When NVMe over Fabrics Meets Arm: Performance and Implications

Yichen Jia
Louisiana State University
yjia@csc.lsu.edu

Eric Anger
Arm Inc.
eric.anger@arm.com

Feng Chen
Louisiana State University
fchen@csc.lsu.edu

***Abstract*—A growing technology trend in the industry is to deploy highly capable and power-efficient storage servers based on the Arm architecture. An important driving force behind this is storage disaggregation, which separates compute and storage to different servers, enabling independent resource allocation and optimized hardware utilization. The recently released remote storage protocol specification, NVMe-over-Fabrics (NVMeoF), makes flash disaggregation possible by reducing the remote access overhead to the minimum. It is highly appealing to integrate the two promising technologies together to build an efficient Arm based storage server with NVMeoF.**

In this work, we have conducted a set of comprehensive experiments to understand the performance behaviors of NVMeoF on Arm-based Data Center SoC and to gain insight into the implications of their design and deployment in data centers. Our experiments show that NVMeoF delivers the promised ultra-low latency. With appropriate optimizations on both hardware and software, NVMeoF can achieve even better performance than direct attached storage. Specifically, with appropriate NIC optimizations, we have observed a throughput increase by up to 42.5% and a decrease of the 95th percentile tail latency by up to 14.6%. Based on our measurement results, we also discuss several system implications for integrating NVMeoF on Arm based platforms. Our studies show that this system solution can well balance the computation, network, and storage resources for data-center storage services. Our findings have also been reported to Arm and Broadcom for future optimizations.

***Index Terms*—NVMe; NVMe over Fabrics; Arm; Flash Memory.**

I. Introduction

Arm processors have been regarded as the platform for mobile and embedded systems for their well-known strength in customizability, cost, and power efficiency [1]. With the recent release of the 64-bit Arm architecture, ARMv8 [2] has significantly improved the computing capability, making it also highly competitive in traditional server environments.

An important application of Arm servers is to host high-speed data stores, such as MongoDB [15] and RocksDB [18], to provide high-speed data services. Such data store systems typically adopt Non-volatile Memory Express (NVMe) based flash SSDs as the storage media for high-throughput and low-latency I/Os. However, the high I/O speed also pushes a significant computing burden on the data store servers.

Storage disaggregation can well address the above-said challenge by separating compute and storage to different servers. *NVMe over Fabrics* (NVMeoF) [17], the recently released protocol standard for accessing NVMe devices over high-speed networks, is particularly attractive. With fast interconnect technologies, such as Remote Direct Memory Access (RDMA), NVMeoF offers an ultra-low remote access latency, enabling very fast network-based storage I/Os.

It is highly appealing to integrate the two promising technologies together and build an efficient Arm-based storage server with NVMeoF. However, we currently have very limited understanding on the field performance of NVMeoF on Arm-based platforms. Although it is known that substantial differences exist between Arm and x86 architectures, it still remains unclear whether such architectural differences have a real impact on the end-to-end performance of disaggregated storage with NVMeoF. Without extensive experimental studies, hardware and software architects are difficult to develop efficient Arm-based storage solutions to meet their infrastructural needs, and it also constrains the industry to obtain first-hand data on the potential performance benefit of deploying Arm-based storage services in real-world scenarios.

In this paper, we present an in-depth study of NVMe and NVMeoF performance on an Arm-based multi-core storage server. In this paper, we use the term *NVMeoF*, or interchangeably *remote flash*, to refer to the NVMe flash SSD that is accessed over network fabric, as opposed to the flash SSD accessed locally over a PCIe link, which is denoted simply as *NVMe* or *local flash*. We benchmark NVMeoF with a set of carefully designed synthetic workloads generated by FIO [7] and run the workloads on both local and remote NVMe devices to characterize their performance behaviors on the Arm platform. We find that NVMeoF is capable of saturating the storage device on the Arm-based server with only moderate CPU utilization. To the best of our knowledge, this work is the first in-depth study of NVMe and NVMeoF performance on Arm-based multi-core server hardware.

Our experimental results show that NVMeoF introduces minimal performance overhead in most cases. We also show that the Network Interface Card (NIC) plays an important role when stress-testing NVMeoF. In contrast to the common expectation, the NIC optimization, such as interrupt moderation, can substantially reduce the 95th percentile tail latency by up to 14.6% while improving the throughput by up to 42.5%. Other factors, such as I/O request size, queue depth, parallelism settings, etc., also play a non-trivial role in the overall performance. We have also observed that with NVMeoF, in some cases, the CPU utilization increases on the

host side, mainly due to the involved kernel level overhead; in the meantime, the CPU utilization decreases on the target side, due to the reduced application-level computation. To improve NVMeoF performance and reduce CPU burdens, we also discuss the potential beneficial changes that can be made to Arm-based servers. Our findings have also been reported to Arm and Broadcom for future optimizations. It is our hope that this work can lay out a foundation for system architects and practitioners to build an NVMeoF based high-speed storage on the Arm architecture.

The rest of paper is organized as follows. Section II gives the background. Section III introduces the methodology. Section IV and V give the experimental results and discuss the related system implications. Related work is presented in Section VI. The final section concludes this paper.

II. Background

A. Arm Servers

As a family of Reduced Instruction Set Computer (RISC) architectures, the Arm architecture [1] has evolved quickly since its first introduction to the market. For their high flexibility, high efficiency, low power consumption, and low price, over the past years, Arm processors have become the dominant platform in mobile and embedded systems, such as smartphones, tablets, and single-board computers, etc. More recently, the 64-bit ARMv8 processors [2], which are designed with significantly enhanced computing capability and optimized for servers, promise to deliver both high performance and low power consumption [4], making it a particularly suitable platform for data center and cloud applications.

B. Non-Volatile Memory Express

Non-Volatile Memory Express (NVMe) [16] is a highly optimized host controller interface for accessing Non-volatile Memory (NVM) devices over PCIe links. NVMe is designed to provide scalable and efficient access for high-speed storage devices, such as NAND flash and the next-generation NVM SSDs. A large number of deep, paired command queues is maintained in the host memory, as the interface between the NVMe driver and the controller. NVMe enables users to fully exploit the performance potential of flash and other high-speed storage by efficiently supporting more processor cores, lanes per device, I/O threads, and I/O queues. The lockless command submission and completion also help keep the latency at a low level. For its high performance and efficiency, NVMe becomes a de facto technology for PCIe based SSD devices.

C. NVMe-over-Fabrics

As an extension to the NVMe standard, NVMe over Fabrics (NVMeoF) [17] enables fast access to remote NVMe devices over various high-speed network fabrics. NVMeoF eliminates unnecessary protocol translations that are originally needed on the I/O path from the host to the remote device, minimizing the overhead involved in remote access. Two main types of transport technologies supported by NVMeoF are Remote Direct Memory Access (RDMA) and Fibre Channel (FC). Our test system uses the RDMA based NVMeoF. RDMA is widely available in both traditional high-performance computing domain and modern data centers. Specifically, NVMeoF uses RDMA to move data from one memory address space to another without intervention of operating system (OS) or processor, resulting in lower overhead and faster response time, with latencies usually within microseconds.

III. Methodology

Our experiments run on a two-node system, including a host machine (initiator) and a storage server (target). The target storage server is a Broadcom 5880X Stingray machine, which features an 8-core 3GHz ARMv8 Cortex-A72 CPU, 3 DDR memory channels (48GB Memory capacity) and 100Gbps full featured NetXtreme NIC. The storage device is a 400GB Intel Data Center P3600 SSD, which is a high-end enterprise flash SSD supporting 2,100 MB/sec and 550 MB/sec bandwidths for sequential read and write operations, respectively. The host machine features an Intel(R) Core(TM) 8-core i7-6700 3.40GHz CPU, 16GB memory and a 50Gbps Broadcom NIC. The host and the target are directly connected through an Leoni 100Gbps ParaLink@23 cable.

We configure the network speed on both host and target sides to be 50Gbps for all the tests, which guarantees that network is not the bottleneck in the experiments. RDMA over Converged Ethernet (RoCEv2) over IPv4 is set up as an NVMeoF transport type, and the port is enabled for the initiator to discover. We benchmark the system using FIO [7].

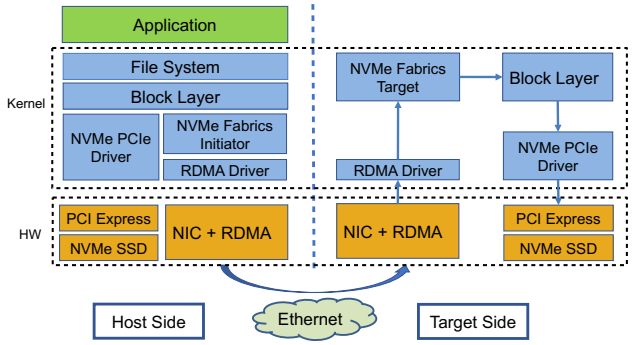

Fig. 1: An Illustration of the System Architecture.

Figure 1 shows the architecture of our experimental system. The applications, such as FIO, run on the host side. There are two storage setups, either directly attached on the host node (x86) or remotely accessed via the network to the target node (Arm). In our setting, the Arm machine serves as the storage node to provide high performance NVMeoF storage service.

Server/Client	Arm/x86	x86/Arm
Bandwidth (Gb/s)	45.42	45.40
Latency (μs)	3.26	3.17

TABLE I: RoCEv2 Performance Evaluation.

Table I shows that the RoCEv2 solution on our experimental system delivers very low latency and high bandwidth over the network. Arm or x86 in the first row represents the machine at the server side, while the other one behaves as the client. We can see that Arm and x86 can provide comparable network performance, as a server or a client. The bandwidth can be as high as 45.42Gb/s and the latency can be as low as 3.17μs, which ensures the network overhead to be minimal during the measurement with NVMeoF.

IV. Experimental Results and Insight

A. Effect of Request Size

According to prior studies [3], [5], [8], request size has a significant impact on the performance of network and storage I/O. We benchmark the NVMeoF storage by varying the sequential read request size from 4 KB to 128 KB with the number of concurrent jobs and IODepth being set to 8 and 1, respectively.

Finding #1 In Figure 2, we can see that for both local access (NVMe) and the remote access (NVMeoF), the latency increases as the request size increases, which is as expected. Compared to NVMe, NVMeoF shows almost identical latency distribution, indicating that it incurs minimal overhead. For example, for the 128 KB requests, the 95th percentile latency is 1,155μs for local access and 1,139μs for remote access, which is 1.39% faster. As shown in Figure 3, the bandwidth increases from 314 MB/sec to 1,361 MB/sec when the request side increases from 4 KB to 128 KB for NVMeoF. NVMeoF causes at most 20% bandwidth penalty, compared to the local access with NVMe.

Figure 4 shows the CPU utilization on the host and the target servers. We can see that as the request size increases from 4 KB to 128 KB, the CPU utilization on the target server decreases to below 2% with NVMeoF. The low CPU utilization indicates that the Arm processor on the target server is powerful enough to handle I/Os of high-speed SSD devices. This is because the storage server with NVMeoF is dedicated for processing storage I/Os only, without involving the application-level computation. Thus, the larger the request size is, the fewer I/O operations are involved and the lower CPU utilization is on the target server.

On the host side, the CPU utilization remains below 8% in all the cases. It reaches the lowest (2.11%) when the request size is 64 KB. However, it jumps to 7.57% when the request size is 128 KB. By looking further at the CPU usage, we can find that the increase is mainly because the kernel level overhead (the SYS time reported by `mpstat`) significantly increases from 1.27% (64KB) to 5.96% (128KB).

Discussion #1 In order to increase the overall bandwidth, developers may increase the request size using techniques such as batching, since NVMeoF is more friendly to large sequential requests. But an excessively large request size could also incur extra overhead to the host CPU. A proper request size should be measured by users to achieve both high performance and low CPU utilization.

B. Effect of Parallelism

Flash SSDs have rich internal parallelism resources [6], but fully exploiting the performance potential still depends on the interface. Different from Serial-Attached SCSI (SAS), where each connection from the CPU core to the flash SSD is limited by the SAS Host Bus Adapter (HBA) and synchronized locking, NVMe enables massive parallelism with up to 65,535 queues and lockless connections, which can provide each CPU core with dedicated queue access to each SSD. As a result, I/O parallelism is expected to have a significant effect on the performance of NVMeoF. We benchmark the NVMeoF storage with different parallelism settings by varying the sequential read job number from 1 to 16 with request size and IODepth being set to 4KB and 128, respectively.

Finding #2 In Figure 5, we can see that the latency increases when the number of concurrent jobs (i.e., parallelism degree) increases. The latency is almost identical between local access and remote access with 1 to 8 jobs. However, when the number of jobs increases to 16, the 20th percentile latency of remote access increases to 3.2ms, compared to 1.6ms for local access, but for the 95th percentile latency, remote access is 6.9ms, which is even 20.7% lower than local access (8.7ms). This is mostly because disaggregating storage from the host effectively removes the I/O-related CPU overhead from the application. The average latencies are identical locally and remotely, being around 4.5ms.

Figure 6 shows that the bandwidth increases to the peak when the number of jobs increases from 1 to 4, and stays at the peak when we continue to increase from 4 jobs to 16 jobs. Compared to local access, the bandwidth penalty introduced by remote access is up to 2.22%. Figure 7 shows the CPU utilization. At the target server side, the CPU utilization increases slightly from 2.82% to 4.35%, as the number of jobs increases from 1 to 16. The CPU utilization at the host side increases from 5.39% to 8.34% accordingly. For local access with NVMe, the CPU utilization reaches the peak, 7.32%, with 8 jobs, which is 0.64 percentage points (p.p.) lower than that on the host, but 3.08 p.p. higher than that on the target.

Discussion #2 For remote access with NVMeoF, increasing parallelism improves the bandwidth significantly, when the parallelism degree is low (less than 4). However, over-parallelization does not bring additional benefits in terms of bandwidth. As for the latency, both average latency and tail latency increase as parallelism increases. However, tail latency (e.g., the 95th percentile latency) of remote access with NVMeoF is observed to be even lower than local access when parallelism degree is high. For application users and developers, it would be beneficial to find an optimal parallelism degree (4 jobs in our case) to achieve both high bandwidth and low latency, according to their Quality of Services (QoS) requirements.

C. Computational Cost

In this section, we analyze the CPU utilization by varying the random write request size from 4 KB to 128 KB with 8 concurrent jobs and IODepth being set to 128.

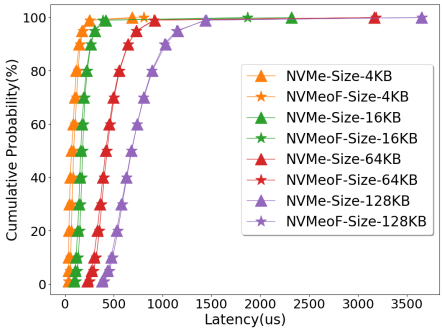

Fig. 2: Latency v.s. Request Size

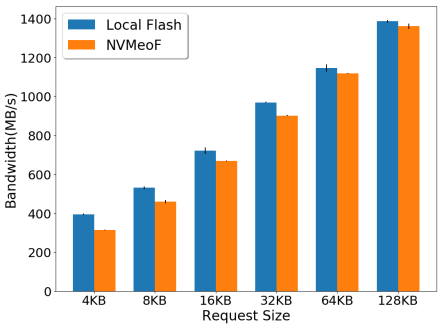

Fig. 3: Bandwidth v.s. Request Size

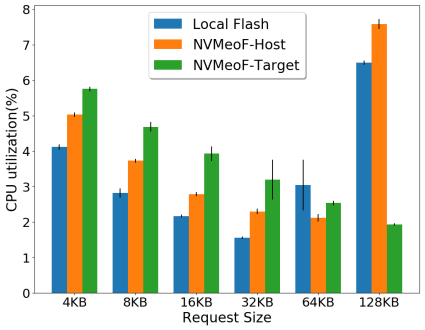

Fig. 4: CPU Utilization v.s. Request Size

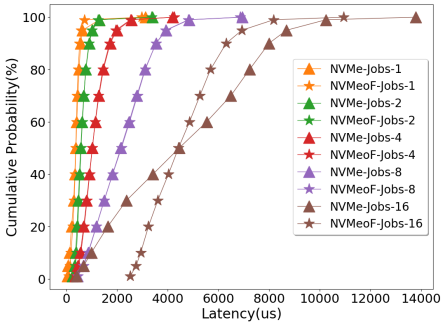

Fig. 5: Latency v.s. Num. of Jobs

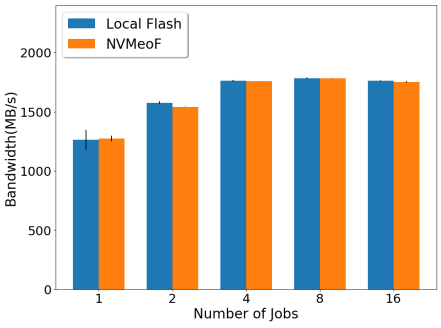

Fig. 6: Bandwidth v.s. Num. of Jobs

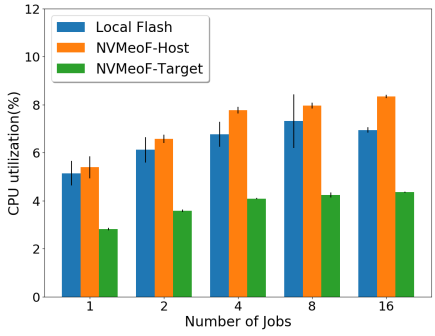

Fig. 7: CPU Utilization v.s. Num. of Jobs

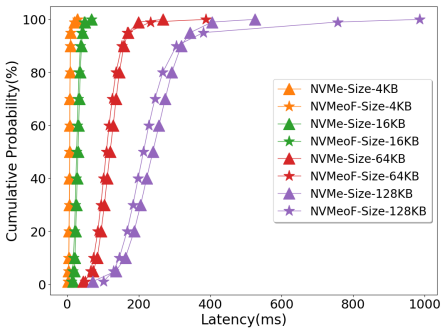

Fig. 8: Latency v.s. RandWrite Size

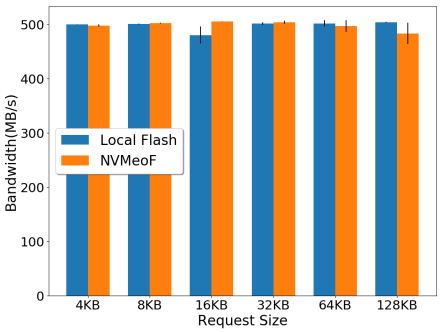

Fig. 9: Bandwidth v.s. RandWrite Size

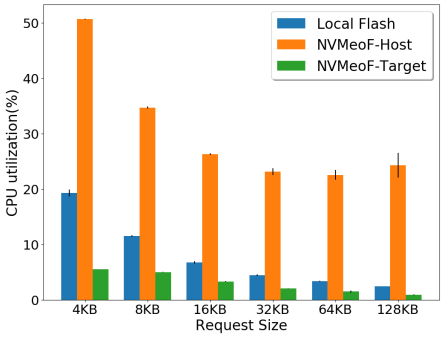

Fig. 10: CPU v.s. RandWrite Size

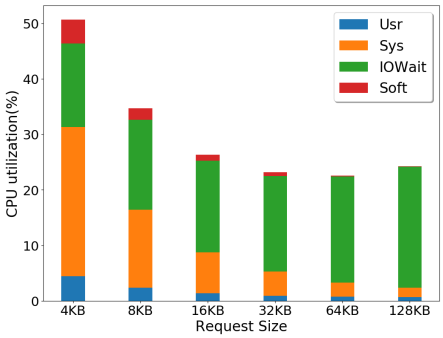

Fig. 11: CPU Utilization Breakdown

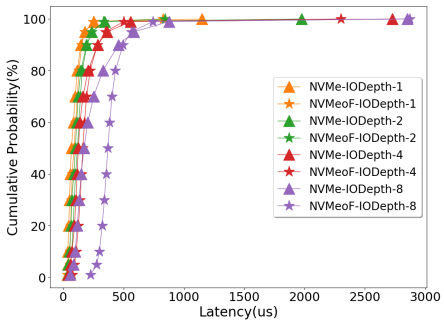

Fig. 12: Latency v.s. IODepth

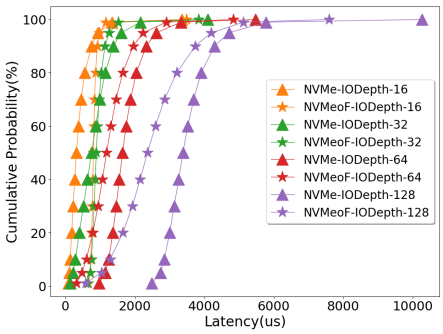

Fig. 13: Latency v.s. IODepth

Finding #3 Figure 9 shows that the bandwidths of local and remote accesses remain stable and comparable at around 500 MB/sec, indicating that with parallel I/Os, the bandwidth can be fully exploited even remotely with NVMeoF. As expected, the latency increases as the request size increases for random writes (see Figure 8). In all the cases, remote access shows a longer tail latency than local access. For example, when request size is 128 KB, the 95th percentile latency for remote accesses is 380ms, in contrast to 343ms for local access. The relatively longer tail latency with NVMeoF can be explained

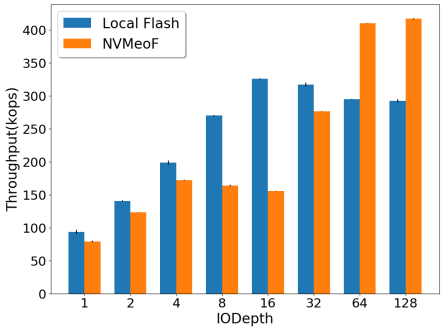

Fig. 14: Throughput v.s. IODepth

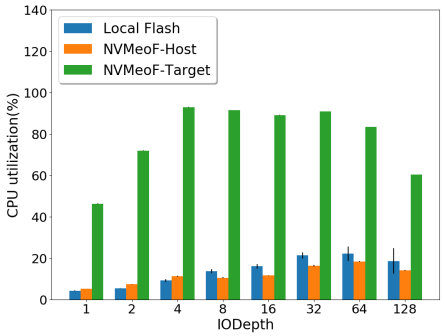

Fig. 15: CPU Utilization v.s. IODepth

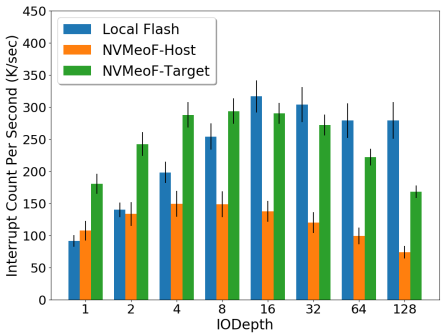

Fig. 16: Interrupts v.s. IODepth

by the observed heavy CPU utilization burden on the host side. For example, as Figure 10 shows, the CPU utilization on the host side is 50.7% compared to 19.2% for the local access when request size is 4 KB.

In order to understand the percentage of CPU utilization while executing different tasks, we use `mpstat` to break down the CPU utilization into four categories, namely USR, SYS, IOWAIT, and SOFT. As we can see in Figure 11, when request size is 4 KB, the SYS (kernel) time is dominant, which is about 26.9%. The IOWAIT time accounts for about 15%. When request size increases, the percentage of IOWAIT time slowly increases, while the percentage of SYS time quickly decreases. This is mostly because processing I/O requests incurs significant kernel level overhead, which causes heavy CPU burdens, and the overhead can be amortized over a large amount of data transfer with a large request size.

Discussion #3 Since small (4KB) random writes incur a high CPU utilization on the NVMeoF host side, organizing large requests can effectively reduce the undesirable CPU burdens. Another possible consideration is to offload the CPU overhead. Right now, both the control plane and the data plane of the NVMeoF driver are running on the CPU, which results in a heavy use of CPU resources. The data plane can be moved to the NIC to offload the CPU burden, while the control plane still runs on the CPU, as discussed in prior talk [10].

D. Effect of CPU Capacity

In order to study the effect of CPU capacity, we only use one core of the Arm server by disabling the rest in this set of experiments. We benchmark the NVMeoF storage by setting the sequential read IODepth from 1 to 128 with 8 concurrent jobs and request size being set to 4 KB.

Finding #4 As shown in Figure 12, when the IODepth increases from 1 to 8, the latency increases and the remote access is slower than the local access in general. For example, when the IODepth is 1, the 95th percentile latency is 178μs for both local and remote access. When the IODepth is 8, for the 50th percentile latency, it is 171μs for local access and 370μs for remote access, which represents about 116.4% overhead. However, the 95th percentile latency is 586μs and 565μs for local and remote access respectively, meaning that NVMeoF reduces the latency by about 3.58%.

When we continue to increase the IODepth from 16 to 128, the latency distribution becomes very different, as shown in Figure 13. When IODepth is 16, the 95th percentile tail latency is 983μs for remote access, which is 6.16% longer than local access (926μs). When IODepth is 32, the latencies of local and remote access get close. When IODepth is 64, the latency of remote access becomes lower than that of local access. For example, the 95th percentile latency when the IODepth is 64 is 2.61ms for local access and is 2.23ms for remote access, which is an improvement of 14.6%. When IODepth equals 128, the 95th percentile latency is 4.71ms for local access and 4.19ms for remote access, meaning that NVMeoF actually reduces the latency by 11.04%.

For local access, the throughput generally increases with the IODepth, despite a slight decrease after reaching the peak at the IODepth of 16. For remote access, the throughput increases when IODepth increases from 1 to 4, and then decreases (about 10%) when IODepth continues to increase to 16. This is because the CPU, instead of the SSD, becomes the bottleneck of the system, since we only use one Arm core in this case (see Figure 15). As we continue to increase IODepth, the throughput increases again and reaches the peak when IODepth is 64 and 128. Compared with local access (292 kop/s), the throughput of remote access with NVMeoF is 416 kop/s when IODepth is 128, which is a substantial improvement of 42.5%.

This performance improvement is mainly due to the interrupt moderation feature on the NIC. Figure 16 shows the number of interrupts generated per second on NVMeoF host side and target side, together with interrupts collected during the test with local flash. As shown in Figure 14 and Figure 16, the system triggers an interrupt for every request to the local SSD. However, when accessing the remote SSD, interrupt moderation on the NIC is enabled. Multiple packets are handled for each interrupt so that the overall interrupt-processing efficiency is improved and the CPU utilization is decreased. As shown in Figure 14 and Figure 15, compared to using a low (32) IODepth, when IODepth is 128, the improved efficiency with interrupt moderation leads to a higher throughput and a lower CPU utilization.

In contrast to the common belief that interrupt moderation would incur an increase of latency, Figure 12 and Figure 13

show that when the IODepth is small (1-8), there is small latency overhead for remote access, but when the IODepth is large (64-128), both the bandwidth and latency will benefit from this feature. Even though interrupt moderation is usually believed to involve performance trade-off between latency and throughput, it is evidently beneficial for both latency and throughput when the requests are intensive with NVMeoF.

Discussion #4 The CPU capacity on the target side and IODepth have a significant impact on the performance. The NIC adapter plays an important role in the NVMeoF performance. Our observations show the effectiveness of enabling interrupt moderation for both low latency and high throughput. We need to identify an optimal NIC adapter configuration based on the workload characteristics and the Quality of Service (QoS) requirements.

V. System Implications

In this section, we discuss several important implications for system designers and practitioners to effectively deploy NVMeoF on Arm-based machines.

(1) **Simplifying the IO stack**. Our experimental results clearly show that the kernel-level execution time generally dominates the CPU usage, especially for small random write operations, which demand a large amount of processing time. Simplifying the kernel-level IO stack, such as replacing kernel-level drivers with user-level drivers [19], we expect to have reduced CPU overhead due to the bypassing of multiple intermediate layers.

(2) **Clustering I/Os**. Handling each single I/O incurs non-trivial processing overhead, especially for high-speed NVMe devices. Application and system designers may consider to merge or convert many small I/Os to a few large ones. Techniques, such as aggressive prefetching and clustering, can help combine small I/Os together, effectively amortizing the processing overhead. Storage and file systems may also reorganize the data layout, such as grouping the related data close together, to create more opportunities for large I/Os.

(3) **Offloading the CPU tasks**. As discussed in prior talk [10], offloading the data plane of NVMeoF to NIC, while only keeping the control plane on CPU, can effectively reduce the high CPU utilization. Although such an approach may incur additional hardware cost, considering the substantially increased efficiency of CPU usage, it is a reasonable investment in a long term for enhancing the overall system performance.

(4) **Enabling interrupt moderation**. Interrupt moderation allows users to manage the rate of interrupts to the CPU during packet transmission and reception. Without interrupt moderation, the system triggers an interrupt for every transmitted and received packet. Although the latency on each packet is minimized with NVMeoF, a large amount of CPU resources is still spent for the interrupt-processing overhead, which is particularly heavy for handling intensive I/O traffic with high-speed storage devices. We have observed that enabling interrupt moderation can effectively reduce the CPU burden and improve both latency and throughput for intensive storage I/O traffic.

(5) **Replacing interrupts with polling**. Prior work [21] has discussed polling and interrupting with NVM devices. Our experiments also show that NVMeoF generates a large amount of interrupts, which incur heavy CPU burden and non-trivial processing overhead. By replacing interrupts with polling, as SPDK [19] does, the CPU usage for interrupt handling is expected to decrease but the polling cost is to be involved. It still needs careful consideration on which manner is most appropriate and efficient. As the storage speed continues to increase (e.g., the 3D XPoint based Intel Optane devices [9]), addressing this question is becoming an imminent need.

VI. Related Work

NVMe over Fabrics (NVMeoF) is a relatively new topic. Limited literature and analysis are available. The design and architecture of NVMeoF can be found in prior talks [11], [13], [14]. Klimovic et al. have proposed a system, called ReFlex, to provide remote flash with nearly identical performance to local flash [12]. Xu et al. have studied the performance of containerized applications on local and remote storage [20]. Guz et al. have compared different storage disaggregation methods, including NVMeoF and iSCSI, on Intel x86 platform [8]. Different from prior work, we have studied the interactions between NVMeoF and the Arm architecture, focusing on the utilization of CPU, storage, and network resources and their performance impact. To the best of our knowledge, this work is the first in-depth study of NVMe and NVMeoF performance on Arm-based multi-core server hardware.

VII. Conclusion

In this paper, we present a comprehensive experimental study of NVMe and NVMeoF on Arm-based server hardware. We find that in most cases, NVMeoF incurs minimal overhead compared to the local NVMe. By setting parallelism properly, NVMeoF can reduce the 95th percentile tail latency by 20.7%, while retaining nearly identical bandwidth. We have also observed that NVMeoF may cause increased CPU utilization on the host side in some cases, but decreases the CPU utilization on the target side due to the reduced application-level computation. By setting the configuration for the NIC properly, NVMeoF can improve the throughput by up to 42.5%, and reduce the 95th percentile tail latency by up to 14.6%, because of optimizations such as interrupt moderation. Our results also show that the current Arm-based storage server is able to deliver sufficient CPU capacity for handling intensive storage I/Os, making it a highly efficient platform for storage disaggregation.

Acknowledgements

We thank the anonymous reviewers for their constructive feedback and insightful comments. We also thank Haresh Sakariya from Broadcom Inc. for his technical support. This paper was partially supported by National Science Foundation under Grants CCF-1453705 and CCF-1629291.

REFERENCES

[1] ARM. ARM Processor. https://www.arm.com/products/processors.

[2] ARM. ARMv8. https://www.arm.com/files/downloads/ARMv8_white_paper_v5.pdf.

[3] P. Balaji. Sockets vs RDMA Interface over 10-Gigabit Networks: An In-depth Analysis of the Memory Traffic Bottleneck. In *RAIT Workshop 04*, 2004.

[4] Cavium. ThunderX2. https://www.cavium.com/product-thunderx2-arm-processors.html.

[5] F. Chen, D. A. Koufaty, and X. Zhang. Understanding Intrinsic Characteristics and System Implications of Flash Memory based Solid State Drives. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '09, June 15-19 2009.

[6] F. Chen, R. Lee, and X. Zhang. Essential Roles of Exploiting Internal Parallelism of Flash Memory based Solid State Drives in High-Speed Data Processing. In *Proceedings of the 17th International Symposium on High Performance Computer Architecture*, HPCA '11, Feb 12-16 2011.

[7] FIO. Fio. https://github.com/axboe/fio.

[8] Z. Guz, H. H. Li, A. Shayesteh, and V. Balakrishnan. NVMe-over-Fabrics Performance Characterization and the Path to Low-overhead Flash Disaggregation. In *Proceedings of the 10th ACM International Systems and Storage Conference*, SYSTOR '17, May 22-24 2017.

[9] Intel. Intel Optane Memory. https://www.intel.com/OptaneMemory.

[10] Kalray. IOProcessor. https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2016/20160811_S304D_Couvert.pdf.

[11] J. Kim and D. Fair. How Ethernet RDMA Protocols iWARP and RoCE Support NVMe over Fabrics. https://www.snia.org/sites/default/files/ESF/How_Ethernet_RDMA_Protocols_Support_NVMe_over_Fabrics_Final.pdf.

[12] A. Klimovic, H. Litz, and C. Kozyrakis. ReFlex: Remote Flash $\approx$ Local Flash. In *Proceedings of the 22nd ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '17, April 8-12 2017.

[13] J. Metz. Let's Talk Fabrics. https://www.snia.org/sites/default/files/ESF/Lets-Talk-Fabrics-NVMe-Over-Fabrics.pdf.

[14] D. Minturn and J. Metz. Under the Hood with NVMe over Fabrics. https://www.snia.org/sites/default/files/ESF/NVMe_Under_Hood_12_15_Final2.pdf.

[15] MongoDB. MongoDB. https://www.mongodb.com/.

[16] NVMe. NVM Express. https://nvmexpress.org/.

[17] NVMeoF. NVMe Over Fabrics. https://www.nvmexpress.org/wp-content/uploads/NVMe_Over_Fabrics.pdf.

[18] RocksDB. RocksDB. https://rocksdb.org/.

[19] SPDK. SPDK. http://www.spdk.io/.

[20] Q. Xu, M. Awasthi, K. T. Malladi, J. Bhimani, J. Yang, and M. Annavaram. Performance Analysis of Containerized Applications on Local and Remote Storage. In *Proceedings of the 33rd International Conference on Massive Storage Systems and Technology*, MSST '17, May 15-19 2017.

[21] J. Yang, D. B. Minturn, and F. Hady. When Poll is Better Than Interrupt. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, FAST '12, Feb 14-17 2012.