# Flash Memory Performance on a Highly Scalable IOV System

Peter Kirkpatrick, Adel Alsaadi, Purnachandar Mididuddi, Prakash Chauhan, Afshin Daghi, Daniel Kim, Sang Kim, K.R. Kishore, Paritosh Kulkarni, Michael Lyons, Kiran Malwankar, Hemanth Ravi, Swaminathan Saikumar, Mani Subramanian, Marimuthu Thangaraj, Arvind Vasudev, Vinay Venkataraghavan, Carl Yang, and Wilson Yung
Aprius, Inc.
peter.kirkpatrick@aprius.com

## Abstract

We present an enterprise-class I/O virtualization (IOV) system, discuss the architecture, and share performance characterization results using extremely high performance flash memory as a load generator. This work describes an IOV system built on the PCI-Express over Ethernet (PCIeOE) protocol, which combines these two ubiquitous, standardized technologies in a novel fashion. By preserving the PCI-Express (PCIe) device and software model, computers interface to the system without modifications to hardware or software. By utilizing 10G Ethernet as a transport, the system integrates with enterprise environments, achieves very high scalability and benefits from the favorable economics of the Ethernet ecosystem. Further, we present a thorough characterization of the latency and performance of the system using very high performance flash memory as an endpoint device. Flash memory serves as a high-intensity traffic generator, and also represents a compelling application of the PCIeOE technology.

## 1. Introduction

The main goal of a modern large scale computing installation is to efficiently support a diverse and bursty set of workloads at the lowest possible operational cost. To support this goal, computing systems are scaling out to dizzying proportions, with the million-server datacenter on the horizon [1]. The economics of such a large system drive the edge nodes (servers) to the minimal set of components, and the networks toward convergence of physical and link layers, with flat topologies [2-4]. The network must further support many protocols such that servers can utilize the access network, multiple tiers of storage, and access other resources such as accelerators and utility I/O.

On this physical infrastructure, server virtualization has increased utilization of the compute node and network interfaces [5-6]. Further, virtual machines enable workload mobility, in terms of the location of execution, availability of service, and elastic scaling to support variance in demand [7-10]. Input/output virtualization (IOV) technologies are evolving to provide the similar benefits for I/O resources; efficient performance, high utilization, service availability, isolation, and elastic scaling [11-17].

Throughout these developments, some common themes persist. The use of standardized technologies ensures that large scale systems interoperate reliably with minimal operational support. Energy efficiency is essential to minimize operational costs and environmental impacts. Resiliency is critical to keep workloads running and to guarantee service level agreements (SLAs). High performance provides a competitive advantage to any organization.

In this work, we present an I/O virtualization system that achieves the goals set forth above, while supporting standard BIOS, operating systems, and PCIe cards. The core technology we have developed is the PCI-Express over Ethernet (PCIeOE) protocol, which enables any PCIe-based resource to reside on the converged Ethernet network, accessible to any server on the network. By preserving the PCI-Express (PCIe) device and software model, computers interface to the system without modifications to hardware or software [18]. By utilizing 10G Ethernet as a transport, the system

integrates with enterprise environments, achieves very high scalability and benefits from the favorable economics that Ethernet enables [19]. The system architecture is discussed, including the resiliency, scaling, and management aspects. A thorough characterization of the system using high performance flash memory is presented, which the authors believe represents the most scalable and highest performance architecture for centralized flash storage.

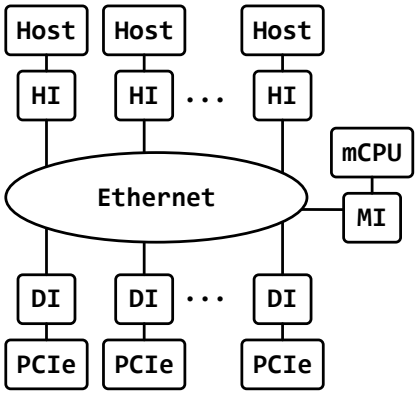


**Figure 1. IOV System structure.**

## 2. PCIeOE System Architecture

The distributed system structure is given in Figure 1. Host interfaces (HI) connect the hosts' PCIe root ports to the Ethernet fabric. Device interfaces (DI) connect PCIe devices to the fabric. A management CPU (mCPU) is connected to the fabric via the management interface (MI), with lower bandwidth for control path traffic only.

The host interface (HI) enables hosts to access distributed PCIe resources across the fabric. A logical block diagram of the device is given in Figure 2. The device presents a PCIe upstream switch port (PCIe 2.0 x4) to the host. Below the switch are PCIe embedded endpoints (EEP) that are enumerated within the host's PCIe domain. The system manager controls mapping of the EEPs to remote resources. Because of the transparent nature of the PCIeOE protocol, hosts always run the unmodified vendor driver for the remote endpoint.

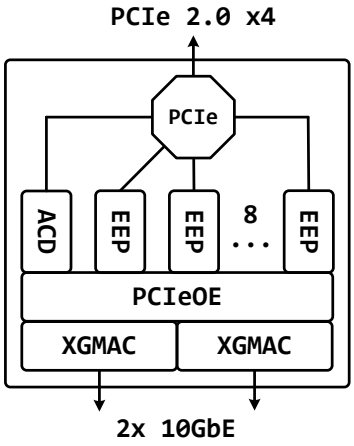


**Figure 2. PCIeOE host interface (HI).**

Downstream PCIe configuration and datapath packets to remote devices are encapsulated by the PCIeOE logic into standard 802.1Q Ethernet frames and load balanced across multiple 10G MACs for transport on the fabric. The device also presents a PCIe endpoint to provide a control path between the system manager and the host OS. This allows the system manager to invoke the PCIe hot plug and rescan mechanism within the host OS. Thus, remote resources may be virtually hot-plugged dynamically into the host PCIe tree.

The host interface resides on a PCIe adapter as shown in Figure 3. The adapter has a PCIe 2.0 x4 card edge and provides two pluggable 10GbE cable interfaces to the Ethernet fabric.

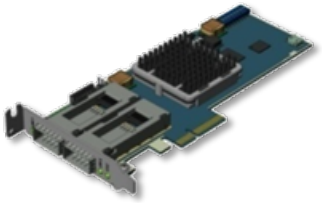

**Figure 3. Host interface adapter.**

A logical view of the device interface (DI) is given in Figure 4. The device interface attaches PCIe devices to the fabric for remote access by hosts. The fabric interface and encapsulation are similar to the HI sections. A unique feature of the device interface is the capability to translate transactions to/from an individual PCIe device attached to the interface into multiple host PCIe domains. The translation is

performed on a per-function basis, which maintains good isolation between resources in the system. This capability enables simultaneous sharing of multifunction devices and SR-IOV endpoints amongst multiple hosts [21]. PCIe packets are passed between the interface and the PCIe device through a downstream PCIe switch port.

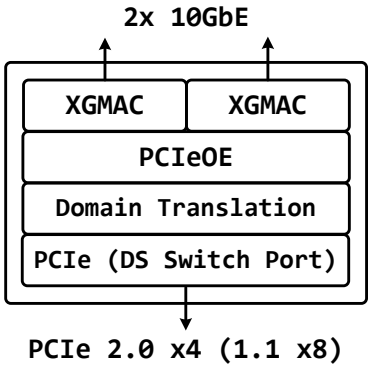


**Figure 4. PCIeOE device interface (DI).**

The x86 management CPU (mCPU) is connected to the fabric by the management interface (MI), which provides an Ethernet interface and performs PCIeOE protocol encapsulation. The mCPU runs the ApriOS operating system, based on the stock CentOS 5.4 Linux distribution with 64-bit 2.6.30 kernel. The standard kernel has been extended to manage a very large PCIe tree including bridges below the device interfaces.

Management of all resources is handled from the management console, where an administrator assigns and configures I/O resources to hosts. Upon system boot or upon hot-plug of I/O devices, the manager enumerates all the PCIe resources on the device interfaces. For multi-function and SR-IOV devices with a privileged driver, this driver runs on the mCPU. This enables configuration of the devices, ranging from simple address and QoS management for LAN/SAN interfaces, to RAID and drive management for storage devices. Hosts see the devices directly and run the unmodified vendor driver.

The current system implementation is shown in Figure 5. Ethernet switching is internal to the chassis, providing 480 Gb/s non-blocking bandwidth from two 24-port low-latency 10GbE switches [22]. Thirty-two 10GbE ports are visible in the rear view, with all-to-all access for hosts to the PCIe slots. The system provides eight full height slots with PCIe 2.0 x4 or PCIe 1.1 x8 interfaces.

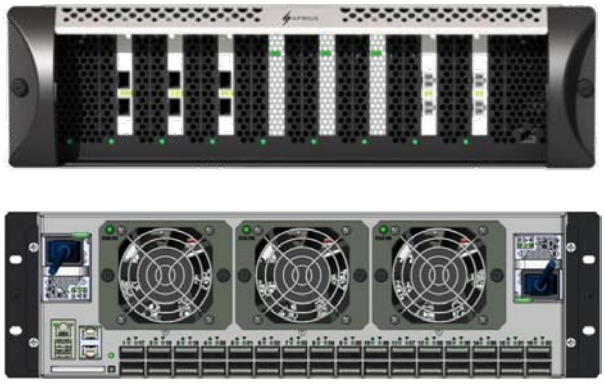

**Figure 5. Front (top) and Rear (bottom) views of the implemented PCIeOE system.**

## 3. PCIeOE Protocol

The central technology we have developed is the PCIeOE protocol. By preserving the standard 802.1Q frame structure, PCIeOE traffic can coexist with Ethernet traffic from other sources on the common datacenter fabric. The encapsulation of a PCIe transaction layer packet (TLP) into an Ethernet frame is shown in Figure 6. The frames carry the PCIeOE Ethertype for identification within the network. For each PCIe TLP, a PCIeOE header is constructed, and together the header and TLP are placed in the frame payload, with padding (if necessary) to meet the minimum Ethernet frame size.

To increase the protocol efficiency, particularly when transmitting small packets, PCIe data link layer packets (DLLPs), used for the ACK/NAK protocol and flow control, are not sent explicitly. PCIeOE ACK/NAK and flow control information is piggybacked along with TLP data. This ensures that after removing 8b10b encoding and adding the PCIeOE header, “10G” of PCIe (e.g. 2.5 GT/s x4 = 10 Gb/s) may be transported bidirectionally using 10G Ethernet without imposing a bandwidth bottleneck. This characteristic holds as the links are scaled to 20G, 40G, and beyond.

One key capability of the protocol is to provide the same resiliency expected from native PCIe,

guaranteeing that any PCIe packet will arrive at its destination in-order, exactly once. Delivery is guaranteed in the event of packet loss or latency experienced in the fabric, using sequence numbers for ordering and a transmit packet replay buffer to recover lost or corrupted packets. Thus, lossless Ethernet is not required for the PCIeOE protocol, although replay adds latency and thus may reduce the performance of an I/O device. If the fabric is lossless, maximum performance can be delivered with high confidence. Given that a "lossless" Ethernet network may still drop packets, and that transmission errors are still possible, the same guaranteed delivery mechanisms are necessary.

PCIeOE also provides the capability to discover and manage hosts and PCIe resources through multiple message types, including a heartbeat message to enhance system level resiliency.

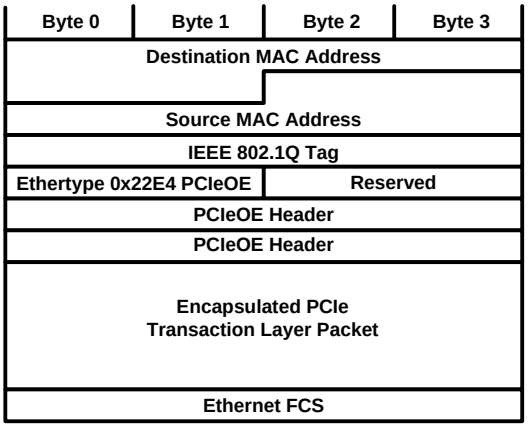


**Figure 6. PCIeOE frame structure.**

## 4. Flash Memory Benchmarking

To establish the potential for centralizing high performance PCIe-based flash memory, a thorough comparison was made between flash performance in a server (native) and in the PCIeOE system. By coupling closely with the CPU/DRAM complex and eliminating storage protocol conversion, PCIe-based flash has proven to be the highest performance non-volatile storage available today, even compared to high performance RAID systems using flash SSD. This performance characteristic is currently driving significant change in computing architecture [23-24].

### 4.1 Experimental Configuration

The server used in this study is an HP DL380 G6, with two Intel Xeon X5670 6-core CPUs at 2.93 GHz, and 12 GB (6x 2 GB) DDR3-1333 DRAM [25]. The platform chipset is an Intel 5520 with PCIe 2.0 root ports. The flash memory devices are FusionIO ioDrive Duo, with 320 GB of SLC NAND flash [26]. These PCIe card form factor devices present two flash "DIMMs" to the host per card. To create the heaviest possible workload for writes, the cards were formatted with 50 % capacity reserved to ensure ample free space for the card to perform NAND management in the background. Before benchmarking at a given block size, the cards were formatted and pre-conditioned using the same size writes, enough to write to the entire capacity of the drive several times over (typically 9x).

The server runs the Windows Server 2008 Enterprise Edition operating system. The micro-benchmarking tool is IOmeter 2006. IOmeter is configured with two workers per DIMM, each with 32 outstanding I/Os for an effective queue depth (QD) of 64 per DIMM. The individual workloads measured are given in Table 1.

| ID | Pattern | Read/Write | Block Size | QD |
|---|---|---|---|---|
| 1 | Rand | 100% Read | 512 B | 64 |
| 2 | Rand | 100% Write | 512 B | 64 |
| 3 | Rand | 70% Read | 4 kB | 64 |
| 4 | Seq | 100% Read | 64 kB | 64 |
| 5 | Seq | 100% Write | 64 kB | 64 |

**Table 1. IOmeter workloads measured.**

Each workload is measured in the server (native) and in the system. The experimental configuration for the native cases is given in Figure 7. Each flash card is installed on a PCIe root port.

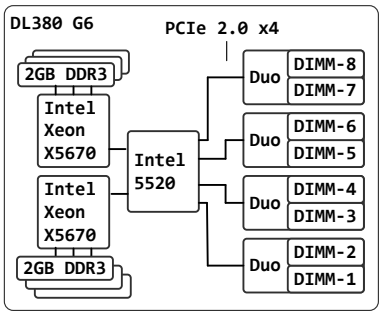


**Figure 7. Native experimental configuration.**

The experimental configuration for the system tests is given in Figure 8. For each flash card that is tested by the benchmark, one host initiator is installed on a system PCIe root port. So for each case (1, 2, or 4 cards), the system provides the equivalent PCIe bandwidth to the native case.

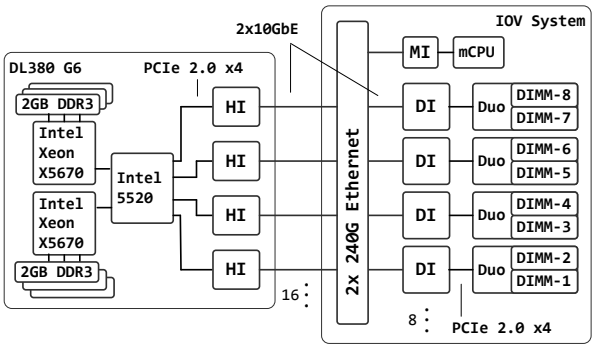


**Figure 8. System test configuration.**

## 4.2 Experimental Results

To verify that CPU utilization was not a limiting factor, the server CPU utilization (average across all cores) was measured during the experiments. The results are given in Figure 9. For all the results provided, the CPU utilization did not exceed 70 %.

The utilization is generally lower in the PCIeOE cases because the increase in storage transaction latency causes the CPU to spend more time waiting for these transactions to complete. The delays are small on the scale of a flash storage transaction (tens of µs) but are more significant when compared to a CPU clock cycle (<1 ns).

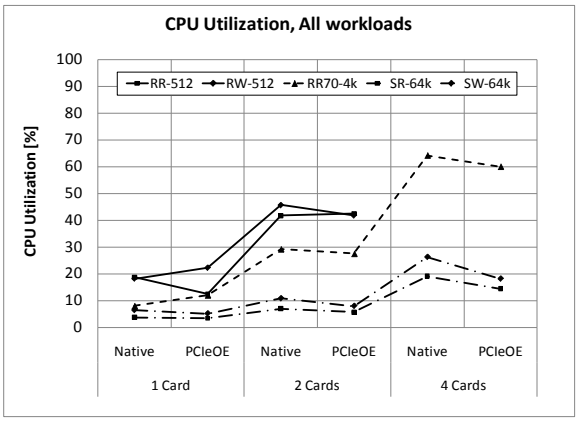


**Figure 9. CPU utilization for all workloads.**

For small block random workloads (512 B), the results are given in Figures 10 and 11. With 4 cards, the IOmeter application hung during the write pre-conditioning (for both native and system cases). Thus, the results for 512 B workloads across 4 cards are not valid and are not included here.

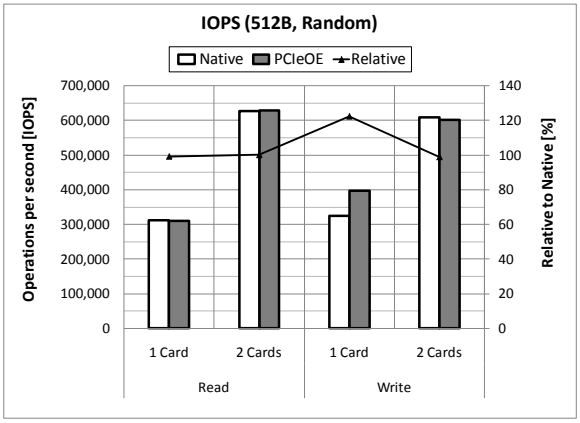


**Figure 10. IOPS for small block workloads.**

The results show that for small transfers, the system performs very close to 100 % of the native cases. For writes to a single card, the system showed increased performance (+20 %), and while this result is repeatable, it is not considered an indicator of broad improvement across various configurations.

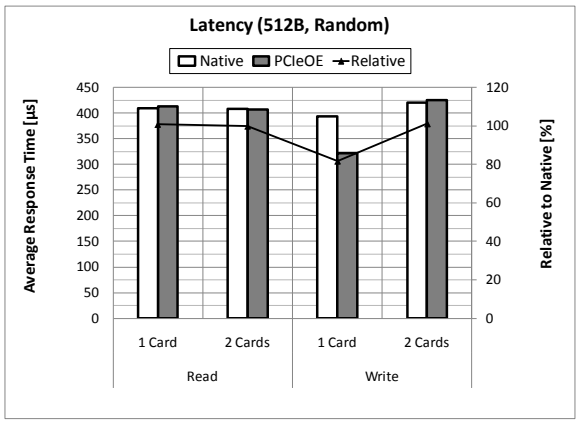


**Figure 11. Latency for small block workloads.**

These results indicate that for small transfers, which are highly sensitive to latency, the small increase in latency due to placing the flash on the network has a trivial impact on performance.

To emulate a typical database workload, the benchmark was configured for a mixed workload of 4 kB blocks, random pattern, and 70 % reads, 30 % writes. The results are given in Figures 11 and 12. For all the cases tested, the system performed very close to 100 % of the native cases (100 ± 0.5 %).

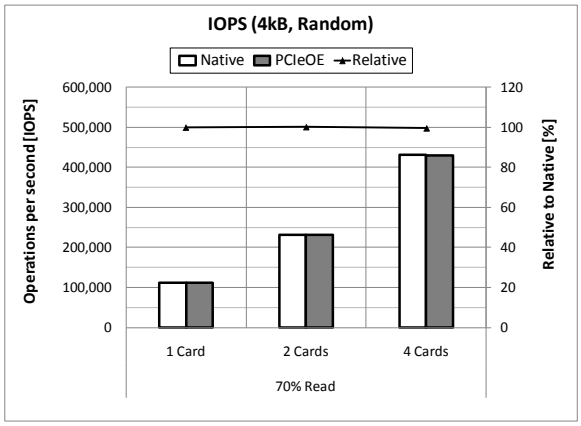


**Figure 12. IOPS for the database workloads.**

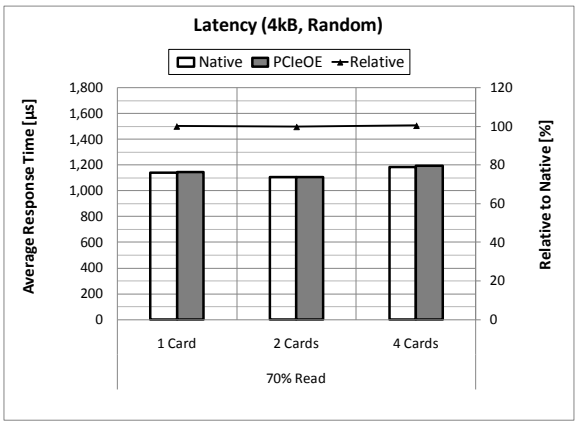


**Figure 13. Latency for the database workloads.**

To provide coverage of a high bandwidth streaming pattern, the benchmark was also run with large sequential (64 kB) workloads. The results are given in Figures 14 and 15. For writes, the system performs between 90-102 % of native. In the case of pure reads, the latency introduced by the system exposes a limitation of the flash card under the default settings. The default settings allow a fixed number of outstanding memory requests from the card DMA engines, and thus the results show 83 % of native performance for 1-2 cards, and slightly lower for 4 cards. It is expected that this condition can be rectified by tuning the card and driver settings, such as increasing the allowed outstanding IO requests and the timing of interrupts.

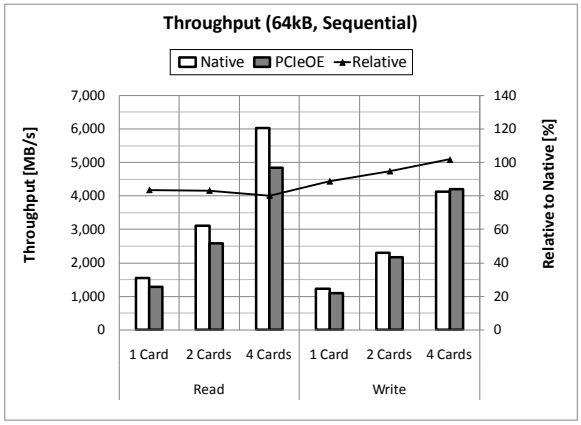


**Figure 14. IOPS for the streaming workloads.**

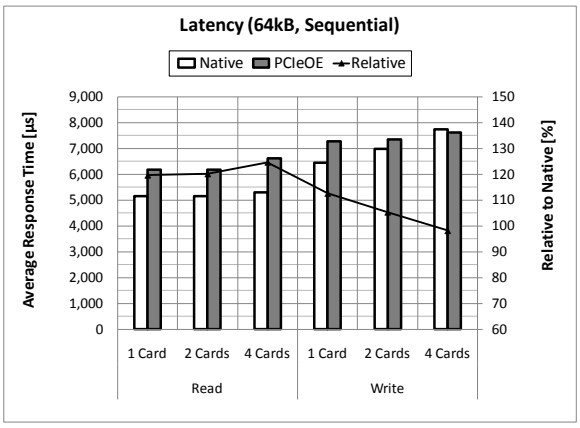


**Figure 15. Results for the streaming workload.**

To characterize the system interfaces under oversubscription, the large streaming workloads were run in an alternate configuration. The streaming workloads were found to be the most stressful for oversubscription due to high bandwidth saturation of the PCIe links. To oversubscribe the host interface (1:2), a single host interface was used to access two cards (1:2). To oversubscribe the device interface, two hosts were used to access a single flash card (2:1), with one DIMM assigned to each host. This scenario represents the compelling use case of sharing the flash card amongst multiple servers. Note that in these experiments, the performance is not CPU-bound, but the hosts have lower-powered CPUs (4-cores, 2.0 GHz), so the performance of the baseline case is slightly lower than the previous results.

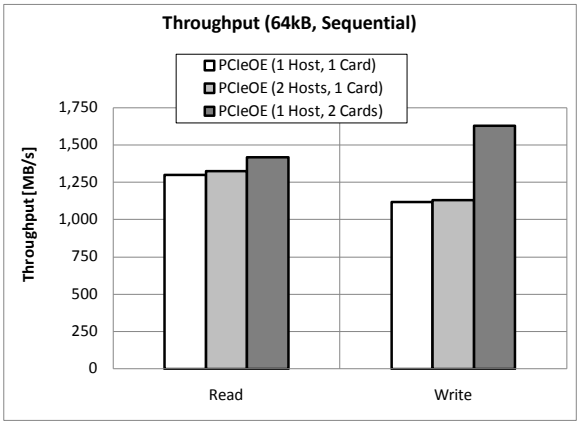


**Figure 16. Results for oversubscribed interfaces.**

The results for oversubscription are given in Figure 16. In the sharing case, under 2:1 oversubscription (PCIeOE 2 Hosts, 1 Card), the host interface performance slightly exceeds the baseline system case (PCIeOE 1 Host, 1 Card), indicating that the device is robust under this condition. Under 1:2 oversubscription (PCIeOE 1 Host, 2 Cards), the device interface performance exceeds the system baseline as well, indicating ideal performance. Note that the performance is not a full 2x because although there is more flash available, the single host interface is still bandwidth constrained for these workloads.

## 5. Related Work

The notion of virtualizing a PCIe device within the bounds of a single host domain has been standardized by the PCI-SIG as the SR-IOV specification [21]. Support for SR-IOV is now included in many commercial PCIe endpoints, from Intel, LSI Logic, Broadcom, Emulex and others [27-30]. Similarly, the extension of this concept to enable access by multiple host domains has resulted in the MR-IOV specification [31]. At least two commercial entities have developed switching products based on the MR-IOV standard, notably NextIO and Virtensys [32-33].

A technology known as ExpressEther has been proposed and developed by the System Platform Research Labs at NEC [34]. This work has detailed an alternate approach to transporting PCIe data on Ethernet, and has presented the performance of a shared 10G Ethernet NIC as a remote resource [35-36].

Another approach to virtualizing and sharing I/O resources include using RDMA as a transport mechanism, which can be switched on various physical layers such as Inifiband or RoCEE, a path taken by Xsigo Systems [37]. Using the RDMA model typically means that the native bus protocol and thus drivers are modified. The additional software layer introduced in this method generally increases flexibility at the cost of additional latency.

Within the PCIeOE system, there are many interesting directions for future study. The performance of the system on more complex Ethernet topologies and co-existence between PCIeOE and other Ethernet-based traffic are important to understand the limits of scalability. The characterization of different PCIe devices and

protocols, as well as higher level (application) benchmarks are also useful to expand insights beyond the micro-benchmarks discussed here.

## 6. Conclusions

In this paper, we presented a highly scalable I/O virtualization system based on the PCIeOE protocol along with measured characterization results using high performance NAND flash. The system architecture enables wide scaling of link bandwidth by utilizing multiple 10G Ethernet interfaces for host access. The architecture enables wide scaling of host and device counts by utilizing dense commodity Ethernet switching as a fabric. By using Ethernet as a transport, the system can be integrated with the ubiquitous converged Ethernet fabric in use in datacenters now and in the future. The system provides easy integration of any PCIe-based resource by preserving the PCIe device and software models, allowing native vendor drivers to be used by hosts.

Thorough testing of the system using NAND flash has indicated that the system and protocol are very robust, low-latency, and high performance. Very demanding NAND flash devices perform at near-native performance levels across a wide range of workloads in the system. For the most common workloads, flash performance in the system is indistinguishable from the native server case. Thus, the system provides a compelling architecture for centralizing and sharing the emerging tier of flash storage in scalable performance computing systems.

## 7. References


[1] Randy H. Katz. Tech Titans Building Boom. In *IEEE Spectrum*, February 2009.

[2] John Kim, William J. Dally, Steve Scott, Dennis Abts. Cost-Efficient Dragonfly Topology for Large-Scale Systems. *OFC/NFOEC 2009*, March 2009.

[3] Dennis Abts, Michael R. Marty, Philip M. Wells, Peter Klausler, Hong Liu. Energy Proportional Datacenter Networks. *ISCA'10*, June 2010.

[4] Pradeep Sindhu. Defining Characteristics of Qfabric. From *http://www.juniper.net/us/en/local/pdf/whitepapers/2000384-en.pdf*, February 2011.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. *Proceedings of SOSP*, Oct. 2003.

[6] Edouard Bugnion, Scott Devine, Kinshuk Govil, and Mendel Rosenblum. Disco: Running Commodity Operating Systems on Scalable Multiprocessors. *ACM Transactions on Computer Systems, Vol. 15, No. 4, Pages 412–447*, November 1997.

[7] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. *Proc. of the 2nd Symp. on Networked Systems Design and Implementation*, May 2005.

[8] Michael Nelson, Beng-Hong Lim, and Greg Hutchins. Fast transparent migration for virtual machines. *Proceedings of the annual conference on USENIX Annual Technical Conference*, April 2005.

[9] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. *University of California, Berkeley Technical Report No. UCB/EECS-2009-28*, February 2009.

[10] Amazon Web Services. From *http://aws.amazon.com*, March 2011.

[11] J. Sugerman, G. Venkitachalan, B.H. Lim. Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor. *Proceedings of the USENIX Annual Technical Conference*, June 2001.

[12] Darren Abramson, Jeff Jackson, Sridhar Muthrasanallur, Gil Neiger, Greg Regnier, Rajesh Sankaran, Ioannis Schoinas, Rich Uhlig, Balaji Vembu, and John Wiegert. Intel virtualization technology for directed I/O. *Intel Technology Journal*, 10(3), August 2006.

[13] Advanced Micro Devices. AMD IOMMU Specification. From *http://support.amd.com/us/Processor_TechDocs/34434-IOMMU-Rev_1.26_2-11-09.pdf*, February 2009.

[14] Julian Satran, Leah Shalev, Muli Ben Yehuda, and Zorik Machulsky. Scalable I/O—A Well-Architected Way to Do Scalable, Secure and Virtualized I/O. *USENIX WIOV 2008*, December 2008.

[15] Joshua LeVasseur, Ramu Panayappan, Espen Skoglund, Christo du Toit, Leon Lynch, and Alex Ward, Dulloor Rao, Rolf Neugebauer and Derek McAuley. Standardized But Flexible I/O for Self-Virtualizing Devices. *USENIX WIOV 2008*, December 2008.

[16] Jose Renato Santos, Yoshio Turner, G. (John) Janakiraman, and Ian Pratt. Bridging the gap between software and hardware techniques for I/O virtualization. *USENIX Annual Technical Conference*, June 2008.
[17] Paul Willmann, Scott Rixner, and Alan L. Cox. Protection strategies for direct access to virtualized I/O devices. *Proceedings of the annual conference on USENIX Annual Technical Conference*, 2008.
[18] Jeffrey Shafer, "I/O Virtualization Bottlenecks in Cloud Computing Today", *USENIX WIOV 2010*, March 2010.
[19] PCI-SIG, "PCI-Express Base Specification 3.0", http://www.pcisig.com/specifications/pciexpress/ November 2010.
[20] IEEE 802.3 Ethernet Working Group, from *http://standards.ieee.org/about/get/802/802.3.html*
[21] PCI-SIG, "Single-Root I/O Virtualization and Sharing 1.1 Specification", *from* *http://www.pcisig.com/specifications/iov/single_root/*, January 2010.
[22] Fulcrum Microsystems, http://www.fulcrummicro.com/.
[23] Adam Leventhal. Flash Storage Today. *ACM Queue*, July/August 2008.
[24] William K. Josephson, Lars A. Bongo, David Flynn, Kai Li. DFS: A File System for Virtualized Flash Storage. 8th USENIX File and Storage Technologies, February 2008.
[25] Hewlett-Packard. Proliant DL380 G6 Server. http://www.hp.com.
[26] FusionIO. ioDrive Duo. http://www.fusionio.com.
[27] Intel. 82599 10Gb Ethernet Controller. http://www.intel.com.
[28] LSI Logic. LSISAS2208 ROC. http://www.lsi.com
[29] Emulex. Emulex OCe11000 UCNA. http://www.emulex.com.
[30] Broadcom. BCM57712 10GbE C-NIC. http://www.broadcom.com.
[31] PCI-SIG, "Multi-Root I/O Virtualization and Sharing 1.0 Specification", *from* *http://www.pcisig.com/specifications/iov/multi-root/*, May 2008.
[32] NextIO. http://www.nextio.com.
[33] Virtensys. http://www.virtensys.com.
[34] Jun Suzuki, Yoichi Hidaka, Junichi Higuchi, Takashi Yoshikawa, Atsushi Iwata. ExpressEther - Ethernet-Based Virtualization Technology for Reconfigurable Hardware Platform. *IEEE HOTI'06*, September 2006.
[35] Nobuyuki Enomoto, Hideyuki Shimonishi, Junichi Higuchi, Takashi Yoshikawa, Atsushi Iwata. High-Speed, Short-Latency Multipath Ethernet Transport for Interconnections. *IEEE HOTI'08*, September 2008.
[36] Jun Suzuki, Yoichi Hidaka, Junichi Higuchi, Teruyuki Baba, Nobuharu Kami, Takashi Yoshikawa. Multi-Root Share of Single-Root I/O Virtualization Compliant Device. *IEEE HOTI'10*, September 2010.
[37] Xsigo Systems. http://www.xsigo/com.