Evaluation of Big Data Analytics Frameworks Using Extended Heaps over Fast Remote Storage Systems

Undergraduate Thesis

Theodoros Pontzouktzidis*
Department of Computer Science, University of Crete, Greece
Author
csd4336@csd.uoc.gr

Abstract

Today big data analytics frameworks running on managed runtimes, such as Java Virtual Machines (JVM) are extensively used in data centers for analyzing large datasets. Existing research has explored extending the managed heap using fast storage devices (e.g., NVMe SSDs) and remote memory. However, a comprehensive comparison of these techniques is lacking, underscoring the need for detailed studies evaluating their performance, and reliability.

In our comparative analysis, we investigate strategies for optimizing block storage access and sizing, focusing on local NVMe SSDs versus remote storage devices and DRAM. We break down and examine the latencies of local and remote storage systems, including SPDK NVMe over Fabrics (NVMe-oF) and Network Block Device (NBD). Using TeraHeap, which extends JVM capabilities by utilizing a second heap on a storage device, we compare its performance with local and remote systems. Our evaluation, conducted using 13 widely-used applications in two real-world big data frameworks, Spark and Giraph, demonstrates that remote NVMe and remote Ram-disk can match the performance of local NVMe.

Keywords: large analytics datasets, large managed heaps, fast storage devices, remote storage, systems performance

1 Introduction

Big data analytics frameworks running on managed runtimes, such as Java virtual machines (JVM) are widely deployed in data-centers to perform data analysis over large amount of datasets. The amount of data increases at a high rate but DRAM capacity in a single server scales slower than data growth. Existing approaches study the extension of the managed heap over fast storage devices (e.g., NVMe SSDs) and remote memory.

Related work that extend the managed heap beyond local DRAM either uses local storage devices, such as TeraHeap [7] or remote memory like Mako [8]. However, current research lacks a comprehensive comparison of these techniques. None of the existing work adequately addresses the pros and cons of each approach. This gap highlights a need for detailed evaluative studies that assess the performance, and reliability of each heap extension implementation method.

In this comparative analysis, we delve into possible strategies for achieving faster and more efficient block storage accessing while ensuring an appropriate size of the block storage. The first aspect of our investigation centers on comparing local storage devices, exemplified by NVMe SSDs, against remote storage devices and remote DRAM memory. NVMe SSDs, offer rapid data access and reliability. However, the efficiency of local storage devices in comparison to remote options remains a question mark and forms the core of our study. Remote DRAM memory, despite its remote positioning, may offer lower latency compared to remote storage devices due to its faster access times. However, its capacity is typically more constrained than storage devices. It's crucial to note the role of technological advancements such as SPDK NVMe over Fabrics (NVMe-oF) [11] in mitigating latency concerns. SPDK NVMe-oF holds promise in reducing device latency, offering a potential solution to bridge the performance gap between different storage modalities.

By employing micro-benchmarks we discern the inherent trade-offs. This evaluation is crucial for comprehensively understanding the landscape of storage solutions and their applicability in real-world scenarios. To assess the performance of block device setups, we leverage TeraHeap. TeraHeap [7] extends the capabilities of the JVM and utilizes a second heap stored on a storage device. This approach allows for a detailed evaluation of storage solutions, providing nuanced insights into their performance characteristics.

The selection among these options requires careful consideration of factors such as workload requirements and infrastructure configurations. The insights acquired by this study will play a pivotal role in informing decision-making processes related to resource allocation and system design, thereby contributing to the ongoing discourse on efficient data management in contemporary computing environments.

2 Background

In this section, we discuss the design of state-of-the art systems that provide access to remote memory and storage. Specifically, in our study we use Network Block Device

*Thesis advised by Angelos Bilas, Professor Department of Computer Science, University of Crete, Greece

(NBD) [9], NVMe over Fabrics (NVMe-oF) [5], and the Storage Performance Development Kit (SPDK) [11].

2.1 Network Block Device (NBD)

Network Block Device (NBD) is a network protocol that can be used to export a block device from a server where the block device resides to a client. Figure 1 shows an overview of an NBD system.

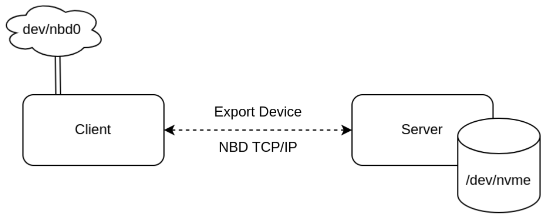

Figure 1. Overview of an NBD system.

The nbd-client usually resides in the OS kernel and exposes a block device interface to the rest of the kernel, so that it may appear as an ordinary, directly-attached storage device. The client passes the block requests to the NBD driver where they are encapsulated as NBD network messages and sent to the server via TCP. Finally the User-Space server upon receiving the NBD request issues standard I/O to the relevant block device and then responds Figure 2 shows a more in depth I/O path using NBD TCP Version. There are multiple NBD implementations, including:

- **nbdkit** is a multi-threaded NBD server with a plugin architecture.
- **libnbd** is a library to aid in writing NBD clients.
- **qemu** contains an **embedded NBD server**, **an embedded NBD client**, and **a standalone NBD server** (qemu-nbd). They maintain a status document of their NBD implementation.
- A **GEOM gate-based client implementation for FreeBSD** exists. It has not seen any updates since 2018, and only implements the client side (any server should run on FreeBSD unmodified, however).
- A **Windows** client implementation exists as part of the RBD implementation of Ceph for Windows.
- **lwNBD** is a NBD server library, targetting bare metal or OS embedded system. It has a plugin architecture.

We will focus on the Network Block Device (TCP version) [9]. We use the TCP version to investigate the overheads added by the TCP protocol stack and compare it with other mechanisms for exporting block devices over the network. With this implementation compiled in the kernel we can use 2 drivers/modules, the nbd-client and the nbd-server.

2.2 NVMe over Fabrics (NVMe-oF)

NVMe over Fabrics (NVMe-oF) [5] is a protocol specification designed for connecting hosts to storage systems over a

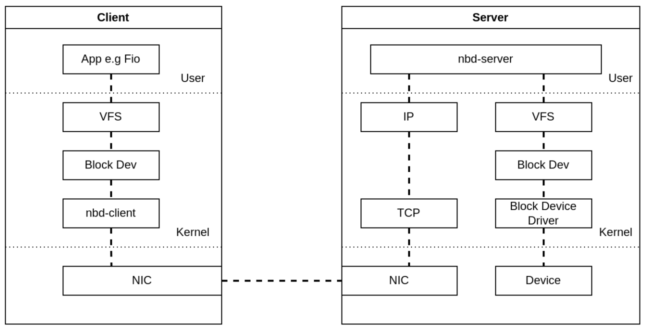

Figure 2. I/O Path of NBD system.

network using the NVMe protocol. It enables the transfer of data between a host computer and a target solid-state storage device or system through network communication. This protocol utilizes NVMe message-based commands to facilitate data transfers, supporting various networking technologies such as Ethernet, Fibre Channel (FC), and InfiniBand.

2.3 Storage Performance Development Kit (SPDK)

The Storage Performance Development Kit (SPDK) [11] is a versatile toolkit tailored for crafting high-performance, scalable storage applications in user-mode environments. Its architecture revolves around several core principles aimed at optimizing performance like User-Space Drivers, Polling Mechanism and lockless I/O handling. At its foundation, SPDK features a user-space, asynchronous NVMe driver designed for zero-copy, highly parallel access to SSDs. This driver, presented as a C library with a simple public header, facilitates seamless integration for developers. Additionally, SPDK offers a user-space block stack library mirroring OS functionalities, including storage device interface unification, queue management for resource constraints, and logical volume administration. SPDK extends its capabilities with NVMe-oF, iSCSI, and vhost servers built atop these foundational components.

SPDK operates entirely in user space, bypassing the kernel entirely for I/O operations. By eliminating the overhead of kernel context switches and system calls, SPDK can achieve significantly higher performance and lower latency compared to traditional storage solutions that rely on kernel-level operations. Memory management in SPDK is highly optimized, utilizing techniques such as large memory buffers, memory pooling, and zero-copy operations. Large memory buffers are allocated upfront, often using huge pages, to ensure contiguous memory allocation and reduce fragmentation. Memory pooling minimizes the overhead of dynamic memory allocation and deallocation, while zero-copy operations eliminate unnecessary data copies, further enhancing performance. Integration with the Data Plane Development Kit (DPDK) enhances SPDK's performance by leveraging

DPDK's [1] efficient packet processing capabilities for networked storage solutions. This integration enables SPDK to handle network I/O with minimal overhead, ensuring high throughput and low latency for storage applications deployed in networked environments. SPDK's support for NVMe-oF extends the NVMe protocol over fabrics, allowing remote access to NVMe storage devices with minimal overhead. This involves sophisticated handling of NVMe commands and data over high-speed networks, optimizing performance and scalability for distributed storage solutions.

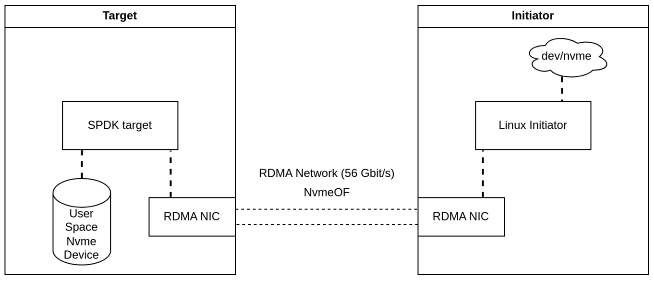

Figure 3. Overview of SPDK NvmeOF Target-Initiator system.

Figure 3 Shows an overview of the SPDK NVMe-oF Target-Initiator system. The process begins with the target unbinding traditional kernel drivers associated with the block device and binding them with SPDK's user-space drivers. This step grants SPDK exclusive control over the block device, enabling optimized I/O handling and performance. Subsequently, the block device, now under the control of SPDK in user space, is exported to the network using the NVMe-oF protocol specification, facilitated by RDMA. RDMA plays a pivotal role in enabling high-speed, low-latency data transfers over the network, allowing direct access to remote system memory without CPU involvement. This process ensures efficient data transmission between the NVMe target and initiator. Additionally, SPDK supports kernel initiators, allowing traditional kernel-based applications to connect to SPDK NVMe-oF targets. Through this mechanism, kernel initiators can access the SPDK-exported block device over the network using standard device nodes such as /dev/nvme.

3 Experimental Methodology

With our evaluation we try to answer:

1. What overheads are present with remote devices in NBD and NVMe-oF.
2. How does data granularity affect remote and local options (e.g. 512 B and 4 KB).
3. Can SPDK NVMe-oF match up against local drive in big data analytics.

For the evaluation, we use two servers with Intel Xeon E5-2630 32 Cores @ 2.4 GHz. Each server uses 256GB of DDR4 DRAM divided on two NUMA nodes, each with 16 threads. For storage we use Samsung 970 EVO Plus 2 TB PCIe Gen 3.0 x4 NVMe SSD. Each server is equipped with Mellanox Technologies ConnectX-3 Network controller MT27500 with fibre ports compliant with the InfiniBand Architecture Specification [4]. The servers run CentOS version 7.9 with Linux kernel version is 5.4.267. Table 1 shows the server specifications.

ID	CPU	DRAM	Device	NIC	Kernel
Server 1	Intel Xeon E5-2630 32 Cores @ 2.4 GHz.	256GB DDR4	Samsung 970 EVO Plus 2 TB PCIe NVMe SSD	Mellanox Technologies ConnectX-3 MT2750	5.4.267
Server 1	Intel Xeon E5-2630 32 Cores @ 2.4 GHz.	256GB DDR4	Samsung 970 EVO Plus 2 TB PCIe NVMe SSD	Mellanox Technologies ConnectX-3 MT2750	5.4.267

Table 1. Server specifications table.

Flexible I/O (FIO) Tester. FIO [6] is a widely used open-source tool in the Linux ecosystem for benchmarking and testing various I/O (input/output) workloads on storage devices. FIO provides detailed output reports, including metrics such as throughput, IOPS (input/output operations per second), latency, and CPU utilization. We configure FIO to use the libaio engine, random reads, direct I/O, 512 B bytes and 4 KB block size, 1 I/O depth and 1 thread to measure the latency of all the device setups shown in Table 2.

Configuration	Description
LOC_n	Local Non-Volatile Memory Express (NVMe) storage drive.
LOC_r	Local Ram-disk.
NBD_n	Network Block Device (NBD) configured to use an NVMe drive over the network.
NBD_r	Network Block Device (NBD) configured to use a Ram-disk over the network.
SPDK_n	Local NVMe drive managed with Storage Performance Development Kit (SPDK) user-space drivers.
SPDK_r	Local Ram-disk managed with SPDK user-space drivers.
OF_n	NVMe over Fabrics (NVMe-oF) using NVMe drives managed with SPDK user-space drivers.
OF_r	NVMe-oF using Ram-disk managed with SPDK user-space drivers.

Table 2. Storage device setups.

Netperf. Netperf [2] is a benchmarking tool used to measure the performance of networking systems. It is designed to provide a standardized method for measuring networking performance between two systems. Netperf allows users to test various aspects of networking performance, such as throughput, latency, and jitter, across different network protocols like TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). We will use Netperf to measure end-to-end latency of TCP (Transmission Control Protocol) To better understand the latency of systems like Network Block Device (NBD) which uses TCP to export the block device to the network.

TeraHeap. TeraHeap [7] is a system that eliminates S/D overhead and expensive GC scans for a large portion of the objects in big data frameworks. TeraHeap enhances the managed runtime environment, particularly the Java Virtual Machine (JVM). It introduces a supplementary heap, designed for high-capacity storage, alongside the primary heap. This secondary heap utilizes fast storage and allows direct access to objects without the need for serialization or deserialization. Additionally, TeraHeap minimizes the garbage collection overhead by preventing the garbage collector from scanning the secondary heap. It takes advantage of frameworks' capability to designate certain objects for off-heap allocation and provides them with a hint-based method for relocating these objects to the secondary heap. We will use TeraHeap as a Real world system. We configure TeraHeap to apply the supplementary heap to local, remote block devices, and remote Ram-disk and compare the performances of each block device system.

ib_read_lat. ib_read_lat is a micro-benchmark from the Perftest tool [3]. specifically designed for measuring InfiniBand (IB) latency. It evaluates the time taken for data to be transferred and received between InfiniBand endpoints. We will use it to measure the latency of transferring data between two Mellanox Technologies ConnectX-3 MT2750 network cards using the IB port.

Workloads and Execution time breakdown. We employ eight memory-intensive tasks from Spark-Bench [12] and five LDBC Graphalytics suites for Giraph [10], generating datasets accordingly. For the remote Ram-disk, we use the five LDBC Graphalytics suites for Giraph and generate a smaller dataset to fit in the limited 100 G Ram-disk exported with SPDK NVMe-oF. Each experiment is run five times, and the average end-to-end execution time is recorded. Time breakdown includes 'other' time, S/D plus I/O time, minor GC time, and major GC time. 'Other' time covers mutator threads, potentially including I/O wait. The profiler operates with minimal overhead. We configure TeraHeap to allocate the first heap (H1) on DRAM and the second heap (H2) over a file in Local or Remote block device or Remote Ram-disk via memory-mapped I/O (mmio) we also limmit DRAM and H1 to a fixed size to create pressure and more I/O to H2. Table 3 shows the DRAM and H1 size in each workload, in Spark and Giraph, accordingly.

Frameworks	Benchmarks	TOTAL DRAM (GB)	H1 (GB)
Spark	Pagerank	80	64
	Connected Components	84	68
	Linear Regression	54	27
	Logistic Regression	54	27
	Triangle Counts	80	64
	Shortest Path	58	42
	SVDPlusPlus	40	24
	SVM	48	32
Giraph	PageRank	85	50
	CDLP	85	60
	WCC	85	60
	BFS	65	35
	SSSP	90	50
Giraph Ram-disk	PageRank	16	12
	CDLP	16	12
	WCC	16	12
	BFS	14	10
	SSSP	24	20

Table 3. TeraHeap workload configuration table.

4 Evaluation

4.1 Storage systems latency and throughput

We use FIO to measure how much latency each different storage system adds. Figure 4 shows the latency with 512 B and 4 KB request size of each storage system.

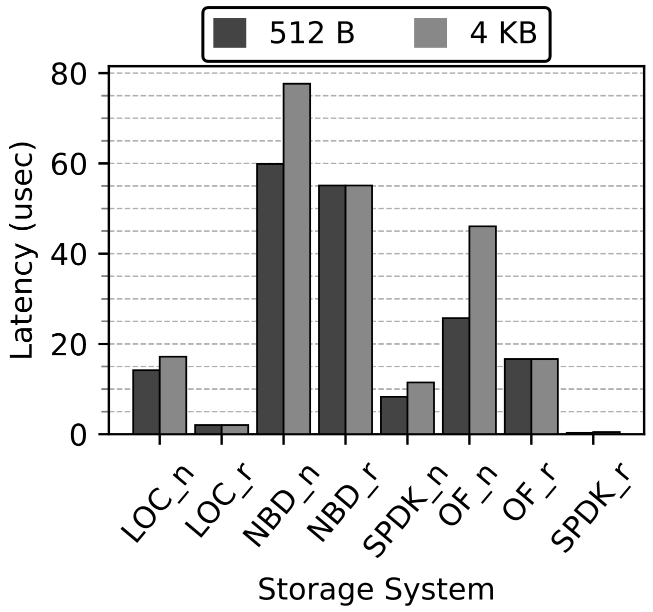

Figure 4. Average latency (usec) of local and remote storage systems with 512 B and 4 KB request size. "r" stands for Ram-disk and "n" stands for NVMe, "OF" stands for NVMe-oF with SPDK User-Space drivers.

We will analyze the results to determine the best setup for evaluating TeraHeap. Focusing on 512 B local SSD (device granularity) FIO reports an average of 14.16 μs latency and for local Ram-disk, FIO reports 2.04 μs 6.94× better. First, we will investigate the remote option NBD (TCP). The exported SSD with NBD average latency is 4.22× higher than local with an average of 59.86 μs. Meaning that we get 45.7 μs more latency with NBD. By looking at the exported Ram-disk with NBD average latency is 1.08× better than the NBD SSD, meaning that there is a common overhead. This overhead can either be the NBD I/O path or high TCP latency. We use Netperf [2] to measure the TCP latency and conduct experiments with 512 B of data in each packet. Netperf reports 37.84 μs average latency. Adding local NVMe and TCP latency we have 52 μs 7.86 μs difference to the NBD (TCP) latency 59.86 μs. We conclude that the primary overhead is TCP, accounting for 63.2% of the average latency in NBD NVMe.

Next, we investigate lowering the latency of the local I/O path to the device by using userspace drivers (SPDK). With userspace drivers, we bypass the layers of the kernel's I/O stack and eliminate context switches [11]. We run FIO on the NVMe with SPDK and get 8.31 μs average latency 1.7× better compared to traditional drivers. We also try to lower the network latency with NVMe-oF NVMe with SPDK User-Space drivers. With the combination of SPDK's user-space optimizations, the efficient protocol design of NVMe-oF compared to TCP and RDMA with Infiniband transport layer we manage to measure 25.69 μs average latency 2.33× better than SSD with NBD.

Moving on SPDK Ram-disk outperforms everything with 0.33 μs average latency, but the NVMe-oF Ram-disk with SPDK User-Space drivers have 16.66 μs average latency 9.03 μs better than the NVMe-oF NVMe and only 1.54× better compared to the local performance where the local Ram-disk is 25.18× better. There is a common overhead in both NVMe-oF setups which is attributed either to the NVMe I/O path or network latency. Previously we measured the NVMe with SPDK User-Space drivers and got 8.31 μs a logical 32.34% of the NVMe-oF NVMe with SPDK latency. We used ib_read_lat [3] to conduct RDMA Read Latency Test with 512 B requests and we measure that the Infiniband ports average latency is 2 μs only 7.78% of the NVMe-oF NVMe with SPDK latency meaning that the high latency can be by the Linux Kernel NVMe-oF Initiator which uses traditional kernel I/O calls to the exported device and then the calls are encapsulated and transported to the target via Infiniband. Unfortunately, SPDK's user-space NVMe-oF initiator [11] doesn't directly create block devices in /dev, SPDK uses its own block device layer (bdev) which will not be compatible for later experiments with TeraHeap.

Similarly, with 4 KB block size we get the same overheads. Here FIO reports more latency for moving the 4 KB data locally and/or across network. The NVMe-oF NVMe with SPDK achieving 25.69 μs and the NVMe-oF Ram-disk with SPDK reaching 16.66 μs average latency (nearly as fast as the local NVMe at 14.16 μs) are suitable to proceed with our evaluation using TeraHeap.

We also use FIO to measure the average throughput each storage system can achieve. Figure 5 shows the average throughput with 512 B and 4 KB request size of each storage system. SPDK_r and LOC_r are missing from this figure because they manage to achieve 28 979 MB/s and 17 510 MB/s average throughput and cover the lower throughput setups that we want to focus.

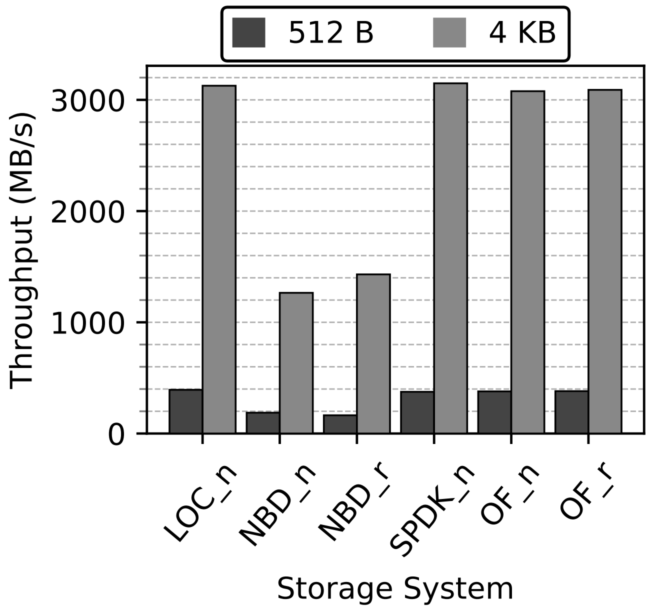

Figure 5. Average Throughput (MB/s) of local and remote storage systems with 512 B and 4 KB request size. "r" stands for Ram-disk and "n" stands for NVMe, "OF" stands for NVMe-oF with SPDK User-Space drivers.

Focusing on 4 KB local SSD FIO reports 3127 MB/s average throughput. Previously we determined that the two best set-ups for latency are NVMe-oF NVMe with SPDK and NVMe-oF Ram-disk with SPDK. Here these setups manage 3078 MB/s and 3090 MB/s average throughput accordingly. Local SSD performs with 1.01× better average throughput. Next NBD is under performing, both SSD and Ram-disk have 2.18× and 2.47× lower average throughput compared to local SSD accordingly. We confirm that The NVMe-oF NVMe with SPDK achieving 3078 MB/s and the NVMe-oF Ram-disk with SPDK reaching 3090 MB/s average throughput (nearly as fast as the local NVMe at 3127 MB/s) are suitable to proceed with our evaluation using TeraHeap.

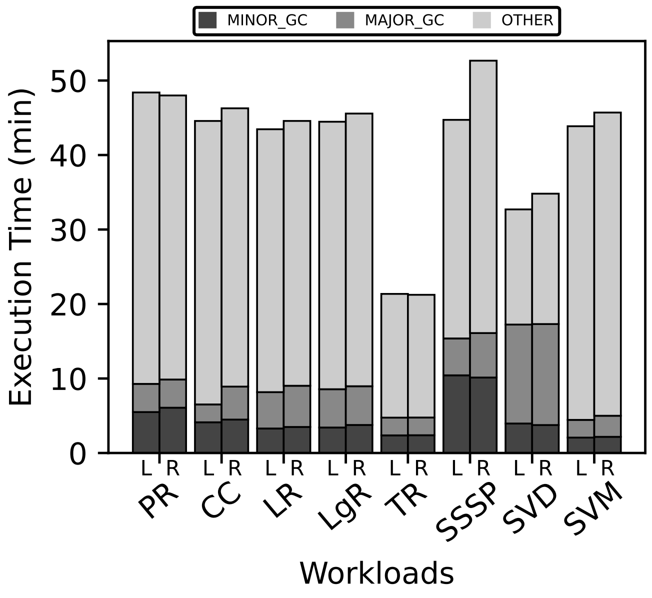

Figure 6. TeraHeap Spark performance. Local NVMe device (L) compared to NVMe-oF exported NVMe with SPDK (R).

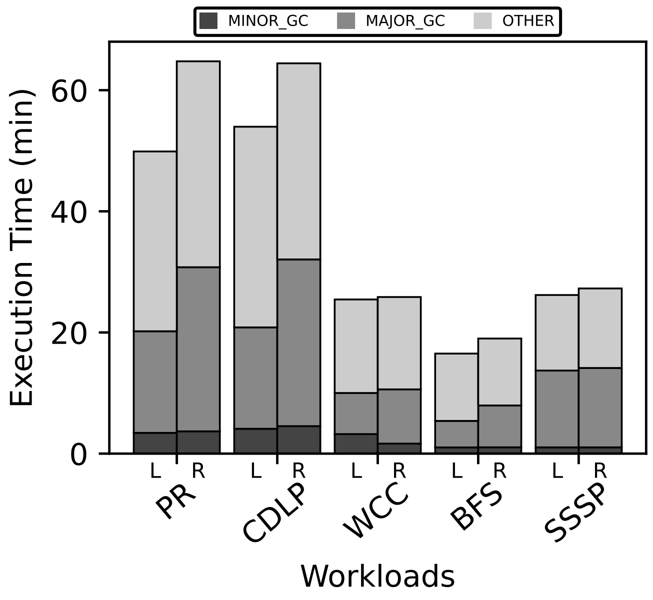

Figure 7. TeraHeap Giraph performance. Local NVMe device (L) compared to NVMe-oF exported NVMe with SPDK (R).

4.2 TeraHeap performance with NVMe-oF SSD

This section compares TeraHeap performance of two setups one with local NVMe device and one with NVMe-oF exported NVMe with SPDK User-Space drivers. Starting with Figure 6. We can see the performance of TeraHeap with Spark workloads for both setups. The two TeraHeap setups local NVMe device (L) and NVMe-oF exported NVMe with SPDK (R) perform similarly with Spark workloads. We can see that the local setup outperforms the remote setup in Connected Components (CC), Linear Regression (LR), Logistic Regression (LgR), Shortest Path (SSSP), SVDPlusPlus (SVD) and SVM making the local setup between 2.4% and 6% faster except Shortest Path (SSSP) where it is 15% faster. There are also cases where the remote setup is slightly better. Workloads Pagerank (PR), Triangle Counts (TR) report that the remote setup is 0.83% and 0.57% quicker accordingly. Spark workloads read objects from H2 but don't change them [7, 12] so heavy write operations are missing.

Next, we run TeraHeap with the Giraph workloads. These workloads read objects from H2 and also change them. Here we expect heavy write operations [7, 10] that can stress the remote setup. Figure 7 illustrates the performance of TeraHeap with Giraph workloads for both setups NVMe device (L) and NVMe-oF exported NVMe with SPDK (R). Here, due to the write operations, the remote setup (R) never manages to exceed the performance of the local setup (L). PageRank (PR), CDLP and BFS workloads are the ones that the remote setup struggles the most here local setup is 22.97% 16.24% and 13.07% quicker accordingly. In the setups mentioned before the remote setup is losing performance due to major GC's taking more time to complete. Following up with WCC and SSSP workloads we can see that the local setup is only 1.54% and 3.97% faster again here major GC's are the reason the remote setup is slower.

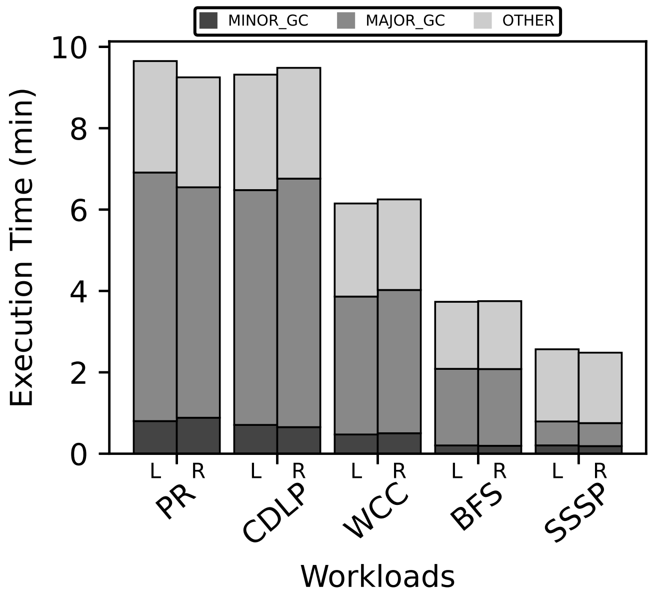

Figure 8. TeraHeap Giraph performance. Local NVMe device (L) compared to NVMe-oF exported Ram-disk with SPDK (R).

4.3 TeraHeap performance with NVMe-oF Ram-disk

This section compares TeraHeap performance of two setups one with local NVMe device and one with NVMe-oF exported Ram-disk with SPDK User-Space drivers. Figure 8 illustrates the performance of TeraHeap with Giraph workloads for both setups. Here the TeraHeap setups local NVMe device (L) and NVMe-oF exported Ram-disk with SPDK (R) perform almost the same with Giraph workloads. We can see that the local setup outperforms the remote setup in CDLP, WCC and BFS making the local setup between 0.44% and 1.75% faster. We can also see cases that the remote Ram-disk is better. Workloads PageRank (PR) and SSSP report that the remote setup is 4.32% and 3.35% quicker accordingly. The lower latency and high throughput of the remote Ram-disk manages to perform well in the giraph workloads compared to the slightly worse NVMe-oF exported NVMe of the previous setup.

4.4 Workload disk statistics

In this section we review the disk statistics of the workload runs. First we examine Spark. figures 9 and 10 show reads and writes accordingly. First, we can confirm that reads are more than writes due to the nature of spark benchmarks. Next focusing on the reads we can see that the Gigabytes read for both the local setup (L) and the remote setup (R) are close.

Next we examine Giraph with NVMe device. figures 11 and 12 show reads and writes accordingly. Focusing on the writes we can see that the data in Gigabytes for both the local setup (L) and the remote setup (R) are close. In the PageRank (PR) and CDLP workloads where the local setup (L) has the best performance compared to the remote setup (R) (figure 7 22.97% and 16.24% faster) we can see more reads (figure 11) for the remote setup due to more major GC's.

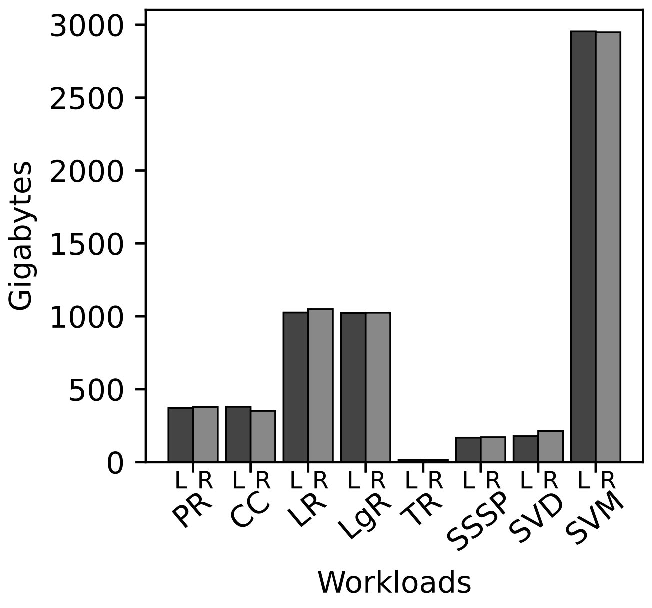

Figure 9. TeraHeap Spark workloads reads (GB). Local NVMe device (L) compared to NVMe-oF exported NVMe with SPDK (R).

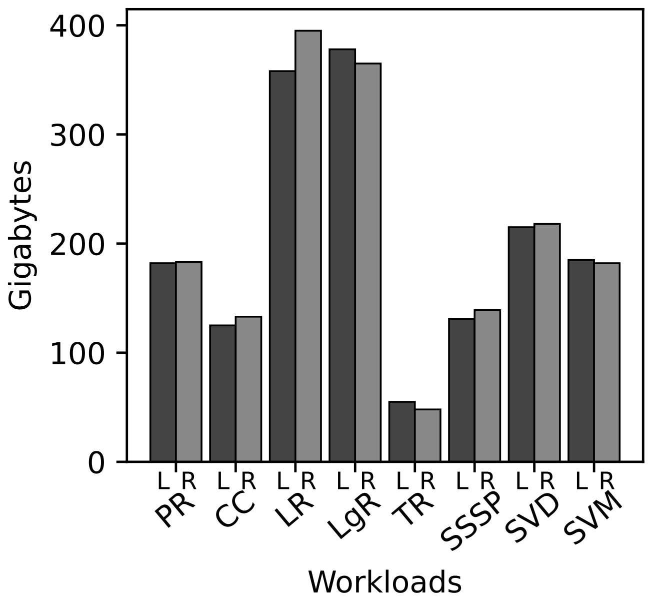

Figure 10. TeraHeap Spark workloads writes (GB). Local NVMe device (L) compared to NVMe-oF exported NVMe with SPDK (R).

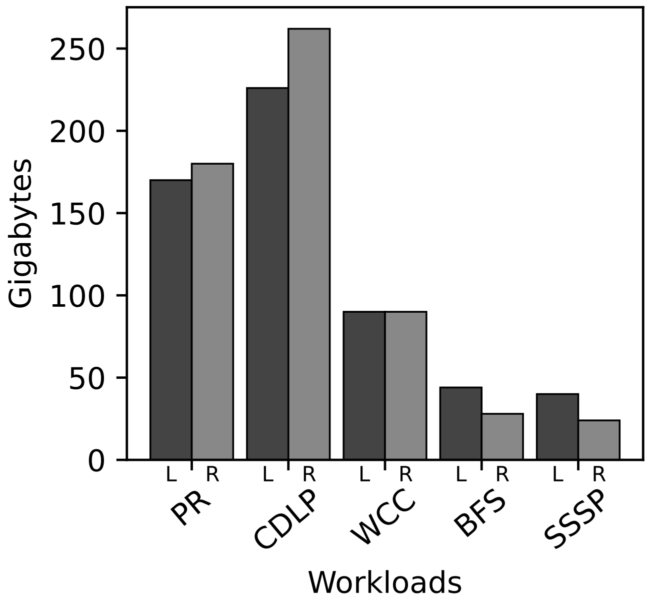

Figure 11. TeraHeap Giraph workloads reads (GB). Local NVMe device (L) compared to NVMe-oF exported NVMe with SPDK (R).

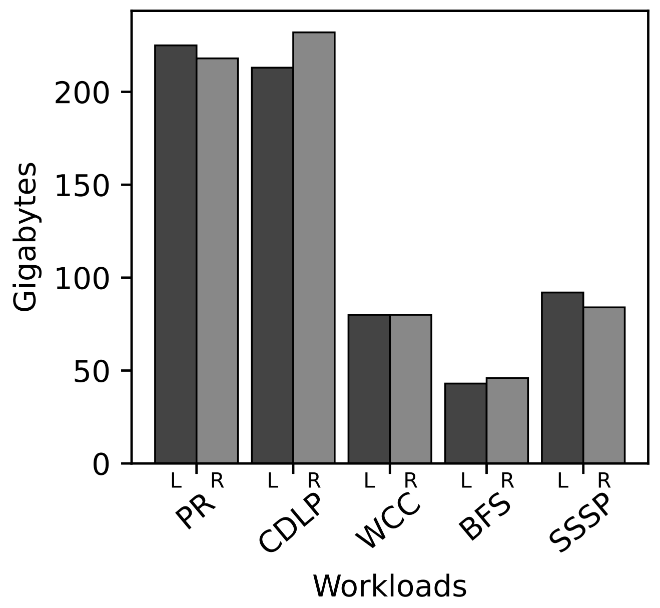

Figure 12. TeraHeap Giraph workloads writes (GB). Local NVMe device (L) compared to NVMe-oF exported NVMe with SPDK (R).

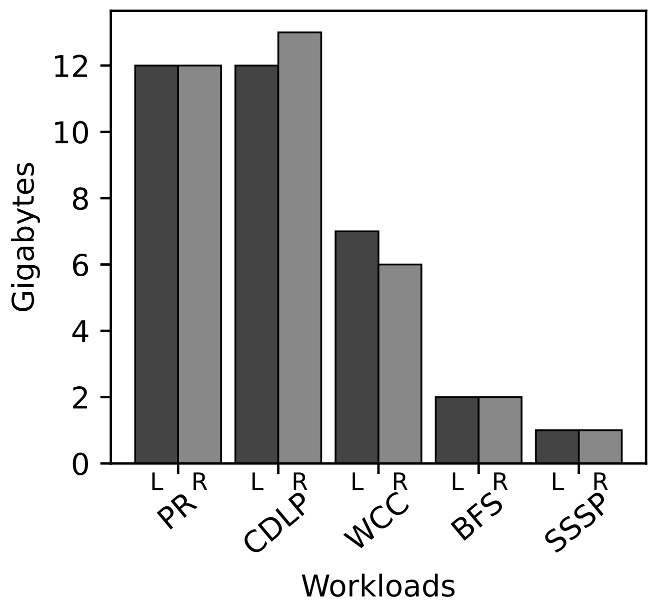

Figure 14. TeraHeap Giraph workloads writes (GB). Local NVMe device (L) compared to NVMe-oF exported Ram-disk with SPDK (R).

Moving on to the Giraph local NVMe device compared to NVMe-oF exported Ram-disk with SPDK. Figures 13 and 14 show reads and writes accordingly. We can see the reads and the writes in Gigabytes for the local setup (L) compared to the remote setup (R) are almost the same.

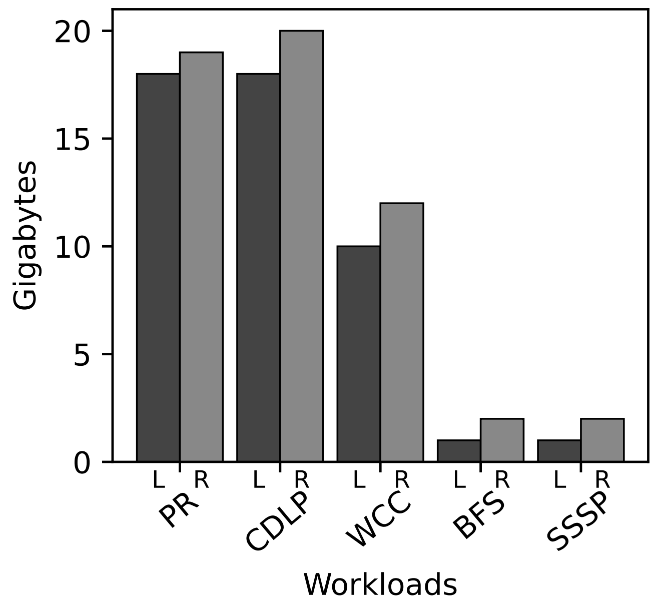

Figure 13. TeraHeap Giraph workloads Reads (GB). Local NVMe device (L) compared to NVMe-oF exported Ram-disk with SPDK (R).

5 Conclusions

Big data analytics frameworks on JVMs are widely used in data centers for large dataset analysis. Research has explored extending the managed heap with fast local storage (e.g., NVMe SSDs) and remote memory, but comprehensive performance and reliability comparisons are lacking. In our comparative analysis, we first employed micro-benchmarks to explore the latency and throughput performance of various storage systems, including local NVMe, remote NVMe, and remote memory. We then selectively compared local device systems with NVMe-oF SPDK NVMe and NVMe-oF SPDK Ram-disk using TeraHeap, which extends the managed heap over these devices. We evaluated these options with 13 widely-used applications in two real-world big data frameworks, Spark and Giraph. The workload reports indicate that NVMe-oF and remote options can match the performance of local ones, yielding very similar results. We found out that the NVMe-oF SPDK NVMe struggles to perform well with the Giraph workloads where there are a lot of writes to objects in the second heap (H2) but the spark workloads have very similar results compared to the local device setup. Finally, we concluded that extending the heap over NVMe-oF SPDK Ram-disk can perform almost the same compared to the local setup in the Giraph workloads. This study provides valuable insights into the performance characteristics of different storage solutions, informing decisions on resource allocation and system design, and contributing to the ongoing discourse on efficient data management in contemporary computing environments.

References

[1] Data plane development kit. https://github.com/DPDK/dpdk.

[2] Netperf. https://github.com/HewlettPackard/netperf.

[3] perftest. https://github.com/linux-rdma/perftest.

[4] Infiniband architecture specification. http://www.infinibandta.org, 2000.

[5] Nvm express over fabrics. https://nvmexpress.org/wp-content/uploads/NVMe-over-Fabrics-1.1a-2021.07.12-Ratified.pdf, 2021.

[6] Jens Axboe. Flexible i/o tester. https://github.com/axboe/fio, 2022.

[7] Iacovos G. Kolokasis, Giannos Evdorou, Shoaib Akram, Christos Kozanitis, Anastasios Papagiannis, Foivos S. Zakkak, Polyvios Pratikakis, and Angelos Bilas. Teraheap: Reducing memory pressure in managed big data frameworks. ASPLOS 2023, page 694–709, New York, NY, USA, 2023. Association for Computing Machinery.

[8] Haoran Ma, Shi Liu, Chenxi Wang, Yifan Qiao, Michael D. Bond, Stephen M. Blackburn, Miryung Kim, and Guoqing Harry Xu. Mako: a low-pause, high-throughput evacuating collector for memory-disaggregated datacenters. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2022, page 92–107, New York, NY, USA, 2022. Association for Computing Machinery.

[9] P.T. A. Marin Lopez, Arturo Garcia Ares. The network block device. *Linux J.*, 2000(73es):40–es, may 2000.

[10] Sherif Sakr, Faisal Moeen Orakzai, Ibrahim Abdelaziz, and Zuhair Khayyat. *Large-Scale Graph Processing Using Apache Giraph*. Springer Publishing Company, Incorporated, 1st edition, 2017.

[11] Ziye Yang, James R. Harris, Benjamin Walker, Daniel Verkamp, Changpeng Liu, Cunyin Chang, Gang Cao, Jonathan Stern, Vishal Verma, and Luse E. Paul. Spdk: A development kit to build high performance storage applications. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 154–161, 2017.

[12] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, page 10, USA, 2010. USENIX Association.