RZ 3906 (#ZUR1609-039) 09/13/2016
Computer Science 5 pages

Research Report

Understanding the Performance of Networked Flash Storage

Animesh Trivedi*, Bernard Metzler*, Jonas Pfefferle*, Patrick Stuedi*, Nikolas Ioannou*, Ioannis Koltsidas*, Thomas R. Gross‡

*IBM Research – Zurich
8803 Rüschlikon
Switzerland

‡Swiss Federal Institute of Technology (ETH)
Zurich

Understanding the Performance of Networked Flash Storage

Animesh Trivedi[1], Bernard Metzler[1], Jonas Pfefferle[1], Patrick Stuedi[1], Nikolas Ioannou[1], Ioannis Koltsidas[1], and Thomas R. Gross[2]

[1]IBM Research, Zurich
[2]ETH, Zurich

Abstract

Availability of high-performance IO devices has led to the development of various IO interfaces/APIs. For many distributed data processing and storage systems that consider integrating high-performance networks (e.g., RDMA) and storage devices (e.g., flash), the performance implications of the available options are not clear upfront. To provide guidance to application developers as well as system designers, we discuss if (or what) *combinations of modern networking and storage stacks can deliver the full performance of flash storage to a networked application?* We report our findings with regard to access latencies, peak performance, IO efficiency, core scaling, and we perform a comparison between a block and an RDMA interface to a flash device.

1 Introduction

The performance of an IO operation in a distributed storage or data processing system is governed by the performance of the storage system as well as the networking technology. Thankfully, the performance and capabilities of modern network and storage devices have improved considerably during the past decade. Consequently, IO stacks of a modern OS such as Linux, have also evolved to deliver these improvements to applications. However, this rapid evolution has also introduced many new parameters, IO dependencies, and new interfaces, whose influence on the performance has not been fully understood. This work, which is triggered by our own curiosity about the implications of integrating high-performance IO devices in a distributed setting, empirically investigates the performance of various networked flash configurations. We now quickly recap the IO developments that made this investigation necessary.

Networking Improvements: The POSIX socket interface has been the de-facto standard for networking API for more than 30 years. However, as the interconnect performance improved (10, 40, 100 Gb/s Ethernet with less than 5 μs access latencies), several research projects have looked into reducing the resource management overhead from the OS [12], improving the small packet performance [6], and the multi-core scalability [7], and achieving a closer integration of network processing and applications [8, 14]. Many of these improvements are now a part of the Linux kernel. As a popular alternative way for efficient network IO, we also report the performance numbers for Remote Direct Memory Access (RDMA) technology, which is being used in a number of systems [11, 4, 10, 14].

Storage Improvements: Modern flash storage can support multiple Gb/s bandwidths with access latencies as low as 15 μs [5]. In a spirit similar to networks, the storage stack of the Linux kernel has evolved to accommodate the hardware improvements by reducing unnecessary OS involvement in the IO and by improving scalability [2]. However, researchers have also observed the parallels between the high-performance networking and storage IO [15, 3]. Hence, apart from the traditional block interface, we also evaluate a Direct Storage Access (DSA) abstraction that provides an RDMA interface to the local storage [13, 9]. The DSA interface offers all key properties of a user-space storage stack [3, 12].

2 Investigation

Aim and Scope: The goal of this investigation is to quantify the combined performance capability of modern IO stacks with high-performance devices. We focus on a client-server environment, where a typical IO pattern consists of a server accepting a data request from a networked client, getting data from the local storage device, and transmitting data back as a response to the requesting client over the network. Such a workload is an integral part of a networked storage system such as, e.g., a key-value server, a datanode in HDFS etc.

Methodology: We investigate IO latencies, peak IO

	Read			Write		
	Latency	IOPS	Bandwidth	Latency	IOPS	Bandwidth
dev-SATA	61.0/117.4 μs	155K/51K	271/258 MB/s	1218.2/89.1 μs	139K/36K	220/189 MB/s
dev-PCIe	34.7/77.6 μs	731K/605K	1.35/1.35 GB/s	15.1/25.3 μs	776K/56K	642/714 MB/s
HS4-blk	108.2/118.0 μs	265K/323K	2.3/2.7 GB/s	668.2/2018 μs	1.6K/1.7K	1.5/1.6 GB/s
HS4-DSA	66.1/66.5 μs	556K/531K	3.1/3.2 GB/s	522.1/522.1 μs	84K/113K	2.0/2.0 GB/s

Table 1: Performance of flash devices. The reported performance numbers for latency and IOPS are for 1 kB request sizes. The bandwidth is measured with 64 kB size. The peak performance is obtained under a device-specific best configuration. The two numbers represent sequential and random accesses respectively.

operations per second (IOPS) and bandwidth at the server. Clients repeatedly ask the server over the network to read or write data into the flash storage. The server reads and writes flash block devices directly. Hence, the performance numbers reported do not include the filesystem overhead. For the latency and the peak IOPS experiments, we use 1 kB buffer size. For the peak bandwidth experiment, the buffer size is 64 kB. The numbers reported are measured on the Linux kernel `v3.13.11` and are the average of 3 runs, each lasting 30 secs with 5 secs of stablization time in between. The performance numbers reported in this section are not meant as a fair quantitative comparison as they represent vastly different hardware/software configurations. Rather, they illustrate the available performance spectrum for networked flash.

Hardware Setup: All experiments use a cluster of 12 IBM x3650 M4 machines containing a dual socket Intel Xeon E5-2690. The machines are connected using three generations of Ethernet networking devices with an Intel I350 Gigabit controller, as well as a Chelsio T4 and a T5 RDMA controller for 1, 10, and 40 Gb/s connectivity respectively. We further include three types of flash storage devices, namely (i) a consumer-grade SATA SSD denoted by *dev-SATA*; (ii) an enterprise-grade commercial PCIe-attached flash denoted by *dev-PCIe*; (iii) a PCIe-attached Hybrid Scalable Solid State Storage (HS4) card [13] that supports the Direct Attached Storage (DSA) interface [9] for a local RDMA access to the flash storage. The DSA interface also support a block device abstraction, which is denoted by HS4-blk. Table 1 and 2 summarize the best local/baseline performance of the storage and networking devices that are evaluated in this paper.

2.1 What is the effect of latency improvments on remote flash accesses?

A complete IO operation on the networked flash includes latencies from the network and the flash. In the past, the performance idiosyncrasies of flash, such as the gaps in the sequential and random accesses etc., were overshadowed by either the high network[1] or flash latencies. However, as both have made progress to lower the access latencies considerably, we revisit the key questions such as if the local or the remote flash access matters? and if there is a performance difference between a random and a sequential access to a networked flash storage?

	TCP/Sockets		RDMA send/recv	
	latency	IOPS	latency	IOPS
Intel 1 GbE	49.0μs	365K	N/A	N/A
T4 10 GbE	16.4μs	549K	13.3μs	750K
T5 40 GbE	12.0μs	620K	8.8μs	1.5M

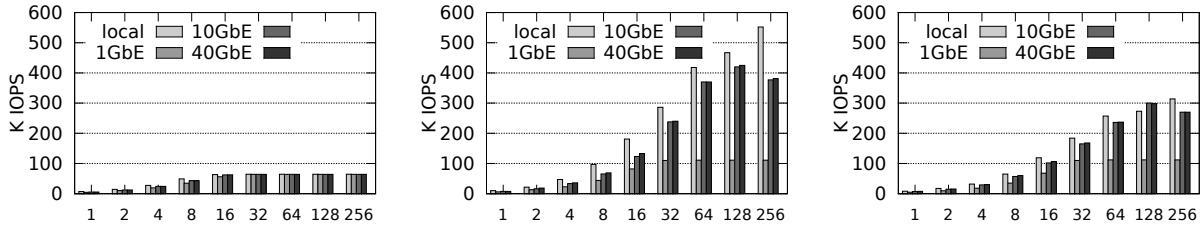
Table 2: Baseline network performance. Our 1GbE Intel NIC does not support RDMA.

	1 GbE	10 GbE	40 GbE
dev-SATA	191/166.4/14.7%	183/133.8/36.7%	157/129.4/21.3%
dev-PCIe	130/126.6/2.6%	101/94/7.4%	92/89.6/2.6%
HS4-blk	217/167/29.9%	142/134.4/5.6%	135/130/3.8%

Table 3: Access latencies in μs for random 1 kB read for various combinations IO devices. The three numbers represent the measured latency, the best latency (calculated as the sum of individual latencies from table 1 and 2), and the difference between the two.

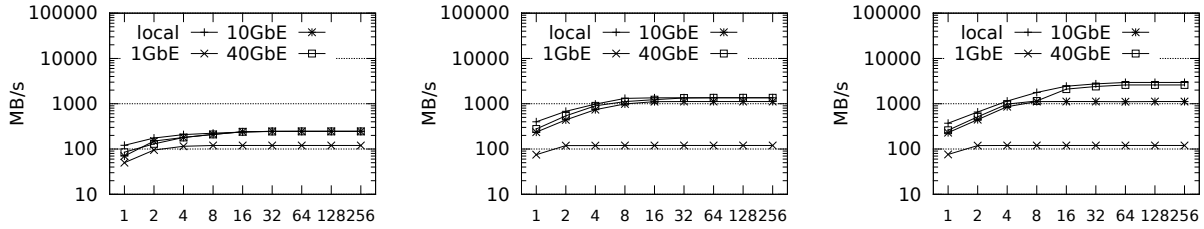
Analysis: Table 3 shows the measured access latencies for a random 1kB access for the various combination of network and storage devices. For PCIe-attached flash and 10, 40 GbE the access latencies remained within 8% of the best predicted latencies. 1 GbE and dev-SSD introduced more overhead than expected. As network latencies continue to improve, we expect network to add minimum overhead to the flash performance. In a second experiment (not shown), we measure the *gap* between the access latencies of random and sequential accesses for various combinations of IO devices. The extend of the performance gap depends on multiple factors such as the

[1] We here refer to the collective latency that includes the core network (e.g., switching) and the end-host protocol processing costs.



(a) dev-SATA (b) dev-PCIe (c) HS4-blk

Figure 1: Scaling of IOPS (with equivalent local performance, i.e. w/o network) with the number of clients.



(a) dev-SATA (b) dev-PCIe (c) HS4-blk

Figure 2: Scaling of bandwidth (with equivalent local performance, i.e. w/o network) with the number of clients.

controller design, the FTL implementation etc. We discover that accessing flash storage with a large gap, such as with dev-PCIe, over a 1 GbE network that has $3-4\times$ higher latencies than 10 GbE and 40 GbE, shrinks the performance gap by $11-41\%$. However, as the network latencies decrease for 10 GbE and 40 GbE, the gap further increases. Hence, we recommend latency-sensitive applications to differentiate between random and sequential accesses and (if possible) trade CPU cycles to efficiently re-order/batch IOs.

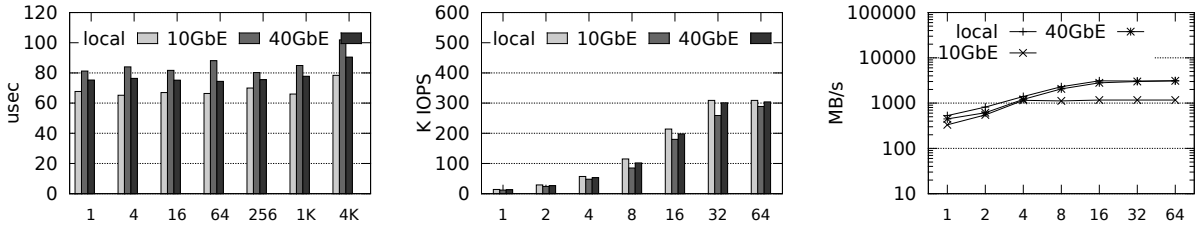
2.2 Can modern IO stacks deliver the full performance of networked flash?

In this section, we look specifically at the peak performance delivered by a combination of flash+network devices to concurrent clients. If the networked access to a flash device is significantly expensive than the local access then we need to investigate and understand the networking overheads. On the other-hand, if the distinction between local and remote flash storage does not matter, then we need to rethink how we build distributed shared data storage and processing systems for high-performance devices [1].

In this experiment clients request a data block (1 kB for IOPS, and 64 kB for bandwidth) at a random flash offset. To represent a data-dependent workload, where a client cannot issue the next request until the last one has finished, clients have only one outstanding request at a time. The server forks a new server process to handle requests for every connected client. This configuration stresses the OS and the IO stacks the most because of overheads from scheduling, context switches, cache contentions, protocol processing etc., and provides only negligible opportunities for the IO cost amortization.

Analysis: Figure 1 shows the scaling of IOPS with the number of concurrent networked clients. For dev-SATA, the peak performance of the flash device (64K IOPS) becomes the bottleneck first, and even a 1 GbE is sufficient to deliver the maximum IOPS allowed by the device. However, for dev-PCIe and HS4-blk, the 1 GbE becomes bottleneck at around 111K IOPS. On an average, the network adds 23% (min-max: 10-32%) and 20% (min-max: 9-31%) overhead for 10 GbE and 40 GbE respectively. For HS4-blk, the network adds 8-15%.

Figure 2 shows the peak bandwidth delivered, and shows a more familiar pattern. In every case, the peak local bandwidth was followed closely (within 10%) by the networked flash bandwidth, until the network became the bottleneck. The 1 GbE network cannot deliver (peak bandwidth: 120 MB/s) the peak performances of any flash device. The 10 GbE networked accommodated dev-SATA (peak bandwidth: 245MB/s), but dev-PCIe (peak bandwidth: 1.35 GB/s) and HS4-blk (peak bandwidth: 2.6 GB/s) necessitated a 40 GbE link. In summary, one can lose up to one-third of the peak IOPS performance for the networked flash storage. This conclusion implies that sources of inefficiencies exist in the networking and



(a) IO latencies vs the buffer size (b) IOPS scaling with number of clients (c) Bandwidth with number of clients

Figure 3: Performance of HS4-DSA interface for local, and over 10GbE and 40GbE RDMA networks.

storage stacks, when dealing with small message sizes (1 kB) that are not large enough to saturate the device capacity. The exact size is device dependent. However, the full peak bandwidth of the flash devices was delivered. Hence, a distinction between a local and a remote storage does not matter, and locality aware optimization can be considered irrelevant [1].

2.3 What is the performance of a networked DSA/RDMA interface?

We now turn our attention to emerging storage interfaces that go beyond the simple block accesses. Examples of these interfaces include NVMe, Moneta-D, and IBM's Direct Storage Access (DSA) interface. In this section, we evaluate the performance of the DSA interface running on top of an IBM HS4 SLC PCIe flash storage. The DSA interface offers the key performance properties of user-space storage (with the support from the hardware, here the HS4 flash storage) as illustrated by the Moneta project [3] and the storage stack of Arrakis OS [12]. The interface is built upon the RDMA networking principles (and even uses the standard OFED/Linux RDMA stack), where each application is given its private set of userspace-mapped storage hardware queues to access flash extents. Reading or writing DSA/HS4 storage is similar to reading or writing a remote memory location using RDMA operations. In this experiment the networked clients also use RDMA operations for high-performance networking. The goal of this exercise is to illustrate the best performance possible.

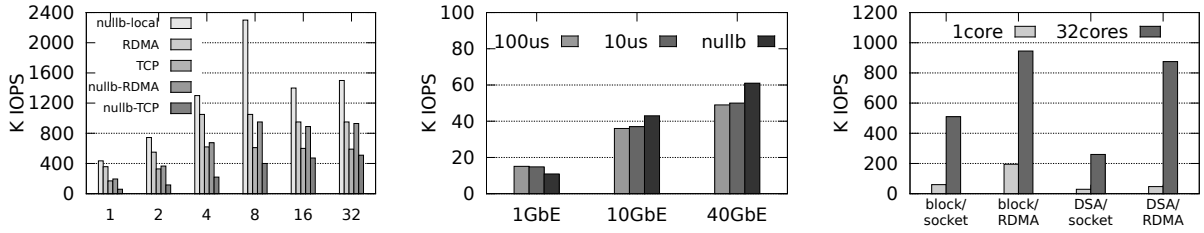
Analysis: Figure 3 summarizes our findings. The HS4/DSA prototype delivered remote flash access latencies that are close to the combined latencies of flash (around 67 μs) and RDMA networks (8-13 μs). In such a setting, a 4 kB page transfer takes about 90 μs with HS4/DSA on T5/40GbE. A major part of this performance improvement comes from IO offloading, another one from the thin software abstraction of RDMA. The peak IOPS performance of networked HS4/DSA follows the local performance more closely, on average 16% to 6% for 10 GbE and 40 GbE respectively. The 40 GbE/RDMA network delivered the full bandwidth of the HS4 device across the network.

2.4 What is the IO efficiency?

We now turn our attention to the cost of high-performance IO. We fix the number of concurrent clients to 64, and for additional comparison, use the null block device from the kernel (`null_blk.ko`).

Scaling of IOPS: Figure 4a shows the scaling of IOPS for the local null block device (nullb), RDMA send/recv operations (RDMA), TCP send/recv operations (TCP), RDMA operations accessing the null block device (RDMA-nullb), and TCP operations accessing the null block device (TCP-nullb). In our setting, a single core delivers about 59K (peak: 510K) and 195K (peak: 950K) IOPS for block IO for TCP and RDMA accesses respectively. In comparison, Arrakis reports 1.1M IOPS for UDP packet processing [12]. However, there are some key differences between the Arrakis IO stack and our setup. First, Arrakis achieved its peak performance under a certain fixed request rates from clients, whereas clients in our experiment do not batch requests. Second, Arrakis performance does not include overheads from the storage accesses. Lastly, the RDMA performance includes cost for executing more complicated TCP state engine. For a memcached server when there is a read hit, Arrakis reports 300K IOPS for a single core, which arguably is better than Linux's 170K IOPS for just TCP send/recv performance.

Effect of storage latency improvements: Figure 4b shows the effect of storage latency improvements on the delivered IOPS for the three networks. We injected controlled delays (no delay, 10 μs, and 100 μs) to mimic a low-latency block device. To our surprise, an order of magnitude latency improvement did not affect the overall IOPS delivered. We believe that this is due to the fact that in a single core experiment, the CPU core (while performing scheduling, context switches, copies etc.) quickly becomes the bottleneck. Improving net-



(a) Scaling of IOPS with number of cores (b) Effect of latency improvement on peak IOPS (c) Efficiency of storage/network offloading

Figure 4: IO efficiency in terms of IOPS delivered under various settings.

work speed from 10 GbE to 40 GbE showed 41% improvement for the null block device.

Efficiency of mixed configurations: Availability of RDMA interfaces for the network and storage gave us opportunity to test the efficiency of the IO offloading. We evaluate a similar 64 client experiment with a null DSA device with a null block device in the following configurations: (a) block for storage and sockets for network; (b) block for storage and RDMA for network; (c) DSA for storage and TCP for network; (d) DSA for storage and RDMA for network. Figure 4c shows our findings. Contrary to our expectations, the offloading of the storage stack yielded small improvements, a conclusion that was also confirmed by measuring the overheads (less than 1.3 μs) from the block layer for the null device. Network offloading to RDMA gives the maximum performance gains (by upto 3.2×). The figure also shows the peak performance (around 900K) when all 32 cores are enabled. However, our analysis only includes simple block accesses. Hence, the results pertaining to the storage overheads will change once more complicated storage operations such as transactions, with file systems are included in the evaluation.

3 Conclusion

In this work we investigated the performance of various flash and network device combinations. Our analysis concludes that (a) low-latency networks expose the performance traits of flash storage which otherwise were eclipsed by large network or storage latencies; (b) networked flash access can lose upto on-third of peak IOPS but the full device bandwidth is delivered; (c) an RDMA interface to the storage and the networks can deliver low-latency remote page accesses (4kB transfer in 90 μs) and upto 900K IOPS for a null block device. As next steps in our investigation, we are looking into performance bottlenecks for small IO operations, breaking down the CPU cost, and evaluating QoS (e.g., the tail effect) when the storage, or the network, or both are loaded.

References

[1] ANANTHANARAYANAN, G., ET AL. Disk-locality in datacenter computing considered irrelevant. In *Proc of the 13th HotOS'11*.

[2] BJØRLING, M., ET AL. Linux block io: Introducing multi-queue ssd access on multi-core systems. In *Proc. of the 6th SYSTOR* (2013), pp. 22:1–22:10.

[3] CAULFIELD, A. M., ET AL. Providing safe, user space access to fast, solid state disks. In *Proc. of the 17th ASPLOS* (2012), pp. 387–400.

[4] DRAGOJEVIĆ, A., ET AL. Farm: Fast remote memory. In *Proc. of the 11th NSDI* (2014), pp. 401–414.

[5] FUSIONIO. iomemory-sx300 datasheet at `http://www.fusionio.com/load/-media-/302x3j/docsLibrary/SX300_DS_Final_v3.pdf`.

[6] HAN, S., ET AL. MegaPipe: a new programming interface for scalable network I/O. In *Proc. of the 10th OSDI* (2012), pp. 135–148.

[7] JEONG, E. Y., ET AL. mtcp: A highly scalable user-level tcp stack for multicore systems. In *Proc. of the 11th NSDI* (2014), pp. 489–502.

[8] LIM, H., ET AL. Mica: A holistic approach to fast in-memory key-value storage. In *Proc. of the 11th NSDI* (2014), pp. 429–444.

[9] METZLER, B., ET AL. The Non-Volatile Memory Verbs Provider (NVP): Using the OFED Framework to access solid state storage. In *2013 OFA International Workshop*.

[10] MITCHELL, C., ET AL. Using one-sided rdma reads to build a fast, cpu-efficient key-value store. In *Proc. of the 2013 USENIX ATC*, pp. 103–114.

[11] OUSTERHOUT, J., ET AL. The case for ramcloud. *Commun. ACM 54*, 7 (July 2011), 121–130.

[12] PETER, S., ET AL. Arrakis: The operating system is the control plane. In *Proc. of the 11th OSDI* (2014), pp. 1–16.

[13] SCHRMANN, F., ET AL. Rebasing I/O for Scientific Computing: Leveraging Storage Class Memory in an IBM BlueGene/Q Supercomputer. In *Supercomputing*, vol. 8488 of *Lecture Notes in Computer Science*. Springer International Publishing, 2014, pp. 331–347.

[14] STUEDI, P., ET AL. Darpc: Data center rpc. In *Proc. of the 5th ACM SOCC* (2014), pp. 15:1–15:13.

[15] TRIVEDI, A., ET AL. Unified high-performance i/o: One stack to rule them all. In *Proc. of the 14th HotOS* (2013).