When NVMe over Fabrics Meets Arm: Performance and Implications

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

Eric Anger
Arm Inc.
Email: Eric.Anger@arm.com

Feng Chen
Louisiana State University
Email: 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 NVMe over Fabrics on Arm-based Data Center SoC and to gain insight into the implications of their designs and deployment in the 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, we have observed throughput increases by 42.5% and the 95th percentile tail latency decreases by 19% in our stress test. Based on our measurement results, we also discuss several design implications on integrating NVMeoF on Arm's platform. Our studies show that this system solution can balance the computation, network and I/O resources for the data-center storage servers. Our findings have also been reported to Arm and Broadcom for future optimizations.

I. Introduction

Arm CPUs have been regarded as the platform of choice in 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 highly competitive in traditional server environments.

An important application of Arm servers is to host high-speed data stores, such as MongoDB [12] and RocksDB [13], 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 host CPUs.

Storage disaggregation can well address the challenge by separating the computing and storage resource usage. *NVMe over Fabrics* (NVMeoF), the recently released protocol standard for accessing NVMe devices over RDMA-capable networks, is particularly attractive. With fast interconnect technologies, 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 a highly efficient Arm-based storage server with NVMeoF. However, we still have limited understanding on the field performance of NVMeoF on Arm based servers. Although there are substantial differences between Arm and x86 architectures, it is unclear whether these difference have a material impact, positive or negative, on NVMe storage performance. The lack of clear performance data not only limits the ability of hardware and software architects to develop efficient Arm solutions, and also constrains the industry to obtain first-hand data on the potential performance benefit of deploying Arm solutions to meet their infrastructural needs. The increasing demand of Arm-based solutions and growing investments in Arm-based platform makes this problem one of the key importance.

We present the first in-depth study of NVMe and NVMeoF performance on a multi-core Arm server hardware. In this paper, we use the term *NVMeoF*, or interchangeably *remote flash*, to refer to the NVMe flash SSD that is accessed over a high-bandwidth network, as opposed to the flash SSD accessed locally over a PCIe link, denoted simply as *Flash*. We benchmark NVMeoF with carefully designed synthetic workloads generated by FIO [6] and run workloads on both local and remote NVMe connections to characterize their performance on Arm based storage server. We find that the NVMeoF is capable of saturating the storage device on Arm-based server with 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 NVMe over Fabrics introduces minimal performance overhead in most cases. We also show that the Network Interface Card (NIC) plays an important role when stress-testing NVMe over Fabrics. Contrast to the common expectation, the NIC optimization, such as interrupt moderation, can substantially reduce the 95th percentile latency by up to 19% while improving the throughput by up to 42.5%. Other factors related to software design and implementation also play a non-trivial role for overall performance. These factors include the I/O sizes, I/O queue depth, parallelism settings, etc. In some cases, the CPU utilization caused by NVMeoF increases significantly on the host side, because of more kernel level overhead; in the

meantime, a noticeable CPU utilization decrease on the target side can always be found, due to the reduced application-level overhead. To improve NVMeoF performance and reduce the CPU burdens, we also discuss the potential beneficial changes that can be made to the 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 shows our experimental results and discusses the design 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 Computing (RISC) architectures, the Arm architecture has evolved significantly since its first introduction to the market. For their high flexibility, high efficiency, low power consumption, and low price, over the past ten years, Arm processors have become the dominant platform in the mobile and embedded systems, such as smartphones, tablets, and single-board computers. More recently, the 64-bit Arm v8 Data Center & Cloud Processors, which are designed with enhanced computing capability, promise to deliver both high performance and low power consumption [4], making it a particularly suitable platform for traditional server systems.

B. Non-Volatile Memory Express

Non-Volatile Memory Express (NVMe) is an optimized, scalable, high-performance, host controller interface for accessing local non-volatile memory devices over PCIe. NVMe is designed to provide efficient access to storage devices built with non-volatile memory, such as flash memory. NVMe is based on a large number of deep, paired Submission and Completion Queues, allocated in host memory. These paired parallel queues are the interface between the NVMe driver and the NVMe controller, which manage them cooperatively. NVMe enables users to fully utilize the performance potential of flash 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 low. While other proprietary protocols exist, NVMe is the most widespread technology for PCIe based SSD devices.

C. NVMe-over-Fabrics

NVMe over Fabrics (NVMeoF) is a recent extension to the NVMe standard that enables access to remote NVMe devices over different network fabrics. It eliminates unnecessary protocol translations along the I/O path to the remote device and keeps the overhead of the remote access minimal. NVMeoF currently supports two types of fabric transport, RDMA and Fibre Channel. RDMA enjoys a wider availability both in traditional HPC domain as well as in modern data centers. Specifically, NVMeoF uses RDMA to move data from one memory address space to another without invoking the OS or the processor. This results in lower overhead and faster response time to queries, with latencies usually within microseconds.

III. Methodology

Our experiments run on a two-node system, including a host machine and a target storage server. The target storage server is a Broadcom 5880X Stingray machine, which features an 8-core 3GHz Arm v8 Coretx-A72 CPU, 3 DDR memory channels (48GB Memory Space) and 100Gbps full featured NetXtreme NIC. The storage device is a single 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. 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 with a 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 a NVMeoF transport type, and the port is enabled for the initiator to discover. We benchmark the system using FIO [6].

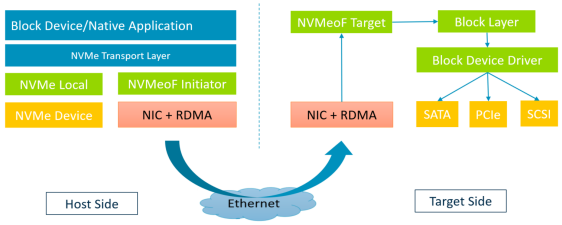

Fig. 1: Benchmarking Architecture

Figure 1 shows the architecture of our test bed. 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 NVMe over Fabrics service.

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

TABLE I: RoCEv2 Performance

Table I shows that the RoCEv2 solution on the Broadcom high performance NIC delivers extremely low latencies and high bandwidth over the network. Arm/x86 in the first row represents the server machine. We can see that the bandwidth can be as high as 45.42Gb/s and the latency can be as low as 3.17us, which ensures the network overhead to be minimal during NVMeoF measurement.

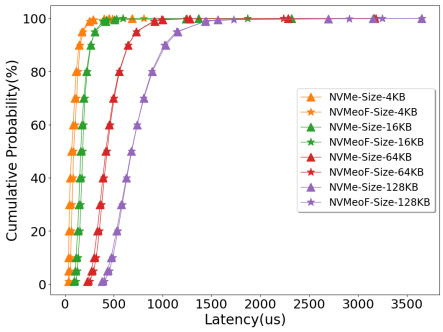

Fig. 2: Latency v.s. Request Size

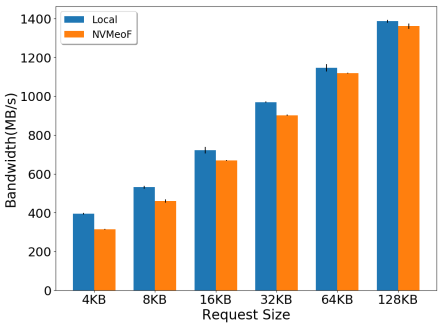

Fig. 3: Bandwidth v.s. Request Size

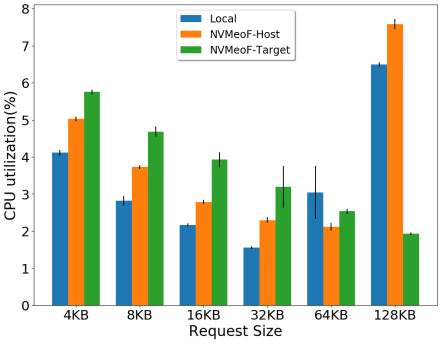

Fig. 4: CPU Utilization v.s. Request Size

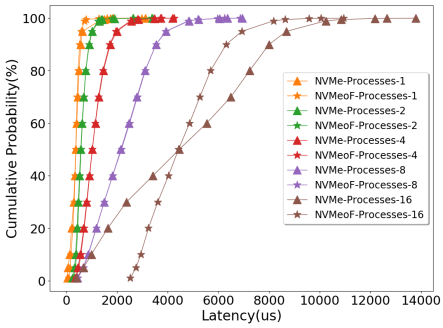

Fig. 5: Latency v.s. Process Number

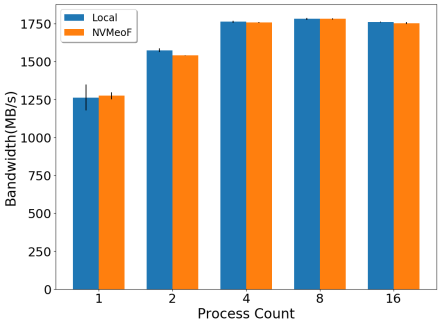

Fig. 6: Bandwidth v.s. Process Number

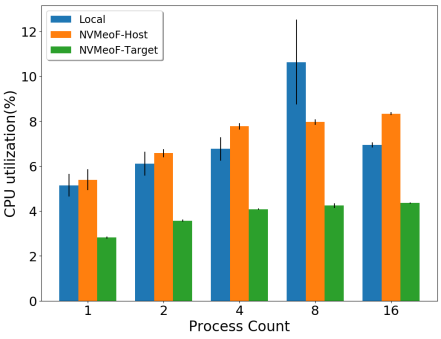

Fig. 7: CPU Utilization v.s. Process Number

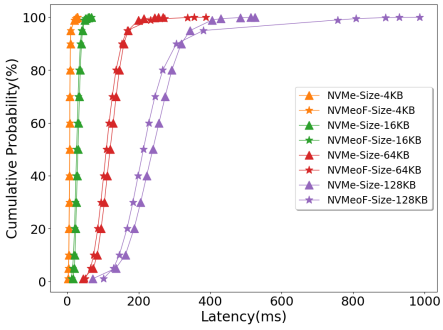

Fig. 8: Latency v.s. RandWrite Size

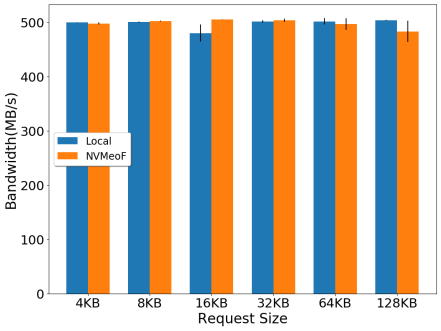

Fig. 9: Bandwidth v.s. RandWrite Size

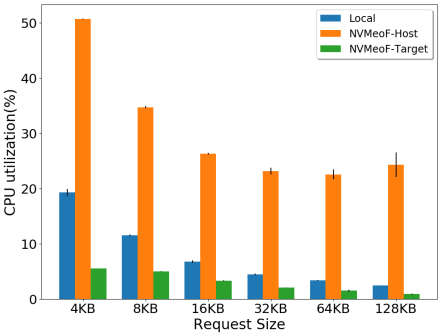

Fig. 10: CPU v.s. RandWrite Size

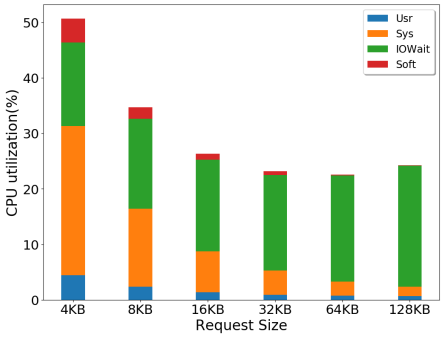

Fig. 11: CPU Utilization Breakdown

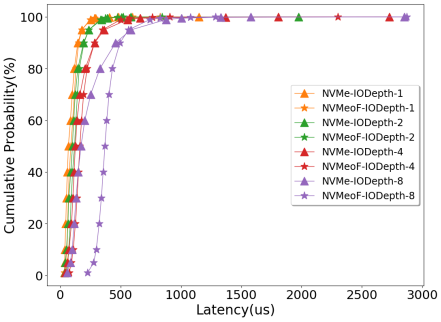

Fig. 12: Latency v.s. IODepth

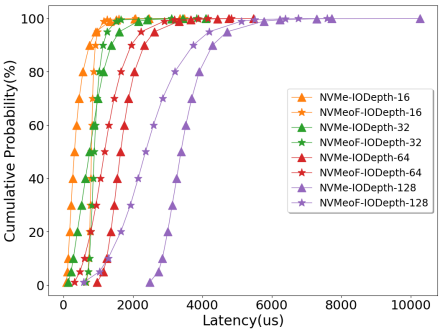

Fig. 13: Latency v.s. IODepth

IV. Experimental Results and Implications

A. *Effect of Request Size*

According to prior studies [3], [5], [7], request size has a significant impact on the performance of network and storage I/O. We benchmark the NVMeoF by varying the sequential read request size from 4 KB to 128 KB with 8 concurrent processes.

Finding #1 In Figure 2, we can see that for both local

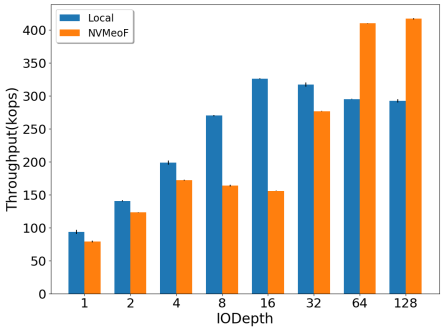

Fig. 14: Throughput v.s. IODepth

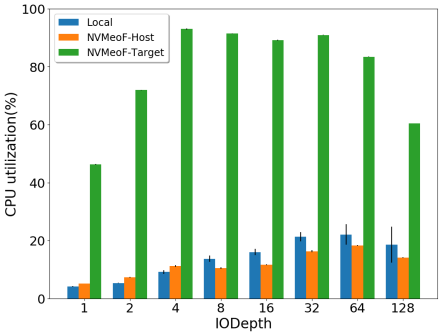

Fig. 15: CPU Utilization v.s. IODepth

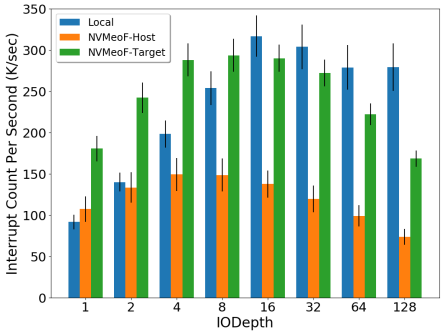

Fig. 16: Interrupts v.s. IODepth

access (NVMe) and the remote access (NVMeoF), the latency increases accordingly as the request size increases. Compared to NVMe, NVMeoF shows almost identical performance, indicating that it incurs minimal overhead. For example, for the 128 KB requests, the 95th percentile latency is 1,139μs for the local access and 1,155μs for the remote access, which adds about 1.5% overhead. 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, which causes at most 20% bandwidth penalty than the local access.

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%. The low CPU utilization also indicates that the Arm processor on the target server is powerful enough to saturate the high-end SSD device. This is because the storage server is dedicated for processing storage I/Os only, without involving the application-level costs. Thus, the larger the request size is, the less I/O operations are involved. On the 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 request size is 128 KB, because the kernel overhead increases from 1.27% to 5.96%.

Discussion #1 At the application level, in order to increase the overall bandwidth, developers need to increase the size of each request with techniques such as batching, since NVMeoF is more friendly to large sequential requests. But an excessively large request size will add extra overhead to the host CPU. So a proper request size should be measured by the users in order to achieve both high performance and low CPU utilization.

B. Effect of Parallelism

Different from SAS (Serial-Attached SCSI), 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 64K queues and lockless connections, which can provide each CPU core with dedicated queue access to each SSD. As a result, parallelism is expected to have a significant effect on the performance of NVMeoF. We benchmark the system with different parallelism settings. We benchmark the NVMeoF by setting the sequential read process number from 1 to 16 with request size being 4 KB.

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

Figure 6 show that the bandwidth increases to the peak when the number of process increase from 1 to 4, and stays the peak when we continue to increase the process number from 4 to 16. The bandwidth penalty of remote access is up to 1.23% by varying the process count. Figure 7 shows the CPU utilization. At the target server side, the CPU utilization increases slightly from 2.82% to 4.35%, as process number increases from 1 to 16. The CPU utilization at the host side increases from 5.39% to 8.34% accordingly. For the local testing, the CPU utilization reaches the peak, 10.63%, when process number is 8, which is 2.67% higher than NVMeoF-Host.

Discussion #2 For remote access with NVMeoF, increasing parallelism improves the bandwidth significantly, when the parallelism degree is small (less than 4). However, over-parallelization does not benefit the performance from bandwidth perspective. 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 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 in our case) to achieve both high bandwidth and low latency, according to their Quality of Services (QoS) requirements.

C. Computational Cost

Based on the observations in Section IV-A and Section IV-B, CPU consumption on NVMeoF host side is heavier

than the local access. In this section, we analyze the CPU utilization by varying the random write request size from 4 KB to 128 KB with 8 concurrent processes.

Finding #3 Figure 9 shows that the bandwidths of local and remote accesses keeps stable and comparable around 500 MB/sec. As expected, the latency increases as the request size increases for random writes (see Figure 8). In all the cases, remote access shows longer tail latency than local access. For example, when request size is 128 KB, the 95th percentile latency for remote accesses increases from 343ms to 380ms than the local access, while the average remote access latency decrease from 235ms to 226ms compared to local access for the 128 KB requests. But the benefit of the average latency comes from high CPU utilization cost 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, SOFT. As we can see from Figure 11, when request size is 4 KB, the SYS (kernel) level execution time is dominant, which is about 26.9%. The IOWAIT execution time is about 15.0%. When the request size increases, the IOWAIT execution time percentage keeps stable, while the SYS level execution time percentage decreases.

Discussion #3 Since CPU utilization on NVMeoF host side increases significantly for small (4 KB) random writes, organizing into large requests can effectively reduce the CPU overhead. Another possible consideration is to offload the CPU overhead. Right now, both the control plane and data plane of the NVMeoF driver are running on CPU, which results in the heavy usage of CPU resource. Instead, the data plane can be moved to NIC, while control plane is still running on CPU to offload the CPU burden, as discussed in prior work [8].

D. Effect of CPU Capacity

From Section IV-A, Section IV-B and Section IV-C, we can see that CPU utilization is low on the Arm machine, which serves as the target side in NVMeoF. In order to study of the effect of CPU capacity, we only use one core of Arm server by disabling the rest. We benchmark the NVMeoF by setting the sequential read IODepth from 1 to 128 with request size being 4 KB.

Finding #4 From Figure 12, when the IODepth increases from 1 to 8, we can see the latency becomes longer in general. The remote access is slower than the local access. For example, when the IODepth is 1, the 95th percentile latency are both 178μs for local and remote access. When the IODepth is 8, the 95th percentile latency are 565μs and 586μs for local and remote access, which adds about 3.7% overhead. However, for the 50th percentile latency, it is 171μs for local access and 370μs for remote access, which adds about 116% overhead. Figure 13 shows that when we continue increasing the IODepth from 16 to 128, the latency distribution has a big change. When IODepth is 16, the 80th percentile latency is 559.4μs for local access and 865μs for remote access, which is about 54.7% overhead. When IODepth is 32, the latency of local and remote access is very close and have overlaps. When IODepth is 64, the latency of remote access becomes smaller than the local access. For example, the 80th percentile latency when the IODepth is 64 is 2.0ms for local access and is 1.6ms for the remote access, which is about 19% improvement. The 80th percentile latency when IODepth equals 128 is 3.9ms for local access and 3.2ms for remote access, which is about 18% improvement.

The bandwidth increases when the IODepth increases for the local test. However, for the remote access, the bandwidth increases when IODepth increases 1 to 4 and 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, as shown in Figure 15. If we continue to increase the number of threads, the bandwidth increases again and reach the peak when the thread number is 64 and 128. This can be explained by the interrupt moderation feature on NIC as shown in Figure 16. Figure 16 shows the number of interrupts issued per second on NVMeoF host side and target side, together with interrupts collected during local test. As shown in Figure 14 and Figure 16, the system triggers an interrupt for every request to 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 15 and Figure 16, when IODepth is 128, the higher interrupt moderation efficiency results in a lower CPU utilization and a higher bandwidth than when IODepth is 32.

In contrast to the typical 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 requests are intensive via NVMeoF.

Discussion #4 The performance of NVMe and NVMeoF is sensitive to the IODepth. Both the latency and bandwidth increase when IODepth increases. Meanwhile, the CPU capacity of the target side has a significant impact on the performance. Moreover, the Ethernet adapter plays an important role in the NVMeoF performance. For example, our observations above show the effectiveness of enabling interrupt moderation for both low latency and high throughput. We need to identify an optimal Ethernet adapter configuration based on the characteristics of their workloads and their Quality of Service (QoS) requirements. Finally, based on our findings, the Storage Performance Development Kit, which is known as SPDK [14], may improve the performance since it eliminate complicated kernel layers and use polling instead of interrupt to deal with the requests. We will leave it as the future work when the tool

is mature.

V. System Implications

In this section, we discuss several important implications for system designers and practitioners to effectively deploy NVMe over Fabrics 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 [14], 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 I/O clustering, can help combine small I/Os into large ones, 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 work [8], offloading the data plane of NVMeoF to NIC, while only keeping the control plane on CPU, can reduce the high CPU utilization. Although such an approach may incur additional cost to the hardware, 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, a large amount of CPU resources is spent for the interrupt-processing overhead, which is particularly heavy for handling intensive I/O traffic with high-speed storage devices. In fact, we have observed that enabling interrupt moderation can effectively reduce the CPU burden and improve both latency and throughput when requests are intensive.

(5) **Replacing interrupts with polling**. Prior work [15] has discussed polling and interrupting with NVM devices. Our experiments also show that NVMeoF generates a large amount of interrupts, which incurs heavy CPU burden and non-trivial processing overhead. By replacing interrupts with polling, as SPDK [14] 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 improve (e.g., the 3D XPoint based Intel Optane devices), addressing this question is becoming an imminent need.

VI. Related Work

NVMe over Fabrics is a relatively new topic. Limited literature and analysis are available. The design and architecture of NVMeoF can be found in prior talks [9], [11]. ReFlex is a software-based system for remote flash access, which provides nearly identical performance to accessing local flash [10]. Guz et al. focus on comparing the overhead of different storage disaggregation method. 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. Different from prior work, we focus on the interaction between CPU, storage and network, considering the architecture of NVMeoF.

VII. Conclusions

In this paper, we present a comprehensive experimental study on NVMe and NVMeoF on Arm based hardware. We find that in most cases, NVMeoF only incurs minimal overhead compared to the local NVMe. By setting parallelism properly, NVMeoF can reduce the tail latency by 20%, while retaining nearly identical bandwidth. For some operations, such as small random writes, NVMeoF causes high CPU utilization on the host side, which mainly comes from kernel level overhead. The CPU capacity on the target side is important for the performance as well. By setting the configuration for the NIC properly for some workloads, NVMeoF can improve the throughput by 42.5%, and reduce the 95th percentile tail latency by 19%, because of interrupt moderation optimizations. Our results also show that the current Arm-based server is able to deliver sufficient CPU capacity for handling storage I/Os, making it a highly efficient platform for storage disaggregation.

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 *In RAIT workshop 04*, page 2004, 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, 2009.

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

[7] 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, pages 16:1–16:9, New York, NY, USA, 2017. ACM.

[8] KALRAY. Ioprocessor. https://www.flashmemorysummit.com/English/Collaterals/Proceedings/2016/20160811_S304D_Couvert.pdf.

[9] 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.

[10] A. Klimovic, H. Litz, and C. Kozyrakis. Reflex: Remote flash local flash. In *Proceedings of ASPLOS'17*, 2017.

[11] 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.

[12] MongoDB. Mongodb. https://www.mongodb.com/.

[13] RocksDB. Rocksdb. https://rocksdb.org/.

[14] SPDK. Spdk. http://www.spdk.io/.

[15] 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, 2012.