optimized by using shared epochs for read accesses and buffering writes for separate exclusive or optimistic epochs (depending on the expected conflicts).

Sharing can efficiently be implemented at the allocation level. For example, shared buffers can be implemented in fast memory and exposed for direct access. Epoch semantics can then be tuned to the application (for example, persistent epochs can be used to aid recovery from crash faults).

Since naming and placement is determined by the application, a user library can choose the best location for each data item. This allows full freedom for local, remote, and job-collective data management. It is thus possible to implement advanced high-performance parallel data access functions such as MPI-IO [26] with DiDAFS.

6 Conclusion

DiDAFS' direct local and remote storage access allows a unified view of high-performance remote memory and high-performance parallel storage, simplifying and accelerating many data-analytics and high performance computing applications. Implementing DiDAFS poses a series of interesting challenges and opportunities for further research, which we can only touch on here:

1. Application-specific cache size, replacement, and prefetch policies. How can the application provide hints to the file system library? How efficient would caching of metadata (not in the block cache) be?
2. Could static analysis be used to derive good prefetch or replacement policies from source codes? Could the file system library be inlined by the compiler and optimized for direct accesses?
3. How can deduplication and copy on write semantics as implemented in Parallax [46] be added in user-space in distributed settings? How would one provide distributed file version histories in DiDAFS?
4. Can programming safety be increased, e.g., avoiding pointers from persistent to volatile storage [14]?

Nevertheless, we feel that DiDAFS is a useful step in extending the separation of control and data planes to a high-performance distributed system.

We thank Marc Snir and Pete Beckman from ANL for inspiring discussions about the problem definition related to the Argo OS.

References

[1] Agrawal, N., Prabhakaran, V., Wobber, T., Davis, J. D., Manasse, M., and Panigrahy, R. Design Tradeoffs for SSD Performance. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference* (Berkeley, CA, USA, 2008), ATC'08, USENIX Association, pp. 57–70.

[2] Akidau, T., Balikov, A., Bekiroğlu, K., Chernyak, S., Haberman, J., Lax, R., McVeety, S., Mills, D., Nordstrom, P., and Whittle, S. MillWheel: Fault-tolerant Stream Processing at Internet Scale. *Proc. VLDB Endow. 6*, 11 (Aug. 2013), 1033–1044.

[3] Alex, L., Todor, E., and Swanson, M. S. Quill: Exploiting Fast Non-Volatile Memory by Transparently Bypassing the File System. Technical report, University of California San Diego (UCDS), 2013.

[4] Amer, A., Long, D., Miller, E., Paris, J.-F., and Schwarz, S. Design issues for a shingled write disk system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on* (May 2010), pp. 1–12.

[5] Anderson, D. C., Chase, J. S., Gadde, S., Gallatin, A. J., Yocum, K. G., and Feeley, M. J. Cheating the I/O Bottleneck: Network Storage with Trapeze/Myrinet. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 1998), ATEC '98, USENIX Association, pp. 12–12.

[6] Arumugam, S., Dobra, A., Jermaine, C. M., Pansare, N., and Perez, L. The DataPath System: A Data-centric Analytic Processing Engine for Large Data Warehouses. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2010), SIGMOD '10, ACM, pp. 519–530.

[7] Bailey, K., Ceze, L., Gribble, S. D., and Levy, H. M. Operating System Implications of Fast, Cheap, Non-volatile Memory. In *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems* (Berkeley, CA, USA, 2011), HotOS'13, USENIX Association, pp. 2–2.

[8] Bauer, M., Treichler, S., Slaughter, E., and Aiken, A. Legion: Expressing Locality and Independence with Logical Regions. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (Los Alamitos, CA, USA, 2012), SC '12, IEEE Computer Society Press, pp. 66:1–66:11.

[9] Blumrich, M. A., Li, K., Alpert, R., Dubnicki, C., Felten, E. W., and Sandberg, J. Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer. In *Proceedings of the 21st Annual International Symposium on Computer Architecture* (Los Alamitos, CA, USA, 1994), ISCA '94, IEEE Computer Society Press, pp. 142–153.

[10] Buzzard, G., Jacobson, D., Mackey, M., Marovich, S., and Wilkes, J. An Implementation of the Hamlyn Sender-managed Interface Architecture. *SIGOPS Oper. Syst. Rev. 30*, SI (Oct. 1996), 245–259.

[11] Caulfield, A. M., Coburn, J., Mollov, T., De, A., Akel, A., He, J., Jagatheesan, A., Gupta, R. K., Snavely, A., and Swanson, S. Understanding the Impact of Emerging Non-Volatile Memories on High-Performance, IO-Intensive Computing. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis* (Washington, DC, USA, 2010), SC '10, IEEE Computer Society, pp. 1–11.

[12] Caulfield, A. M., Mollov, T. I., Eisner, L. A., De, A., Coburn, J., and Swanson, S. Providing Safe, User Space Access to Fast, Solid State Disks. *SIGARCH Comput. Archit. News 40*, 1 (Mar. 2012), 387–400.

[13] Chen, E., Apalkov, D., Diao, Z., Driskill-Smith, A., Druist, D., Lottis, D., Nikitin, V., Tang, X., Watts, S., Wang, S., Wolf, S., Ghosh, A., Lu, J., Poon, S., Stan, M., Butler, W., Gupta, S., Mewes, C., Mewes, T., and Visscher, P. Advances and Future Prospects of Spin-Transfer Torque Random Access Memory. *IEEE Transactions on Magnetics 46*, 6 (June 2010), 1873–1878.

[14] Coburn, J., Caulfield, A. M., Akel, A., Grupp, L. M., Gupta, R. K., Jhala, R., and Swanson, S. NV-Heaps: Making Persistent Objects Fast and Safe with Next-generation, Non-volatile Memories. *SIGPLAN Not. 46*, 3 (Mar. 2011), 105–118.

[15] Cully, B., Wires, J., Meyer, D., Jamieson, K., Fraser, K., Deegan, T., Stodden, D., Lefebvre, G., Ferstay, D., and Warfield, A. Strata: Scalable High-performance Storage on Virtualized Non-volatile Memory. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2014), FAST'14, USENIX Association, pp. 17–31.

[16] DeBergalis, M., Corbett, P., Kleiman, S., Lent, A., Noveck, D., Talpey, T., and Wittle, M. The

Direct Access File System. In *Proceedings of the 2Nd USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2003), FAST'03, USENIX Association, pp. 13–13.

[17] Dubois, M., Scheurich, C., and Briggs, F. Memory Access Buffering in Multiprocessors. In *25 Years of the International Symposia on Computer Architecture (Selected Papers)* (New York, NY, USA, 1998), ISCA '98, ACM, pp. 320–328.

[18] El-Ghazawi, T., Carlson, W., Sterling, T., and Yelick, K. *UPC: Distributed Shared-Memory Programming*. Wiley-Interscience, 2003.

[19] Eshel, M., Haskin, R., Hildebrand, D., Naik, M., Schmuck, F., and Tewari, R. Panache: A Parallel File System Cache for Global File Access. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2010), FAST'10, USENIX Association, pp. 12–12.

[20] Fatahalian, K., Horn, D. R., Knight, T. J., Leem, L., Houston, M., Park, J. Y., Erez, M., Ren, M., Aiken, A., Dally, W. J., and Hanrahan, P. Sequoia: Programming the Memory Hierarchy. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing* (New York, NY, USA, 2006), SC '06, ACM.

[21] Freitas, R. F., and Wilcke, W. W. Storage-class Memory: The Next Storage System Technology. *IBM J. Res. Dev. 52*, 4 (July 2008), 439–447.

[22] Ganger, G. R., Engler, D. R., Kaashoek, M. F., Briceño, H. M., Hunt, R., and Pinckney, T. Fast and Flexible Application-level Networking on Exokernel Systems. *ACM Trans. Comput. Syst. 20*, 1 (Feb. 2002), 49–83.

[23] Gerstenberger, R., Besta, M., and Hoefler, T. Enabling Highly-scalable Remote Memory Access Programming with MPI-3 One Sided. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (New York, NY, USA, 2013), SC '13, ACM, pp. 53:1–53:12.

[24] Gharachorloo, K., Lenoski, D., Laudon, J., Gibbons, P., Gupta, A., and Hennessy, J. Memory Consistency and Event Ordering in Scalable Shared-memory Multiprocessors. *SIGARCH Comput. Archit. News 18*, 2SI (May 1990), 15–26.

[25] Gibson, G. A., Nagle, D. F., Amiri, K., Butler, J., Chang, F. W., Gobioff, H., Hardin, C., Riedel, E., Rochberg, D., and Zelenka, J. A Cost-effective, High-bandwidth Storage Architecture. *SIGPLAN Not. 33*, 11 (Oct. 1998), 92–103.

[26] Gropp, W., Hoefler, T., Thakur, R., and Lusk, E. *Using Advanced MPI: Modern Features of the Message-Passing Interface*. MIT Press, Nov. 2014.

[27] Herlihy, M., and Moss, J. E. B. Transactional Memory: Architectural Support for Lock-free Data Structures. *SIGARCH Comput. Archit. News 21*, 2 (May 1993), 289–300.

[28] Intel. Intel Virtualization Technology for Directed I/O (VT-d) Architecture Specification, September 2013.

[29] Josephson, W. K., Bongo, L. A., Li, K., and Flynn, D. DFS: A File System for Virtualized Flash Storage. *Trans. Storage 6*, 3 (Sept. 2010), 14:1–14:25.

[30] Lee, E. K., and Thekkath, C. A. Petal: Distributed Virtual Disks. *SIGOPS Oper. Syst. Rev. 30*, 5 (Sept. 1996), 84–92.

[31] Ligon, W. B., . I., and Ross, R. B. Implementation and Performance of a Parallel File System for High Performance Distributed Applications. In *Proceedings of the 5th IEEE International Symposium on High Performance Distributed Computing* (Washington, DC, USA, 1996), HPDC '96, IEEE Computer Society.

[32] Lowell, D. E., and Chen, P. M. Free Transactions with Rio Vista. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 1997), SOSP '97, ACM, pp. 92–101.

[33] Magoutis, K. The optimistic direct access file system: Design and network interface support. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture (HPCA'02)* (2002).

[34] Magoutis, K., Addetia, S., Fedorova, A., Seltzer, M. I., Chase, J. S., Gallatin, A. J., Kisley, R., Wickremesinghe, R., and Gabber, E. Structure and Performance of the Direct Access File System. In *Proceedings of the General Track of the Annual Conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2002), ATEC '02, USENIX Association, pp. 1–14.

[35] Murray, D. G., McSherry, F., Isaacs, R., Isard, M., Barham, P., and Abadi, M. Naiad: A Timely Dataflow System. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2013), SOSP '13, ACM, pp. 439–455.

[36] Nightingale, E. B., Elson, J., Fan, J., Hofmann, O., Howell, J., and Suzue, Y. Flat Datacenter Storage. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2012), OSDI'12, USENIX Association, pp. 1–15.

[37] Oral, S., Wang, F., Dillow, D., Shipman, G., Miller, R., and Drokin, O. Efficient Object Storage Journaling in a Distributed Parallel File System. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2010), FAST'10, USENIX Association, pp. 11–11.

[38] Peter, S., Li, J., Zhang, I., Ports, D. R. K., Woos, D., Krishnamurthy, A., Anderson, T., and Roscoe, T. Arrakis: The Operating System is the Control Plane. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)* (Broomfield, CO, Oct. 2014), USENIX Association, pp. 1–16.

[39] Schmuck, F., and Haskin, R. GPFS: A Shared-Disk File System for Large Computing Clusters. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2002), FAST '02, USENIX Association.

[40] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and Noveck, D. NFS Version 4 Protocol, 2000.

[41] Spector, A. Z. Performing Remote Operations Efficiently on a Local Computer Network. *Commun. ACM 25*, 4 (Apr. 1982), 246–260.

[42] Stonebraker, M. Operating system support for database management. *Communications of the ACM 24*, 7 (1981), 412–418.

[43] Trivedi, A., Stuedi, P., Metzler, B., Pletka, R., Fitch, B. G., and Gross, T. R. Unified High-performance I/O: One Stack to Rule Them All. In *Proceedings of the 14th USENIX Conference on Hot Topics in Operating Systems* (Berkeley, CA, USA, 2013), HotOS'13, USENIX Association, pp. 4–4.

[44] 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.

[45] von Eicken, T., Basu, A., Buch, V., and Vogels, W. U-Net: A User-level Network Interface for Parallel and Distributed Computing. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 1995), SOSP '95, ACM, pp. 40–53.

[46] Warfield, A., Ross, R., Fraser, K., Limpach, C., and Hand, S. Parallax: Managing Storage for a Million Machines. In *Proceedings of the 10th Conference on Hot Topics in Operating Systems - Volume 10* (Berkeley, CA, USA, 2005), HOTOS'05, USENIX Association, pp. 4–4.

[47] Welsh, M., Basu, A., and von Eicken, T. Incorporating Memory Management into User-Level Network Interfaces. Tech. rep., Cornell University, Ithaca, NY, USA, 1997.

[48] White, S. J., and DeWitt, D. J. QuickStore: A High Performance Mapped Object Store. *The VLDB Journal 4*, 4 (Oct. 1995), 629–673.

[49] Wong, H.-S., Raoux, S., Kim, S., Liang, J., Reifenberg, J. P., Rajendran, B., Asheghi, M., and Goodson, K. E. Phase Change Memory. *Proceedings of the IEEE 98*, 12 (Dec 2010), 2201–2227.

[50] Xie, C., Su, C., Kapritsos, M., Wang, Y., Yaghmazadeh, N., Alvisi, L., and Mahajan, P. Salt: Combining ACID and BASE in a Distributed Database. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2014), OSDI'14, USENIX Association, pp. 495–509.

[51] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing* (Berkeley, CA, USA, 2010), HotCloud'10, USENIX Association, pp. 10–10.

[52] Zhang, J., Behzad, B., and Snir, M. Optimizing the Barnes-Hut Algorithm in UPC. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* (New York, NY, USA, 2011), SC '11, ACM, pp. 75:1–75:11.