BlueDBM: Distributed Flash Storage for Big Data Analytics

SANG-WOO JUN, MING LIU, and SUNGJIN LEE, Massachusetts Institute of Technology
JAMEY HICKS, JOHN ANKCORN, and MYRON KING, Quanta Research Cambridge
SHUOTAO XU and ARVIND, Massachusetts Institute of Technology

Complex data queries, because of their need for random accesses, have proven to be slow unless all the data can be accommodated in DRAM. There are many domains, such as genomics, geological data, and daily Twitter feeds, where the datasets of interest are 5TB to 20TB. For such a dataset, one would need a cluster with 100 servers, each with 128GB to 256GB of DRAM, to accommodate all the data in DRAM. On the other hand, such datasets could be stored easily in the flash memory of a rack-sized cluster. Flash storage has much better random access performance than hard disks, which makes it desirable for analytics workloads. However, currently available off-the-shelf flash storage packaged as SSDs does not make effective use of flash storage because it incurs a great amount of additional overhead during flash device management and network access. In this article, we present BlueDBM, a new system architecture that has flash-based storage with in-store processing capability and a low-latency high-throughput intercontroller network between storage devices. We show that BlueDBM outperforms a flash-based system without these features by a factor of 10 for some important applications. While the performance of a DRAM-centric system falls sharply even if only 5% to 10% of the references are to secondary storage, this sharp performance degradation is not an issue in BlueDBM. BlueDBM presents an attractive point in the cost/performance tradeoff for Big Data analytics.

CCS Concepts: • **Computer systems organization** → **Cloud computing;** Reconfigurable computing; • **Hardware** → *Memory and dense storage*; • **Networks** → *Storage area networks;*

Additional Key Words and Phrases: Wireless sensor networks, media access control, multichannel, radio interference, time synchronization

ACM Reference Format:
Sang-Woo Jun, Ming Liu, Sungjin Lee, Jamey Hicks, John Ankcorn, Myron King, Shuotao Xu, and Arvind. 2016. BlueDBM: Distributed flash storage for big data analytics. ACM Trans. Comput. Syst. 34, 3, Article 7 (June 2016), 31 pages.
DOI: http://dx.doi.org/10.1145/2898996

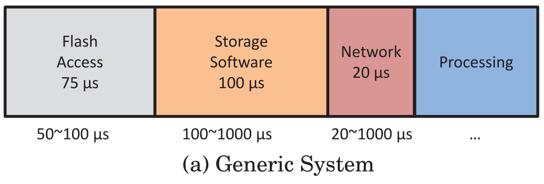
1. INTRODUCTION

By many accounts, complex analysis of Big Data is going to be the biggest economic driver for the IT industry. For example, Google has predicted flu outbreaks by analyzing social network information a week faster than the CDC [Google 2014], analysis of

This work is based on a prior publication in ISCA 2015.

Authors' addresses: S.-W. Jun, 32-G836, Stata Center, MIT, 32 Vassar Street, Cambridge, MA, USA; email: wjun@csail.mit.edu; M. Liu, 32-G836, Stata Center, MIT, 32 Vassar Street, Cambridge, MA, USA; email: ml@csail.mit.edu; S. Lee, Room 1010, High-Tech Building, Inha University, 100 inharo, Nam-gu, Incheon, Republic of Korea; email: sungjin.lee@inha.ac.kr; J. Hicks, Accelerated Tech, Inc., Cambridge, MA, USA; email: jamey.hicks@accelerated.tech; J. Ankcorn, 32-G870, Stata Center, MIT, 32 Vassar Street, Cambridge, MA, USA; email: jca@alum.mit.edu; M. King, 38 Ashland St., Arlington MA 02476; email: mdk@alum.mit.edu; S. Xu, 32-G836, Stata Center, MIT, 32 Vassar Street, Cambridge, MA, USA; email: shuotao@csail.mit.edu; Arvind, 32-G866, Stata Center, MIT, 32 Vassar Street, Cambridge, MA, USA; email: arvind@csail.mit.edu.

DOI: http://dx.doi.org/10.1145/2898996



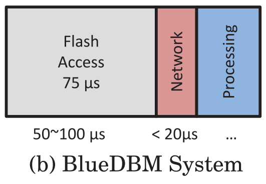


Fig. 1. Latency breakdown of distributed flash access.

Twitter data can reveal social upheavals faster than journalists, Amazon is planning to use customer data for anticipatory shipping of products [Spiegel et al. 2011], and real-time analysis of a personal genome may significantly aid in diagnostics. Big Data analytics are potentially going to have a revolutionary impact on the way scientific discoveries are made.

Big Data, by definition, doesn't fit in personal computers or DRAM of even moderate-size clusters. Since the data may be stored on hard disks, latency and throughput of storage access are of primary concern. Historically, this has been mitigated by organizing the processing of data in a highly sequential manner. However, complex queries cannot always be organized for sequential data accesses, and thus high-performance implementations of such queries pose a great challenge. One approach to solving this problem is *RAMCloud* [Ousterhout et al. 2010], where the cluster has enough collective DRAM to accommodate the entire dataset in DRAM. In this article, we explore a much cheaper alternative where Big Data analytics can be done with reasonable efficiency in a single rack with distributed flash storage, which has much better random access performance than hard disks. We call our system BlueDBM and it provides the following capabilities:

(1) A 20-node system with large enough flash storage to host Big Data workloads up to 20 TBs
(2) Uniformly low-latency access into a network of storage devices that form a global address space
(3) Capacity to implement user-defined in-store processing engines
(4) Flash card design, which exposes an interface to make application-specific optimizations in flash accesses

The BlueDBM architecture includes three major architectural modifications: (1) an in-store processing engine, (2) low-latency storage area network integrated into the storage controller, and (3) file system and flash management software optimized across multiple hardware and software layers. Such a system improves system characteristics in the following ways:

Latency: BlueDBM achieves extremely low-latency access into distributed flash storage. Network software stack overhead is removed by integrating the storage area network into the storage device, and network latency is reduced using a faster network fabric. It reduces the storage access software latency with cross-layer optimizations or removes it completely via an in-storage processing engine. It reduces processing time using application-specific accelerators. The latency improvements are depicted in Figure 1.

Bandwidth: Bandwidth of storage access is increased. Flash chips are organized into many buses to improve parallelism, and a fast PCIe link is used instead of the slower SATA interface. While it is often challenging to write software that is capable of consuming data from flash storage at multigigabytes per second, an application-specific accelerator in the storage can consume data at device speed, instead of being bound by the software performance or the performance of the PCIe link to the host.

Power: Power consumption of the overall system is reduced. An in-store processor sometimes removes the need for data to be moved to the host, reducing power consumption related to data movement. Flash storage consumes far less power compared to DRAM of comparable capacity. Application-specific accelerators are also more power efficient than a general-purpose CPU or GPU.

Cost: Cost of the overall system is reduced. The engineering cost of constructing a separate accelerator is reduced by integrating the accelerator into the storage device. The cost of flash storage is also much lower than DRAM of comparable capacity.

Our preliminary experimental results show that for some applications, BlueDBM performance is an order of magnitude better than a conventional cluster where SSDs are used only as a disk replacement. BlueDBM establishes an architecture whose price-performance-power characteristics provide an attractive alternative for doing similar-scale applications in a RAMCloud.

As we will discuss in the related work section, almost every element of our system is present in some commercial system. Yet our system architecture as a whole is unique. The main contributions of this work are as follows: (1) design and implementation of a scalable flash-based system with a global address space, in-store computing capability, and a flexible intercontroller network; (2) a hardware-software codesign environment for incorporating user-defined in-store processing engines; (3) performance measurements that show the advantage of such an architecture over using flash as a drop-in replacement for disks; and (4) demonstration of a complex data analytics appliance that is much cheaper and consumes an order of magnitude less power than the cloud-based alternative.

The rest of the article is organized as follows: In Section 2, we explore some existing research related to our system. In Section 3, we describe the architecture of our rack-level system, and in Section 4, we describe the software interface that can be used to access flash and the accelerators. In Section 5, we describe a hardware implementation of BlueDBM and show our results from the implementation in Section 6. In Section 7, we describe and evaluate some example accelerators we have built for the BlueDBM system. Section 8 summarizes our article.

2. RELATED WORK

In Big-Data-scale workloads, building a cluster with enough DRAM capacity to accommodate the entire dataset can be desirable but expensive. An example of such a system is RAMCloud, which is a DRAM-based storage for large-scale data center applications [Ousterhout et al. 2010; Rumble et al. 2014]. RAMCloud provides more than 64TB of DRAM storage distributed across over 1,000 servers networked over high-speed interconnect. Although RAMCloud provides 100 to 1,000 times better performance than disk-based systems of similar scale, its high energy consumption and high price per GB limit its widespread use except for extremely performance- and latency-sensitive workloads. Furthermore, the overhead of a widely distributed processing platform becomes high quickly, making it difficult to make full use of the total computational power of the cluster [McSherry et al. 2015]. Even with a scalable processing platform, the overhead may be so high that running a less scalable software on fewer machines might sometimes be a faster solution.

NAND-Flash-based SSD devices are gaining traction as a faster alternative to disks, and close the performance gap between DRAM and persistent storage. SSDs are an order of magnitude cheaper than DRAM and an order of magnitude faster than disk. Many existing database and analytics softwares have shown improved performance with SSDs [Dai 2010; Kang et al. 2013b; Lee et al. 2008]. Several SSD-optimized analytics softwares, such as the SanDisk Zetascale [SanDisk 2014], have demonstrated promising performance while using SSD as the primary data storage. Many commercial

SSD devices have adopted a high-performance PCIe interface in order to overcome the slower SATA bus interface designed for disk [FusionIO 2014a; Memory 2014; Intel 2014]. Attempts to use flash as a persistent DRAM alternative by plugging it into a RAM slot are also being explored [Technologies 2014].

SSD storage devices have been largely developed to be a faster drop-in replacement for disk drives. This backward compatibility has helped their widespread adoption. However, additional software and hardware is required to hide the difference in device characteristics [Agrawal et al. 2008]. Due to the high performance of SSDs, even inefficiencies in the storage management software becomes significant, and optimizing such software has been under active investigation. Moneta [Caulfield et al. 2010] modifies the operating system's storage management components to reduce software overhead when accessing NVM storage devices. Willow [Seshadri et al. 2014] provides an easy way to augment SSD controllers with additional interface semantics that make better use of SSD characteristics, in addition to a backward-compatible storage interface. Attempts to remove the translation layers and letting the database make high-level decisions [Hardock et al. 2013] have also shown to be beneficial. Work such as nameless writes [Arpaci-Dusseau et al. 2010] and application-managed flash [Lee et al. 2016] move the translation layer out of the storage device and into the file system, which reduces the storage device complexity and also improves performance.

Due to their high performance, SSDs also affect network requirements. The latency to access disk over Ethernet was dominated by the disk seek latency. However, in an SSD-based cluster, the storage access latency could even be lower than network access. These concerns are being addressed by faster network fabrics such as 10GbE and Infiniband [Association 2014], and by low-overhead software protocols such as RDMA [Liu et al. 2003; Islam et al. 2012; Rahman et al. 2014; Woodall et al. 2006; Liu et al. 2003; Rahman et al. 2013] or user-level TCP stacks that bypass the operating system [Jeong et al. 2014; Honda et al. 2014]. QuickSAN [Caulfield and Swanson 2013] is an attempt to remove a layer of software overhead by augmenting the storage device with a low-latency NIC, so that remote storage access does not need to go through a separate network software stack. Works such as DFS [Josephson et al. 2010] achieve high performance in a distributed flash storage setting by virtualizing distributed flash storage into a single large address space, allowing a much simpler distributed file system implementation.

Another important attempt to accelerate SSD storage performance is in-store processing, where some data analytics are offloaded to embedded processors inside SSDs. These processors have extremely low-latency access to storage and helped overcome the limitations of the storage interface bus. The idea of in-store processing itself is not new. Intelligent disks (IDISKs) connected to each other using serial networks [Keeton et al. 1998] and Active Disks [Acharya et al. 1998] with the capability to run application-specific programs on disk drives have been proposed in 1998, and adding processors to disk heads to do simple filters was suggested as early as in the 1970s [Leilich et al. 1978; Ozkarahan et al. 1975; Banerjee et al. 1979]. However, performance improvements of such special-purpose hardware did not justify their cost at the time.

In-store processing is seeing new light with the advancement of fast flash technology. Devices such as Smart SSDs [Do et al. 2013; Kang et al. 2013a; Seshadri et al. 2014] and Programmable SSDs [Cho et al. 2013] have shown promising results, but gains are often limited by the performance of the embedded processors in such power-constrained devices. Embedding reconfigurable hardware in storage devices is being investigated as well. For example, Ibex [Woods et al. 2014] is a MySQL accelerator platform where a SATA SSD is coupled with an FPGA. Relational operators such as selection and group-by are performed on the FPGA whenever possible; otherwise, they are forwarded to software. Companies such as IBM/Netezza [Singh and Leonhardi 2011] offload operations such as filtering to a reconfigurable fabric near storage. On the other end of the spectrum, systems such as XSD [Cho et al. 2013] embeds a GPU into an SSD controller and demonstrates high-performance accelerating MapReduce.

Building specialized hardware for databases has been extensively studied and productized. Companies such as Oracle [Oracle 2014] have used FPGAs to offload database queries. FPGAs have been used to accelerate operations such as hash index lookups [Kocberber et al. 2013]. Domain-specific processors for database queries are being developed [Sukhwani et al. 2012; Woods et al. 2013], including Q100 [Wu et al. 2014] and LINQits [Chung et al. 2013]. Q100 is a data-flow-style processor with an instruction set architecture that supports SQL queries. LINQits mapped a query language called LINQ to a set of accelerated hardware templates on a heterogeneous SoC (FPGA + ARM). Both designs exhibited order of magnitude performance gains at lower power, affirming that specialized hardware for data processing is advantageous. However, unlike BlueDBM, these architectures accelerate computation on data that is in DRAM. Accelerators have also been placed in-path between the network and processor to perform operations at wire speed [Mueller et al. 2009] or to collect information such as histogram tables without overhead [István et al. 2014].

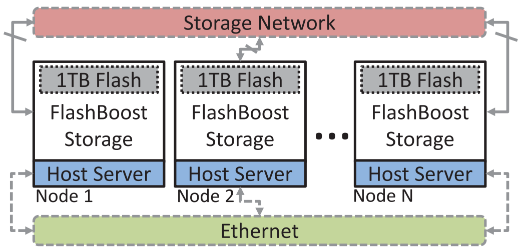
Incorporating reconfigurable hardware accelerators into large data centers is also being investigated actively. Microsoft recently has built and demonstrated the power/performance benefits of an FPGA-based system called Catapult [Putnam et al. 2014]. Catapult uses a large number of homogeneous servers, each augmented with an FPGA. The FPGAs form a network among themselves via high-speed serial links so that large jobs can be mapped to groups of FPGAs. Catapult was demonstrated to deliver much faster performance while consuming less power, compared to a normal DRAM-centric cluster. BlueDBM has similar goals in terms of reconfigurable hardware acceleration, but it uses flash devices to accelerate lower-cost systems that do not have enough collective DRAM to host the entire dataset.

This system improves upon our previous BlueDBM prototype [Jun et al. 2014], which was a four-node system with less than 100GB of slow flash. It was difficult to extrapolate the performance of real applications from the results obtained from our previous prototype, because of both its size and different relative performance of various system components. The current generation of BlueDBM has been built with the explicit goal of running real applications and is freely available to the community for developing Big Data applications.

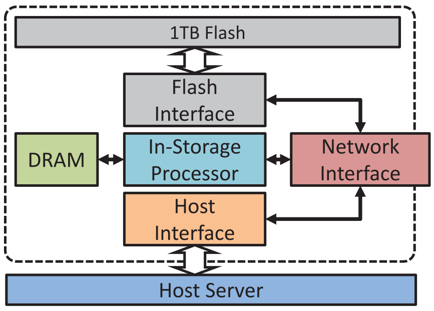
3. SYSTEM ARCHITECTURE

The BlueDBM architecture is a homogeneous cluster of host servers coupled with a BlueDBM storage device (see Figure 2(a)). Each BlueDBM storage device is plugged into the host server via a PCIe link, and it consists of flash storage, an in-store processing engine, multiple high-speed network interfaces, and on-board DRAM. The host servers are networked together using Ethernet or other general-purpose networking fabric. The host server can access the BlueDBM storage device via a host interface implemented over PCIe. It can directly communicate either with the flash interface, to treat is as a raw storage device, or with the in-store processor, to perform computation on the data. When the host wants to access remote storage, instead of requesting data from a remote host, it can directly request data from the remote storage device over the integrated storage network. This removes multiple passes of the network and storage software stacks, improving performance greatly.

The in-store processing engine has access to four major services: the flash interface, network interface, host interface, and on-storage DRAM buffer. Figures 2(a) and 2(b) show the four services available to the in-store processor. In the following sections, we



(a) BlueDBM overall architecture



(b) BlueDBM node architecture

Fig. 2. BlueDBM architecture.

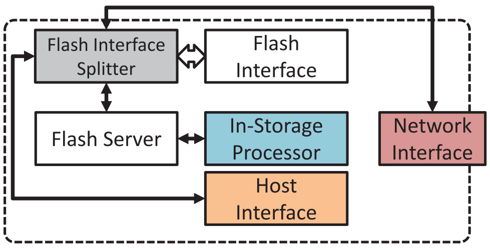


Fig. 3. BlueDBM flash controller organization.

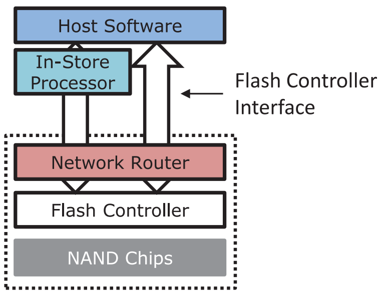
describe the flash interface, network interface, and host interface in order. We omit the DRAM buffer because there is nothing special about its design.

3.1. Flash Controller

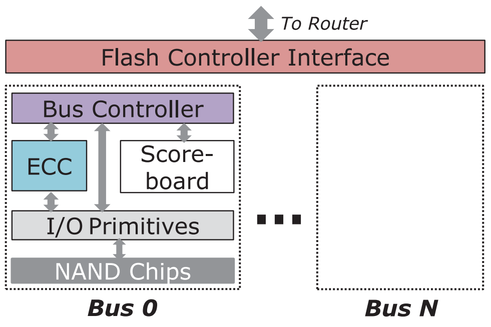
Flash devices or SSDs achieve high bandwidth by grouping multiple flash chips into several channels, all of which can operate in parallel. Because NAND flash has limited program/erase cycles and frequent errors, complex flash management algorithms are required to guarantee reliability. These include wear leveling, garbage collection, bit error correction and bad block management. These functions are typically handled by multiple ARM-based cores in the SSD controller. The host side interface of an SSD is typically SATA or PCIe, using AHCI or NVMe protocols to communicate with the host. SSDs are viewed as a typical block device to the host operating system, and its internal architecture and management algorithms are completely hidden.

However, this additional layer of management duplicates some file system functions and adds significant latency [Lee et al. 2015]. Furthermore, in a distributed storage environment, such as BlueDBM, independent flash devices do not have a holistic view of the system and thus cannot efficiently manage flash. Finally, in-store processors that we have introduced in BlueDBM would also incur performance penalties if passing through this extra layer. Thus, in BlueDBM, we chose to shift flash management away from the device and into the file system/block device driver (discussed in Section 4).

3.1.1. Interface for High-Performance Flash Access. Figure 3 describes the architecture of the flash access interface. A host software or in-store processor can access flash through a special flash controller interface designed for highly optimized performance. Figure 4(a) describes the layers involved in flash access. The software or in-store processor has a very low-latency access into remote flash storage by using the controller network.



(a) Flash access stack



(b) Flash controller internal architecture

Fig. 4. BlueDBM flash architecture.

If the flash command includes a node index, the controller network can transparently route the commands and data to and from the remote node.

Our flash controller exposes a low-level, thin, fast, and bit-error-corrected hardware interface to raw NAND flash chips, buses, blocks, and pages. This has the benefit of (1) cutting down on access latency from the network and in-store processors, (2) exposing all degrees of parallelism of the device, and (3) allowing higher-level system stacks (file system, database storage engine) to more intelligently manage data. The flash controller exposes the following commands:

(1) *ReadPage(tag, node, bus, chip, block, page):* Reads a flash page.
(2) *WritePage(tag, node, bus, chip, block, page):* Writes a flash page. Pages must be erased before being written, and writes within a block must be sequential.
(3) *EraseBlock(tag, node, bus, chip, block):* Erases a flash block. It returns an error status if the block is bad.

Some basic functionalities are kept inside the flash controller, namely, bus/chip-level scheduling and ECC. The internal architecture of the flash controller is described in Figure 4(b). Higher-level flash management functions such as garbage collection and wear leveling are implemented in the higher layers, usually the device driver or the host software.

To access the flash, the user first issues a flash command with the operation, the address, and a tag to identify the request. For writes, the user then awaits for a write data request from the controller scheduler, which tells the user that the flash controller is ready to receive the data for that write. The user will send the write data corresponding to that request in 128-bit bursts. The controller returns an acknowledgment once the write is finished. For read operations, the data is returned in 128-bit bursts along with the request tag. For maximum performance, the controller may send these data bursts *out of order* with respect to the issued request and *interleaved* with other read requests. Thus, completion buffers may be required on the user side to maintain FIFO characteristics. Furthermore, we note that to saturate the bandwidth of the flash device, multiple commands must be in-flight at the same time, since flash operations can have latencies of 50μs or more.

Flash requests coming into the controller are distributed by bus address, and the bus controller uses a scoreboard to schedule parallel operations onto the flash chips for maximum bandwidth. The scheduler works in a priority round-robin fashion, rotating to select the first request that has the highest priority among all the chips, and enqueues it for execution. We prioritize short command/address bursts on the bus over long data transfers, and older requests over newer ones. For ECC, we use RS(255, 243)

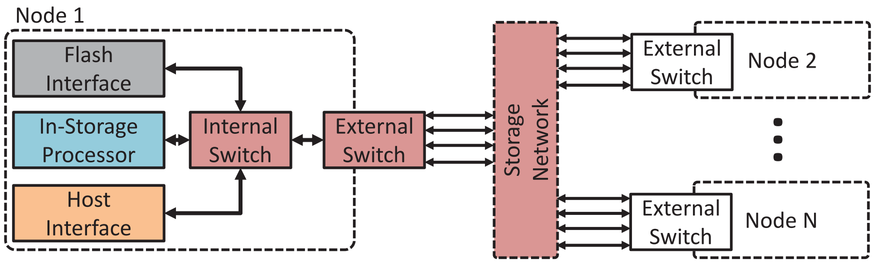


Fig. 5. BlueDBM network architecture.

Reed-Solomon codes for simplicity. The Reed-Solomon decoder could cause variable-time stalls on the bus. Therefore, one commonly uses BCH or LDPC codes on a production system.

3.1.2. Multiple Access Agents. Multiple hardware endpoints in BlueDBM may need shared access to this flash controller interface. For example, a particular controller may be accessed by local in-store processors, local host software over PCIe DMA, or remote in-store processors over the network. Thus, we implemented a Flash Interface Splitter with tag renaming to manage multiple users (Figure 3). In addition, to ease development of a hardware accelerator, we also provide an optional Flash Server module as part of BlueDBM. This server converts the out-of-order and interleaved flash interface into multiple simple in-order request/response interfaces using page buffers. It also contains an Address Translation Unit that maps file handles to incoming streams of physical addresses from the host. The in-store processor simply makes a request with the file handle, offset, and length, and the Flash Server will perform the flash operation at the corresponding physical location. The software support for this function is discussed in Section 4. The Flash Server's width, command queue depth, and number of interfaces is adjustable based on the application.

3.2. Storage Controller Network

BlueDBM provides a low-latency high-bandwidth transport layer network infrastructure across all BlueDBM storage devices in the cluster, using a simple design with low buffer requirements. BlueDBM storage devices form a separate network among themselves via high-performance serial links. The BlueDBM network is a packet-switched mesh network, in which each storage device has multiple network ports and is capable of routing packets across the network without requiring a separate switch or router. In addition to routing, the storage network supports transport layer functionality such as flow control and virtual channels while maintaining high performance and extremely low latency. For data traffic between the storage devices, the integrated network ports remove the overhead of going to the host software to access a separate network interface.

Figure 5 shows the network architecture. Switching is done at two levels, the internal switch and the external switch. The internal switch routes packets between local components. The external switch accesses multiple physical network ports and is responsible for forwarding data from one port to another in order to relay a packet to its next hop. It is also responsible for relaying inbound packets to the internal switch and relaying outbound packets from the internal switch to a correct physical port.

Due to the multiple ports on the storage nodes, the BlueDBM network is flexible and can be configured to implement various topologies, as long as there is a sufficient number of ports on each node. Figure 6 shows some example topologies. To implement

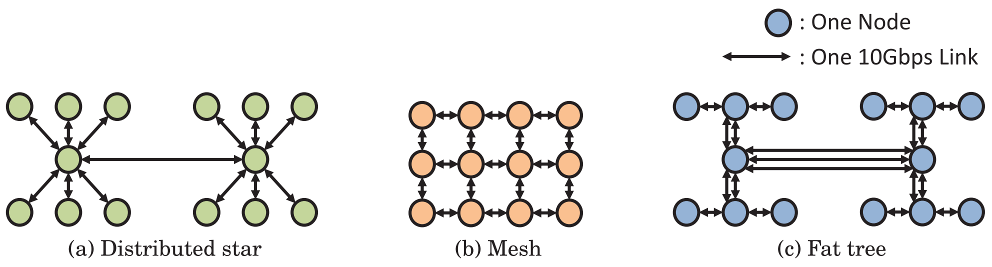


Fig. 6. Any network topology is possible as long as there are sufficient network ports.

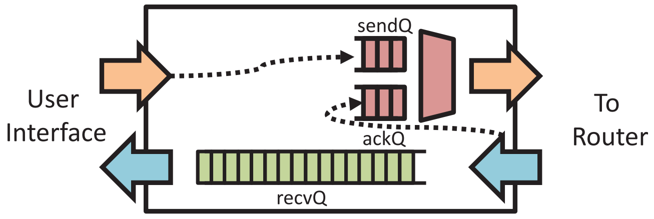


Fig. 7. Logical endpoint architecture.

a different topology, the physical cables between each node have to be rewired, but the routing across a topology can be configured dynamically by the software.

3.2.1. Transport Layer. The BlueDBM network infrastructure exposes transport layer functionality including virtual channel semantics to the users by providing it with up to hundreds of *logical endpoints*. The number of endpoints are determined at design time by setting a parameter, and all endpoints share the physical network. Each endpoint is parameterized with a unique index identifier that does not need to be contiguous. Figure 7 shows the internal architecture of an endpoint.

Whenever a packet is received by an endpoint, it checks a table of packets received per source node to determine if it is time to send a flow control credit to the source node. If the send budget of the source node is predicted to have become small enough and there is enough space on the local receive buffer, it enters a packet into the ack queue and marks the amount of space as allocated. The size of the receive buffer and flow control credits is parameterized for individual endpoints, so that precious on-chip buffer resources can be conserved while meeting performance requirements of individual endpoints. For example, high-throughput-data endpoints can have large buffers and large flow control credits, while lower-throughput endpoints such as some command or status paths can have smaller buffers with smaller flow control credits.

Each endpoint exposes two interfaces, `send` and `receive`. An in-store processor can send data to a remote node by calling `send` with a pair of data and destination node index, or receive data from remote nodes by calling `receive`, which returns a pair of data and source node index. These interfaces provide back pressure, so that each endpoint can be treated like a FIFO interface across the whole cluster. Such intuitive characteristics of the network ease development of in-store processors.

In order to maintain extremely low network latency, each endpoint is given a choice whether to use end-to-end flow control. If the developer is sure that a particular virtual link will always drain on the receiving end, flow end-to-end flow control can be omitted for that endpoint. However, if the receiver fails to drain data for a long time, the link-level back-pressure may cause related parts of the network to block. On the other hand, an endpoint can be configured with end-to-end flow control, which will only send

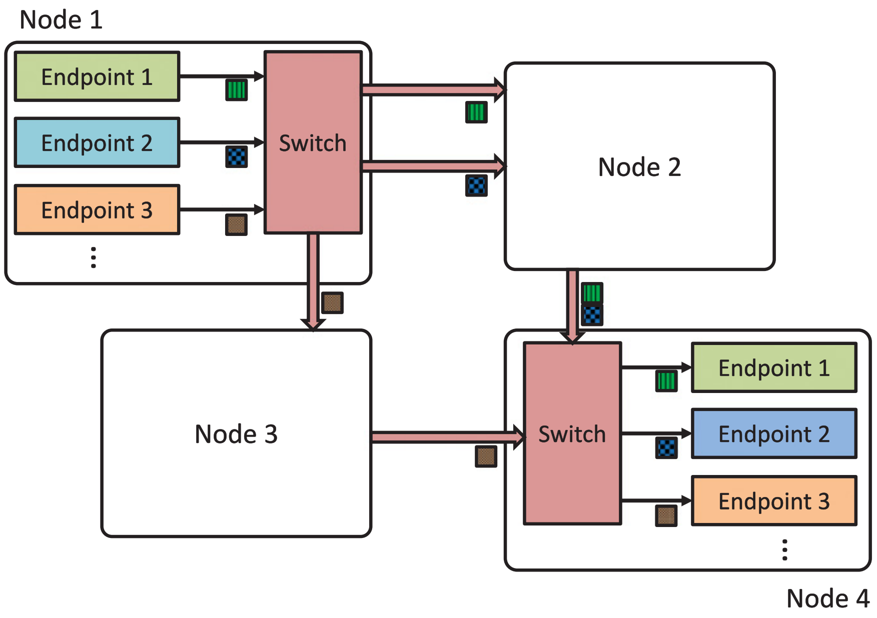


Fig. 8. Packets from the same endpoint to a destination maintain FIFO order.

data when there is space on the destination endpoint. This will ensure safety but may result in higher latency due to flow control packets, unless a big enough buffer space is allocated to hide it.

3.2.2. Network Layer. In order to make maximum use of the bandwidth of the network infrastructure while keeping resource usage to a minimum, the BlueDBM network implements deterministic routing for each logical endpoint. This means that all packets originating from the same logical endpoint that are directed to the same destination node follow the same route across the network, while packets from a different endpoint directed to the same destination node may follow a different path. Figure 8 shows packet routing in an example network. The benefit of this approach is that packet traffic can be distributed across multiple links while maintaining the order of all packets from the same endpoint. If packets from the same endpoint are allowed to take different paths, it would require a completion buffer, which may be expensive in an embedded system. For simplicity, the BlueDBM network does not implement a discovery protocol and relies on a network configuration file to populate the routing tables.

3.2.3. Link Layer. The link layer manages physical connections between network ports in the storage nodes. The most important aspect of the link layer is the simple token-based flow control implementation. This provides back-pressure across the link and ensures that packets will not drop if the data rate is higher than what the network can manage, or if the data cannot be received by the destination node that is running slowly.

3.3. Host Interface

The in-store processing core can be accessed from the host server over either a direct interface that supports RPC and DMA operations or a file system abstraction built on top of the direct interface. The file system interface is described in detail in Section 4.

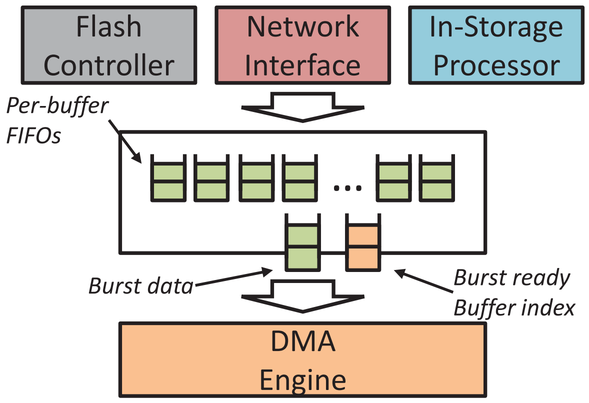


Fig. 9. Host-FPGA interface over PCIe.

These interfaces also include a field for a target node index, so a host can also access the processing core of a remote device using the same interface.

In order to parallelize requests and maintain high performance, the host interface provides the software with 128 page buffers, each for reads and writes. When writing a page, the software will request a free write buffer, copy data to the write buffer, and send a write request over RPC with the physical address of the destination flash page. The buffer will be returned to the free queue when the hardware has finished reading the data from the buffer. When reading a page, the software will request a free read buffer and send a read request over RPC with the physical address of the source flash page. The software will receive an interrupt with the buffer index when the hardware has finished writing to software memory.

Using DMA to write data to the storage device is straightforward to parallelize, but parallelizing reads is a bit more tricky due to the characteristics of flash storage. When writing to storage, the DMA engine on the hardware will read data from each buffer in order in a contiguous stream. So having enough requests in the request queue is enough to make maximum use of the host-side link bandwidth. However, data reads from flash chips on multiple buses in parallel can arrive interleaved at the DMA engine. Because the DMA engine needs to have enough contiguous data for a DMA burst before issuing a DMA burst, some reordering may be required at the DMA engine. This becomes even trickier when the device is using the integrated network to receive data from remote nodes, where they might all be coming from different buses. To fix this issue, we provide a dual-ported buffer in hardware, which has the semantics of a vector of FIFOs, so that data for each request can be enqueued into its own FIFO until there is enough data for a burst. Figure 9 describes the structure of the host interface for flash reads.

4. SYSTEM SOFTWARE ORGANIZATION

BlueDBM provides a set of software interfaces that aims to achieve two goals: (1) conveniently use existing software without modification and (2) provide a general framework for implementing application-specific in-store processors. In order to use existing software without modification, BlueDBM provides a generic block device driver interface in its storage and also a file system interface. In order to provide a general framework for in-store processors, BlueDBM uses Connectal, a hardware-software codesign framework to simplify the development of hardware accelerators.

When the file system is being used, the file system is in charge of the mappings of each file into the storage device, and the file system provides the in-store processor

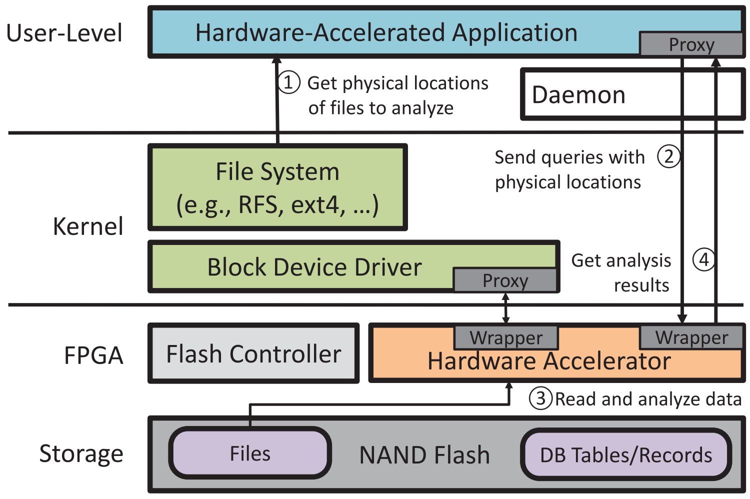


Fig. 10. System software organization.

the locations of each page of a file. However, for in-store processors to achieve high performance, they sometimes need to have direct access to the flash storage instead of having to consult the file system for every page location. For this purpose, a developer can use the Connectal framework to use custom software interfaces for the in-store processor. The software architecture of BlueDBM can be seen in Figure 10.

4.1. Flash-Aware File System

Instead of using a transparent Flash Translation Layer (FTL) to provide a disk-like interface to the file system, BlueDBM moves many of the FTL functionalities to a flash-aware file system, allowing the file system to use high-level information and make intelligent decisions. Commercial SSDs incorporate an FTL inside the flash device controller to manage flash and maintain a block-device view to the operating system. However, common file systems manage blocks in a fashion optimized for hard disks. SSDs use the FTL to emulate block device interfaces for compliance with operating systems, performing logical-to-physical mapping and garbage collection, which require large DRAM and incur many extra I/Os. Some file systems have tried to remedy this by refactoring the I/O architecture and offloading most of the FTL functions into a flash-optimized log-structured file system. A prominent example of this is RFS [Lee et al. 2015]. Unlike conventional FTL designs where the flash characteristics are hidden from the file system, RFS performs some functionalities of an FTL, including logical-to-physical address mapping and garbage collection. This achieves better garbage collection efficiency at a much lower memory requirement. The file system interface in BlueDBM is built on the same paradigm.

For compatibility with existing software, BlueDBM also offers a full-fledged FTL implemented in the device driver, similar to FusionIO's device driver. This allows us to use well-known Linux file systems (e.g., ext2/3/4) as well as database systems (directly running on top of a block device) with BlueDBM.

The BlueDBM software allows developers to easily make use of fast in-storage processing without any efforts to write their own custom interfaces manually. Figure 10 shows how user-level applications access hardware accelerators. In the BlueDBM software stack, user-level applications can query the file system for the physical locations of files on the flash (see (1) in Figure 10). This was made possible because the file

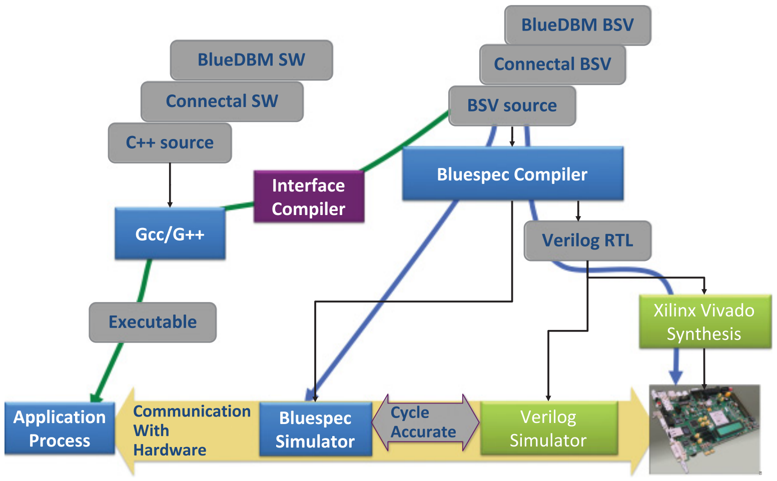


Fig. 11. BlueDBM software design flow.

system maintains the flash mapping information. Applications can then provide in-storage processors with a stream of physical addresses (see (2)), so that the in-storage processors can directly read data from flash with low latency (see (3)). The results are sent to software memory and the user application can be notified (see (4)).

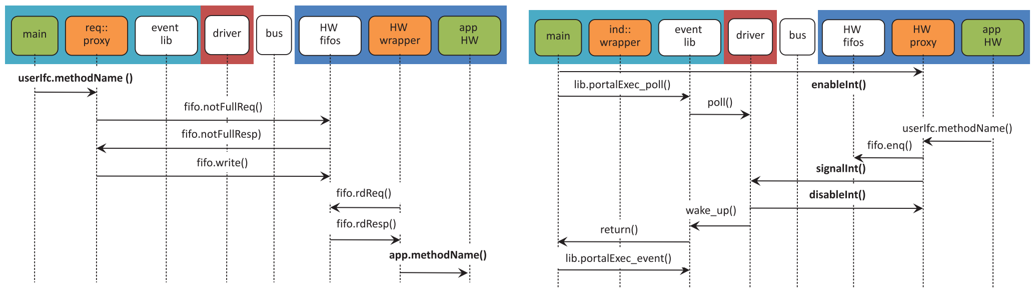
It is worth noting that, in BlueDBM, all the user requests, including both user queries and data, are sent to the hardware directly, bypassing almost all of the operating system's kernels, except for essential driver modules. This helps us to avoid deep OS kernel stacks that often cause long I/O latencies. It is also common that multiple instances of a user application may compete for the same hardware acceleration units. For efficient sharing of hardware resources, BlueDBM runs a scheduler that assigns available hardware acceleration units to competing user applications. In our implementation, a simple FIFO-based policy is used for request scheduling.

4.2. Connectal

BlueDBM uses the Connectal [King et al. 2015] framework for accelerator development. Connectal consists of (1) a library of hardware and software components including support for software/hardware communication and shared memory; (2) an interface compiler that generates C, C++, and BSV; (3) a universal device driver that works for all accelerators; and (4) a unified dependency build system for hardware accelerators.

In the Connectal design flow, shown in Figure 11, accelerated applications consist of software and hardware components that communicate via asynchronous remote method invocation. As shown in the figure, BlueDBM applications use Connectal and BlueDBM software libraries and application-specific source code. On the hardware side, accelerators use Connectal and BlueDBM BSV libraries and accelerator-specific BSV.

In BSV, developers declare the methods of a module with *interfaces*, which correspond to Java interfaces or C++ abstract base class declarations. With Connectal, accelerator developers declare interfaces using BSV as an interface definition language. Connectal generates C, C++, and BSV stubs (proxies and wrappers) from these interface declarations. The C and C++ stubs may be used in user-space applications, and the C stubs may also be used in kernel drivers. Automatically generating stubs from a BSV interface specification enables developers to easily iterate their design, knowing that the respective C, C++, and BSV compilers will ensure that both software and hardware



(a) Application software invokes hardware via a generated software proxy, hardware FIFOs, and a generated hardware mapper

(b) Application hardware invokes software via a generated hardware proxy, hardware FIFOs and a generated software mapper

Fig. 12. Hardware software communication in the Connectal hardware-software codesign framework.

are using the interfaces consistently. Connectal provides an option to bundle the FPGA bitstream into a section of the application's executable file, so that the hardware is automatically initialized when the application is invoked.

These proxies and wrappers are connected via transports such as memory-mapped hardware FIFOs, whose implementation is provided as a Connectal BSV library. Accelerated applications map the hardware FIFOs into the process address space, so that software may communicate with its accelerator with no kernel involvement. Figure 12 contains a message sequence chart showing traces of the low-level messages needed for (1) software to invoke hardware methods and (2) hardware to call back to software. The *poll()* system call enables software to wait for an interrupt from hardware.

The same generated proxy and wrapper code may be used for communication between software and simulators and between applications and daemons moderating access to the hardware, using TCP or Unix sockets or shared memory.

In addition to the remote method call support, Connectal also provides hardware access to the host's DRAM via bus master operations. Connectal Bluespec libraries include direct memory access modules along with a memory management unit enabling hardware to reference memory with linear offsets, matching what is used in software without requiring physically contiguous DRAM pages. The memory interface is specialized for the hardware platform, enabling the same sharing of memory between software, simulators, and FPGAs.

Device driver development requires knowledge of kernel APIs, the concurrency model, and the tool chain, and presents another barrier to accelerator development. In addition, if the device driver is specific to the accelerator, then any changes to the accelerator may require changes to the device driver as well as the user-mode software. To avoid these problems, Connectal provides a universal device driver so that accelerator developers do not have to write any device drivers unless their application is in the kernel. The driver maps the hardware FIFOs into memory and enables user-space software to wait for interrupts signaling a message from the hardware. Because the support for mapping hardware addresses to user space and suspending thread execution pending an interrupt is completely generic, this driver can be used unmodified for all developed designs, removing the need for detailed understanding of Linux kernel driver construction.

Connectal automates the coordinated build of hardware, software, and backend parameters via a simple single makefile-based dependency build environment. The dependency build approach speeds up incremental engineering work by only

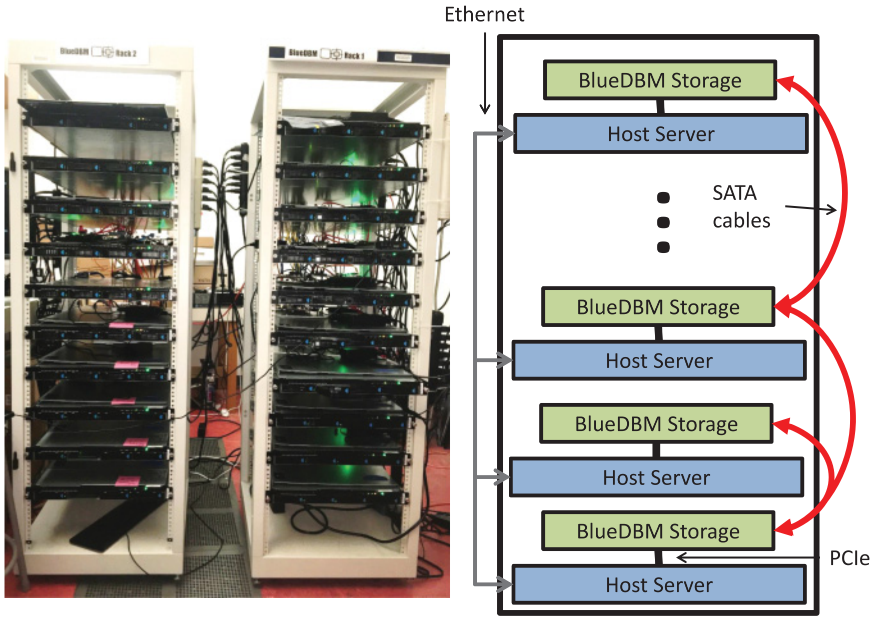


Fig. 13. A 20-node BlueDBM cluster.

regenerating the components that changed in the last revision, leading to a large reduction in elapsed time to validate changes.

Connectal enhances design portability, enabling unmodified hardware designs to run on different simulators and different FPGA boards, and to be built using different back-end workflows by factoring out platform and toolchain dependencies. Connectal supports Xilinx and Altera FPGAs, Linux and Android operating systems, and software written in C and C++.

5. HARDWARE IMPLEMENTATION

We have built a 20-node BlueDBM cluster to explore the capabilities of the architecture. Figure 13 shows the photo of our implementation.

In our implementation of BlueDBM, we have used an FPGA to implement the in-store processor and also the flash, host, and network controllers. However, the BlueDBM architecture should not be limited to an FPGA-based implementation. Development of BlueDBM was done in the high-level hardware description language Bluespec. It is possible to develop in-store processors in any hardware description language, as long as they conform to the interface exposed by the BlueDBM system services. Most of the interfaces are latency-insensitive FIFOs with back-pressure. Bluespec provides a lot of support for such interfaces, making in-store accelerator development easier.

The cluster consists of 20 rack-mounted Xeon servers, each with 24 cores and 50GB of DRAM. Each server also has a Xilinx VC707 FPGA development board connected via a PCIe connection. Each VC707 board hosts two custom-built flash boards with SATA connectors. The VC707 board, coupled with two custom flash boards, is mounted on top of each server. The host servers run the Ubuntu distribution of Linux. Figure 14 shows the components of a single node. One of the servers also had a 512GB Samsung M.2 PCIe SSD for performance comparisons.

We used Connectal [King et al. 2015] and its PCIe Gen 1 implementation for the host link. Connectal is a hardware-software codesign framework built by Quanta Research.

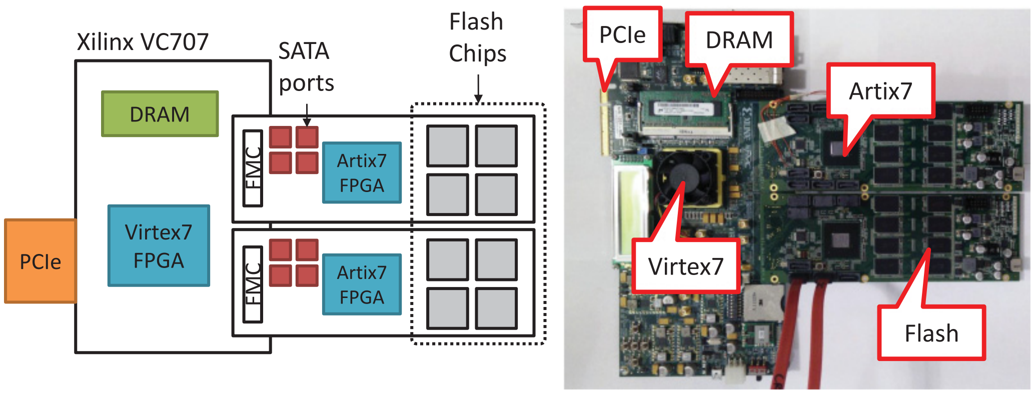


Fig. 14. A BlueDBM storage node.

Table I. Host Virtex 7 Resource Usage

Module Name	#	LUTs	Registers	RAMB36	RAMB18
Flash Interface	1	1,389	2,139	0	0
Network Interface	1	29,591	27,509	0	0
DRAM Interface	1	11,045	7,937	0	0
Host Interface	1	88,376	46,065	169	14
Virtex-7 Total		135,271	135,897	224	18
		(45%)	(22%)	(22%)	(1%)

Connectal reads the interface definition file written by the programmer and generates glue logic between hardware and software. Connectal automatically generates an RPC-like interface from developer-provided interface specification, as well as a memory-mapped DMA interface for high-bandwidth data transfer. Connectal's Gen 1 PCIe implementation that we used to implement BlueDBM caps our performance at 1.6GB/s reads and 1GB/s writes, which is a reasonable performance for a commodity flash storage device. We are also exploring the benefits of a faster host link including an in-house implementation of a lightweight PCIe gen 2 DMA engine, which is the fastest host interface that the VC707 board supports. Newer versions of Connectal also support PCIe Gen 2 with its faster performance.

The FPGA resource usage of the Virtex 7 FPGA chip on the VC707 board is shown in Table I. As can be seen, there is still enough space for accelerator development on the Virtex FPGA.

5.1. Custom Flash Board

We have designed and built a high-capacity custom flash board with high-speed serial connectors, with the help of Quanta Inc. and Xilinx Inc.

Each flash card has 512GB of NAND flash storage and a Xilinx Artix 7 chip, and plugs into the host FPGA development board via the FPGA Mezzanine Card (FMC) connector. The flash controller and Error-Correcting Code (ECC) are implemented on this Artix chip, providing the Virtex 7 FPGA chip on the VC707 a logical error-free access into flash. The communication between the flash board and the Virtex 7 FPGA is done by a four-lane aurora channel, which is implemented on the GTX/GTP serial transceivers included in each FPGA. This channel can sustain up to 3.3GB/s of bandwidth at 0.5μs latency. The flash board also hosts eight SATA connectors, four of which pin out the high-speed serial ports on the host Virtex 7 FPGA, and four of which pin out the

Table II. Flash Controller on Artix 7 Resource Usage

Module Name	#	LUTs	Registers	BRAM
Bus Controller	8	7,131	4,870	21
→ ECC Decoder	2	1,790	1,233	2
→ Scoreboard	1	1,149	780	0
→ PHY	1	1,635	607	0
→ ECC Encoder	2	565	222	0
SerDes	1	3,061	3,463	13
Artix-7 Total		75,225 (56%)	62,801 (23%)	181 (50%)

Table III. BlueDBM Estimated Power Consumption

Component	Power (Watts)
VC707	30
Flash Board x2	10
Xeon Server	200
Node Total	240

high-speed serial ports on the Artix 7 chip. The serial ports are capable of 10Gbps and 6.6Gbps of bandwidth, respectively.

The FPGA resource usage of each of the two Artix-7 chips is shown in Table II; 46% of the I/O pins were used either to communicate with the FMC port or to control the flash chips.

5.2. Network Infrastructure

In our BlueDBM implementation, network links are implemented over the low-latency serial transceivers. By implementing routing in the hardware and using a low-latency network fabric, we were able to achieve very high performance, with less than $0.5\mu s$ of latency per network hop, and near 10Gbps of bandwidth per physical link. Our hardware implementation pins out eight physical ports per storage node, so the aggregate network bandwidth available to a node reaches up to 8GB/s, including packet overhead.

In our implementation, pairs of two physical ports are paired together to form one logical link, resulting in a fan-out of four links per node. Each logical link supports a bandwidth of 20Gbps including packet overhead, and 17Gbps of useful bandwidth excluding packet overhead. We chose to group pairs of physical ports together because a single link of 10Gbps was too slow to network flash devices capable of speeds over 2GB/s.

5.3. Power Consumption

Table III shows the overall power consumption of the system, which was estimated using values from the data sheet. Each Xeon server includes 24 cores and 50GBs of DRAM. Thanks to the low power consumption of the FPGA and flash devices, BlueDBM adds less than 20% of power consumption to the system.

6. RAW SYSTEM PERFORMANCE

This section evaluates the raw system performance of the BlueDBM implementation. We measured the performance of various aspects of the system including the bandwidth and latency of distributed flash access. The experiments include measuring the raw performance of the controller network, latency and bandwidth of accessing distributed flash storage, and the performance of a file system that can take advantage of the changed interface we expose to the software. The evaluation of application-specific accelerators on the in-store processor is discussed in detail in Section 7.

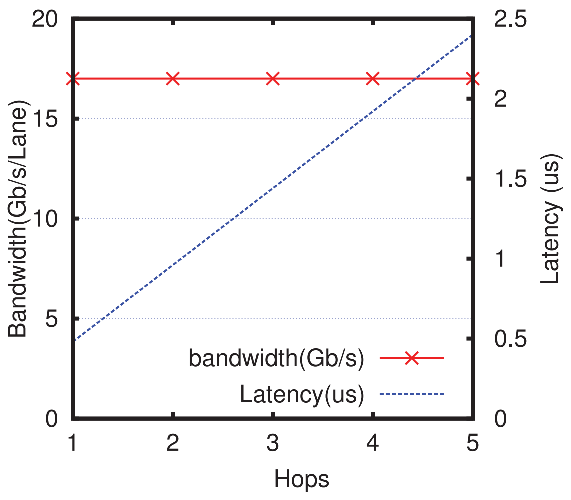


Fig. 15. BlueDBM intercontroller network performance.

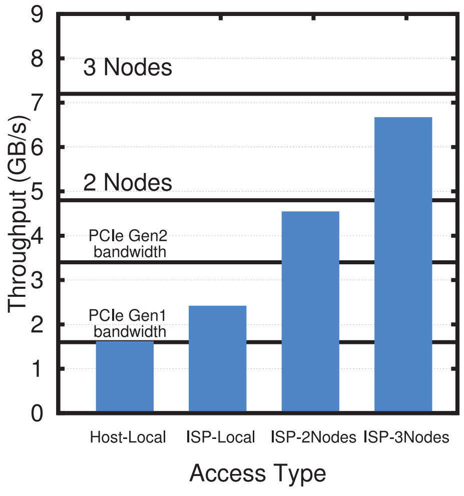


Fig. 16. Bandwidth of data access in BlueDBM.

6.1. Network Performance

We measured the performance of the network by transferring a single stream of 128-bit data packets through multiple nodes across the network in a noncontentious traffic setting. The maximum physical link bandwidth is 20Gbps, and per-hop latency is 0.48μs. Figure 15 shows that we are able to sustain 17Gbps of bandwidth per stream across multiple network hops. This shows that the protocol overhead is around 15%. The latency is 0.48μs per network hop, and the end-to-end latency is simply a multiple of network hops to the destination.

Each node in our BlueDBM implementation includes a fan-out of four network ports, so each node can have an aggregate full duplex bandwidth of 8.5GB/s. With such a high fan-out, it would be unlikely for a remote node in a rack-class cluster to be over 4 hops, or 2μs away. In a naive ring network of 20 nodes with two links each to next and previous nodes, the average latency to a remote node is 5 hops, or 2.5μs. The ring throughput is 34Gbps. *Assuming a flash access latency of 50μs, such a network will only add 5% latency in the worst case, giving the illusion of a uniform access storage.*

6.2. Remote Storage Access Latency

We measured the latency of remote storage access by reading an 8K page of data from the following sources using the storage controller network:

(1) FtoISP: From remote flash storage to in-store processor
(2) FtoHost: From remote flash storage to host server
(3) FtoRHost: From remote flash storage to host server via remote host server
(4) DtoRHost: From remote DRAM to host server via remote host server

In each case, the request is sent from either the host server or the in-store processor on the local BlueDBM node. In the third and fourth case, the request is processed by the remote server, instead of the remote in-store processor, adding extra latency. However, data is always transferred back via the storage controller network. We could have also measured the accesses to remote servers via Ethernet, but that latency is at least 100 times of the integrated network and will not be particularly illuminating.

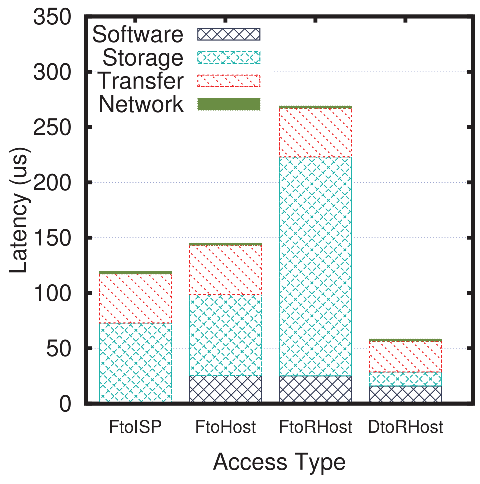


Fig. 17. Latency of remote data access in BlueDBM.

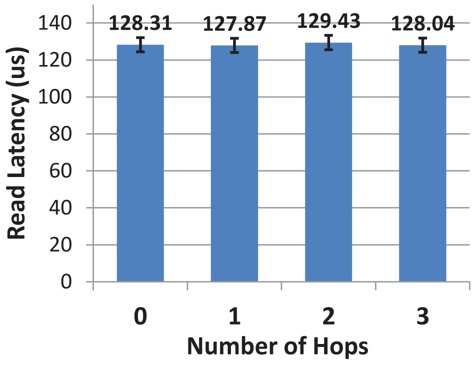


Fig. 18. Latency of remote data access over multiple hops.

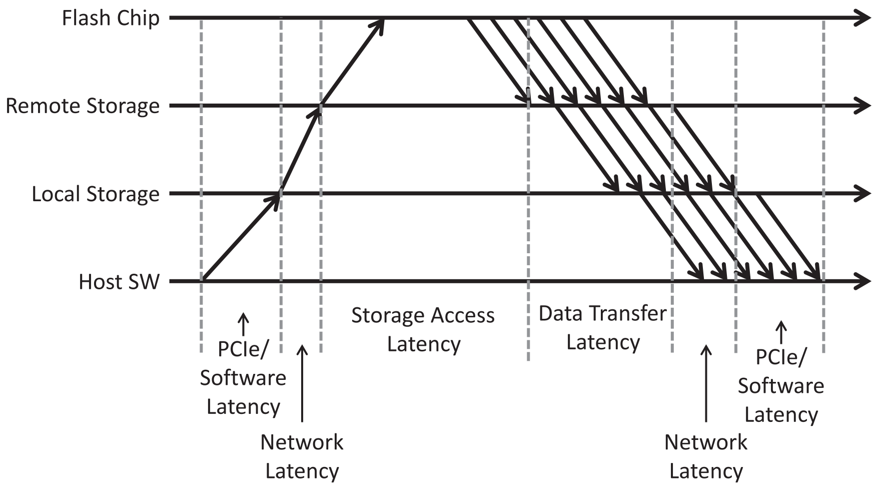


Fig. 19. Breakdown of remote storage access latency.

The latency is broken up into four components as shown in Figure 19. First is the local software overhead of accessing the network interface. Second is the storage access latency, or the time it takes for the first data byte to come out of the storage device. Third is the amount of times it takes to transfer the data until the last byte is sent back over the network, and last is the network latency.

Figure 17 shows the exact latency breakdown for each experiment. Notice in all four cases, the network latency is insignificant. Figure 18 further shows that the measured latency of remote data access over multiple network hops is uniform. The data transfer latency is similar except when data is transferred from DRAM (DtoRHost), where it is slightly lower. Notice that except in the case of FtoISP, storage access incurs the additional overhead of PCIe and host software latencies. If we compare FtoISP to

FtoRHost, we can see the benefits of an integrated storage network, as the former allows overlapping the latencies of storage and network access.

6.3. Storage Access Bandwidth

We measured the bandwidth of BlueDBM by sending a stream of millions of random read requests for 8KB size pages to local and remote storage nodes and measuring the elapsed time to process all of the requests. We measured the bandwidth under the following scenarios:

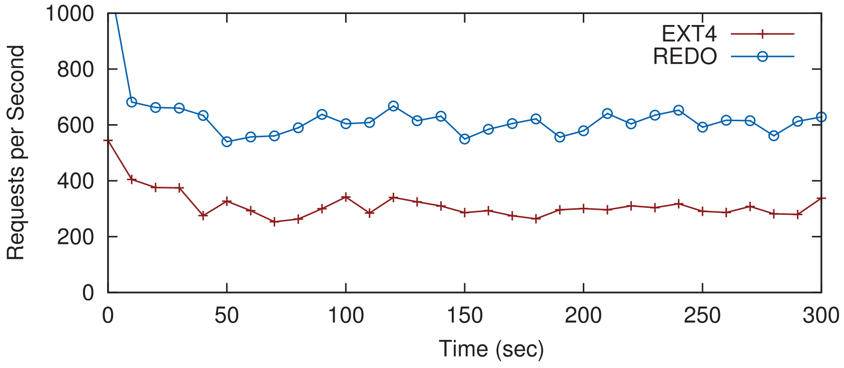
(1) Host-Local: Host sends requests to the local flash and all data is streamed returned over PCIe.
(2) ISP-Local: Host sends requests to the local flash and all data is consumed at the local in-store processor.
(3) ISP-2Nodes: Like ISP-Local except 50% of the requests are sent to a remote flash controller. One network link connects the two nodes.
(4) ISP-3Nodes: Like ISP-Local except 33% of the requests are sent to each of the two remote flash controllers. Two network links connect each remote controller to the local controller.

Figure 16 shows the read bandwidth performance for each of these cases. Our design of the flash card provides 1.2GB/s of bandwidth per card. Therefore, in theory, if both cards are kept completely busy, 2.4GB/s should be the maximum sustainable bandwidth from the in-store processor, and this is what we observe in the ISP-Local experiment. In our Host-Local experiment, we observed only 1.6GB/s of bandwidth. This is because this is the maximum bandwidth our PCIe Gen 1 implementation can sustain. If we had used a PCIe Gen 2 interface, the host would have been able to sustain the performance of a single storage node. In ISP-2Nodes, the aggregate bandwidth of two flash devices should add up to 4.8GB/s, but we only observe about 4.5GB/s, because remote storage access is limited by the 17Gbps serial link. With two nodes, even a PCIe Gen 2 link would not have been able to fully sustain the bandwidth of the storage device. In ISP-3Nodes, the aggregate bandwidth of three flash devices should add up to 7.2GB/s, but we only observe about 6.6GB/s because the aggregate bandwidth of the two serial links connecting the remote controllers is limited to 34Gbps, or 4.25GB/s.

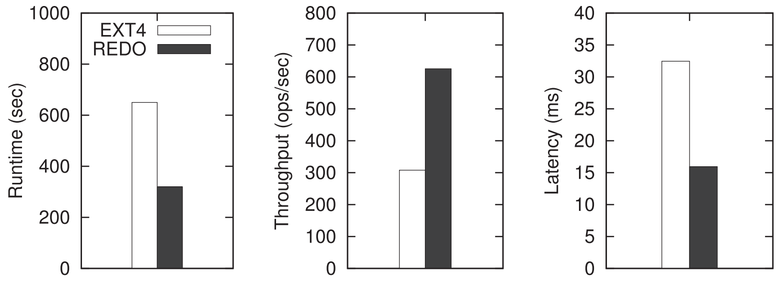
What these sets of experiments show is that in order to make full use of flash storage, some combination of fast networks, fast host connections, and low software overhead is necessary. These requirements can be somewhat mitigated if we make use of in-store computing capabilities, which is what we discuss next.

6.4. Flash-Aware File System Performance

We demonstrated BlueDBM's compatibility with host software and the benefits of exposing flash characteristics by running a flash-aware file system called REDO [Lee et al. 2015] and comparing its performance to the same system running a generic ext4 file system running on top of the BlueDBM block device driver. REDO is a log-structured file system that contains built-in flash management functionalities. By removing redundancies that arise from separately using FTL and traditional file systems, REDO can achieve higher performance using fewer hardware resources. We ran the popular Yahoo Cloud Serving Benchmark (YCSB) [Cooper et al. 2010] with MySQL+InnoDB at default settings. YCSB was configured to perform 200,000 updates to 750,000 records in the database. We compare two I/O stack configurations: (1) BlueDBM+REDO file system and (2) BlueDBM + host page-level FTL + ext4 file system. The latter configuration emulates a traditional SSD I/O architecture. Measurements are shown in Figure 20.



(a) Requests per second over time



(b) Runtime (c) Throughput (d) Latency

Fig. 20. Experimental results with database applications.

It should be noted that the software configuration used in this experiment is designed to emphasize the performance difference between REDO and a conventional file system, not to achieve the highest performance with the given hardware. For example, MySQL could have been tuned to have larger buffer space and reduce storage I/O traffic, which would have improved performance greatly. However, such a configuration would not show clearly the performance difference between a flash-optimized file system and a conventional file system.

We see that REDO doubles the performance of FTL+ext4 in both throughput and latency. This gain primarily stems from a reduced number of I/O operations that REDO performs for the same workload. By merging file system and FTL functions, REDO can cut down on redundant and unnecessary I/Os in garbage collection while maximizing the parallelism of the flash device. REDO is one of many examples of OS-level and user-level software that can take advantage of the raw flash interface provided by BlueDBM.

7. APPLICATION ACCELERATION

In this section, we demonstrate the performance and benefits of the BlueDBM architecture for application-specific accelerators. We present three examples: nearest-neighbor search, graph traversal, and string search. Nearest-neighbor search and string search demonstrate the performance benefits of high random access bandwidth and hardware acceleration of comparison functions. Graph traversal uses a latency-bound problem to demonstrate the benefits of low-latency access into distributed storage.

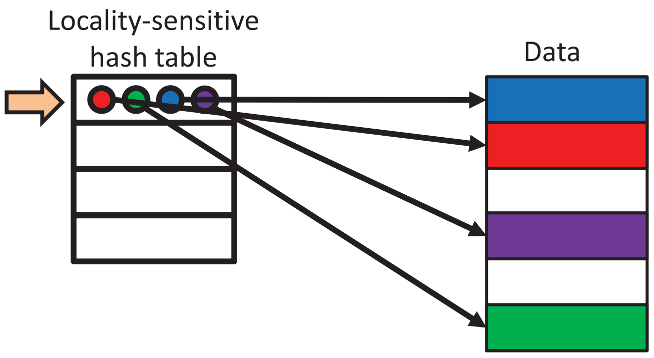


Fig. 21. Data accesses in LSH are randomly distributed.

7.1. Nearest-Neighbor Search

Description: Nearest-neighbor search is required by many applications, for example, image querying. One of the modern techniques in this field is Locality-Sensitive Hashing (LSH) [Gionis et al. 1999]. LSH hashes the dataset using multiple hash functions, so that similar data is statistically likely to be hashed to similar buckets. When querying, the query is hashed using the same hash functions, and only the data in the matching buckets are actually compared. The bulk of the work during a query process is traversing hash buckets and reading the corresponding data to perform distance calculation. Because data pointed to by the hash buckets are most likely scattered across the dataset, access patterns are quite random (see Figure 21).

We have built an LSH query accelerator, where all of the data is stored in flash and the distance calculation is done by the in-store processor on the storage device. The software first loads the query data item into the comparison accelerator and then sends a stream of addresses from a hash bucket. The system returns the index of the data item most closely matching the query to the software. Since we do not expect any performance difference for queries emanating from two different hash buckets, we simply send out a million nearest-neighbor searches for the same query.

We implemented three different distance comparison metrics with increasingly complex distance calculation functions: Hamming distance, cosine similarity, and joint-histogram-based image comparison. The Hamming distance example calculates the Hamming distance between 8KB data items, and the cosine similarity example compares vectors of 8,000 one-byte values using the cosine similarity metric. The image comparison example compares 128-by-128 pixel images stored in a bitmap format of 48KB each. The comparator first runs the raw image through a Sobel filter to determine the "edgeness" of each pixel location and generates a 4-dimensional histogram of using the edgeness and RGB values of each pixel location. The distance between two images is calculated by comparing the difference between their histograms.

In this study, we were interested in evaluating and comparing the benefits of flash storage (as opposed to DRAM) and in-store processors. We also wanted to compare the BlueDBM design with off-the-shelf SSDs with PCIe interface. The following experiments aim to evaluate the performance of each system during various access patterns, such as random or sequential access, and when accesses are partially serviced by secondary storage.

We have used a commercially available M.2 mPCIe SSD whose performance, for 8KB accesses, was limited to 600MB/s. Since BlueDBM performance is much higher (2.4GB/s), we also conducted several experiments with BlueDBM throttled to 600MB/s to match its performance. Since performance should scale linearly with the number of

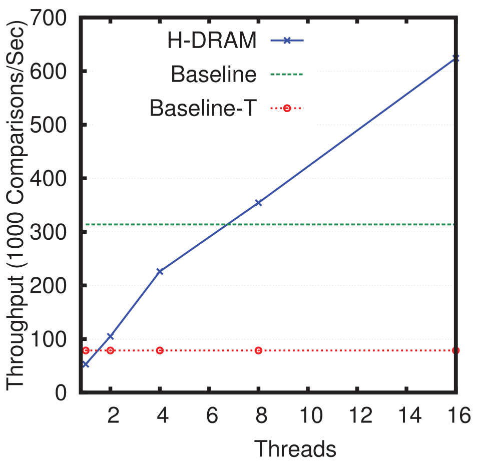


Fig. 22. Nearest neighbor with BlueDBM against DRAM.

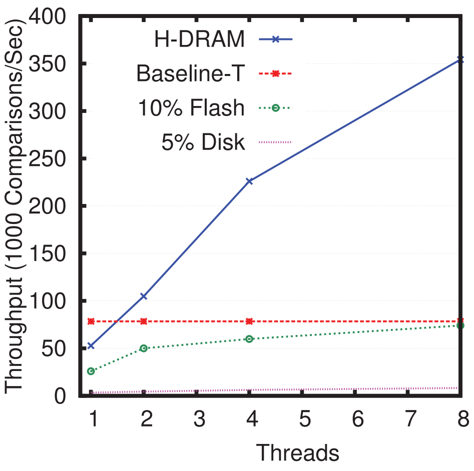


Fig. 23. Nearest neighbor with mostly DRAM.

nodes for this application, we concentrated on various configurations in a single-node setting.

Hamming Distance Evaluation:

First, we used Hamming distance as the distance metric, which is computationally simple. The following list shows the system configurations that were compared:

(1) Baseline: BlueDBM with in-store acceleration
(2) Baseline-T: Throttled BlueDBM with in-store acceleration
(3) Baseline-TS: Throttled: Multithreaded software on multicore host accessing Throttled BlueDBM as storage
(4) H-DRAM: Multithread software on multicore host accessing host DRAM as storage
(5) 10% Flash: Same as H-DRAM with 10% accesses to SSD
(6) 5% Disk: Same as H-DRAM with 5% accesses to HDD
(7) H-RFlash: Multithreaded software on multicore host accessing off-the-shelf SSD
(8) H-SFlash: Same as H-RFlash except data accesses are artificially arranged to be sequential.

Figure 22 shows the relative performance of a throttled BlueDBM (Baseline-T) and multithreaded software accessing data on host DRAM (H-DRAM), with Baseline BlueDBM. The baseline performance we observed on BlueDBM was 320K Hamming Comparisons per second. There are two important takeaways from this graph: (1) With a very simple function such as Hamming distance, BlueDBM can keep up with DRAM-resident data for up to four threads, because the host is getting compute bound. However, as more threads are added, performance will scale, until DRAM bandwidth becomes the bottleneck. Since DRAM bandwidth as compared to flash bandwidth is high, DRAM-based processing wins with enough resources. (2) Native flash speed matters; that is, when flash performance is throttled to one-fourth of the maximum, the performance drops accordingly. The relationship between flash performance and application performance will not be so simple if flash was being accessed by software.

To make the comparisons fair, we conducted a set of experiments shown in Figures 23, 24, and 25 using throttled BlueDBM as the baseline.

Results of 5% Disk and 10% Flash experiments shown in Figure 23 show that the performance of a DRAM-centric configuration (H-DRAM) falls off sharply if even a small

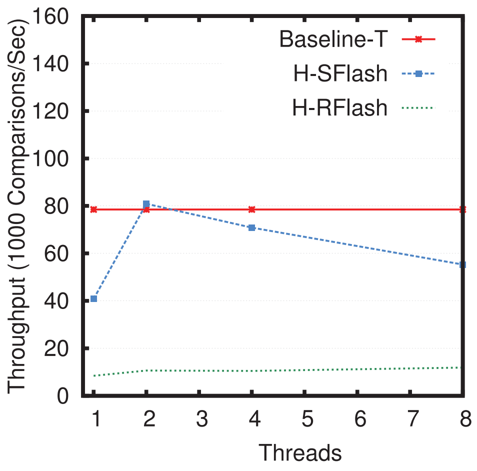


Fig. 24. Nearest neighbor with off-the-shelf SSD.

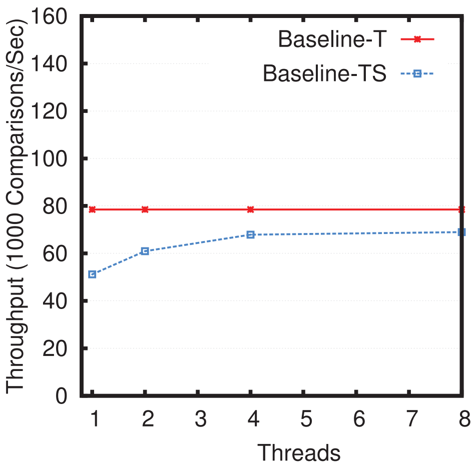


Fig. 25. Nearest neighbor with in-store processing.

fraction of data does not reside in DRAM. Assuming eight threads, the performance drops from 350K Hamming Comparisons per second to <80K and <10K Hamming Comparisons per second for 10% Flash and 5% Disk, respectively. At least one commercial vendor has observed similar phenomena and claimed that even when 40% of data fits on DRAM, the performance of Hadoop decreases by an order of magnitude [FusionIO 2014b]. Complex queries on DRAM show high performance only as long as all the data fits in DRAM.

The Off-the-Shelf SSD experiment H-RFlash results in Figure 24 showed that its performance is poor compared to even-throttled BlueDBM. However, when we artificially arranged the data accesses to be sequential, the performance improved dramatically, sometimes matching throttled BlueDBM. This suggests that the Off-the-Shelf SSD may be optimized for sequential accesses.

Figure 25, comparing Baseline-T and Baseline-TS, shows the advantage of accelerators. In this example, the accelerator advantage is at least 20%. Had we not throttled BlueDBM, the advantage would have been 30% or more. This is because the software is bottlenecked by the PCIe bandwidth at 1.6GB/s, while the in-store processor can process data at full flash bandwidth. We expect this advantage to be larger for applications requiring more complex accelerators Compared to a fully flash-based execution, BlueDBM performs an order of magnitude faster.

More Complex Examples:

We also compared the performance of various system configurations using more complex and computationally intensive distance metrics. We collected the performance of nearest-neighbor search using Hamming distance, cosine similarity, and image comparison as distance metrics on the following system configurations:

(1) Random Disk+SW: Data stored on disk, processed using software
(2) BlueDBM: Data stored on BlueDBM, processed using hardware
(3) DRAM+SW: Data stored completely on DRAM, processed using software

Configurations with software processing were configured to use the optimal number of threads in order to demonstrate maximum achievable performance. Figure 26 shows the collected results, normalized against the DRAM-based system, which is the upper bound of performance.

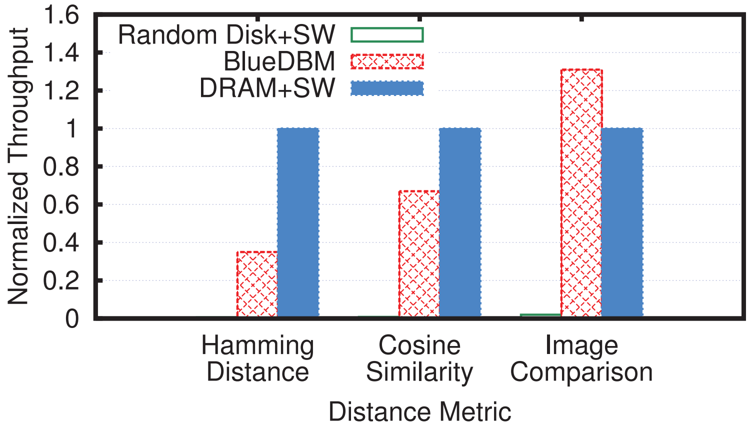


Fig. 26. BlueDBM relative performance increases with more complex distance metrics.

As distance metrics become more complex, the relative performance of the disk-based system, or BlueDBM increases, an increasing amount of time is spent on processing the data, rather than fetching it from storage. Due to terrible random access performance of hard disks, the disk-based system still shows low performance. However, BlueDBM's fast flash storage allows examples with a still simple distance metric such as cosine similarity to show performance rivaling the DRAM-based system's performance. With a more complex distance metric such as image comparison, the computation overhead becomes a bigger bottleneck than flash storage performance. In such a situation, fast flash storage with a fast hardware accelerator near the storage can exceed the performance of a DRAM-based system. We think the examples shown here cover a wide range of meaningful applications and show that BlueDBM can be a desirable platform for accelerating this class of applications.

7.2. Graph Traversal

Description: Efficient graph traversal is a very important component of any graph processing system. Fast graph traversal enables solving many problems in graph theory, including maximum flow, shortest path, and graph search. It is also a latency-bound problem because one often cannot predict the next node to visit until the previous node is visited and processed. We demonstrate the performance benefits of our BlueDBM architecture by implementing distributed graph traversal that takes advantage of the in-store processor and the integrated storage network, which allows extremely low-latency access into both local and remote flash storage.

Evaluation: Graph traversal algorithms often involve dependent lookups. That is, the data from the first request determines the next request, like a linked-list traversal at the page level. Since such traversals are sensitive to latency, we conducted the experiments with settings that are similar to the settings in Section 6.2.

(1) FtoISP: In-store processor requests data from remote storage over integrated network.
(2) FtoHost: Software requests data from remote storage over integrated network.
(3) FtoRHost: Software requests data from remote software to read from flash
(4) D+50%F: Store requests data from remote software: 50% chance of hitting flash.
(5) D+30%F: Store requests data from remote software: 30% chance of hitting flash.
(6) DtoRHost: Software requests data from remote software: Data read from DRAM.

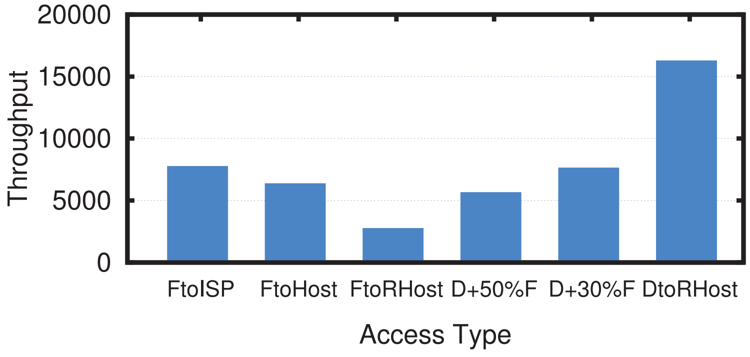


Fig. 27. Reducing access latency of distributed flash storage improves graph traversal performance.

As expected, the results in Figure 27 show that the integrated storage network and in-store processor together show almost a factor-of-3 performance improvement over generic distributed SSD. This performance difference is large enough that even when 50% of the accesses can be accommodated by DRAM, performance of BlueDBM is still much higher.

The performance difference between *FtoHost* and *FtoRHost* illustrates the benefits of using the integrated network to reduce a layer of software access. Performance of *FtoISP* compared to *FtoHost* shows the benefits of further reducing software overhead by having the ISP manage the graph traversal logic.

7.3. String Search

Description: String search is a common operation in analytics, often used in database table scans, DNA sequence matching, and cheminformatics. It is primarily a sequential read-and-compare workload. We examine its performance on BlueDBM with assistance from in-store Morris-Pratt (MP) string search engines [Morris and Pratt 1970] fully integrated with the file system, flash controller, and application software. The software portion of string search initially sets up the accelerator by transferring the target string pattern (needle) and a set of precomputed MP constants over DMA. Then it consults the file system for a list of physical addresses of the files to search (haystack). This list is streamed to the accelerator, which uses these addresses to request for pages from the flash controller. The accelerated MP engines may operate in parallel either by searching multiple files or by dividing up the haystack into equal segments (with some overlaps). This choice depends on the number of files and size of each file. Since four read commands can saturate a single flash bus, we use four engines per bus to maximize the flash bandwidth. Only search results are returned to the server.

Evaluation: We compared our implementation of hardware-accelerated string search running on BlueDBM to the Linux Grep utility querying for exact string matches running on both SSD and hard disk. Processing bandwidth and server CPU utilizations are shown in Figure 28. We observe that the parallel MP engines in BlueDBM are able to process a search at 1.1GB/s, which is 92% of the maximum sequential bandwidth of a single flash board. Using BlueDBM, the query consumes almost no CPU cycles on the host server since the query is entirely offloaded and only the location of matched strings is returned, which we assume is a tiny fraction of the file (0.01% is used in our experiments). This is 7.5 times faster than software string search (Grep) on hard disks, which is I/O bound by disk bandwidth and consumes 13% CPU. On SSD, software string search remains I/O bound by the storage device, but CPU utilization increases significantly to 65% even for this type of simple streaming compare operation. This

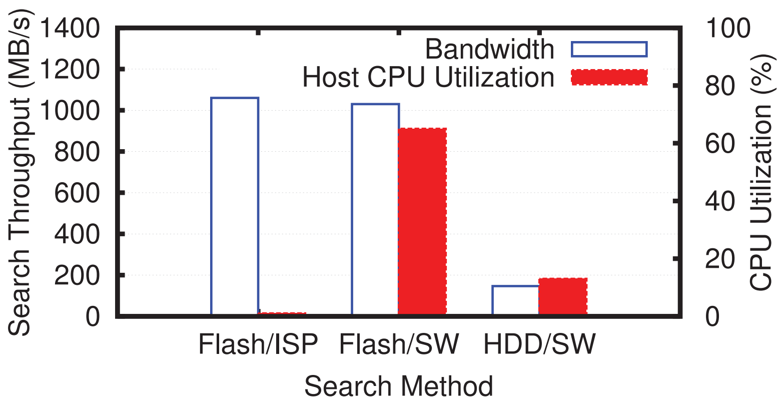


Fig. 28. String search bandwidth and CPU utilization.

high utilization is problematic because string search is often only a small portion of more complex analytic queries that can quickly become compute bound. As we have shown in the results, BlueDBM can effectively alleviate this by offloading search to the in-store processor, thereby freeing up the server CPU for other tasks.

8. CONCLUSION AND FUTURE WORK

Big Data analytics usually require a large amount of fast random access memory and computation. When a working set spills over the DRAM capacity of a cluster and starts accessing disk storage, the whole performance of the cluster falls sharply. A natural solution is building a large enough cluster with enough collective DRAM to accommodate the working set. Such a cluster often becomes prohibitively large, in terms of both capital and operational cost. It also becomes difficult to run software that makes efficient use of the total computation capabilities of a large cluster. Flash storage is an attractive alternative to DRAM in this regard, due to its fast random access performance, low power consumption, and low cost per GB. However, flash storage, packaged as off-the-shelf SSDs, suffer performance penalties in order to be backward compatible with older magnetic disk storage devices.

As a solution, we have presented BlueDBM, a distributed flash store for Big Data analytics that uses flash storage, in-store processing, and integrated networks for cost-effective analytics of large datasets. BlueDBM provides a uniformly low-latency access into a network of storage devices that form a global address space, by refactoring the storage and network software layers. It also provides the capacity to implement user-defined in-store processing engines, which can fully sustain the performance flash storage provides.

A rack-size BlueDBM system is likely to be an order of magnitude cheaper and less power hungry than a cluster with enough DRAM to accommodate 10TB to 20TB of data. For example, our 20-node BlueDBM platform can accommodate 20TB of data, while more than 100 comparable machines would be required to accommodate the same amount of data in DRAM. Additionally, while the performance of DRAM-centric systems falls rapidly if even a small fraction of data has to reside in secondary storage, this problem is greatly mitigated in a BlueDBM-like architecture because flash-based systems with 10TB to 20TB of storage are very affordable.

We have demonstrated the performance benefits of BlueDBM using various examples on large amounts of data in comparison to both generic flash-based system without such architectural improvements and DRAM-centric systems. Our results show that a platform such as BlueDBM is a viable platform for Big Data analytics in terms of performance and cost.

Our current implementation uses an FPGA to implement most of the new architectural features, that is, in-store processors, integrated network routers, and flash controllers. It is straightforward to implement most of these features using ASICs and provide some in-store computing capability via general-purpose processors. This will simultaneously improve the performance and lower the power consumption even further. Notwithstanding such developments, we are developing tools to make it easy to develop in-store processors for the reconfigurable logic inside BlueDBM.

We are currently developing several new applications, including *Distributed Key-Value Store Acceleration* by using the fast intercontroller networks and offloading query processing to in-store processors, and *Large Scale Sparse-Matrix Based Linear Algebra Acceleration*. One of the important goals of these projects is to identify some important operations of these workloads and provide the user with a library of hardware accelerated functions, reducing the overhead of the programmer in developing in-store processor cores. We plan to collaborate with other research groups to explore more applications.

ACKNOWLEDGMENTS

This work was partially funded by Quanta (Agmt. Dtd. 04/01/05), Samsung (Res. Agmt. Eff. 01/01/12), Lincoln Laboratory (PO7000261350), and Intel (Agmt. Eff. 07/23/12). We also thank Xilinx for their generous donation of VC707 FPGA boards and FPGA design expertise. We especially thank Xilinx engineer Rick Ballantyne for his expertise in the custom board design. We would also like to acknowledge our long-term collaboration with professor Jihong Kim and his group at Seoul National University.

REFERENCES

Anurag Acharya, Mustafa Uysal, and Joel Saltz. 1998. *Active Disks*. Technical Report. Santa Barbara, CA.

Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. 2008. Design tradeoffs for SSD performance. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference (ATC'08)*. USENIX Association, Berkeley, CA, 57–70.

Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Vijayan Prabhakaran. 2010. Removing the costs of indirection in flash-based SSDs with nameless writes. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Storage and File Systems (HotStorage'10)*. USENIX Association, Berkeley, CA, 1–5. http://dl.acm.org/citation.cfm?id=1863122.1863123

Infiniband Trade Association. 2014 (accessed November 18, 2014). *Infiniband*. http://www.infinibandta.org.

Jayanta Banerjee, David K. Hsiao, and Krishnamurthi Kannan. 1979. DBC: A database computer for very large databases. *IEEE Trans. Comput.* C-28, 6 (June 1979), 414–429. DOI:http://dx.doi.org/10.1109/TC.1979.1675381

Adrian M. Caulfield, Arup De, Joel Coburn, Todor I. Mollow, Rajesh K. Gupta, and Steven Swanson. 2010. 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)*. IEEE Computer Society, 385–395. DOI:http://dx.doi.org/10.1109/MICRO.2010.33

Adrian M. Caulfield and Steven Swanson. 2013. QuickSAN: A storage area network for fast, distributed, solid state disks. *SIGARCH Comput. Archit. News* 41, 3 (June 2013), 464–474. DOI:http://dx.doi.org/10.1145/2508148.2485962

Benjamin Y. Cho, Won Seob Jeong, Doohwan Oh, and Won Woo Ro. 2013. XSD: Accelerating MapReduce by harnessing the GPU inside an SSD. In *Proceedings of the 1st Workshop on Near-Data Processing.*

Eric S. Chung, John D. Davis, and Jaewon Lee. 2013. LINQits: Big data on little clients. In *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA'13)*. ACM, New York, NY, 261–272. DOI:http://dx.doi.org/10.1145/2485922.2485945

Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC'10)*. ACM, New York, NY, 143–154. DOI:http://dx.doi.org/10.1145/1807128.1807152

Jason Dai. 2010. Toward efficient provisioning and performance tuning for Hadoop. *Proc. Apache Asia Roadshow* 2010 (2010), 14–15.

Jaeyoung Do, Yang-Suk Kee, Jignesh M. Patel, Chanik Park, Kwanghyun Park, and David J. DeWitt. 2013. Query processing on smart SSDs: Opportunities and challenges. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD'13)*. ACM, New York, NY, 1221–1230. DOI:http://dx.doi.org/10.1145/2463676.2465295

FusionIO. 2012 (Accessed November 22, 2014). *Using HBase with ioMemory*. http://www.fusionio.com/white-papers/using-hbase-with-iomemory.

FusionIO. 2014 (Accessed November 18, 2014). *FusionIO*. http://www.fusionio.com.

Aristides Gionis, Piotr Indyk, Rajeev Motwani, and others. 1999. Similarity search in high dimensions via hashing. *VLDB* 99, 518–529.

Google. 2011 (Accessed November 18, 2014). *Google Flu Trends*. http://www.google.org/flutrends.

Sergej Hardock, Ilia Petrov, Robert Gottstein, and Alejandro Buchmann. 2013. NoFTL: Database systems on FTL-less flash storage. *Proc. VLDB Endow.* 6, 12 (Aug. 2013), 1278–1281. DOI:http://dx.doi.org/10.14778/2536274.2536295

Michio Honda, Felipe Huici, Costin Raiciu, Joao Araujo, and Luigi Rizzo. 2014. Rekindling network protocol innovation with user-level stacks. *SIGCOMM Comput. Commun. Rev.* 44, 2 (April 2014), 52–58. DOI:http://dx.doi.org/10.1145/2602204.2602212

Intel. 2014 (Accessed November 18, 2014). *Intel Solid-State Drive Data Center Family for PCIe*. http://www.intel.com/content/www/us/en/solid-state-drives/intel-ssd-dc-fa mily-for-pcie.html.

Nusrat S. Islam, Md. W. Rahman, Jithin Jose, Raghunath Rajachandrasekar, Hao Wang, Hari Subramoni, Chet Murthy, and Dhabaleswar K. Panda. 2012. High performance RDMA-based design of HDFS over InfiniBand. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC'12)*. IEEE Computer Society Press, Los Alamitos, CA, Article 35, 35 pages.

Zsolt István, Louis Woods, and Gustavo Alonso. 2014. Histograms as a side effect of data movement for big data. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. ACM, 1567–1578.

Eun Young Jeong, Shinae Woo, Muhammad Jamshed, Haewon Jeong, Sunghwan Ihm, Dongsu Han, and KyoungSoo Park. 2014. mTCP: A highly scalable user-level TCP stack for multicore systems. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation (NSDI'14)*. USENIX Association, Berkeley, CA, 489–502.

William K. Josephson, Lars A. Bongo, Kai Li, and David Flynn. 2010. DFS: A file system for virtualized flash storage. *Trans. Storage* 6, 3, Article 14 (Sept. 2010), 25 pages. DOI:http://dx.doi.org/10.1145/1837915.1837922

Sang-Woo Jun, Ming Liu, Kermin Elliott Fleming, and Arvind. 2014. Scalable multi-access flash store for big data analytics. In *Proceedings of the 2014 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'14)*. ACM, New York, NY, 55–64. DOI:http://dx.doi.org/10.1145/2554688.2554789

Seok-Hoon Kang, Dong-Hyun Koo, Woon-Hak Kang, and Sang-Won Lee. 2013b. A case for flash memory SSD in hadoop applications. *Int. J. Control Autom.* 6, 1 (2013).

Yangwook Kang, Yang-suk Kee, Ethan L. Miller, and Chanik Park. 2013a. Enabling cost-effective data processing with smart ssd. In *2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST'13)*. IEEE, 1–12.

Kimberly Keeton, David A. Patterson, and Joseph M. Hellerstein. 1998. A case for intelligent disks (IDISKs). *SIGMOD Rec.* 27, 3 (Sept. 1998), 42–52. DOI:http://dx.doi.org/10.1145/290593.290602

Myron King, Jamey Hicks, and John Ankcorn. 2015. Software-driven hardware development. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA'15)*. ACM, New York, NY, 13–22. DOI:http://dx.doi.org/10.1145/2684746.2689064

Onur Kocberber, Boris Grot, Javier Picorel, Babak Falsafi, Kevin Lim, and Parthasarathy Ranganathan. 2013. Meet the walkers: Accelerating index traversals for in-memory databases. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-46)*. ACM, New York, NY, 468–479. DOI:http://dx.doi.org/10.1145/2540708.2540748

Sungjin Lee, Jihong Kim, and Arvind. 2015. Refactored design of I/O architecture for flash storage. *Comput. Archit. Lett.* 14, 1 (Jan. 2015), 70–74. DOI:http://dx.doi.org/10.1109/LCA.2014.2329423

Sungjin Lee, Ming Liu, Sangwoo Jun, Shuotao Xu, Jihong Kim, and Arvind. 2016. Application-managed flash. In *14th USENIX Conference on File and Storage Technologies (FAST'16)*. USENIX Association, Santa Clara, CA, 339–353. http://usenix.org/conference/fast16/technical-sessions/presentation/lee.

Sang-Won Lee, Bongki Moon, Chanik Park, Jae-Myung Kim, and Sang-Woo Kim. 2008. A case for flash memory ssd in enterprise database applications. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. ACM, 1075–1086.

Hans-Otto Leilich, Günther Stiege, and Hans Christoph Zeidler. 1978. A search processor for data base management systems. In *4th International Conference on Very Large Data Bases.* 280–287.

Jiuxing Liu, Jiesheng Wu, Sushmitha P. Kini, Pete Wyckoff, and Dhabaleswar K. Panda. 2003. High performance RDMA-based MPI Implementation over InfiniBand. In *Proceedings of the 17th Annual International Conference on Supercomputing (ICS'03)*. ACM, New York, NY, 295–304. DOI:http://dx.doi.org/10.1145/782814.782855

Frank McSherry, Michael Isard, and Derek G. Murray. 2015. Scalability! but at what cost? In *Proceedings of the 15th USENIX Conference on Hot Topics in Operating Systems (HOTOS'15)*. USENIX Association, Berkeley, CA. http://dl.acm.org/citation.cfm?id=2831090.2831104

Violin Memory. 2014 (Accessed November 18, 2014). *Violin Memory*. http://www.violin-memory.com.

James Morris Jr. and Vaughan Pratt. 1970. *A Linear Pattern-Matching Algorithm*. TR-40, Comptr Ctr., U of California, Berkeley, Calif.

Rene Mueller, Jens Teubner, and Gustavo Alonso. 2009. Streams on wires: A query compiler for FPGAs. *Proc. VLDB Endow.* 2, 1 (Aug. 2009), 229–240. DOI:http://dx.doi.org/10.14778/1687627.1687654

Oracle. 2014 (Accessed November 18, 2014). *Exadata Database Machine*. https://www.oracle.com/engineered-systems/exadata/index.html.

John Ousterhout, Parag Agrawal, David Erickson, Christos Kozyrakis, Jacob Leverich, David Mazières, Subhasish Mitra, Aravind Narayanan, Guru Parulkar, Mendel Rosenblum, Stephen M. Rumble, Eric Stratmann, and Ryan Stutsman. 2010. The case for RAMClouds: Scalable high-performance storage entirely in DRAM. *SIGOPS Oper. Syst. Rev.* 43, 4 (Jan. 2010), 92–105. DOI:http://dx.doi.org/10.1145/1713254.1713276

Esen A. Ozkarahan, Stewart A. Schuster, and Kenneth C. Smith. 1975. RAP - An associative processor for database management. In *American Federation of Information Processing Societies: 1975 National Computer Conference*. 379–387. DOI:http://dx.doi.org/10.1145/1499949.1500024

Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth, Gopal Jan, Gray Michael, Haselman Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi, and Xiao Doug Burger. 2014. A reconfigurable fabric for accelerating large-scale datacenter services. *SIGARCH Comput. Archit. News* 42, 3 (June 2014), 13–24. DOI:http://dx.doi.org/10.1145/2678373.2665678

Md. W. Rahman, Nusrat S. Islam, Xaoyi Lu, Jithin Jose, Hari Subramoni, Hao Wang, and Dhabaleswar K. Panda. 2013. High-performance RDMA-based design of hadoop mapreduce over InfiniBand. In *International Workshop on High Performance Data Intensive Computing (HPDIC'13), in Conjunction with IEEE International Parallel and Distributed Processing Symposium (IPDPS'13)*.

Md. W. Rahman, Xiaoyi Lu, Nusrat S. Islam, and Dhabaleswar K. Panda. 2014. HOMR: A hybrid approach to exploit maximum overlapping in MapReduce over high performance interconnects. In *Proceedings of the 28th ACM International Conference on Supercomputing (ICS'14)*. ACM, New York, NY, 33–42. DOI:http://dx.doi.org/10.1145/2597652.2597684

Stephen M. Rumble, Ankita Kejriwal, and John Ousterhout. 2014. Log-structured memory for DRAM-based storage. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST'14)*. USENIX Association, Berkeley, CA, 1–16.

SanDisk. 2014 (Accessed November 22, 2014). *Sandisk ZetaScale Software*. http://www.sandisk.com/enterprise/zetascale/.

Sudharsan Seshadri, Mark Gahagan, Sundaram Bhaskaran, Trevor Bunker, Arup De, Yanqin Jin, Yang Liu, and Steven Swanson. 2014. Willow: A user-programmable SSD. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation (OSDI'14)*. USENIX Association, Berkeley, CA, 67–80.

Malcolm Singh and Ben Leonhardi. 2011. Introduction to the IBM netezza warehouse appliance. In *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research (CASCON'11)*. IBM Corp., Riverton, NJ, 385–386.

Joel R. Spiegel, Michael T. McKenna, Girish S. Lakshman, and Paul G. Nordstrom. 2011. Method and system for anticipatory package shipping. (Dec. 27 2011). US Patent 8,086,546.

Bharat Sukhwani, Hong Min, Mathew Thoennes, Parijat Dube, Balakrishna Iyer, Bernard Brezzo, Donna Dillenberger, and Sameh Asaad. 2012. Database analytics acceleration using FPGAs. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT'12)*. ACM, New York, NY, 411–420. DOI:http://dx.doi.org/10.1145/2370816.2370874

Diablo Technologies. 2014 (Accessed November 18, 2014). *Diablo Technologies*. http://www.diablo-technologies.com/.

Tim S. Woodall, Galen M. Shipman, George Bosilca, Richard L. Graham, and Arthur B. Maccabe. 2006. High performance RDMA protocols in HPC. In *Proceedings of the 13th European PVM/MPI User's Group Conference on Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI'06)*. Springer-Verlag, Berlin, 76–85. DOI:http://dx.doi.org/10.1007/11846802_18

Louis Woods, Zsolt Istvan, and Gustavo Alonso. 2013. Hybrid FPGA-accelerated SQL query processing. In *2013 23rd International Conference on Field Programmable Logic and Applications (FPL'13)*. DOI:http://dx.doi.org/10.1109/FPL.2013.6645619

Louis Woods, Zsolt Istvan, and Gustavo Alonso. 2014. Ibex - an intelligent storage engine with support for advanced SQL Off-loading. In *Proceedings of the 40th International Conference on Very Large Data Bases (VLDB'14)*. 963–974.

Lisa Wu, Andrea Lottarini, Timothy K. Paine, Martha A. Kim, and Kenneth A. Ross. 2014. Q100: The architecture and design of a database processing unit. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'14)*. ACM, New York, NY, 255–268. DOI:http://dx.doi.org/10.1145/2541940.2541961

Received February 2016; accepted March 2016