BlueDBM: An Appliance for Big Data Analytics

Sang-Woo Jun† Ming Liu† Sungjin Lee† Jamey Hicks⋆
John Ankcorn⋆ Myron King⋆ Shuotao Xu† Arvind†

Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology†
Quanta Research Cambridge⋆
{wjun,ml,chamdoo,shuotao,arvind}@csail.mit.edu†
{jamey.hicks,john.ankcorn,myron.king}@qrclab.com⋆

Abstract

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 20 TB. For such a dataset, one would need a cluster with 100 servers, each with 128GB to 256GBs 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. In this paper we present BlueDBM, a new system architecture which has flash-based storage with in-store processing capability and a low-latency high-throughput inter-controller network. 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 ram-cloud system falls sharply even if only 5%~10% of the references are to the secondary storage, this sharp performance degradation is not an issue in BlueDBM. BlueDBM presents an attractive point in the cost-performance trade-off for Big Data analytics.

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 CDC [13]; Analysis of twitter data can reveal social upheavals faster than journalists; Amazon is planning to use customer data for anticipatory shipping of products [43]; Real-time analysis of personal genome may significantly aid in diagnostics. Big Data analytics are potentially going to have 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 is 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 *ram cloud* [34], where the cluster has enough collective DRAM to accommodate the entire dataset in DRAM. In this paper, 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 accesses 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. Near-uniform 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.

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 unambiguously establishes an architecture whose price-performance-power characteristics provide an attractive alternative for doing similar scale applications in a ram cloud.

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: (1) Design and implementation

ISCA '15, June 13 - 17, 2015, Portland, OR, USA

ISBN 978-1-4503-3402-0/15/06$15.00
DOI: http://dx.doi.org/10.1145/2749469.2750412

of a scalable flash-based system with a global address space, in-store computing capability and a flexible inter-controller 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. (4) Demonstration of a complex data analytics appliance which is much cheaper and consumes an order of magnitude less power than the cloud-based alternative.

The rest of the paper 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 paper.

2. Related Work

In Big Data scale workloads, building a cluster with enough DRAM capacity to accommodate the entire dataset can be very desirable but expensive. An example of such a system is RAMCloud, which is a DRAM-based storage for large-scale datacenter applications [34, 39]. RAMCloud provides more than 64TBs of DRAM storage distributed across over 1000 servers networked over high-speed interconnect. Although RAMCloud provides 100 to 1000 times better performance than disk-based systems of similar scale, its high energy consumption and high price per GB limits its widespread use except for extremely performance and latency-sensitive workloads.

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 price compared to DRAM, and an order of magnitude faster performance compared to disk. Many existing database and analytics software has shown improved performance with SSDs [8, 21, 27]. Several SSD-optimized analytics softwares, such as the SanDisk Zetascale [40] have demonstrated promising performance while using SSD as the primary data storage. Many commercial SSD devices have adopted high-performance PCIe interface in order to overcome the slower SATA bus interface designed for disk [11, 30, 16]. Attempts to use flash as a persistent DRAM alternative by plugging it into a RAM slot are also being explored [45].

SSD storage devices have been largely developed to be a faster drop-in replacement for disk drives. This backwards compatibility has helped their widespread adoption. However, additional software and hardware is required to hide the difference in device characteristics [1]. 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 [4] modifies the operating system's storage management components to reduce software overhead when accessing NVM storage devices. Willow [41] provides an easy way to augment SSD controllers with additional interface semantics that make better use of SSD characteristics, in addition to a backwards compatible storage interface. Attempts to remove the translation layers and let the databse make high-level decisions [14] have shown to be beneficial.

Due to their high performance, SSDs also affect the network requirements. The latency to access disk over Ethernet was dominated by the disk seek latency. However, in a 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 [2], and by low-overhead software protocols such as RDMA [29, 17, 38, 46, 29, 37] or user-level TCP stacks that bypass the operating system [19, 15]. QuickSAN [5] 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.

Another important attempt to accelerate SSD storage performance is in-store processing, where some data analytics is 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 (IDISK) connected to each other using serial networks have been proposed in 1998 [23], and adding processor to disk heads to do simple filters have been suggested as early as in the 1970s [28, 35, 3]. 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 [9, 22, 41] and Programmable SSDs [6] 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 [48] 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 [42] offload operations such as filtering to a reconfigurable fabric near storage. On the other end of the spectrum, systems such as XSD [6] embeds a GPU into a SSD controller, and demonstrates high performance accelerating MapReduce.

Building specialized hardware for databases have been extensively studied and productized. Companies such as Oracle [33] have used FPGAs to offload database queries. FPGAs have been used to accelerate operations such as hash index lookups [25]. Domain-specific processors for database queries are being developed [44, 47], including Q100 [49] and LINQits [7]. 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 very advantageous. However, unlike BlueDBM, these architectures accelerate computation on data that is in DRAM. Accelerators have also been placed in-path between network and processor to perform operations at wire speed [32], or to collect information such as histogram tables without overhead [18].

Incorporating reconfigurable hardware accelerators into large datacenters is also being investigated actively. Microsoft recently has built and demonstrated the power/performance benefits of an FPGA-based system called Catapult [36]. 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 ram cloud 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 [20], which was a 4-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 will be 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 1). 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 either directly communicate with the flash interface, to treat is as a raw storage device, or with the in-store processor to perform computation on the data.

The in-store processing engine has access to four major services: The flash interface, network interface, host interface and the on-storage DRAM buffer. Figure 1 and Figure 2 shows the four services available to the in-store processor. In the following sections we 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 Interface

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 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 [26]. 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 file system/block device driver (discussed in Section 4).

3.1.1. Interface for High Performance Flash Access 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 (i) cutting down on access latency from the network and in-store processors; (ii) exposing all degrees of parallelism of the device and (iii) allowing higher level system stacks (file system, database storage engine) to more intelligently manage data.

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 acknowledgement once 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 50 μs or more.

3.1.2. Multiple Access Agents Multiple hardware endpoints in BlueDBM may need shared access to this flash controller

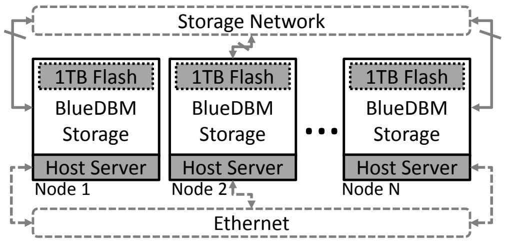

Figure 1: BlueDBM overall architecture

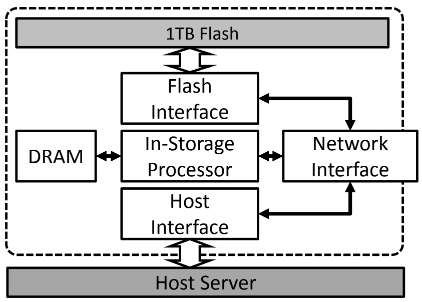

Figure 2: BlueDBM node architecture

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 hardware in-store processors, 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. Integrated Storage Network

BlueDBM provides a low-latency high-bandwidth network infrastructure across all BlueDBM storage devices in the cluster,

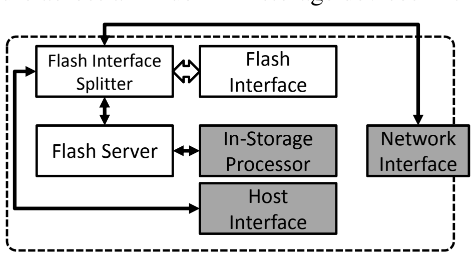

Figure 3: Flash interface

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 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 removes the overhead of going to the host software to access a separate network interface.

Figure 4 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 very flexible and can be configured to implement various topologies, as long as there is sufficient number of ports on each node. Figure 5 shows some example topologies. To implement a different topology the physical cables between each node has to be re-wired, but the routing across a topology can be configured dynamically by the software.

3.2.1. Logical Endpoint The BlueDBM network infrastructure exposes virtual channel semantics to the users of the network by providing it with multiple *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 that does not

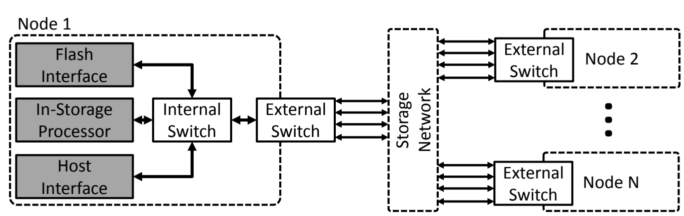

Figure 4: Network architecture

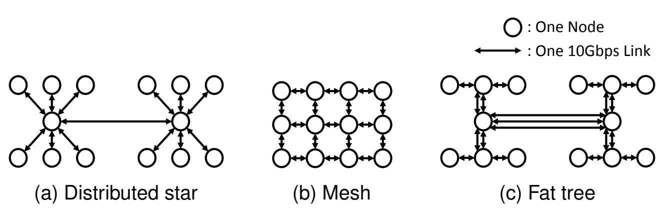

Figure 5: Any network topology is possible as long as it requires less than 8 network ports per node

need to be contiguous. 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.

3.2.2. 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 which is running slowly.

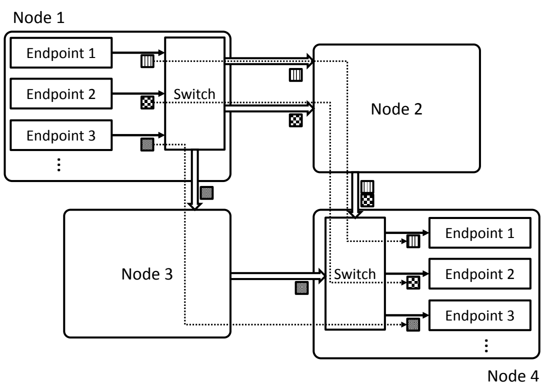

Figure 6: Packets from the same Endpoint to a destination maintain FIFO order

3.2.3. Routing Layer In order to make maximum use of the bandwidth of the network infrastructure while keeping resource usage to a minimal, 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 6 shows packet routing in an example network. The benefits 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.

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 to only send data when there is space on the destination endpoint, which will assure safety but result in higher latency due to flow control packets, and more memory usage for buffers.

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.

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 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 7 describes the structure of the host interface for flash reads.

4. Software Interface

In BlueDBM, we aim to provide a set of software interfaces that support the execution of any existing application as well as modified applications that leverage the in-store processors in the system. Furthermore, software layers in BlueDBM must perform flash management functions since we chose to expose a raw flash interface in hardware for higher efficiency (pre-

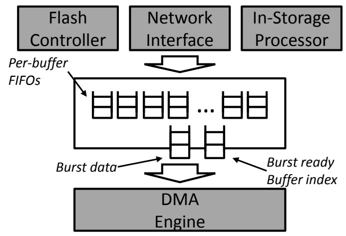

Figure 7: Host-FPGA interface over PCIe

viously discussed in Section 3.1). The software architecture is shown in Figure 8. Three interfaces are supplied to the user application: (i) a file system interface, (ii) a block device driver interface and (iii) an accelerator interface.

We first discuss the file system. Commercial SSDs incorporate a Flash Translation Layer (FTL) inside the flash device controller to manage flash and maintains 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 lots of extra I/Os. Some file systems have tried to remedy this by refactoring the I/O architecture in order to offload most of the FTL functions into a flash-optimized log-structured file system. A prominent example of this is RFS [26]. Unlike conventional FTL designs where the flash characteristics are hidden from the file system, RFS performs some functionality of an FTL, including logical-to-physical address mapping and garbage collection. This achieves better garbage collection efficiency at 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 Fusion IO's 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 8 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 8). This was made possible because the file system maintains the 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 very 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 kernel, except for essential driver modules. This helps us to avoid deep OS kernel stacks that often cause long I/O latencies. It is also very 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.

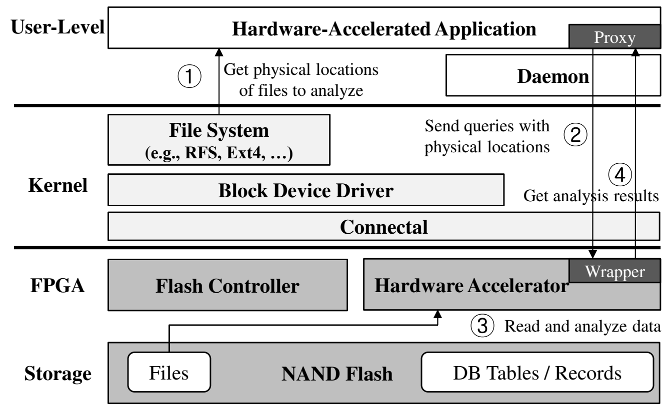

Figure 8: Software interface

5. Hardware Implementation

We have built a 20-node BlueDBM cluster to explore the capabilities of the architecture. Figure 9 shows the photo of our implementation.

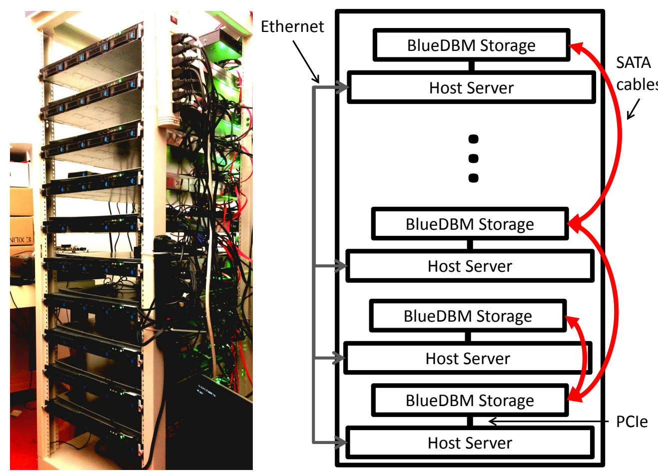

Figure 9: A 20-node BlueDBM cluster

In our implementation of BlueDBM, we have used a Field Programmable Gate Array (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 backpressure. 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 50GBs 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 10 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 [24] and its PCIe Gen 1 implementation for the host link. Connectal is a hardware-software codesign framework built by Quanta Research. Connectal reads the interface definition file written by the programmer and generates glue logic between hardware and software. Connectal automatically generates RPC-like interface from developer-provided interface specification, as well as a memory-mapped DMA interface for high bandwidth data transfer. Connectal's PCIe implementation caps our performance at 1.6GB/s reads and 1GB/s writes, which is a reasonable performance for a commodity flash storage device. In the future we will also explore the benefits of a faster host link including later generation PCIe links.

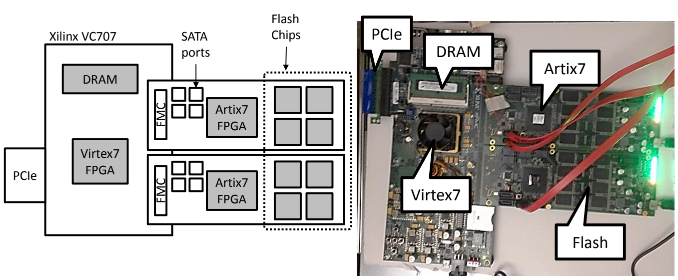

Figure 10: A BlueDBM storage node

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 512GBs 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) is 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 4-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 $0.5\mu s$ latency. The flash board also hosts 8 SATA connectors, 4 of which pin out the high-speed serial ports on the host Virtex 7 FPGA, and 4 of whch pin out the high-speed serial ports on the Artix 7 chip. The serial ports are capable of 10Gbps and 6.6Gbps of bandwidth, respectively.

5.2. Network Infrastructure

In our BlueDBM implementation, the link is implemented over the low-latency serial transceivers. By implementing routing in the hardware and using a very 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 link. Our implementation has a network fan-out of 8 ports per storage node, so the aggregate network bandwidth available to a node reaches up to 8GB/s, including packet overhead.

5.3. Software Interface

Our host interface is implemented using Connectal [24]. Connectal provides a PCIe Gen 1 endpoint and driver pair, and provides up to 1.6GB/s DMA read to host DRAM bandwidth and 1GB/s of DMA write from host DRAM bandwidth. Reading or writing data from the host buffers were done by DMA read/write engines implemented in the Connectal framework. In our BlueDBM implementation, there are four read engines and four write engines each, in order to more easily make maximum use of the PCIe bandwidth.

6. Evaluation

This section evaluates the characteristics of the BlueDBM implementation.

6.1. FPGA Resource Utilization

The FPGA resource usage of each of the two Artix-7 chips are shown in Table 1. 46% of the I/O pins were used either to communicate with the FMC port or to control the flash chips.

Module Name	#	LUTs	Registers	BRAM
Bus Controller	8	7131	4870	21
→ ECC Decoder	2	1790	1233	2
→ Scoreboard	1	1149	780	0
→ PHY	1	1635	607	0
→ ECC Encoder	2	565	222	0
SerDes	1	3061	3463	13
Artix-7 Total		75225 (56%)	62801 (23%)	181 (50%)

Table 1: Flash controller on Artix 7 resource usage

The FPGA resource usage of the Virtex 7 FPGA chip on the VC707 board is shown in Table 2. As it can be seen, there is still enough space for accelerator development on the Virtex FPGA.

Module Name	#	LUTs	Registers	RAMB36	RAMB18
Flash Interface	1	1389	2139	0	0
Network Interface	1	29591	27509	0	0
DRAM Interface	1	11045	7937	0	0
Host Interface	1	88376	46065	169	14
Virtex-7 Total		135271 (45%)	135897 (22%)	224 (22%)	18 (1%)

Table 2: Host Virtex 7 resource usage

6.2. Power Consumption

Table 3 shows the overall power consumption of the system, which were estimated using values from the datasheet. 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.

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

Table 3: BlueDBM estimated power consumption

6.3. 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 non-contentious traffic setting. The maximum physical link bandwidth is 10Gbps, and per-hop latency is 0.48 μs. Figure 11 shows that we are able to sustain 8.2Gbps of bandwidth per stream across multiple network hops. This shows that the protocol overhead is under 18%. The latency is 0.48 μs per network hop, 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 8 network ports, so each node can have an aggregate full duplex bandwidth of 8.2GB/s. With such a high fan-out, it would be unlikely that a remote node in a rack-class cluster to be over 4 hops, or 2 μs away. In a naive ring network of 20 nodes with 4 lanes each to next and previous nodes, the average latency to a remote node is 5 hops, or 2.5 μs. The ring throughput is 32.8 Gbps. *Assuming a flash access latency of 50 μs, such a network will only add 5% latency in the worst case, giving the illusion of a uniform access storage.*

6.4. 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 integrated storage network:

1. ISP-F: From in-store processor to remote flash storage;
2. H-F: From host server to remote flash storage;
3. H-RH-F: From host server to remote flash storage via its host server.
4. H-D: From host server to remote DRAM;

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 integrated storage network. We could have also measured the accesses to remote servers via Ethernet, but that latency is at least 100x of the integrated network, and will not be particularly illuminating.

The latency is broken up into four components as shown in Figure 14. 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 12 shows the exact latency breakdown for each experiment. Notice in all 4 cases, the network latency is insignificant. The data transfer latency is similar except when data is transferred from DRAM (H-D), where it is slightly lower. Notice that except in the case of ISP-F, storage access incurs the additional overhead of PCIe and host software latencies. If we compare ISP-F to H-RH-F, we can see the benefits of an integrated storage network, as the former allows overlapping the latencies of storage and network access.

6.5. 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. Only one serial 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 serial links connect each remote controller to the local controller.

Figure 13 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 implementation can sustain. In ISP-2Nodes, the aggregate bandwidth of two flash devices should add up to 4.8GB/s, but

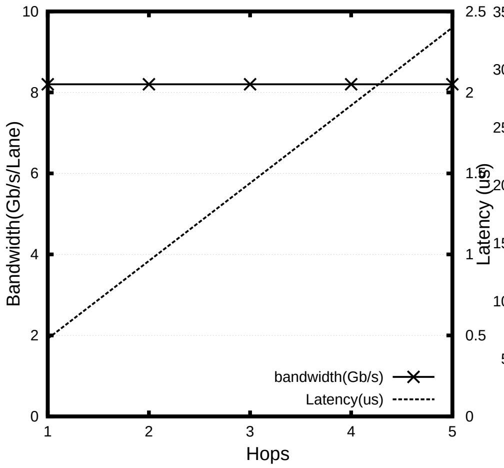

Figure 11: BlueDBM integrated network performance

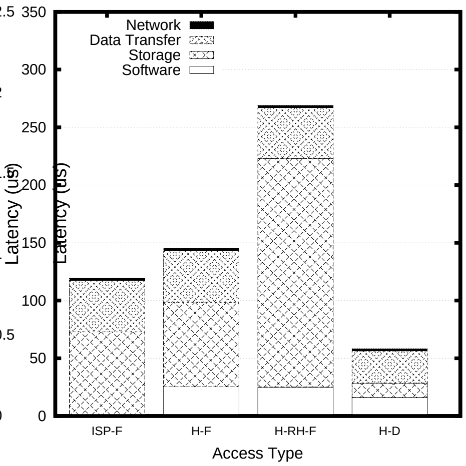

Figure 12: Latency of remote data access in BlueDBM

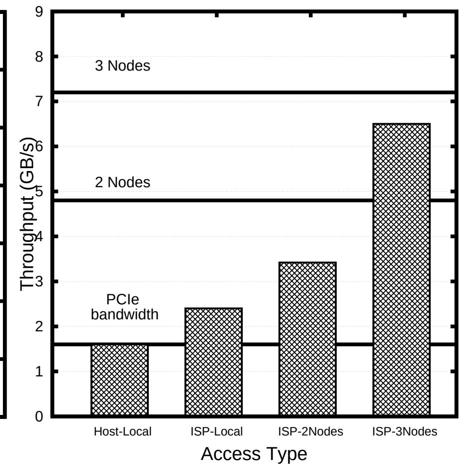

Figure 13: Bandwidth of data access in BlueDBM

we only observe about 3.4GB/s, because remote storage access is limited by the single 8Gbps-serial link. In ISP-3Nodes, the aggregate bandwidth of three flash devices should add up to 7.2GB/s, but we only observe about 6.5GB/s because the aggregate bandwidth of the four serial links connecting the remote controllers is limited to 32.8Gbps (=4.1GB/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.

7. Application Acceleration

In this section, we demonstrate the performance and benefits of the BlueDBM architecture by presenting some accelerator demonstrations.

7.1. Nearest Neighbor Search

Description: Nearest neighbor search is required by many applications, e.g., image querying. One of the modern techniques in this field is Locality Sensitive Hashing [12]. 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

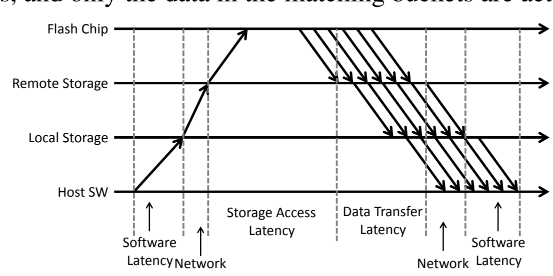

Figure 14: Breakdown of remote storage access latency

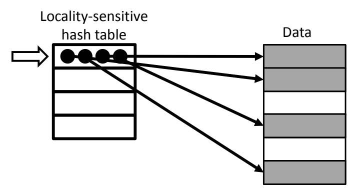

Figure 15: Data accesses in LSH are randomly distributed

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 15).

We have built a 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. For simplicity, we assume 8KB data items, and calculate the hamming distance between the query data and each of the items in the hash bucket. The software sends a stream of addresses from a hash bucket along with the query data page, and the system returns the index of the data item most closely matching the query. 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.

Evaluation: 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 aims 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. Since performance should

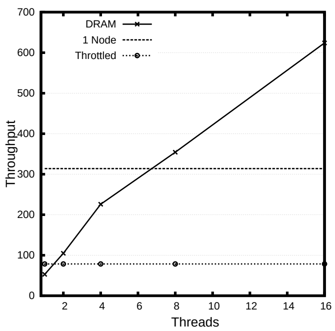

Figure 16: Nearest neighbor with BlueDBM up to two nodes

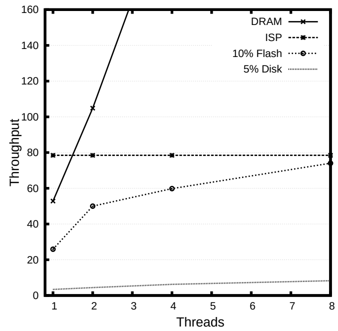

Figure 17: Nearest neighbor with mostly DRAM

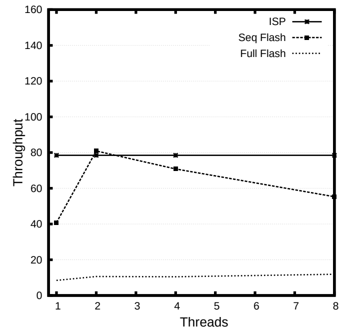

Figure 18: Nearest neighbor with off-the-shelf SSD

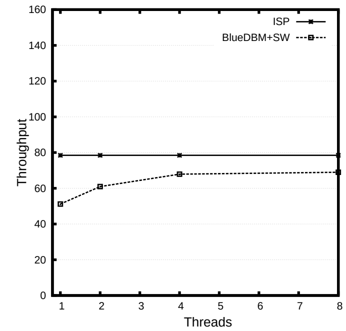

Figure 19: Nearest neighbor with in-store processing

scale linearly with the number of nodes for this application, we concentrated on various configurations in a single node setting:

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

Figure 16 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) BlueDBM can keep up with DRAM-resident data for up to 4 threads, because 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 very high, DRAM-based processing wins with enough resources. (2) Native flash speed matters i.e., when flash performance is throttled to 1/4th 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 17, 18, 19 using throttled BlueDBM as the baseline.

Results of DRAM + 5% Disk and DRAM + 10% Flash experiments shown in Figure 17 show that the performance of ram cloud (H-DRAM) falls off very sharply if even a small fraction of data does not reside in DRAM. Assuming 8 threads, the performance drops from 350K Hamming Comparisons per second to $<$ 80K and $<$ 10K Hamming Comparisons per second for DRAM + 10% Flash and DRAM + 5% Disk, respectively. At least one commercial vendor has observed similar phenomena and claimed that even when 40% of data fits on DRAM, performance of Hadoop decreases by an order of magnitude [10]. 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 18 showed that its performance is poor as 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 19 comparing Baseline-T and H-F Throttled 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 while the in-store processor can process data at full flash bandwidth, the software will be bottlenecked by the PCIe bandwidth at 1.6GB/s. 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.

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 very 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 advantages 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

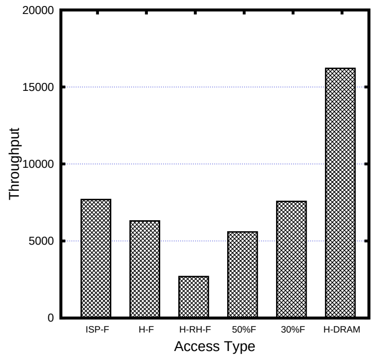

Figure 20: Graph traversal performance

the next request, like a linked-list traversal at the page level. Since such traversals are very sensitive to latency, we conducted the experiments with settings that are very similar to the settings in Section 6.4.

1. IPS-F: In-store processor requests data from remote storage over integrated network
2. H-F: Software requests data from remote storage over integrated network
3. H-RH-F: Software requests data from remote software to read from flash
4. DRAM + 50% F: Store requests data from remote software. 50% chance of hitting flash
5. DRAM + 30% F: Store requests data from remote software. 30% chance of hitting flash
6. H-DRAM: Software requests data from remote software. Data read from DRAM

As expected the results in Figure 20 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 *H-F* and *H-RH-F* illustrates the benefits of using the integrated network to reduce a layer of software access. Performance of *ISP-F* compared to *H-F* 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 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 [31] 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 4 read commands can saturate a single flash bus, we use 4 engines per bus to maximize the flash bandwidth. Only search results are returned to the server.

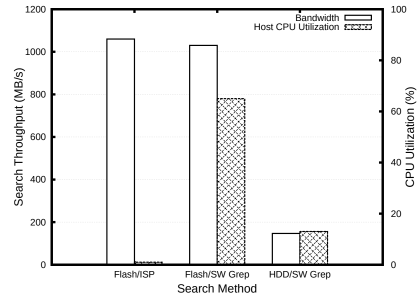

Figure 21: String search bandwidth and CPU utilization

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 21. 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 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 are returned, which we assume is a tiny fraction of the file (0.01% is used in our experiments). This is 7.5x 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 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

We have presented BlueDBM, an appliance for Big Data analytics that uses flash storage, in-store processing and integrated networks for cost-effective analytics of large datasets. A rack-size BlueDBM system is likely to be an order of magnitude cheaper and less power hungry than a cloud based system with enough DRAM to accommodate 10TB to 20TB of data. We have demonstrated the performance benefits of BlueDBM using simple examples on large amounts of data in comparison to a generic flash-based system without such architectural improvements. We have also shown that the performance of a system which relies on data being resident in DRAM, falls rapidly if even a small fraction of data has to reside in secondary storage. BlueDBM like architecture does not suffer from this problem because flash based systems with 10TB to 20TB of storage are very affordable.

Our current implementation uses an FPGA to implement most of the new architectural features, that is, in-store processors, integrated network routers, 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 or planning to develop several new applications including: *SQL Database Acceleration* by offloading query processing and filtering to in-store processors, *Sparse-Matrix Based Linear Algebra Acceleration* and *BlueDBM-Optimized MapReduce*, which attempts to optimize data flow of MapReduce to best fit an SSD-based cluster with in-store processors. We plan to collaborate with other research groups to explore more applications.

9. Acknowledgements

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.

References

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, “Design tradeoffs for ssd performance,” in *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, ser. ATC’08. Berkeley, CA, USA: USENIX Association, 2008, pp. 57–70. Available: http://dl.acm.org/citation.cfm?id=1404014.1404019

[2] I. T. Association, *Infiniband*, 2014 (Accessed November 18, 2014). Available: http://www.infinibandta.org

[3] J. Banerjee, D. Hsiao, and K. Kannan, “Dbc: A database computer for very large databases,” *Computers, IEEE Transactions on*, vol. C-28, no. 6, pp. 414–429, June 1979.

[4] A. M. Caulfield, A. De, J. Coburn, T. I. Mollow, R. K. Gupta, and S. Swanson, “Moneta: A high-performance storage array architecture for next-generation, non-volatile memories,” in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO ’43. Washington, DC, USA: IEEE Computer Society, 2010, pp. 385–395. Available: http://dx.doi.org/10.1109/MICRO.2010.33

[5] A. M. Caulfield and S. Swanson, “Quicksan: A storage area network for fast, distributed, solid state disks,” *SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 464–474, Jun. 2013. Available: http://doi.acm.org/10.1145/2508148.2485962

[6] B. Y. Cho, W. S. Jeong, D. Oh, and W. W. Ro, “Xsd: Accelerating mapreduce by harnessing the gpu inside an ssd,” 2013.

[7] E. S. Chung, J. D. Davis, and J. Lee, “Linqits: Big data on little clients,” in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA ’13. New York, NY, USA: ACM, 2013, pp. 261–272. Available: http://doi.acm.org/10.1145/2485922.2485945

[8] J. Dai, “Toward efficient provisioning and performance tuning for hadoop,” *Proceedings of the Apache Asia Roadshow*, vol. 2010, pp. 14–15, 2010.

[9] J. Do, Y.-S. Kee, J. M. Patel, C. Park, K. Park, and D. J. DeWitt, “Query processing on smart ssds: Opportunities and challenges,” in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD ’13. New York, NY, USA: ACM, 2013, pp. 1221–1230. Available: http://doi.acm.org/10.1145/2463676.2465295

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

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

[12] A. Gionis, P. Indyk, R. Motwani *et al.*, “Similarity search in high dimensions via hashing,” in *VLDB*, vol. 99, 1999, pp. 518–529.

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

[14] S. Hardock, I. Petrov, R. Gottstein, and A. Buchmann, “Noftl: Database systems on ftl-less flash storage,” *Proc. VLDB Endow.*, vol. 6, no. 12, pp. 1278–1281, Aug. 2013. Available: http://dx.doi.org/10.14778/2536274.2536295

[15] M. Honda, F. Huici, C. Raiciu, J. Araujo, and L. Rizzo, “Rekindling network protocol innovation with user-level stacks,” *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 52–58, Apr. 2014. Available: http://doi.acm.org/10.1145/2602204.2602212

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

[17] N. S. Islam, M. W. Rahman, J. Jose, R. Rajachandrasekar, H. Wang, H. Subramoni, C. Murthy, and D. K. Panda, “High performance rdma-based design of hdfs over infiniband,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC ’12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 35:1–35:35. Available: http://dl.acm.org/citation.cfm?id=2388996.2389044

[18] Z. István, L. Woods, and G. Alonso, “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, 2014, pp. 1567–1578.

[19] E. Y. Jeong, S. Woo, M. Jamshed, H. Jeong, S. Ihm, D. Han, and K. Park, “mtcp: A highly scalable user-level tcp stack for multicore systems,” in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI’14. Berkeley, CA, USA: USENIX Association, 2014, pp. 489–502. Available: http://dl.acm.org/citation.cfm?id=2616448.2616493

[20] S.-W. Jun, M. Liu, K. E. Fleming, and Arvind, "Scalable multi-access flash store for big data analytics," in *Proceedings of the 2014 ACM/SIGDA International Symposium on Field-programmable Gate Arrays*, ser. FPGA '14. New York, NY, USA: ACM, 2014, pp. 55–64. Available: http://doi.acm.org/10.1145/2554688.2554789

[21] S.-H. Kang, D.-H. Koo, W.-H. Kang, and S.-W. Lee, "A case for flash memory ssd in hadoop applications," *International Journal of Control and Automation*, vol. 6, no. 1, 2013.

[22] Y. Kang, Y.-s. Kee, E. L. Miller, and C. Park, "Enabling cost-effective data processing with smart ssd," in *Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on*. IEEE, 2013, pp. 1–12.

[23] K. Keeton, D. A. Patterson, and J. M. Hellerstein, "A case for intelligent disks (idisks)," *SIGMOD Rec.*, vol. 27, no. 3, pp. 42–52, Sep. 1998. Available: http://doi.acm.org/10.1145/290593.290602

[24] M. King, J. Hicks, and J. Ankcorn, "Software-driven hardware development," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: ACM, 2015, pp. 13–22. Available: http://doi.acm.org/10.1145/2684746.2689064

[25] O. Kocberber, B. Grot, J. Picorel, B. Falsafi, K. Lim, and P. Ranganathan, "Meet the walkers: Accelerating index traversals for in-memory databases," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-46. New York, NY, USA: ACM, 2013, pp. 468–479. Available: http://doi.acm.org/10.1145/2540708.2540748

[26] S. Lee, J. Kim, and Arvind., "Refactored design of i/o architecture for flash storage," *Computer Architecture Letters*, vol. PP, no. 99, pp. 1–1, 2014.

[27] S.-W. Lee, B. Moon, C. Park, J.-M. Kim, and S.-W. Kim, "A case for flash memory ssd in enterprise database applications," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 2008, pp. 1075–1086.

[28] H. Leilich, G. Stiege, and H. C. Zeidler, "A search processor for data base management systems," in *Fourth International Conference on Very Large Data Bases, September 13-15, 1978, West Berlin, Germany.*, 1978, pp. 280–287.

[29] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda, "High performance rdma-based mpi implementation over infiniband," in *Proceedings of the 17th Annual International Conference on Supercomputing*, ser. ICS '03. New York, NY, USA: ACM, 2003, pp. 295–304. Available: http://doi.acm.org/10.1145/782814.782855

[30] V. Memory, *Violin Memory*, 2014 (Accessed November 18, 2014). Available: http://www.violin-memory.com

[31] J. Morris Jr and V. Pratt, *A linear pattern-matching algorithm*, 1970.

[32] R. Mueller, J. Teubner, and G. Alonso, "Streams on wires: A query compiler for fpgas," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 229–240, Aug. 2009. Available: http://dx.doi.org/10.14778/1687627.1687654

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

[34] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman, "The case for ramclouds: Scalable high-performance storage entirely in dram," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 4, pp. 92–105, Jan. 2010. Available: http://doi.acm.org/10.1145/1713254.1713276

[35] E. A. Ozkarahan, S. A. Schuster, and K. C. Smith, "RAP - an associative processor for database management," in *American Federation of Information Processing Societies: 1975 National Computer Conference, 19-22 May 1975, Anaheim, CA, USA*, 1975, pp. 379–387. Available: http://doi.acm.org/10.1145/1499949.1500024

[36] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. Prashanth, G. Jan, G. Michael, H. S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Yi, and X. D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," *SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 13–24, Jun. 2014. Available: http://doi.acm.org/10.1145/2678373.2665678

[37] M. W.-u. Rahman, N. Islam, X. Lu, J. Jose, H. Subramoni, H. Wang, and D. Panda, "High-performance rdma-based design of hadoop mapreduce over infiniband," in *International Workshop on High Performance Data Intensive Computing (HPDIC), in conjunction with IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2013.

[38] M. W.-u. Rahman, X. Lu, N. S. Islam, and D. K. D. Panda, "Homr: A hybrid approach to exploit maximum overlapping in mapreduce over high performance interconnects," in *Proceedings of the 28th ACM International Conference on Supercomputing*, ser. ICS '14. New York, NY, USA: ACM, 2014, pp. 33–42. Available: http://doi.acm.org/10.1145/2597652.2597684

[39] S. M. Rumble, A. Kejriwal, and J. Ousterhout, "Log-structured memory for dram-based storage," in *Proceedings of the 12th USENIX Conference on File and Storage Technologies*, ser. FAST'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 1–16. Available: http://dl.acm.org/citation.cfm?id=2591305.2591307

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

[41] S. Seshadri, M. Gahagan, S. Bhaskaran, T. Bunker, A. De, Y. Jin, Y. Liu, and S. Swanson, "Willow: A user-programmable ssd," in *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 67–80. Available: http://dl.acm.org/citation.cfm?id=2685048.2685055

[42] M. Singh and B. Leonhardi, "Introduction to the ibm netezza warehouse appliance," in *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research*, ser. CASCON '11. Riverton, NJ, USA: IBM Corp., 2011, pp. 385–386. Available: http://dl.acm.org/citation.cfm?id=2093889.2093965

[43] J. Spiegel, M. McKenna, G. Lakshman, and P. Nordstrom, "Method and system for anticipatory package shipping," Dec. 27 2011, uS Patent 8,086,546. Available: http://www.google.com/patents/US8086546

[44] B. Sukhwani, H. Min, M. Thoennes, P. Dube, B. Iyer, B. Brezzo, D. Dillenberger, and S. Asaad, "Database analytics acceleration using fpgas," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '12. New York, NY, USA: ACM, 2012, pp. 411–420. Available: http://doi.acm.org/10.1145/2370816.2370874

[45] D. Technologies, *Diablo Technologies*, 2014 (Accessed November 18, 2014). Available: http://www.diablo-technologies.com/

[46] T. S. Woodall, G. M. Shipman, G. Bosilca, R. L. Graham, and A. B. Maccabe, "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*, ser. EuroPVM/MPI'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 76–85. Available: http://dx.doi.org/10.1007/11846802_18

[47] L. Woods, Z. Istvan, and G. Alonso, "Hybrid fpga-accelerated sql query processing," in *Field Programmable Logic and Applications (FPL), 2013 23rd International Conference on*, Sept 2013, pp. 1–1.

[48] L. Woods, Z. Istvan, and G. Alonso, "Ibex - an intelligent storage engine with support for advanced sql off-loading," in *Proceedings of the 40th International Conference on Very Large Data Bases*, ser. VLDB '14, 2014, pp. 963–974.

[49] L. Wu, A. Lottarini, T. K. Paine, M. A. Kim, and K. A. Ross, "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*, ser. ASPLOS '14. New York, NY, USA: ACM, 2014, pp. 255–268. Available: http://doi.acm.org/10.1145/2541940.2541961