Scalable Multi-Access Flash Store for Big Data Analytics

Sang-Woo Jun[*], Ming Liu[*], Kermin Elliott Fleming[†], Arvind[*]
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139
{wjun, ml, arvind}@csail.mit.edu[*]
{kermin.fleming}@intel.com[†]

ABSTRACT

For many "Big Data" applications, the limiting factor in performance is often the transportation of large amount of data from hard disks to where it can be processed, i.e. DRAM. In this paper we examine an architecture for a scalable distributed flash store which aims to overcome this limitation in two ways. First, the architecture provides a high-performance, high-capacity, scalable random-access storage. It achieves high-throughput by sharing large numbers of flash chips across a low-latency, chip-to-chip backplane network managed by the flash controllers. The additional latency for remote data access via this network is negligible as compared to flash access time. Second, it permits some computation near the data via a FPGA-based programmable flash controller. The controller is located in the datapath between the storage and the host, and provides hardware acceleration for applications without any additional latency. We have constructed a small-scale prototype whose network bandwidth scales directly with the number of nodes, and where average latency for user software to access flash store is less than 70μs, including 3.5μs of network overhead.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Distributed systems*

General Terms

Design Measurement Performance

Keywords

Storage system, Big Data, FPGA networks, SSD, Flash

1. INTRODUCTION

We have entered the "Big Data" age. The evolution of computer networks and the increasing scale of electronic integration into our daily lives has lead to an explosion of data to be analyzed. Thanks to the steady pace of Moore's Law, our computing abilities on these data have been growing as well. However, effective computation requires a very low-latency random access into the data. As a result, it is highly desirable for the entire working set of the problem to fit in main memory to achieve good performance.

However, modern Big Data application datasets are often too large to be cached in the main memory of any host at a reasonable cost. Instead, they are spread among multiple machines in a cluster interconnected with some network fabric, and often also stored in a cheaper, higher density secondary storage such as hard disks. This means data often has to be accessed from a secondary storage device over a network, each of which has significantly higher access latency than local main memory. The primary performance bottleneck is the seek time of magnetic disks, which has to be amortized by large sequential data access. As a result, the storage device characteristics in large part dictated the design of the rest of the system.

With the recent advancement of low latency and high bandwidth flash devices as alternatives to disks, the performance bottleneck has shifted from the storage device to the network latency and software overhead. As a result, modern high performance storage systems need to optimize all aspects of the system, including the storage, network and software stack. Current attempts to increase the performance of storage systems include use of hardware implementation of the network stack and better I/O fabric. However, even state-of-the-art networked storage systems still suffer hundreds of microseconds of latency. This large gap between the performance of main memory and storage often limits our capacity to process large amounts of data.

Another facet of high performance storage systems under active investigation is providing a computation fabric on the storage itself, effectively transporting computation capabilities to where the storage is, instead of moving large amounts of data to be processed. However, the processing power that can be put on a storage device within the power budget is often limited, and its benefits are sometimes limited when heavy computation is required. For applications that require heavy computation, it is effective to use hardware acceleration to assist data processing. Due to the high development and production cost of dedicated ASIC accelerator chips, reconfigurable hardware fabrics such as FPGAs are popular choices for implementing power-efficient application-specific accelerators.

In this work, we propose a novel high-performance storage architecture, which we call BlueDBM (Blue Database Machine). The high-level goal of BlueDBM is to provide a high-performance storage system that accelerates the processing of very large datasets. The BlueDBM design aims to achieve the following goals:

- **Low Latency, High Bandwidth**: To increase the performance of response-time sensitive applications, the network should add negligible latency to the overall system while maintaining high bandwidth.

- **Scalability**: Because Big Data problems are constantly increasing in size, the architecture should be scalable to higher capacity and node count.

- **Low-Latency Hardware Acceleration**: In order to reduce data transport and alleviate computationally bound problems, the platform should provide very low-latency hardware acceleration.

- **Application Compatibility**: As a general storage solution for Big Data, existing applications should run on top of our new storage hardware without any modification.

- **Multi-accessibility**: In order to accommodate distributed data processing applications, the system should be capable of handling multiple simultaneous requests from many different users.

The BlueDBM architecture distributes high performance flash storage among computational nodes to provide a scalable, high-performance and cost-effective distributed storage. In order to achieve this, BlueDBM introduces a low-latency and high-speed network directly between the flash controllers. The direct connection between controllers not only reduces access latency by removing the network software stack, but also allows the flash controllers to mask the network latency within flash access latency. As we will demonstrate, controller-to-controller latencies in such a network can be insignificant compared to flash access latencies, giving us the potential to expose enormous storage capacity and bandwidth with performance characteristics similar to a locally attached PCIe flash drive.

To further improve the effectiveness of the storage system, BlueDBM includes a FPGA-based reconfigurable fabric for implementing hardware accelerators near storage. Because the reconfigurable fabric is located in the datapath of the storage controller through which data has to travel anyways, no latency overhead is introduced from using the accelerators.

The key contribution of this paper is a novel storage architecture for Big Data applications, which include a low-latency communication link directly between flash controllers and a platform for accelerator implementation on the flash controller itself. We demonstrate the characteristics of such an architecture on a 4-node prototype system. We are also engaged in building a much larger system based on the architecture.

To test these ideas, we have constructed a small 4-node prototype of our system using commercially available FPGAs [10] coupled to a custom flash array, networked using high speed inter-FPGA serial links. The prototype has an average latency to client applications of about 70μs, which is an order of magnitude lower than existing distributed flash systems such as CORFU [13], and rivals the latency of a local SSD. Our shared 4-node prototype provides 4x the bandwidth of a single flash card with marginal impact on access latency. We also implemented a word counting application with hardware accelerator support from our storage platform, showing 4x performance increase over a pure software implementation. We are currently building a newer system employing the same ideas but with more modern hardware, which will deliver an order of magnitude performance increase over the prototype system.

The rest of the paper is organized as follows. Section 2 provides background on storage deployment and FPGA based acceleration in Big Data. Sections 3 to 5 describes the system in detail, and Section 6 discusses the prototype we have built to demonstrate the performance of the system. Section 7 provides the experimental results obtained from the prototype and its evaluation. Section 8 concludes the paper.

2. RELATED WORK

Storage systems that require high capacity are usually constructed in two ways: (i) building a Storage Area Network (SAN) or (ii) using a distribute file system. In a SAN, large amounts of storage are placed in a storage node such as a RAID server, and these storage devices are connected together using a dedicated network (i.e. SAN), providing the abstraction of locally attached disk to the application servers. However, the physical storage network is usually ethernet based running on protocols such as iSCSI or FCoE, which adds milliseconds of software and network latency. An alternative organization is to distribute the storage among the application hosts and use the general purpose network along with a distributed file system (e.g., NFS [7], Lustre [6], GFS [18]) to provide a file-level sharing abstraction. This is popular with distributed data processing platforms such as MapReduce [3]. While a distributed file system is cheap and scalable, the software overhead of concurrency control and the high-latency congestion-prone general purpose network degrades performance. Nevertheless, traditionally, these network, software and protocol latencies are tolerable because they are insignificant compared to the seek latency of magnetic disks.

This is changing with recent developments in high-performance flash devices. Large flash storage offer two benefits over magnetic disks, namely superior random read performance and low power consumption, while still providing very high density. Such advantages make them alternatives to magnetic disks. Flash chips offer access latency in the order of tens of microseconds, which is several orders of magnitude shorter than the 10 to 20 millisecond disk seek time. By organizing multiple chips in parallel, very high throughput can be obtained. As a result, the storage device is no longer a bottleneck in high capacity storage systems. Instead, other parts of the system such as network latency and software stack overhead now have a prominent impact on performance. It has been shown that in a disk-based distributed storage system, non-storage components are responsible for less than 5% of the total latency, while in a flash-based system, this number rises to almost 60% [14].

Flash has its own drawbacks as well. Its characteristics include limited program erase cycles, coarse-grain block level erases, and low write throughput. As a result of these characteristics, hardware (e.g., controllers, interfaces) and software (e.g., file systems, firmware) traditionally designed for hard disks are often suboptimal for flash. Much research has gone into developing techniques such as intelligent address translation in the Flash Translation Layer (FTL) to control area under use [12] [24] [26]. Our storage architecture is similarly motivated by and designed for these flash characteristics.

Recent efforts such as CORFU [13] attempts to build distributed file systems tailored for flash storage characteristics, but still suffers millisecond-level latency. Other attempts such as QuickSAN [14] have studied directly connecting flash storage to the network in order to bypass some of the software latency. This brings down the latency of the system to hundreds of microseconds. We hope to further improve performance by removing protocol and software overhead.

In data centers, several research efforts have suggested providing side-channels for communication between nodes within the data center to alleviate and bypass network congestion. [19] attempts to resolve congestion using software architectural approaches. Halperin et al.[20] examine adding wireless links to data-centers as an auxiliary communication mechanism.

Moving computation to data in order to circumvent the I/O limitations has been proposed in the past. Computation in main memory (e.g. Computational RAM [16]) has been studied, but it failed to see much light due to the fast advancement of I/O interfaces. However, in light of power consumption walls and Big Data, moving computation to high capacity secondary storage is becoming an attractive option. Samsung has already demonstrated the advantages of having a small ARM processor on the storage device itself [22] for in store computation. They have shown power and performance benefits of offloading I/O tasks from host CPU to the storage device. However, benefits are only seen if the offloaded task have low computing complexity and high data selectivity because of the weak ARM processor.

FPGAs have been gaining popularity as application specific accelerators for Big Data due to its low power consumption, flexibility and low cost. FPGA accelerators are currently being used in database management systems [4], in providing web services [15], as well as in other domains such as machine learning [23] and bioinformatics [27].

3. SYSTEM ARCHITECTURE

The distributed flash store system that we propose is composed of a set of identical storage nodes. Each node is a flash storage device coupled with a host PC via a high-speed PCIe link. The storage device consists of flash chips organized into busses, controlled by a flash controller implemented on reconfigurable FPGA fabric. The storage devices are networked via a dedicated storage network implemented on multi-gigabit low latency serial links using SERialize/DESerializer (SERDES) functionality provided within the FPGA fabric. The host servers are networked using generic Ethernet communications. The construction of the system is shown in Figure 1.

To use BlueDBM, the host PCs run high level applications (e.g. databases) which generate read/write commands to the file system. The file system forwards the requests to the locally attached FPGA, which fulfills the requests by accessing either the local or remote flash boards. Data is globally visible and accessible from all host PCs, and the address space is shared and unified among all nodes. Alternatively, the application may issue commands to instruct a hardware accelerator on the FPGA to process the data directly from the flash controllers.

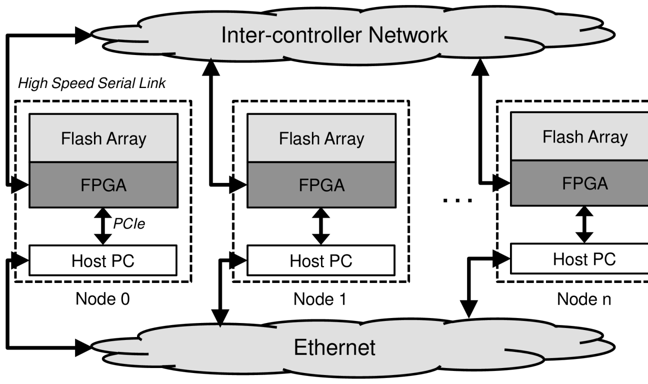


Figure 1: BlueDBM top level system diagram consisting of multiple storage nodes connected using high speed serial links forming an inter-controller network

This organization fulfills our goal of (i) low latency/high bandwidth by using parallel flash chips, PCIe and high-speed transceivers coupled with a thin networking protocol; (ii) scalability through homogeneous nodes and a network protocol that maintains low latency over multiple hops and is topologically flexible; (iii) low-latency hardware acceleration by providing a hook to software to invoke accelerator operations on data without passing through host; (iv) application compatibility by providing a generic file system and exposing the abstraction of a single unified address space to the applications; and (v) multi-accessibility by providing multiple entry points to storage via many host PCs.

The hardware and software stacks of the system are presented in Figure 2. Hardware running on the FPGA has 5 key components: (i) client interface, (ii) address mapper, (iii) flash controller, (iv) inter-FPGA router and (v) accelerator. The client interface handles the communication of data and commands with the host over PCIe. Together with the driver and file system software on the host, they provide a shared unified address space abstraction to the user application. The address mapper maps areas in the logical address space to each node in the network. The flash controller includes a simple flash translation layer to access the raw NAND chips on the flash board. The router component implements a thin protocol for communication over the high speed inter-FPGA SERDES links. Finally, accelerators may be placed before and after the router for local or unified access to data. The flash controller and related components are explain in detail below. The inter-FPGA network and accelerators are explained separately in the next sections.

3.1 File System, Client Interface and Address Mapper

The client interface module on the FPGA works in concert with the driver and file system software on the host server to handle I/O requests and responses over PCIe. We implemented a generic file system using FUSE [1]. FUSE intercepts file system command made to its mount point and

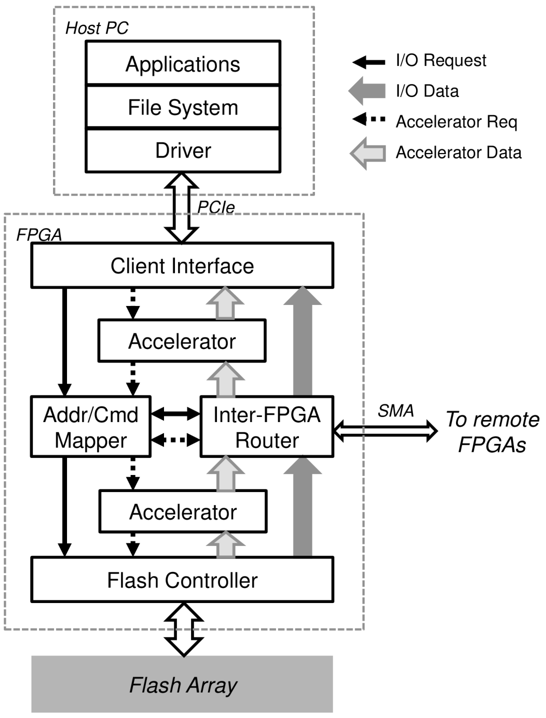


Figure 2: Hardware and software stack of a single node

allows us to convert file system commands into load/store operations to our flash device. Currently the entire combined storage of all nodes in the system is translated to a single flat address space.

For each I/O request, the address mapper module determines which storage node the data resides in. All storage nodes in the system need to agree on the same mapping scheme, which is currently defined programmatically. Due to the low latency serial communication fabric, there is little difference in performance between fetching from a local node or fetching from a remote node. Therefore, the current mapping scheme focuses on utilizing as much device parallelism as possible, by striping the address space such that adjacent page addresses are mapped to different storage nodes.

3.2 Flash Controller

We use a flash array for fast random access storage. These arrays are populated with NAND chips organized into several buses with chips on each bus sharing control and data paths. The flash board exposes raw chip interfaces to the FPGA via GPIOs and the flash controller accesses the chips. The architecture of the flash controller is shown in Figure 3. We use an independent chip controller and scheduler per bus to communicate with the raw flash chips. Not only can buses be accessed in parallel, data operations on different flash chips on the same bus may be overlapped to hide latency. We designed the flash controller to take advantage of these properties. In addition, we use a tag renaming table and a data switch to manage sharing among multiple hosts in a distributed setting.

The system includes a simple implementation of the Flash Translation Layer (FTL). The current FTL focuses only on providing maximum read performance through parallel access to as many flash chips as possible. We achieve this by simply permutating a portion of the logical address bits such that spatially local requests have a high probability of being mapped to different bus controllers, buses and chips thereby improving overall system throughput.

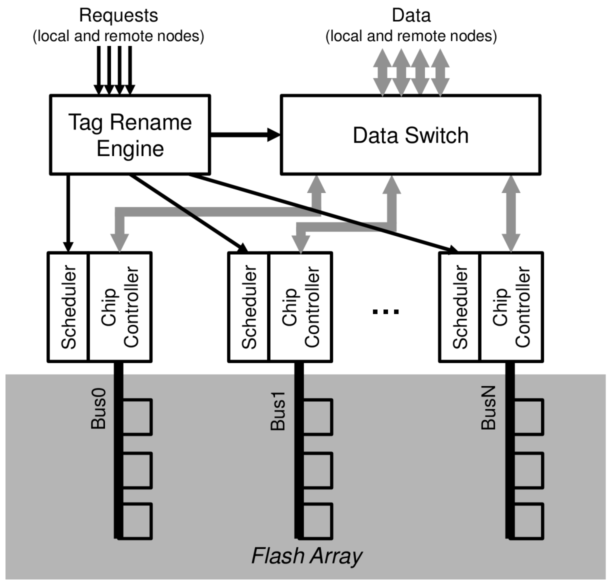


Figure 3: Flash controller featuring a scheduler and chip controller per bus, virtualized using a tagging mechanism

3.3 Controller Virtualization and Communication

Because multiple clients can access all storage in the system through a very thin layer of controllers, there needs to be an efficient way to match the commands against the data flowing in and out of the flash chips. For example, data being read from the flash device needs to be routed back to where the read command originated. A possible solution is to implement a distributed agreement protocol between each node, but this is complex and requires additional data transfer across the network. Instead, we use a two-layer tagging scheme to keep track of this information.

In the first layer, each command that is issued from a client interface is given a 8-bit tag value. A list of unoccupied tags are kept in a free tag queue. We dequeue when a new request is issued, and enqueue back when a request retires. On the command issuer side, this tag is correlated with information such as the request page address in a tag mapping table structure. When the request needs to be processed at a remote node, the tag is sent to the target node with the rest of the request information. However, because each node keeps a separate list of free tags, there can be tag collisions at the remote node. This is solved using a second layer of tagging scheme, which translates the original tag to the target node's unique local tag. The first layer tag is stored in another tag map table with information such as the request source and the original tag value. After the request has been handled, the data is sent back to the request origin node tagged with the original tag value it was requested with, so it can be reused for future operations.

4. INTER-CONTROLLER NETWORK

Conventional computer networking infrastructures, like Ethernet and TCP, are designed to provide general purpose functionality and operate at large scales over long distances. As a result, they come at a cost of significant time and processing overhead. But this cost was often overlooked in the past when constructing SANs, because the latency of magnetic disks dominated over the network infrastructure . However, in the case of flash-based deployment, such networking overhead becomes a serious issue. Furthermore, conventional method of networking storage devices requires the storage traffic to share the host-side network infrastructure. This results in reduced effective bandwidth, because the link between the host and its storage has to be shared for local and remote data. Finally, because the network and storage management are composed separately, the combined latency adds up to hundreds of microseconds of latency.

BlueDBM solves these issues by having a dedicated storage data network directly connecting the flash controllers to each other. For this purpose, we constructed a simple packet-switched storage network protocol over a high-speed serial link provided by the FPGA package. The protocol was implemented completely inside the FPGA, and provides sub-microsecond latency communication between controllers. Because the flash controller manages the storage device as well as the network, all data transport of words within a page could be pipelined, effectively hiding the network latency of accessing a page. The protocol includes a flooding peer discovery functionality, allowing hot-plugging of nodes into any topology required by the application.

4.1 Routing Layer

Figure 4 depicts the architecture of the router. The routing mechanism for our storage network is a packet-switched protocol that resembles a simplified version of the Internet Protocol. Each node maintains a routing table of all nodes in the network, where each row contains information including which physical link a packet should take to get to that node and how many network hops away it is. This table is populated on-line via a flooding discovery protocol. This allows hotplugging nodes into the network while the system is live, and also to automatically maintain the shortest path between all pairs of nodes.

The networking infrastructure is constructed such that the flash controller or accelerators can declare in code their own virtual communications links of various widths, and the router will organize and schedule packets in order to multiplex all of the virtual lanes onto a single physical link. This not only allows the writing of simple, easy to understand code for the network infrastructure, but also provides a clean and efficient abstraction of the network for the accelerator platform. An accelerator can declare, at compile time, multiple virtual links according to its requirements, reducing the burden of network management in accelerator development.

4.2 Physical Layer

In our current implementation we make use of the high-speed serial transceivers provided in the FPGA silicon as the physical link. The transceivers provide not only high bandwidth and low latency, they also provide relatively reliable data transport over up to two meters, which is sufficient in data center racks.

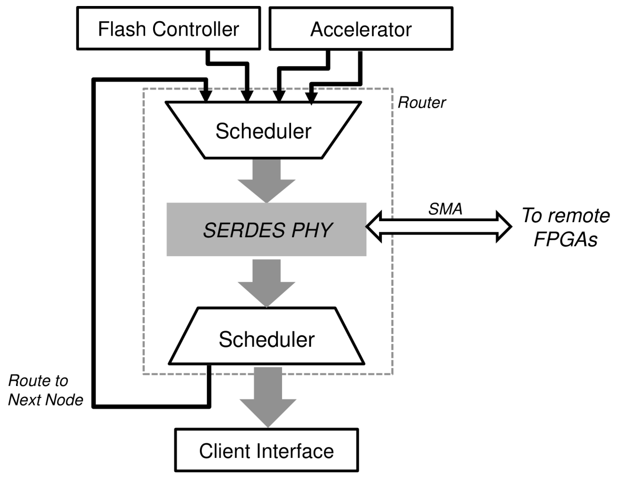


Figure 4: Inter-node network router.

Please note that the choice of physical communication fabric in our system is flexible. While in this particular system, we do not choose general-purpose media such as ethernet or Infiniband [5] as a physical transport for performance reasons, we could have easily chosen other such high-speed communications fabrics given that it is supported by the FPGA or hardware. Heterogeneous controller networks can also be constructed. For example, ethernet could be used across racks in a data center, while high speed inter-FPGA serial links can be used within a rack.

5. CONTROLLER AS A PROGRAMMABLE ACCELERATOR

In order to enable extremely low latency acceleration, our system provides a platform for implementing accelerators along the datapath of the storage device (Figure 5, right). One advantage is that operations on the data can be completed faster with dedicated hardware. In addition, because the combined throughput of the BlueDBM cluster can easily surpass the bandwidth of any single hostside link (i.e. PCIe), accelerators that filter or compress data can be used to process more data than the hostside link fabric allows. This setup is more advantageous compared to using the accelerator as a separate appliance (Figure 5, left), where data must be transported from storage to the accelerator via the host, and then transported back after computation.

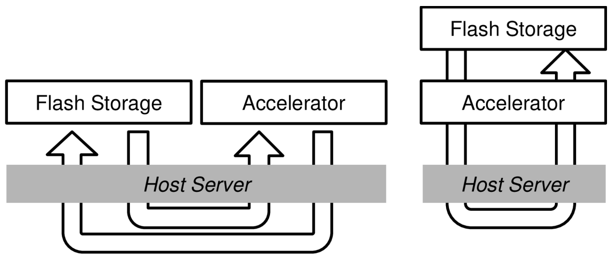


Figure 5: Flow of data when using an accelerator as a separate appliance (left) versus an accelerator in the data path of the storage device (right)

5.1 Two-Part Implementation of Accelerators

In BlueDBM, accelerators can be implemented both before and after the inter-FPGA router (Figure 2). The *local* accelerator, which is located between the flash storage and the router, is used to implement functions that only require parts of the data. For example, compressing pages before writing them to flash. The *global* accelerator located between the router and the client interface implements higher-level functionalities that require a global view of data. Examples includes table join operations in a database accelerator, or the word counting example that will be described shortly. Both accelerators can work in concert, for example to implement compression and decompression algorithms, to reduce the amount of data transported over the inter-FPGA link.

5.2 Example Accelerators

To demonstrate the accelerator architecture, we have implemented a simple word counting accelerator on the BlueDBM platform. The accelerator exposes an interface to specify the word being counted, in the form a FUSE virtual file. Once the word is registered, the accelerator accesses its virtual access points to storage and network to count the number of the registered word in all storage devices in the network. The resulting output can also be accessed via a FUSE virtual file. An example invocation of the accelerator looks like the following:

```
echo "would" > fuse/input ; cat fuse/output
```

We have already implemented other effective application specific FPGA-based hardware accelerators serving as separate appliances to a host machine. These include application specific compression [21], database query accelerators and network link compression algorithms. Most of these can be ported to the BlueDBM platform with minor modifications, and we are in the process of doing this. We expect both compute and data bound applications to see notable performance improvements with these accelerators.

6. PROTOTYPE SYSTEM

Our prototype flash system, a photo of which is shown in Figure 6(a), is based around the Xilinx ML605 board and our custom built flash board. The ML605 board and the flash board is coupled using the FPGA Mezzanine Card (FMC) connector, as seen in Figure 6(b), and plugged into a PCIe slot on the host server. The implementation overhead was greatly reduced by building on top of an abstraction layer [25], which mapped physical physical device details into logical services. We have also used similar abstractions of the serial network [17] for early functionality, but eventually implemented a routing protocol in favor of dynamic reconfiguration of the network.

Each flash board hosts 16GB of flash storage arranged in four parallel buses comprised of 8 512MB Micron SLC flash chips. An on-board Xilinx CPLD is used to decode command signals for all buses. Our custom flash board uses slightly older, less dense flash chips with asynchronous interfaces. These older chips with asynchronous signalling lowers the throughput of each flash bus to a maximum of around 25MB/s, even though our controllers can handle chips with much larger bandwidths.

We network the processing nodes of our system by way of the Virtex-6 GTX high speed transceivers. Each transceiver

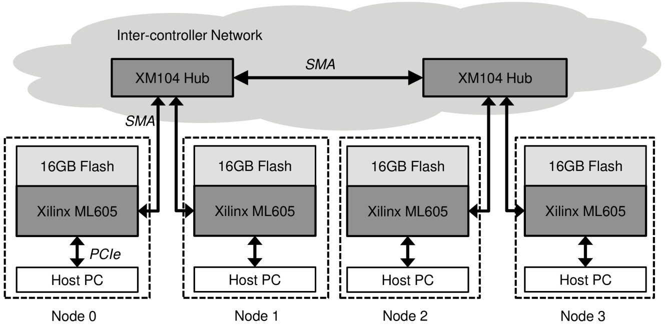


Figure 7: Prototype physical implementation with 4 storage nodes and 2 hubs

is capable of transporting up to 5Gbps. Unfortunately, our flash cards utilize the same mezzanine connector as most of the ML605's serial transceivers. As a result, each node can only connect to one other node via the only remaining SMA port on the ml605 board. Therefore, the prototype uses a tree topology shown in Figure 7, and connects the processing nodes using extra ML605s which act as hubs. These ML605s do not have attached flash cards, enabling us to use most of its transceivers for inter-FPGA communication using the Xilinx XM104 connectivity card.

Hosts in our system run the Ubuntu distribution of Linux. We use the file system FUSE[1] to interface with our storage system though eventually we plan to implement a true file system.

7. RESULTS

Using our prototype system, we first characterize the inter-controller network. Then we examine the raw latency and throughput of the entire storage system. Finally, we measure the performance of simple applications running on the system, taking advantage of multi-accessibility and accelerators.

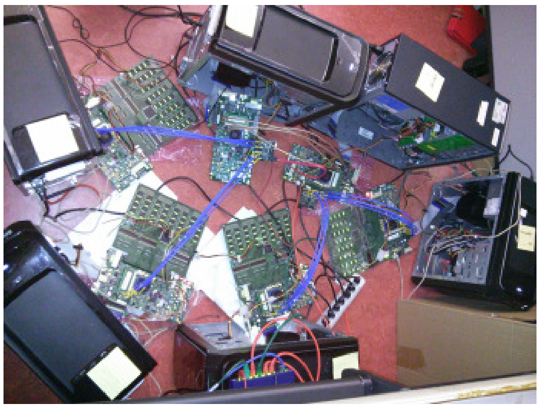
7.1 FPGA Resource Usage

The approximate area breakdown of each node in our flash system is shown in Table 1. Our design, which is largely unoptimized for either timing or area, is dominated by its use of large buffers. This area corresponds to approximately 35% of the resources of the medium sized Virtex-6 chip. The rest of the area is free to be used for accelerator implementation.

Most structures in our system are constant in size regardless of the number of processing nodes in the system. Notable exceptions to this scaling include the routing table and packet header size. However, even with a thousand-node system, we can easily fit the routing table within a few BRAMs on the FPGA given that each entry is merely 128 bits. Packet headers will require 10 bits to identify the source/destination node in a thousand-node system, which means a corresponding increase in FIFO sizes. However, this area increase remains insignificant compared to the rest of the design on the FPGA. Thus we are able to scale to thousands of nodes without significant impact on area.

7.2 Network Performance

Figure 8 summarizes the typical throughput and latency characteristics of our inter-FPGA network architecture. We


(a) Four-node prototype system

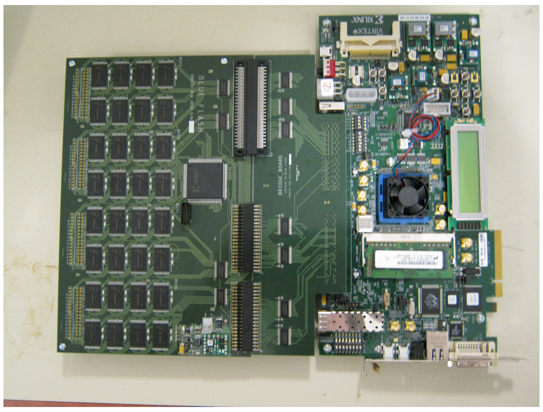

(b) ML605 and attached flash card

Figure 6: Prototype system

	LUTS	Registers	BRAM
Client Interface	17387	17312	51
Flash Controller	10972	8542	151
Networking	24725	27530	16
Total	53084 (35%)	53384 (17%)	218 (52%)

Table 1: Synthesis metrics for controller components at 100MHz.

achieve approximately 450MB/s or 70% of the theoretical link bandwidth with average packet latency of around 0.5μs per hop. Latency scales linearly with the number of hops traversed because we maintain flow-control on a per-hop basis, as opposed maintaining flow-control on the end-to-end traversal. Considering that the typical latency of a flash read is several tens of microseconds, requests in our network can, in theory, traverse dozens of nodes before the network latency becomes a significant portion of the storage read latency, potentially enabling the addressing of multiple terabytes worth of data across many nodes.

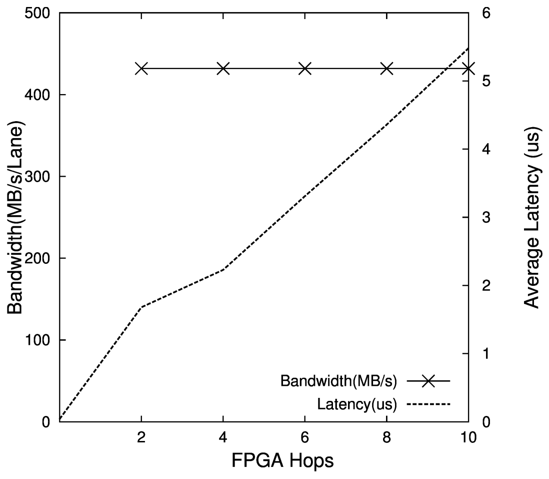


Figure 8: Throughput and latency of our inter-FPGA network using a 5Gbps SERDES connection on the Virtex-6 ML605.

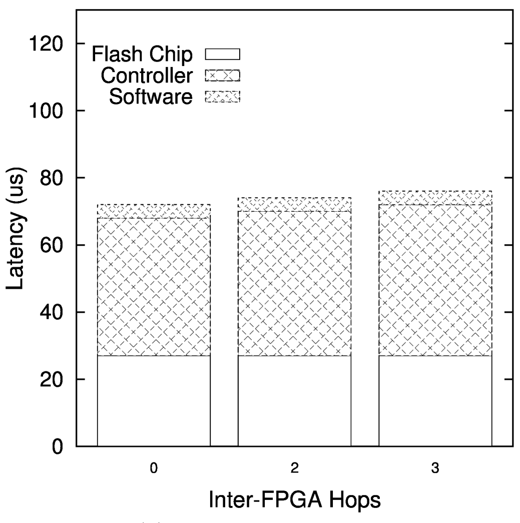
In our current system, each compute node only supports a single lane connection due to physical constraints discussed previously. However, the maximum bandwidth per chip in the latest generation of FPGAs tops 10GB/s per chip for moderately sized FPGAs [11]. Based on this, we believe that BlueDBM can scale to hundreds of processing nodes while delivering average-case performance similar to a good commodity SSD array and best-case performance rivalling or surpassing local PCIe SSDs such as FusionIO [2]. Indeed, we are currently building a much bigger machine that will demonstrate the BlueDBM architecture at a much larger scale.

7.3 Raw Latency

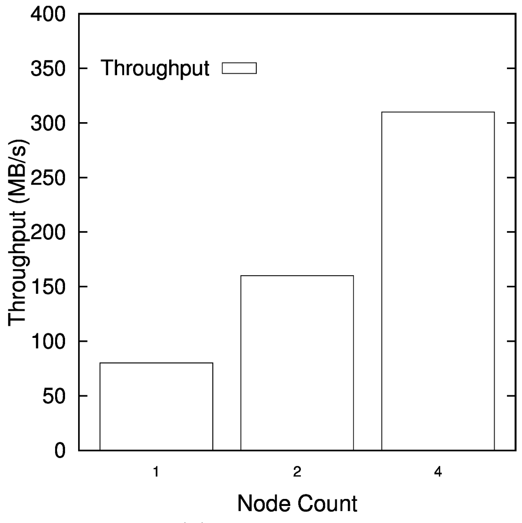
Figure 9(a) shows the average read latency of our storage system from the perspective of the host user application. It is the average time from when a request is made for a single 2048-byte page to when the entire page is received by the application. Latency is measured by making repeated blocking requests one at a time, to random pages, which may reside on different nodes, buses and blocks.

The total read latency can be broken down into flash chip latency, controller latency and software latency. Chip latency is the access time of a single flash chip and is a characteristic of the NAND flash device. Our SLC chips average around 27μs. The controller latency is incurred in moving the data from a flash chip to the appropriate client interface, and includes the inter-FPGA network latency. The software latency accounts for the time to transfer the page across the PCIe bus and through the driver and FUSE file system.

From Figure 9(a), we observe that flash chip latency remains constant with more nodes as expected. Because by construction, the underlying storage network is abstracted away, the driver and file system layers are thin and simple. Their latencies total 4μs and also remains constant with increasing number of nodes. The network latency is minimal compared to flash latency as shown previously. Moreover, the tight coupling between the inter-FPGA network and the flash controller means that the network does not have to wait for the entire page to be read from the controller before sending it. Network latency can be hid by pipelining individual words streamed out of the controller across the network. As a result, the end-to end latency of fetching a page from



(a) Page access latency



(b) Throughput

Figure 9: Raw latency and throughput measurements of our 4-node prototype

remote storage is much less than the sum of storage and network latencies accounted for separately. In our prototype, end-to-end page read latency increase is a marginal 2μs per additional network hop. We expect this trend to continue for larger networks. The total latency of our system is an order of magnitude lower than existing networking solutions such as Ethernet or fibre channel.

7.4 Raw Throughput

Figure 9(b) shows the sequential read throughput of the system with increasing number of distributed nodes. Throughput was measured by running a benchmark on a single host that requests a continuous stream of pages from the total address space in order. Each request is serviced by either a local or a remote node depending on the mapping of the requested address.

The throughput of our system achieves linear scaling with the addition of more storage nodes. A single node provides 80 MB/s of bandwidth. A 2-node system doubles the bandwidth to 160 MB/s, while a 4-node system further scales up to 310 MB/s, or 3.8x the speed of a single node. The reason the performance is not a full 4x is because our prototype implementation of the PCIe driver is hitting its maximum throughput. Future iterations of the system will remove this limitation. It is conceivable that by adding more storage nodes, we can achieve throughput and capacity comparable to commercial SAN or PCIe flash products such FusionIO or PureStorage Flash Array [8], but at a much lower dollars per gigabyte.

It should be noted that the throughput of a single node of the prototype system is limited by the low throughput of the custom flash boards. With modern flash chips ($\tilde{2}$00 MB/s per chip) organized into more buses, we would be able to achieve the same linear scaling at much higher bandwidth, until we saturate the bandwidth of PCIe or the inter-FPGA links. We are currently designing a new flash board to build a faster and larger system for real-world big data applications.

7.5 Multi-Access Performance Scaling

Effective multi-access capability of a storage system is crucial in a distributed processing environment such as MapReduce. We demonstrate this ability by running computationally heavy workloads on multiple consumer nodes and measuring the achieved performance of the system.

Figure 10 shows the performance scaling of the system in a multi-access setting. Throughput is shown normalized against a single-access scenario in a four-node system. It can be seen that the total bandwidth delivered by the system linearly increases with the number of consumers. This is because the total available bandwidth of the system exceeds the amount a single server node could process. This shows that our system is an effective way to share device storage and bandwidth among multiple hosts, where each host may not always require the maximum bandwidth from the storage device, because for example, it is doing computation on the data or waiting for external input.

However, if all nodes are constantly requesting maximum throughput, we will not see linear scaling. In such a case, the total throughput of the system will saturate at the maximum internal bandwidth, after which node throughput will decrease. It is worthy to note that this is a baseline experiment to demonstrate the raw performance of the system, without advanced hot block management features such as DRAM caching or deduplication. After such advanced features are implemented, we expect to show better performance even on bandwidth intensive workloads.

7.6 Application-Specific Acceleration

Figure 11 shows the performance results of the word counting application, implemented with (i) an in-datapath hardware accelerator, (ii) an off-datapath hardware accelerator treated as a separate appliance, and (iii) software only. All experiments were run on the two-node configuration, where the maximum bandwidth is 140MB/s. It can be seen that while the accelerator on the datapath makes almost maximum use of the device bandwidth at 128MB/s, the software implementation of the application is not nearly as fast

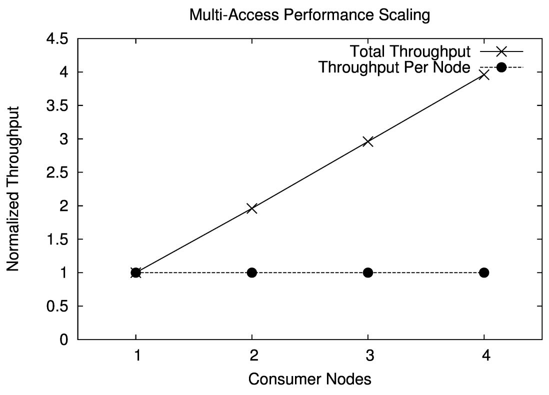


Figure 10: Performance scaling in multi-access scenario

(31MB/s), because it is bound by the CPU. Even the hardware accelerator, implemented as a separate appliance suffers significant throughput loss because of the overhead involved in streaming the fetched data into the accelerator.

Because out host server provides only one PCIe slot, we could not implement the off-datapath accelerator as a physically separate appliance. Instead, the accelerator shares the FPGA fabric and PCIe link of the flash controller. In order to utilize the accelerator, data read from flash storage is transferred back to the FPGA, this time to the accelerator implementation instead of the flash controller. However, even though the flash controller and accelerator share some of the same resources, because they share no control structure inside the FPGA and the direction of heavy data transfer on PCIe is different, we do not think the performance characteristics of this configuration is very different from a physically separate implementation.

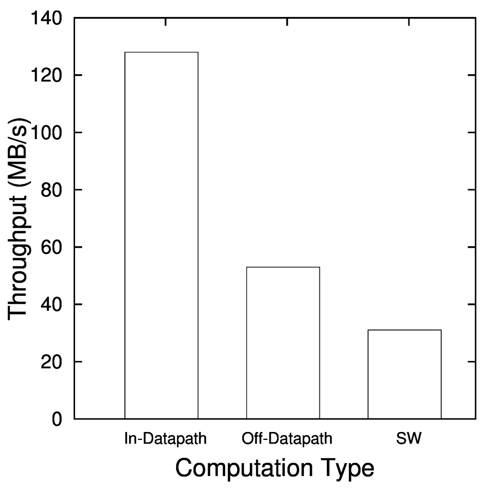


Figure 11: Word counting accelerator performance scaling

8. CONCLUSION AND FUTURE WORK

"Big Data" processing requires high-performance storage architectures. In this paper, we have examined an architecture for a scalable distributed flash store, wherein each node possesses a moderate amount of storage resources and reconfigurable fabric for accelerator implementation, and is connected to other nodes by way of a low-latency and high-bandwidth controller-to-controller network. We have demonstrated that by having the inter-FPGA network connecting the controllers directly, each node is able to access remote storage with negligible performance degradation. Not only does the controller-to-controller network provide pooling of storage capacity, but it also allows combining the throughput of all nodes on the network, resulting in linear throughput scaling with more nodes. We also demonstrated that offloading computation into the storage controller as an accelerator provides performance benefits against implementing acceleration as a separate appliance.

We are in the process of building a 20-node BlueDBM rack-level system using more modern, faster and higher capacity flash boards with newer Xilinx VC707 FPGA boards [11]. The new flash board is planned to deliver more than 1GB/s of throughput per storage node, and the server-side PCIe bandwidth will perform at more than 3GB/s. The new system will be used to explore real Big Data problems at the storage hardware level. Some planned improvements and experimentations include:

Improved FTL: Our current system is designed for read-intensive applications. We have thus far assumed that writes occur infrequently. Our next step is to optimize writes to flash memory by designing wear leveling, garbage collection, write amplification reduction algorithms specifically for a controller networked flash storage system.

DRAM Caching: We can cache reads and writes to the SSD in DRAM on the FPGA board. This can reduce writes to the flash and improve performance. We may use a cache coherence protocol to synchronize the cache of individual nodes. Additionally, because of our low latency inter-FPGA network, we could create a shared global DRAM cache from DRAM of all the nodes and dynamically partition them according to the workload characteristics.

Database Acceleration: Existing applications can already take advantage of BlueDBM's distributed flash store, but we aim to further accelerate database management systems such as Postgres or SciDB [9] by offloading database operations to the FPGA. Specifically, filtering, aggregation and compression tasks could be done directly at the storage level.

We believe our system holds great promise both for high-speed rack-level storage networking and for large-scale application acceleration in Big Data.

9. ACKNOWLEDGEMENTS

This work was funded in part by Quanta (#6922986) and Samsung (#6925093). We also thank Xilinx for their generous donation of hardware.

10. REFERENCES

[1] FUSE: Filesystem in Userspace. `http://fuse.sourceforge.net/`. Accessed: Sept. 2013.

[2] FusionIO. `http://www.fusionio.com`. Accessed: Sept. 2013.

[3] Hadoop Distributed File System. `http://hadoop.apache.org/docs/stable/hdfs_user_guide.html`. Accessed: Sept. 2013.

[4] IBM Netezza. http://www-01.ibm.com/software/data/netezza/. Accessed: Sept. 2013.

[5] Infiniband. http://www.infinibandta.org/. Accessed: Sept. 2013.

[6] Lustre. http://wiki.lustre.org/index.php/Main_Page. Accessed: Sept. 2013.

[7] NFS Specifications. http://tools.ietf.org/html/rfc1094. Accessed: Sept. 2013.

[8] PureStorage FlashArray. http://www.purestorage.com/flash-array/. Accessed: Sept. 2013.

[9] SciDB. http://scidb.org/. Accessed: Sept. 2013.

[10] Virtex-6 FPGA ML605 Evaluation Kit. http://www.xilinx.com/products/devkits/EK-V6-ML605-G.htm. Accessed: Sept. 2013.

[11] Virtex-7 Datasheet. http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf. Accessed: Sept. 2013.

[12] M. Balakrishnan, A. Kadav, V. Prabhakaran, and D. Malkhi. Differential raid: rethinking raid for ssd reliability. In *Proceedings of the 5th European conference on Computer systems*, EuroSys '10, pages 15–26, New York, NY, USA, 2010. ACM.

[13] M. Balakrishnan, D. Malkhi, V. Prabhakaran, T. Wobber, M. Wei, and J. D. Davis. Corfu: a shared log design for flash clusters. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, NSDI'12, pages 1–1, Berkeley, CA, USA, 2012. USENIX Association.

[14] A. M. Caulfield and S. Swanson. Quicksan: a storage area network for fast, distributed, solid state disks. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pages 464–474, New York, NY, USA, 2013. ACM.

[15] S. R. Chalamalasetti, K. Lim, M. Wright, A. AuYoung, P. Ranganathan, and M. Margala. An fpga memcached appliance. In *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*, FPGA '13, pages 245–254, New York, NY, USA, 2013. ACM.

[16] D. Elliott, M. Stumm, W. M. Snelgrove, C. Cojocaru, and R. McKenzie. Computational ram: Implementing processors in memory. *IEEE Des. Test*, 16(1):32–41, Jan. 1999.

[17] K. E. Fleming, M. Adler, M. Pellauer, A. Parashar, Arvind, and J. Emer. Leveraging Latency-Insensitivity to Ease Multiple FPGA Design. In *20th Annual ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA 2012)*, February 2012.

[18] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

[19] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: a scalable and flexible data center network. *Commun. ACM*, 54(3):95–104, Mar. 2011.

[20] D. Halperin, S. Kandula, J. Padhye, P. Bahl, and D. Wetherall. Augmenting data center networks with multi-gigabit wireless links. *SIGCOMM Comput. Commun. Rev.*, 41(4):38–49, Aug. 2011.

[21] S. W. Jun, K. Fleming, M. Adler, and J. Emer. Zip-io: Architecture for application-specific compression of big data. In *Field-Programmable Technology (FPT), 2012 International Conference on*, pages 343–351, 2012.

[22] Y. Kang, Y. suk Kee, E. Miller, and C. Park. Enabling cost-effective data processing with smart ssd. In *Mass Storage Systems and Technologies (MSST), 2013 IEEE 29th Symposium on*, 2013.

[23] K. Nagarajan, B. Holland, A. George, K. Slatton, and H. Lam. Accelerating machine-learning algorithms on fpgas using pattern-based decomposition. *Journal of Signal Processing Systems*, 62(1):43–63, 2011.

[24] S. Nath and A. Kansal. Flashdb: dynamic self-tuning database for nand flash. In *Proceedings of the 6th international conference on Information processing in sensor networks*, IPSN '07, pages 410–419, New York, NY, USA, 2007. ACM.

[25] A. Parashar, M. Adler, K. Fleming, M. Pellauer, and J. Emer. LEAP: A Virtual Platform Architecture for FPGAs. In *CARL '10: The 1st Workshop on the Intersections of Computer Architecture and Reconfigurable Logic*, 2010.

[26] D. Tsirogiannis, S. Harizopoulos, M. A. Shah, J. L. Wiener, and G. Graefe. Query processing techniques for solid state drives. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, SIGMOD '09, pages 59–72, New York, NY, USA, 2009. ACM.

[27] F. Xia, Y. Dou, G. Lei, and Y. Tan. Fpga accelerator for protein secondary structure prediction based on the gor algorithm. *BMC Bioinformatics*, 12(Suppl 1):S5, 2011.