QuickSAN: A Storage Area Network for Fast, Distributed, Solid State Disks

Adrian M. Caulfield Steven Swanson
Computer Science and Engineering Department
University of California, San Diego
{acaulfie,swanson}@cs.ucsd.edu

ABSTRACT

Solid State Disks (SSDs) based on flash and other non-volatile memory technologies reduce storage latencies from 10s of milliseconds to 10s or 100s of microseconds, transforming previously inconsequential storage overheads into performance bottlenecks. This problem is especially acute in storage area network (SAN) environments where complex hardware and software layers (distributed file systems, block severs, network stacks, etc.) lie between applications and remote data. These layers can add hundreds of microseconds to requests, obscuring the performance of both flash memory and faster, emerging non-volatile memory technologies.

We describe QuickSAN, a SAN prototype that eliminates most software overheads and significantly reduces hardware overheads in SANs. QuickSAN integrates a network adapter into SSDs, so the SSDs can communicate directly with one another to service storage accesses as quickly as possible. QuickSAN can also give applications direct access to both local and remote data without operating system intervention, further reducing software costs. Our evaluation of QuickSAN demonstrates remote access latencies of 20 μs for 4 KB requests, bandwidth improvements of as much as 163$\times$ for small accesses compared with an equivalent iSCSI implementation, and 2.3-3.0$\times$ application level speedup for distributed sorting. We also show that QuickSAN improves energy efficiency by up to 96% and that QuickSAN's networking connectivity allows for improved cluster-level energy efficiency under varying load.

1. INTRODUCTION

Modern storage systems rely on complex software and interconnects to provide scalable, reliable access to large amounts of data across multiple machines. In conventional, disk-based storage systems the overheads from file systems, remote block device protocols (e.g., iSCSI and Fibre Channel), and network stacks are tiny compared to the storage media access time, so software overheads do not limit scalability or performance.

The emergence of non-volatile, solid-state memories (e.g., NAND flash, phase change memory, and spin-torque MRAM) changes this landscape completely by dramatically improving storage media performance. As a result, software shifts from being the least important component in overall storage system performance to being a critical bottleneck.

The software costs in scalable storage systems arise because, until recently, designers could freely compose existing system components to implement new and useful storage system features. For instance, the software stack running on a typical, commodity storage area network (SAN) will include a file system (e.g., GFS [29]), a logical volume manager (e.g., LVM), remote block transport layer client (e.g., the iSCSI initiator), the client side TCP/IP network stack, the server network stack, the remote block device server (e.g., the iSCSI target), and the block device driver. The combination of these layers (in addition to the other operating system overheads) adds 288 μs of latency to a 4 kB read. Hardware accelerated solutions (e.g., Fibre Channel) eliminate some of these costs, but still add over 90 μs per access.

Software latencies of hundreds of microseconds are inconsequential for disk accesses (with a hardware latency measured in many milliseconds), but modern NAND flash-based solid state disks (SSDs) measure latencies in the 10s of microseconds and future PCM- or STTM-based devices [4, 32] will deliver $<$10 μs latencies. At those speeds, conventional software and block transport latencies will cripple performance.

This paper describes a new SAN architecture called *QuickSAN*. QuickSAN re-examines the hardware and software structure of distributed storage systems to minimize software overheads and let the performance of the underlying storage technology shine through.

QuickSAN improves SAN performance in two ways. First, it provides a very low-latency, hardware-accelerated block transport mechanism (based on 10 Gbit ethernet) and integrates it directly into an SSD. Second, it extends a recently proposed OS bypass mechanism [5] to allow an application to access remote data without (in most cases) any intervention from the operating system while still enforcing file system protections.

We examine two different structures for QuickSAN. *Centralized QuickSAN* systems mirror conventional SAN topologies that use a centralized data storage server. *Distributed QuickSAN* enables distributed storage by placing one slice of a globally shared storage space in each client node. As a result, each node has very fast access to its local slice, and retains fast, direct access to the slices at the other nodes as well. In both architectures, clients see storage as a single, shared storage device and access data via a shared-disk file system and the network.

We evaluate QuickSAN using GFS2 [12] and show that it reduces software and block transfer overheads by 94%, reduces overall latency by 93% and improves bandwidth by 14.7$\times$ for 4 KB requests. We also show that leveraging the fast access to local storage that distributed QuickSAN systems provide can improve performance by up to 3.0$\times$ for distributed sorting, the key bottleneck

ISCA '13 Tel-Aviv, Israel

Component	Existing interfaces Latency R/W (μs) iSCSI	Existing interfaces Latency R/W (μs) Fibre Channel	QuickSAN Latency R/W (μs) Kernel	QuickSAN Latency R/W (μs) Direct
OS Entry/Exit & Block IO	4.0 / 3.4		3.8 / 3.4	0.3 / 0.3
GFS2	0.1 / 1.5		1.4 / 2.3	n/a
Block transport	284.0 / 207.7	86.0 / 85.9	17.0/16.9	17.0/16.9
4 kB non-media total	288.1 / 212.6	90.1 / 90.8	22.2/22.6	17.2/17.1

Table 1: *Software and block transport latencies* Software stack and block transport latencies add overhead to storage accesses. Software-based SAN interfaces like iSCSI takes 100s of microseconds. Fibre Channel reduces this costs, but the additional latency is still high compared to the raw performance of fast non-volatile memories. The Fibre Channel numbers are estimates based on [1] and measurements of QuickSAN.

in MapReduce systems. We also measure QuickSAN's scalability, its impact on energy efficiency, and the impact of adding mirroring to improve reliability. Finally, we show how distributed QuickSAN can improve cluster-level efficiency under varying load.

The remainder of this paper is organized as follows. Section 2 describes existing SAN-based storage architectures and quantifies the software latencies they incur. Section 3 describes QuickSAN's architecture and implementation in detail. Section 4 places QuickSAN in the context of previous efforts. Section 5 evaluates both centralized and distributed QuickSAN to compare their performance on a range of storage workloads. Section 6 presents our conclusions.

2. MOTIVATION

Non-volatile memories that offer order of magnitude increases in performance require us to reshape the architecture of storage area networks (SANs). SAN-based storage systems suffer from two sources of overhead that can obscure the performance of the underlying storage. The first is the cost of the device-independent layers of the software stack (e.g., the file system), and the second is the *block transport cost*, the combination of software and interconnect latency required to access remote storage. These costs range from 10s to 100s of microseconds in modern systems.

As non-volatile memories become more prevalent, the cost of these two components grows relative to the cost of the underlying storage. For disks, with latencies typically between 7 and 15 ms, the costs of these layers is almost irrelevant. Flash-based SSDs, however, have access times under 100 μs and SSDs based on more advanced memories will have $<$10 μs latencies. In these systems, the cost of conventional storage software and block transport will completely dominate storage costs.

This section describes the sources of these overheads in more detail and places them in context of state-of-the-art solid-state storage devices.

2.1 Storage overheads

Figure 1 shows the architecture of two typical SAN-based shared-disk remote storage systems. Both run under Linux and GFS2 [12], a distributed, shared-disk file system. In Figure 1(a) a software implementation of the iSCSI protocol provides block transport. An iSCSI target (i.e., server) exports a block-based interface to a local storage system. Multiple iSCSI client initiators (i.e., clients) connect to the target, providing a local interface to the remote device.

Fibre Channel, Figure 1(b), provides block transport by replacing the software below the block interface with a specialized SAN card that communicates with Fibre Channel storage devices. Fibre Channel devices can be as simple as a hard drive or as complex as a multi-petabyte, multi-tenancy storage system, but in this work we focus on performance-optimized storage devices (e.g., the Flash Memory Arrays [30] from Violin Memory).

The values in Table 1 measure the OS and block transport latency for a 4 kB request from the applications perspective for iSCSI and Fibre Channel. The first three rows are the generic OS storage stack. The fourth row measures the minimum latency for a remote storage access starting at the block interface boundary in Figure 1, and excluding the cost of the actual storage device. These latencies seem small compared to the 7-11 ms required for the disk access, but next-generation SSDs fundamentally alter this balance.

2.2 The Impact of Fast SSDs

Fast non-volatile memories are already influencing the design of storage devices in the industry. As faster memories become available the cost of software and hardware overheads on storage accesses will become more pronounced.

Commercially available NAND flash-based SSDs offer access latencies of tens of microseconds, and research prototypes targeting more advanced memory technologies have demonstrated latencies as low as 4-5 μs [5].

Violin Memory's latest 6000 Series Flash Memory Arrays [30] can perform read accesses in 80-100 μs and writes in 20 μs with aggressive buffering. FusionIO's latest ioDrives [10] do even better: 68 μs for reads and 15 μs for writes, according to their data sheets. Similar drives from Intel [17] report 65 μs for reads and writes.

At least two groups [4, 5, 32] have built prototype SSDs that use DRAM to emulate next-generation memories like phase-change memories, spin-torque MRAMs, and the memristor. These devices can achieve raw hardware latencies of between 4 and 5 μs [5, 32].

Figure 2 illustrates the shift in balance between the cost of underlying storage and the cost of software and the block transport layers. For a disk-based SAN, these overheads represent between 1.2 and 3.8% of total latency. For flash-based SSDs the percentage climbs as high as 59.6%, and for more advanced memories software overheads will account for 98.6-99.5% of latency.

Existing software stacks and block transport layers do not provide the low-latency access that fast SSDs require to realize their full potential. The next section describes our design for QuickSAN that removes most of those costs to expose the full performance of SSDs to software.

3. QuickSAN

This section describes the hardware and software components of QuickSAN. The QuickSAN hardware adds a high-speed network interface to a PCIe-attached SSD, so SSDs can communicate directly with one another and retrieve remote data to satisfy local IO requests. The QuickSAN software components leverage this capability (along with existing shared-disk file systems) to eliminate most of the software costs described in the previous section.

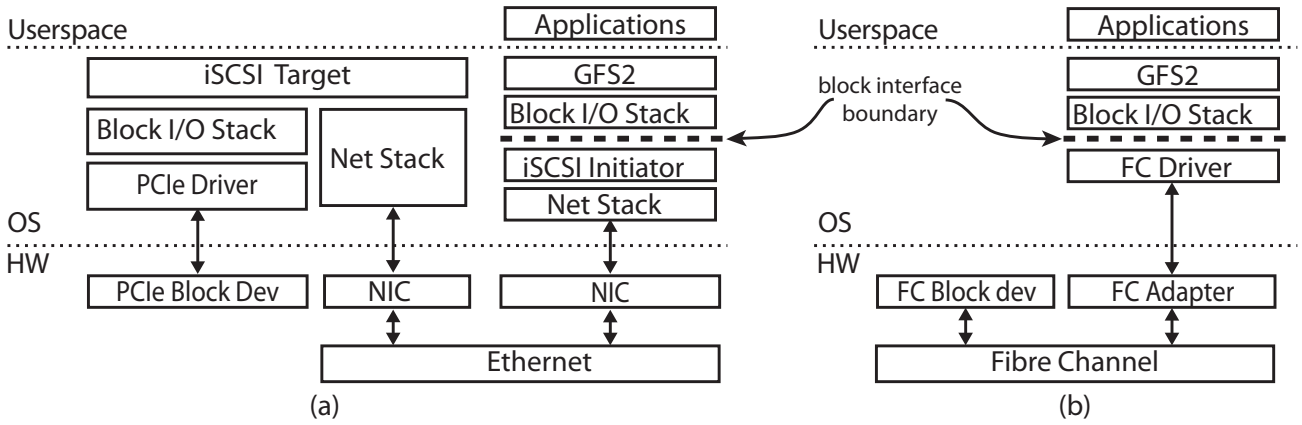

Figure 1: ***Existing SAN architectures*** **Existing SAN systems use software-based architectures like software iSCSI (a) or specialized hardware solutions like Fibre Channel (b). Both hide ample complexity behind the block device interface they present to the rest of the system. That complexity and the software that sits above it in OS both add significant overheads to storage access.**

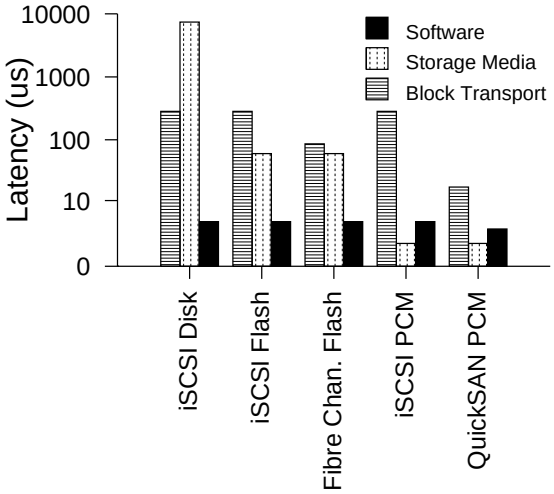

Figure 2: ***Shifting bottlenecks*** **As storage shifts from disks to solid-state media, software and block transport account for a growing portion of overall latency. This shift requires us to rethink conventional SAN architectures.**

3.1 QuickSAN Overview

We explore two different architectures for QuickSAN storage systems. The first resembles a conventional SAN storage system based on Fibre Channel or iSCSI. A central server (at left in Figure 3) exports a shared block storage device, and client machines access that device via the network. We refer to this as the *centralized* architecture.

The second, *distributed*, architecture replaces the central server with a distributed set of SSDs, one at each client (Figure 3, right). The distributed SSDs combine to expose a single, shared block device that any of the clients can access via the network.

To explore ways of reducing software overheads in these two architectures, we have implemented two hardware devices: A customized network interface card (the QuickSAN NIC) and a custom SSD (the QuickSAN SSD) that integrates both solid state storage and QuickSAN NIC functionality. In some of our experiments we also disable the network interface on the QuickSAN SSD, turning it into a normal SSD.

The QuickSAN NIC is similar to a Fibre Channel card: It exports a block device interface and forwards read and write requests over a network to a remote storage device (in our case, a QuickSAN SSD). The QuickSAN SSD responds to remote requests without communicating with the operating system, so it resembles a Fibre Channel client and target device.

The QuickSAN SSD also exports a block device interface to the host system. The interface's block address space includes the SSD's local storage and the storage on the other QuickSAN SSDs it is configured to communicate with. The QuickSAN SSD services accesses to local storage directly and forwards requests to external storage to the appropriate QuickSAN SSD.

This organization allows QuickSAN SSDs to communicate in a peer-to-peer fashion, with each SSD seeing the same global storage address space. This mode of operation is not possible with conventional SAN hardware.

QuickSAN further reduces software latency by applying the OS-bypass techniques described in [5]. These techniques allow applications to bypass the system call and file system overheads for common-case read and write operations while still enforcing file system protections.

The following subsections describe the implementation of the QuickSAN SSD and NIC in detail, and then review the OS bypass mechanism and its implications for file system semantics.

3.2 The QuickSAN SSD and NIC

The QuickSAN NIC and the QuickSAN SSD share many aspects of their design and functionality. Below, we describe the common elements of both designs and then the SSD- and NIC-specific hardware. Then we discuss the hardware platform we used to implement them both.

Common elements The left half of Figure 4 contains the common elements of the SSD and NIC designs. These implement a storage-like interface that the host system can access. It supports read and write operations from/to arbitrary locations and of arbitrary sizes (i.e., accesses do not need to be aligned or block-sized) in 64 bit address space.

The hardware includes the host-facing PIO and DMA interfaces, the request queues, the request scoreboard, and internal buffers. These components are responsible for accepting IO requests, executing them, and notifying the host when they are complete. Communication with the host occurs over a PCIe 1.1x8 interface, which

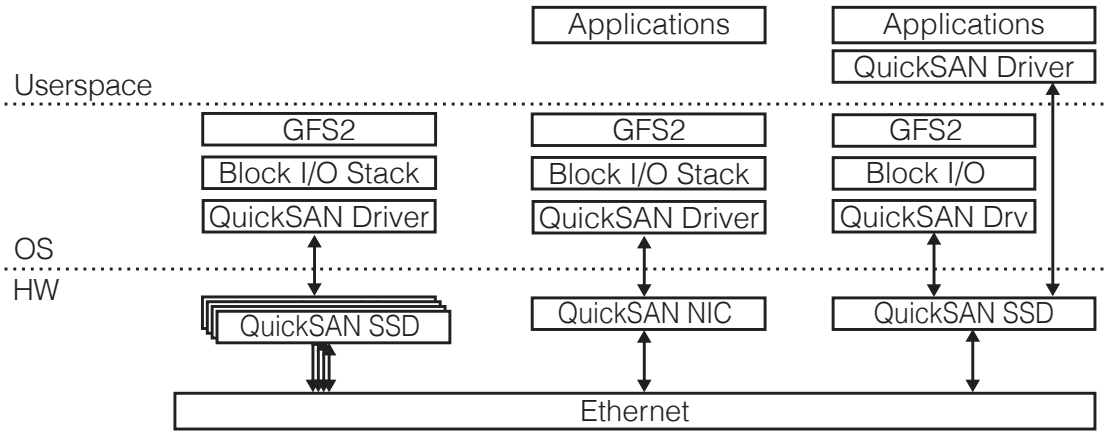

Figure 3: *QuickSAN configurations* QuickSAN supports multiple storage topologies and two software interfaces. At left, a single machine hosts multiple QuickSAN SSDs, acting a central block server. The center machine hosts a QuickSAN NIC that provides access to remote SSDs. The machine at right hosts a single SSD and is poised to access its local (for maximum performance) or remote data via the userspace interface.

runs at 2 GB/s, full-duplex. This section also includes the virtualization and permission enforcement hardware described below (and in detail in [5]).

Data storage The storage-related portions of the design are shown in the top-right of Figure 4. The SSD contains eight high-performance, low-latency non-volatile memory controllers attached to an internal ring network. The SSD's local storage address space is striped across these controllers with a stripe size of 8 kB.

In this work, the SSD uses emulated phase change memory, with the latencies from [19] — 48 ns and 150 ns for array reads and writes, respectively. The array uses start-gap wear leveling [23] to distribute wear across the phase change memory and maximize lifetime.

The network interface QuickSAN's network interface communicates over a standard 10 Gbit CX4 ethernet port, providing connectivity to other QuickSAN devices on the same network. The network interfaces plays two roles: It is a source of requests, like the PCIe link from the host, and it is also a target for data transfers, like the memory controllers.

The link allows QuickSAN to service remote requests without operating system interaction on the remote node and even allows access to storage on a remote node when that node's CPU is powered off (assuming the SSD has an independent power supply).

QuickSAN requires a lossless interconnect. To ensure reliable delivery of network packets, QuickSAN uses ethernet but employs flow control as specified in the IEEE 802.3x standard. Fibre Channel over ethernet (FCoE) uses the same standards to provide reliable delivery. Ethernet flow control can interact poorly with TCP/IP's own flow control and is usually disabled on data networks, but a more recent standard, 802.1qbb, extends flow control to cover multiple classes of traffic and alleviates these concerns. It is part of the "data center bridging" standard pushing for converged storage and data networks in data centers [7].

Ethernet flow control provides the necessary reliability guarantees that QuickSAN needs, but it runs the risk of introducing deadlock into the network if insufficient buffering is available. For example, deadlock can occur if an SSD must pause incoming traffic due to insufficient buffer space, but it also requires an incoming response from the network before it can clear enough buffer space to unpause the link.

We have carefully designed QuickSAN's network interface to ensure that it always handles incoming traffic without needing to send an immediate response on the network, thus breaking the conditions necessary for deadlock. We guarantee sufficient buffer space for (or can handle without buffering) all incoming small (non-payload bearing) incoming messages. Read requests allocate buffers at the source before sending the request, and a dedicated buffer is available for incoming write requests, which QuickSAN clears without needing to send an outgoing packet.

Userspace access The host-facing interface of the QuickSAN SSD and NIC includes support for virtualization and permissions enforcement, similar to the mechanisms presented in [5]. These mechanisms allow applications to issue read and write requests directly to the hardware without operating system intervention while preserving file system permissions. This eliminates, for most operations, the overheads related to those software layers.

The virtualization support exposes multiple, virtual hardware interfaces to the NIC or SSD, and the operating system assigns one virtual interface to each process. This enables the process to issue and complete IO operations. A user space library interposes on file operations to provide an almost (see below) POSIX-compliant interface to the device.

The permissions mechanism associates a permissions table with each virtual interface. The hardware checks each request that arrives via the PCIe interface against the permission table for the appropriate interface. If the interface does not have permission to perform the access, the hardware notifies the process that the request has failed.

The operating system populates the permission table on behalf of the process by consulting the file system to retrieve file layout and permission information.

Permission checks in QuickSAN occur at the source NIC or SSD rather than at the destination. This represents a trade off between scalability and security. If permission checks happened at the destination, the destination SSD would need to store permission records for all remote requestors and would need to be aware of remote virtual interfaces. Checking permissions at the source allows the number of system wide permission entries and virtual interfaces to

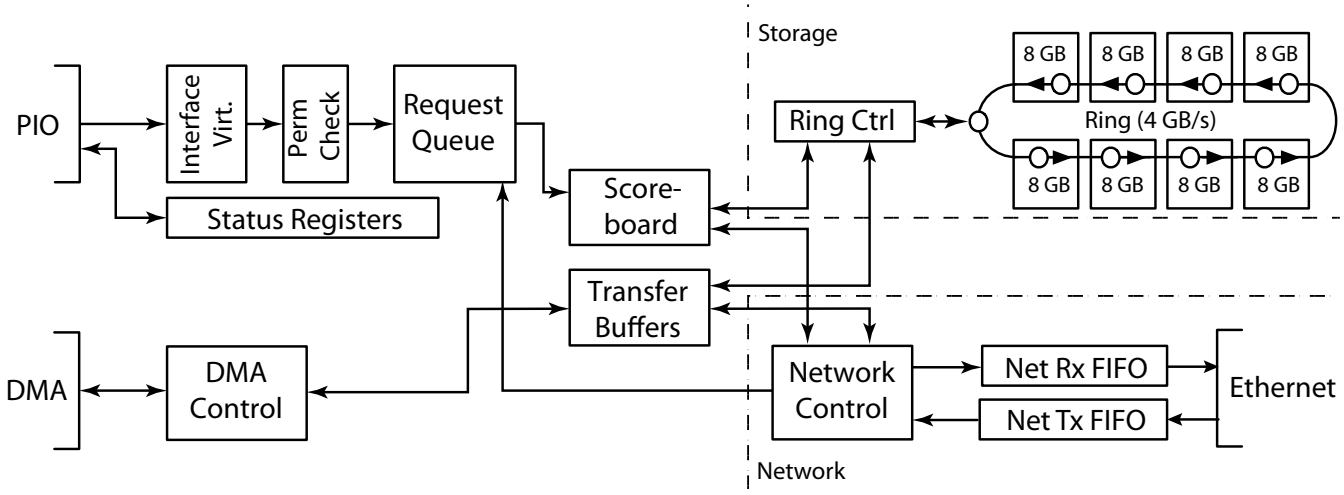

Figure 4: ***QuickSAN's internal architecture*** **The QuickSAN NIC and SSD both expose a virtualized, storage interface to the host system via PCIe (at left). The network interface attaches to a 10 Gbit network port (bottom-right), while the QuickSAN SSD adds 64 GB of non-volatile storage (top-left). The device services requests that arrive via either the network or host interface and forwards requests for remote storage over the network.**

scale with the number of clients. A consequence is that QuickSAN SSDs trust external requests, an acceptable trade-off in most clustered environments.

Since the userspace interface provides direct access to storage, it can lead to violations of the atomicity guarantees that POSIX requires. For local storage devices hardware support for atomic writes [8] can restore atomicity. For distributed storage systems, however, achieving atomicity requires expensive distributed locking protocols, and implementing these protocols in hardware seems unwieldly.

Rather than provide atomicity in hardware, we make the observation that most applications that require strong consistency guarantees and actively share files (e.g., databases) rely on higher-level, application-specific locking primitives. Therefore, we relax the atomicity constraint for accesses through the userspace interface. If applications require the stronger guarantees, they can perform accesses via the file system and rely on it to provide atomicity guarantees. The userspace library automatically uses the file system for append operations since they must be atomic.

Implementation We implemented QuickSAN on the BEE3 prototyping platform [2]. The BEE3 provides four FPGAs which each host 16 GB of DDR2 DRAM, two 10 Gbit ethernet ports, and one PCIe interface. We use one of the ethernet ports and the PCIe link on one FPGA for external connectivity. The design runs at 250 MHz.

To emulate the performance of phase change memory using DRAM, we used a modified DRAM controller that allows us to set the read and write latency to the values given earlier.

3.3 QuickSAN software

Aside from the userspace libray described above, most of the software that QuickSAN requires is already commonly available on clustered systems.

QuickSAN's globally accessible storage address space can play host to conventional, shared-disk file systems. We have successfully run both GFS2 [12] and OCSF2 [21] on both the distributed and local QuickSAN configurations. We expect it would work with most other shared-disk file systems as well. Using the userspace interface requires that the kernel be able to query the file system for file layout information (e.g., via the fiemap ioctl), but this support is common across many file systems.

In a production system, a configuration utility would set control registers in each QuickSAN device to describe how the global storage address space maps across the QuickSAN devices. The hardware uses this table to route requests to remote devices. This software corresponds to a logical volume management interface in a conventional storage system.

4. RELATED WORK

Systems and protocols that provide remote access to storage fall into several categories. Network-attached storage (NAS) systems provide file system-like interfaces (e.g., NFS and SMB) over the networks. These systems centralize metadata management, avoiding the need for distributed locking, but their centralized structure limits scalability.

QuickSAN most closely resembles existing SAN protocols like Fibre Channel and iSCSI. Hardware accelerators for both of these interfaces are commonly available, but iSCSI cards are particularly complex because they typically also include a TCP offload engine. Fibre Channel traditionally runs over a specialized interconnect, but new standards for Converged Enhanced Ethernet [7] (CEE) allow for Fibre Channel over lossless Ethernet (FCoE) as well. QuickSAN uses this interconnect technology. ATA over Ethernet (AoE) [16] is a simple SAN protocol designed specifically for SATA devices and smaller systems.

Systems typically combine SAN storage with a distributed, shared-disk file system that runs across multiple machines and provides a single consistent file system view of stored data. Shared-disk file systems allow concurrent access to a shared block-based storage system by coordinating which servers are allowed to access particular ranges of blocks. Examples include the Global File System (GFS2) [29], General Parallel File System [26], Oracle Cluster File System [21], Clustered XFS [27], and VxFS [31].

Parallel file systems (e.g., Google FS [13], Hadoop Distributed File System [28], Lustre [18], and Parallel NFS [15]) also run across multiple nodes, but they spread data across the nodes as well. This gives applications running at a particular node faster access to local storage.

Although QuickSAN uses existing shared-disk file systems, it actually straddles the shared-disk and parallel file system paradigms depending on the configuration. On one hand, the networked QuickSAN SSDs nodes appear as a single, distributed storage device that fits into the shared disk paradigm. On the other, each host will have a local QuickSAN SSD, similar to the local storage in a parallel file system. In both cases, QuickSAN can eliminate the software overhead associated with accessing remote storage.

Parallel NFS (pNFS) [15] is a potential target for QuickSAN, since it allows clients to bypass the central server for data accesses. This is a natural match for QuickSAN direct access capabilities.

QuickSAN also borrows some ideas from remote DMA (RDMA) systems. RDMA allows one machine to copy the contents of another machine's main memory directly over the network using a network protocol that supports RDMA (e.g., Infiniband). Previous work leveraged RDMA to reduce processor and memory IO load and improve performance in network file systems [3, 33]. RamCloud [22] utilizes RDMA to reduce remote access to DRAM used as storage. QuickSAN extends the notion of RDMA to storage and realizes many of the same benefits, especially in configurations that can utilize its userspace interface.

QuickSAN's direct-access capabilities are similar to those provided by Network Attached Secure Disks (NASD) [14]. NASD integrated SAN-like capabilities into storage devices directly exposed them over the network. QuickSAN extends this notion by also providing fast access to the local host to fully leverage the performance of solid state storage.

Alternate approaches to tackling large-scale data problems have also been proposed. Active Storage Fabrics [9], proposes combining fast storage and compute resources at each node. Processing accelerators can then run locally within the node. QuickSAN also allows applications to take advantage of data locality, but the architecture focuses on providing low-latency access to all of the data distributed throughout the network.

5. RESULTS

This section evaluates QuickSAN's latency, bandwidth, and scalability along with other aspects of its performance. We measure the cost of mirroring in QuickSAN to improve reliability, and evaluate its ability to improve the energy efficiency of storage systems. Finally, we measure the benefits that distributed QuickSAN configurations can provide for sorting, a key bottleneck in MapReduce workloads. First, however, we describe the three QuickSAN configurations we evaluated.

5.1 Configurations

We compare performance across both centralized and distributed SAN architectures using three different software stacks.

For the centralized topology, we attached four QuickSAN SSDs to a single host machine and expose them as a single storage device. Four clients share the resulting storage device. In the distributed case, we attach one QuickSAN SSD to each of four machines to provide a single, distributed storage device. In both cases, four clients access the device. In the distributed case, the clients also host data.

	Local		Remote	
	Read	Write	Read	Write
iSCSI	92.3	111.7	296.7	236.5
QuickSAN-OS	15.8	17.5	27.0	28.6
QuickSAN-D	8.3	9.7	19.5	20.7

Table 2: *QuickSAN latency* Request latencies for 4 KB transfer to each of the configurations to either local or remote storage. The distributed configurations have a mix of local and remote accesses, while the centralized configurations are all remote.

We run three different software stacks on each of these configurations.

iSCSI This software stack treats the QuickSAN SSD as a generic block device and implements all SAN functions in software. In the centralized case, Linux's clustered Logical Volume Manager (cLVM) combines the four local devices into a single logical device and exposes it as an iSCSI target. In the distributed case, each machine exports its local SSD as an iSCSI target, and cLVM combines them into a globally shared device. Client machines use iSCSI to access the device, and issue requests via system calls.

QuickSAN-OS The local QuickSAN device at each client (a NIC in the centralized case and an SSD in the distributed case) exposes the shared storage device as a local block device. Applications access the device through the normal system call interface.

QuickSAN-D The hardware configuration is the same as QuickSAN-OS, but applications use the userspace interface to access storage.

Our test machines are dual-socket Intel Xeon E5520-equipped servers running at 2.27 GHz. The systems run CentOS 5.8 with kernel version 2.6.18. The network switch in all experiments is a Force10 S2410 CX4 10 GBit Ethernet Switch. In experiments that include a file system, we use GFS2 [12].

5.2 Latency

Reducing access latency is a primary goal of QuickSAN, and Table 2 contains the latency measurements for all of our configurations. The measurements use a single thread issuing requests serially and report average latency.

Replacing iSCSI with QuickSAN-OS reduces remote write latency by 92% – from 296 μs to 27 μs. Savings for reads are similar. QuickSAN-D reduces latency by an additional 2%. Based on the Fibre Channel latencies in Table 1, we expect that the saving for QuickSAN compared to a Fibre Channel-attached QuickSAN-like SSD would be smaller – between 75 and 81%.

5.3 Bandwidth

Figure 5 compares the sustained, mean per-host bandwidth for all six configurations for random accesses of size ranging from 512 bytes to 256 kB. Comparing the distributed configurations, QuickSAN-D outperforms the iSCSI baseline by 72.7× and 164.4× for small 512 B reads and writes, respectively. On larger 256 kB accesses, QuickSAN-D's advantage shrinks to 4.6× for reads and 4.4× for writes.

For QuickSAN, the distributed architecture outperforms the centralized architecture by 100 MB/s per node for 4 kB requests and maintains a 25 MB/s performance advantage across all request sizes for both kernel and userspace interfaces. Userspace write performance follows a similar trend with a 110 MB/s gain at 4 kB and at least a 45 MB/s gain for other sizes.

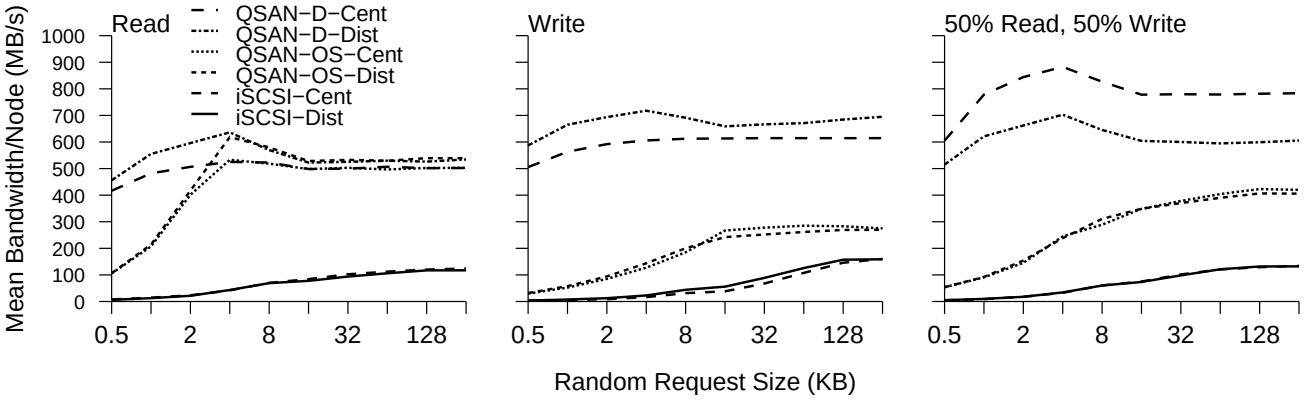

Figure 5: ***QuickSAN bandwidth*** **Eliminating software and block transport overheads improves bandwidth performance for all configurations. QuickSAN-D's userspace interface delivers especially large gains for small accesses and writes.**

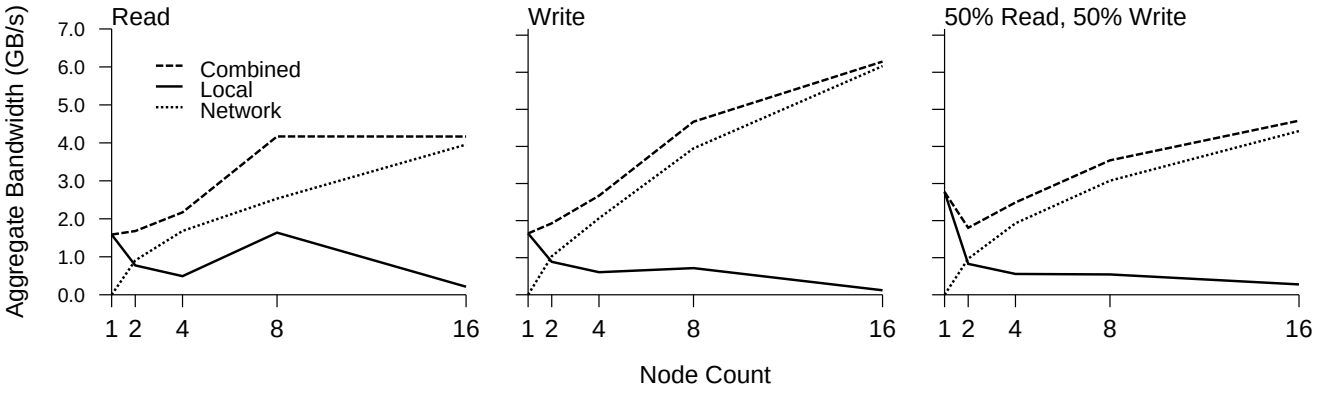

Figure 6: ***QuickSAN bandwidth scaling*** **For random requests spread across the shared storage address space, aggregate bandwidth improves as more nodes are added. Local bandwidth scales more slowly since a smaller fraction of accesses go to local storage as the number of nodes grows.**

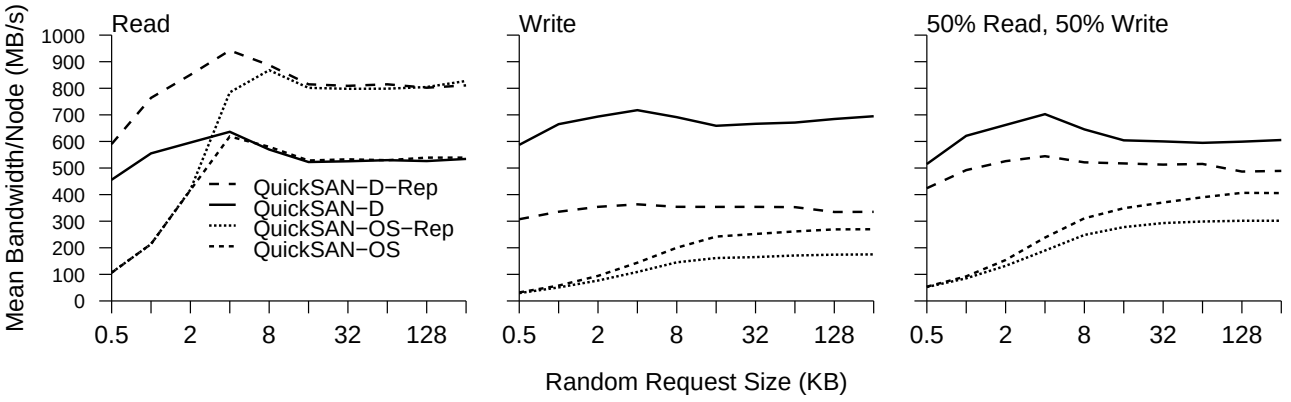

Figure 7: ***The impact of replication on bandwidth*** **Mirroring in QuickSAN improves bandwidth for read operations because two SSDs can service any given request and, in some cases, the extra copy will be located on the faster, local SSD. Write performance drops because of the extra update operations mirroring requires.**

Write performance through the kernel is much lower due to the distributed file system overheads. Some of that cost goes towards enforcing atomicity guarantees. The performance advantage for the distributed configuration stems from having fast access to a portion of the data and from better aggregate network bandwidth to storage.

Interestingly, for a 50/50 mix of reads and writes under QuickSAN-D, the centralized architecture outperforms the distributed architecture by between 90 and 180 MB/s (17-25%) per node across all request sizes. This anomaly occurs because local and remote accesses compete for a common set of buffers in the QuickSAN SSD. This limits the SSD's ability to hide the latency of remote accesses. In the centralized case, the QuickSAN NICs dedicate all their buffer space to remote accesses, hiding more latency, and improving performance. The read/write workloads exacerbates this problem for the distributed configuration because the local host can utilize the full-duplex PCIe link, putting more pressure on local buffers than in write- or read-only cases. The random access nature of the workload also minimizes the benefits of having fast access to local storage, a key benefit of the distributed organization.

The data show peak performance at 4 kB for most configurations. This is the result of a favorable interactions between the available buffer space and network flow control requirements for transfer 4 kB and smaller. As requests get larger, they require more time to move into and out of the transfer buffers causing some contention. 4 kB appears to be the ideal request size for balancing performance buffer access between the network and the internal storage.

5.4 Scaling

QuickSAN is built to scale to large clusters of nodes. To evaluate its scalability we ran our 4 kB, random access benchmark on between one and sixteen nodes. There are two competing trends at work: As the number of nodes increases, the available aggregate network bandwidth increases. At the same time, the fraction of the random accesses that target a nodes local storage drops. For this study we change the QuickSAN architecture slightly to include only 16 GB of storage per node, resulting in slightly decreased per-node memory bandwidth.

Figure 6 illustrates both trends and plots aggregate local and remote bandwidth across the nodes as well as total bandwidth. Aggregate local bandwidth scales poorly for small node counts as more requests target remote storage. Overall, bandwidth scales well for the mixed read/write workload: Quadrupling node count from two to eight increase total bandwidth by 2.0× and network bandwidth by 3.2×. Moving from two to eight nodes increases total bandwidth by 2.5× and network bandwidth by 4.5×. Writes experience similar improvements, two to sixteen node scaling increases total bandwidth by 3.3× and network bandwidth by 5.9×. For reads, total and network bandwidth scale well from two to eight nodes (2.5× and 2.8×, respectively), but the buffer contention mentioned above negatively impacts bandwidth. Additional buffers would remedy this problem. Network bandwidth continues to scale well for reads with two to sixteen node scaling producing a 4.4× increase.

Latency scaling is also an important consideration. As node counts grow, so must the number of network hops required to reach remote data. The minimum one-way latency for our network is 1 μs which includes the delay from two network interface cards, two network links, and the switch. Our measurements show that additional hops add 450 ns of latency each way in the best case.

5.5 Replication

SAN devices (and all high-end storage systems) provide data integrity protection in the form of simple mirroring or more complex RAID- or erasure code-based mechanisms. We have implemented mirroring in QuickSAN to improve data integrity and availability. When mirroring is enabled, QuickSAN allocates half of the storage capacity on each QuickSAN SSD as a mirror of the primary portion of another SSD. QuickSAN transparently issues two write requests for each write. QuickSAN can also select from any replicas to service remote read requests using a round-robin scheme, although it always selects a local replica, if one is available. With this scheme, QuickSAN can tolerate the failure of any one SSD.

Mirroring increases write latency by 1.7 μs for 512 B accesses and 7.7 μs for 4 KB accesses in both QuickSAN-D and QuickSAN-OS. The maximum latency impact for QuickSAN is 1.6 ×. Figure 7 measures the impact of mirroring on bandwidth. Sustained write bandwidth through the kernel interface drops by 23% for 4 kB requests with a maximum overhead of 35% on large transfers. The userspace interface, which has lower overheads and higher throughput, experiences bandwidth reductions of between 49 and 52% across all request sizes, but write bandwidth for 512 B requests is still 9.5 ×better than QuickSAN-OS without replication.

Replication improves read bandwidth significantly, since it spreads load more evenly across the SSDs and increases the likelihood that a copy of the data is available on the local SSD. For 4 KB accesses, performance rises by 48% for QuickSAN-D and 26% for QuickSAN-OS. Adding support for QuickSAN to route requests based on target SSD contention would improve performance further.

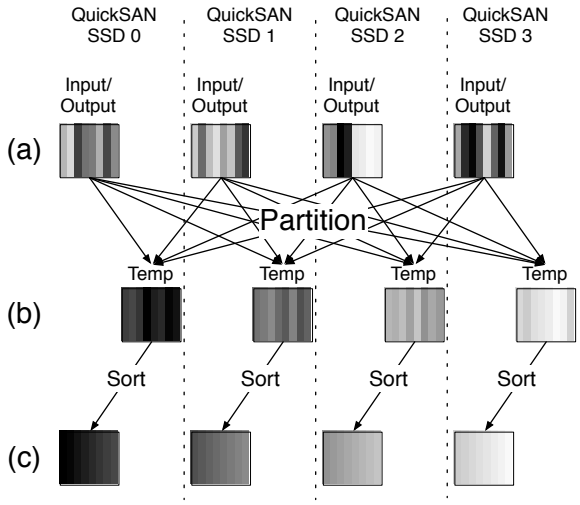

Figure 8: *Sorting on QuickSAN* Our sorting algorithm uses a partition stage followed by parallel, local sorts. The input and output stages utilize the fast access to local storage the QuickSAN provides, while the partitioning stage leverages SSD-to-SSD communicatio.

5.6 Sorting on QuickSAN

Our distributed QuickSAN organization provides non-uniform access latency depending on whether an access targets data on the local SSD or a remote SSD. By leveraging information about where data resides, distributed applications should be able to realize significant performance gains. Many parallel file systems support this kind of optimization.

To explore this possibility in QuickSAN, we implemented a distributed external sort on QuickSAN. Distributed sort is an impor-

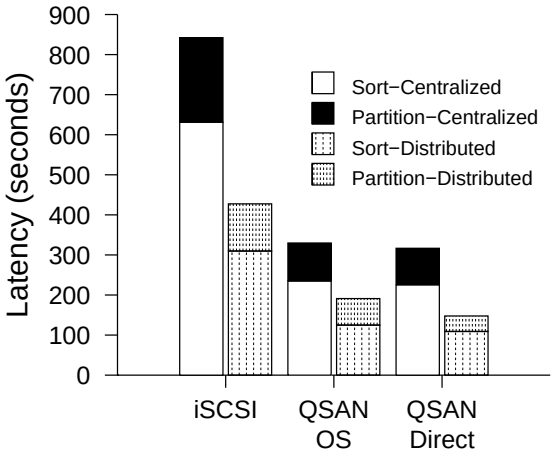

Figure 9: ***Sorting on QuickSAN*** **Removing iSCSI's software overheads improves performance as does exploiting fast access to local storage that a distributed organization affords.**

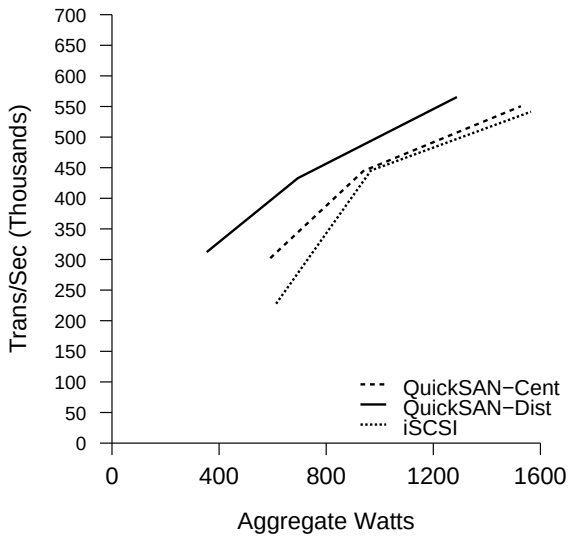

Figure 11: ***Application-level efficiency*** **QuickSAN's ability to access remote data without host intervention allows for more efficient server consolidation. QuickSAN provides better overall efficiency with any number of servers.**

tant benchmark in part because MapReduce implementations rely on it to partition the results of the map stage. Our implementation uses the same approach as TritonSort [25], and other work [24] describes how to implement a fast MapReduce on top of an efficient, distributed sorting algorithm.

Our sort implementation operates on a single 102.4 GB file filled with key-value pairs comprised of 10-byte keys and 90-byte values. The file is partitioned across the eight QuickSAN SSDs in the system. A second, temporary file is also partitioned across the SSDs.

Figure 8 illustrates the algorithm. In the first stage, each node reads the values in the local slice the input file (a) and writes each of them to the portion of the temporary file that is local to the node on which that value will eventually reside (b). During the second stage, each node reads its local data into memory, sorts it, and writes the result back to the input file (c), yielding a completely sorted file. We apply several optimizations described in [25], including buffering writes to the temporary file to minimize the number of writes and using multiple, smaller partitions in the intermediate file (c) to allow the final sort to run in memory.

Figure 9 displays the average runtime of the partition and sort stages of our sorting implementation. The data show the benefits of the distributed storage topology and the gains that QuickSAN provides by eliminating software and block transport overheads. For distributed QuickSAN, the userspace driver improves performance 1.3× compared to the OS interface and 2.9× compared to iSCSI. Although the distributed organization improves performance for all three interfaces, QuickSAN-D sees the largest boost – 2.14× versus 1.96× for iSCSI and 1.73× for the QuickSAN-OS interface.

5.7 Energy efficiency

Since software runs on power-hungry processors, eliminating software costs for IO requests can significantly reduce the energy that each IO operation requires. Figure 10 plots the Joules per 4 kB IOP across the three distributed storage configurations. To collect the numbers, we measured server power at the power supply. We assume a QuickSAN SSD implemented as a product would consume 24 W (the maximum power consumed by a FusionIO ioDrive [11]), and, conservatively, that power does not vary with load. The network switch's 150 W contribution does not vary significantly with load. The trends for the centralized configurations are similar.

The graph illustrates the high cost of software-only implementations like iSCSI. The iSCSI configurations consume between 6.2 and 12.7× more energy per IOP than the QuickSAN-OS and between 13.2 and 27.4× more than QuickSAN-D.

The data for writes and the mixed read/write workload also demonstrates the energy cost of strong consistency guarantees. For writes, accessing storage through the file system increases (QuickSAN-OS) energy costs by 4× relative to QuickSAN-D.

5.8 Workload consolidation

An advantage of centralized SANs over distributed storage systems that spread data across many hosts is that shutting down hosts to save power does not make any data unreachable. Distributed storage systems, however, provide fast access to local data, improving performance.

The QuickSAN SSD's ability to service requests from remote nodes, even if their host machine is powered down can achieve the best of both worlds: Fast local access when the cluster is under heavy load and global data visibility to support migration.

To explore this application space, we use four servers running a persistent key-value store (MemCacheDB [6]). During normal, fully-loaded operation each server runs a single instance of the MemCacheDB. We use four machines running memslap [20] to drive the key-value stores on the four machine under test.

As system load drops, we can migrate the key-value store to another server. On their new host, they can access the same data and transparently resume operation. We assume that a front-end steering mechanism redirects traffic as necessary using DNS. We can perform the same migration in both the centralized QuickSAN and centralized iSCSI configurations.

We measure the performance and power consumption of the system under three different configurations: centralized iSCSI, centralized QuickSAN, and distributed QuickSAN.

Figure 11 plots performance versus total power consumption for each system. The data show that the distributed QuickSAN reduces the energy cost of each request by 58 and 42% relative to the iSCSI

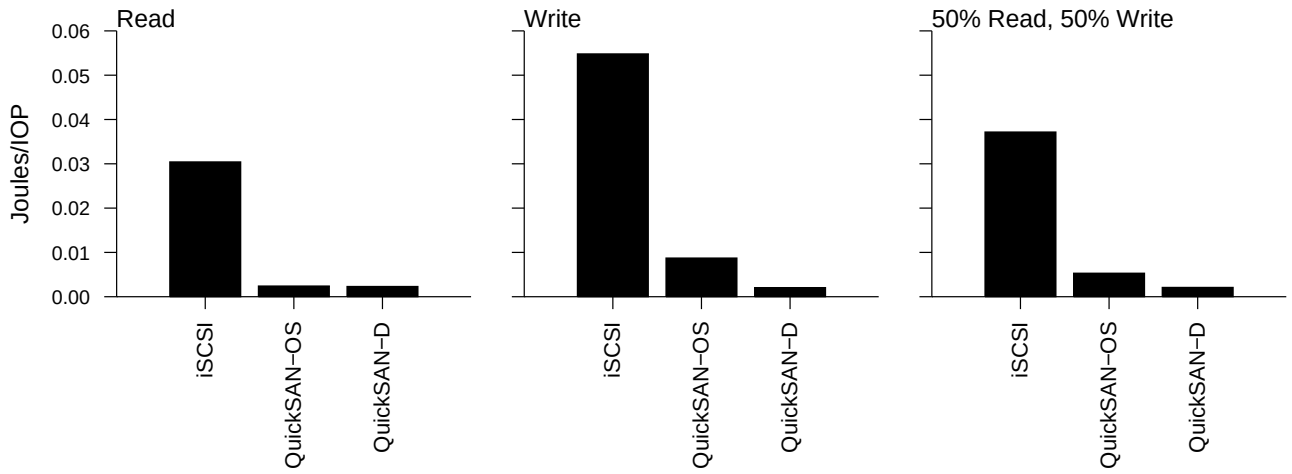

Figure 10: *QuickSAN energy efficiency* Depending on the software stack and block transport mechanism, the energy cost of storage access varies. Removing the high software overheads of iSCSI account for most of the gains, but the userspace interface saves 76% of energy for writes compared to the kernel version.

and centralized QuickSAN implementations, respectively.

6. CONCLUSION

We have described QuickSAN a new SAN architecture designed for solid state memories. QuickSAN integrates network functionality into a high-performance SSD to allow access to remote data without operating system intervention. QuickSAN reduces software and block transport overheads by between 82 and 95% compared it Fibre Channel and iSCSI-based SAN implementations and can improve bandwidth for small requests by up to 167×. We also demonstrated that QuickSAN can improve energy efficiency by 58% compared to a iSCSI-based SAN. QuickSAN illustrates the ongoing need to redesign computer system architectures to make the best use of fast non-volatile memories.

Acknowledgements

We would like to thank the ISCA program committee for their useful feedback, Maxim Adelman for our useful discussion about Fibre Channel performance, and Xilinx for their generous hardware donations. This work was supported, in part, by a Google Faculty Research Award.

7. REFERENCES

[1] M. Adelman. Principle Engineer, Violin Memory. Personal communication.

[2] http://www.beecube.com/platform.html.

[3] B. Callaghan, T. Lingutla-Raj, A. Chiu, P. Staubach, and O. Asad. Nfs over rdma. In *Proceedings of the ACM SIGCOMM workshop on Network-I/O convergence: experience, lessons, implications*, NICELI '03, pages 196–208, New York, NY, USA, 2003. ACM.

[4] A. M. Caulfield, A. De, J. Coburn, T. I. Mollov, 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*, MICRO '43, pages 385–395, Washington, DC, USA, 2010. IEEE Computer Society.

[5] A. M. Caulfield, T. I. Mollov, L. Eisner, A. De, J. Coburn, and S. Swanson. Providing Safe, User Space Access to Fast, Solid State Disks. In *Proceeding of the 17th international conference on Architectural support for programming languages and operating systems*, New York, NY, USA, March 2012. ACM.

[6] S. Chu. Memcachedb. http://memcachedb.org/.

[7] Cisco. Lossless 10 gigabit ethernet: The unifying infrastructure for san and lan consolidation, 2009.

[8] J. Coburn, T. Bunker, R. K. Gupta, and S. Swanson. From ARIES to MARS: Reengineering Transaction Management for Next-Generation, Solid-State Drives. Technical Report CS2012-0981, Department of Computer Science & Engineering, University of California, San Diego, June 2012. http://csetechrep.ucsd.edu/Dienst/UI/2.0/Describe/ncstrl.ucsd_cse/CS2012-0981.

[9] B. G. Fitch, A. Rayshubskiy, M. C. Pitman, T. J. C. Ward, and R. S. Germain. Using the Active Storage Fabrics Model to Address Petascale Storage Challenges. In *Proceedings of the 4th Annual Workshop on Petascale Data Storage*, PDSW '09, pages 47–54, New York, NY, USA, 2009. ACM.

[10] http://www.fusionio.com/.

[11] iodrive2 data sheet. http://www.fusionio.com/data-sheets/iodrive2/.

[12] http://sourceware.org/cluster/gfs/.

[13] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, 2003.

[14] G. A. Gibson, D. F. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka. A cost-effective, high-bandwidth storage architecture. In *Proceedings of the eighth international conference on Architectural support for programming languages and operating systems*, ASPLOS-VIII, pages 92–103, New York, NY, USA, 1998. ACM.

[15] D. Hildebrand and P. Honeyman. Exporting Storage Systems in a Scalable Manner with pNFS. In *Symposium on Mass Storage Systems*, pages 18–27, 2005.

[16] S. Hopkins and B. Coile. Aoe (ata over ethernet), 2009. http://support.coraid.com/documents/AoEr11.txt.

[17] http://www.intel.com/content/www/us/en/solid-state-drives/ssd-910-series-specification.html.

[18] P. Koutoupis. The lustre distributed filesystem. *Linux J.*, 2011(210), Oct. 2011.

[19] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting phase change memory as a scalable dram alternative. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pages 2–13, New York, NY, USA, 2009. ACM.

[20] Memcached. http://memcached.org/.

[21] https://oss.oracle.com/projects/ocfs2/.

[22] D. Ongaro, S. M. Rumble, R. Stutsman, J. Ousterhout, and M. Rosenblum. Fast crash recovery in ramcloud. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 29–41, New York, NY, USA, 2011. ACM.

[23] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali. Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling. In *MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 14–23, New York, NY, USA, 2009. ACM.

[24] A. Rasmussen, V. T. Lam, M. Conley, G. Porter, R. Kapoor, and A. Vahdat. Themis: an i/o-efficient mapreduce. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 13:1–13:14, New York, NY, USA, 2012. ACM.

[25] A. Rasmussen, G. Porter, M. Conley, H. V. Madhyastha, R. N. Mysore, A. Pucher, and A. Vahdat. Tritonsort: a balanced large-scale sorting system. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI'11, pages 3–3, Berkeley, CA, USA, 2011. USENIX Association.

[26] F. B. Schmuck and R. L. Haskin. GPFS: A Shared-Disk File System for Large Computing Clusters. In *USENIX Conference on File and Storage Technologies*, pages 231–244, 2002.

[27] Cxfs. http://www.sgi.com/products/storage/software/cxfs.html.

[28] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop Distributed File System. In *Symposium on Mass Storage Systems*, 2010.

[29] S. R. Soltis, G. M. Erickson, K. W. Preslan, M. T. O'Keefe, and T. M. Ruwart. The Global File System: A File System for Shared Disk Storage. *IEEE Transactions on Parallel and Distributed Systems*, 1997.

[30] Violin memory 6000 series flash memory arrays. http://www.violin-memory.com/products/6000-flash-memory-array/.

[31] http://www.symantec.com/cluster-file-system.

[32] J. Yang, D. B. Minturn, and F. Hady. When poll is better than interrupt. In *in proceedings of the 10th USENIX Conference on File and Storage Technologies*, February 2012.

[33] W. Yu, S. Liang, and D. K. Panda. High performance support of parallel virtual file system (pvfs2) over quadrics. In *Proceedings of the 19th annual international conference on Supercomputing*, ICS '05, pages 323–331, New York, NY, USA, 2005. ACM.