Performance Impact of NVMe-Over-TCP on HDFS Workloads

Nikita Sharma
Department of Computer Science
University of Texas at Austin
Austin, USA
nikitasharma@utexas.edu

Ruihao Li, Qinzhe Wu, Lizy K. John
Department of Electrical and Computer Engineering
University of Texas at Austin
Austin, USA
{liruihao,qw2699}@utexas.edu,ljohn@ece.utexas.edu
0000-0002-7092-2401,0000-0002-7988-1431,0000-0002-8747-5214

***Abstract*—Storage is one of the important components in datacenters. As the data volume rises and the service scale grows, some workloads like database demand increasing amount of storage. While a single server can only host a limited number of disks, distributed file systems (e.g., Hadoop Distributed File System referred to as HDFS) enable accessing disks mounted on the other servers in the cluster, satisfying the storage requirements. On the other side, NVMe-over-Fabric protocols (e.g., NVMe-over-TCP) have been released as a solution on the device level to provide access to remote NVMe disks. Therefore, for those applications developed on top of HDFS, there are at least two choices to make use of the storage resources distributed in datacenters. A concern is whether NVMe-over-TCP will hurt the performance. The evaluation in this paper reveals that the performance degradation NVMe-over-TCP caused on HDFS-based workloads is limited, suggesting NVMe-over-TCP a performant and economical solution in datacenter design to support workloads that needs a lot of storage (such as database applications).**

***Index Terms*—storage, datacenter, NVMe-over-TCP, Hadoop Distributed File System.**

I. Introduction

Datacenters are rapidly growing and expanding, and the demand placed on these datacenters is growing at unprecedented rates. One of the essential components of a datacenter is storage. There needs to be sufficient storage that can support the massive scale at which data is growing and applications are running. However, the amount of disks that a single machine could host is limited; so servers in the cluster are usually configured to share the disks between each other as remote storage. In addition to the capacity requirement, the latency of accessing the storage (local or remote) should ideally not be the cause of a bottleneck in applications and the backing storage should allow for high throughput. There have been a few efforts on increasing the capacity and accelerating the access of the storage(e.g., caching [1], partitioning [2], scheduling [3], virtualization [4], failure prediction [5] and so on).

One of the emerging trends is the adoption of Non-Volatile-Memory Express (NVMe) disks for storage disaggregation. NVMe has become one of the industry standards for Solid-State Drives (SSDs), and has shown excellent performance on real-world workloads such as database applications [6]. By 2020, more than 90% of the SSD disks used in hyperscale datacenters are NVMe disks [7]. The NVMe and the derived NVMe-over-Fabric (NVMeoF) specifications [8] provide a low latency and scalable interface for software to communicate with non-volatile memory across multiple transports (e.g., PCI-Express, RDMA, TCP). In NVMe architecture [9], there are paired submission queues for NVMe commands, and completion queues for NVMe device acknowledgements in between the user applications and the NVMe controller. Similarly, the NVMe device management is accomplished by the submission-completion queue structure as well. In the same NVMe subsystem, there could exist more than one NVMe controller, as well as more than one NVMe namespace, and a namespace could span multiple NVMe controllers. Each NVMe host has the access to one or more NVMe controllers. NVMeoF connects a host server to storage that uses the NVMe protocol over a network fabric (e.g., Ethernet, Infiniband etc.), so that servers in a cluster could host and share a great amount of storage. Prior to the release of NVMeoF, solutions have been practiced on the file system level (e.g., Hadoop Distributed File System, HDFS [10]), however, a past study [11] shows that with NVMe-over-RDMA, the throughput of HDFS could be improved a lot ($2.55\times$). This makes NVMeoF beneficial especially for storage disaggregation. Specifically, this paper focuses on NVMe-over-TCP, a protocol that builds upon NVMeoF using an underlying TCP fabric. Compared to other NVMeoF protocols (e.g., NVMe-over-RDMA, NVMe-over-InfiniBand), NVMe-over-TCP has the best adaptability, that where there are commercial network connections and TCP works, we can deploy NVMe-over-TCP.

There have been studies on other NVMeoF schemes (§ II), while to our knowledge, NVMe-over-TCP has not been well studied, despite of the low barrier to set up and deploy NVMe-over-TCP (as described above, unlike other NVMeoF schemes, NVMe-over-TCP needs no additional network facilities). Therefore, the following questions are worth exploring, which this paper tries to answer:

1) Whether with the minimum hardware requirement (i.e., no addition to the legacy Ethernet network fabric), NVMe-over-TCP can make accessing remote NVMe disks as fast as accessing local NVMe disks?

2) Whether the disks mounted via NVMe-over-TCP are able to serve as many concurrent read/write operations (in another word, operation throughput) as the local NVMe disks?
3) If there is performance degradation caused by NVMe-over-TCP, whether it would be still in a tolerable range for datacenter workloads? Or NVMe-over-TCP could avoid degrading the performance but at what cost?

In this paper we run several microbenchmarks as well as a database application on a NVMe-over-TCP setup in order to evaluate its impact on performance (latency as well as throughput). More specifically, we look at the performance of NVMe-over-TCP with the popular distributed file system framework, HDFS. If performance overheads with NVMe-over-TCP is low, it would allow for massive increases in the storage capacity and flexibility of HDFS.

The contributions of the paper are as follows:

1) We evaluate multiple options of using NVMe disks including deploying with NVMe-over-TCP. Specifically, the experiments compare 1) local NVMe disk; 2) remote NVMe-over-TCP disk; 3) remote NVMe disk with HDFS;
2) We perform workload characterization on both microbenchmarks and a database application. The evaluation result suggests NVMe-over-TCP has negligible performance overhead so datacenter operators could deploy more NVMe disks via NVMe-over-TCP protocol, achieving the benefits of large storage capacity without degrading performance.

II. Related Work

Accessing remote storage does come with high overheads. However, it also allows for the size of storage to scale rapidly especially in response to the growing demand. Therefore, researchers have been evaluating the benefits and drawbacks of various NVMeoF systems. With the advanced network facility support (e.g., InfiniBand, converged Ethernet), it has been shown that NVMe-over-RDMA has negligible performance degradation in the storage disaggregation setting, and is much better than iSCSI [12], [13]. Under ARM architecture, NVMe-over-RDMA is even able to achieve better performance than direct attached storage if both the hardware and software are optimized [14]. Nevertheless, most of the prior studies are limited to NVMeoF with specialized hardware (e.g., Infiniband, RDMA), where it is not surprising for those NVMeoF protocols to obtain comparable performance as local NVMe disks due to the advantages of the fabric support. Nevertheless, the latest released NVMe-over-TCP protocol has not been systematically studied yet.

III. Methodology

This section covers how we setup the system and the experiments we conduct to evaluate the system performance with the microbenchmarks as well as the database application.

A. System Configurations

The specifications of the experimental platform are listed in Table I. The servers we used in the experiments are equipped with high-end 32-core ARM processors, and 128GB DDR4 DRAMs. There is one 800GB NVMe disk attached to each server via Generation 3 PCI-Express ports. Servers are connected in a local network (only one switch in between) and the network cards have sufficient bandwidth. The kernel modules that provide NVMe-over-TCP stack, as well as kernel, software versions are all listed in Figure 1.

TABLE I: System Specifications

Processors	32× ARMv8 64-bit CPU cores
Memory	128GB DDR4-2666
Disk	PCIe Gen3 ×4 800GB SSD
NIC	PCIe 2.1 2.5GT/s GbE NIC
kernel modules	nvmet, nvmet_tcp
kernel version	`Linux 5.4.0-80-generic`
OS version	`Ubuntu 20.04.3 LTS`
Hadoop version	`hadoop-3.3.1`
RocksDB version	`6.23.2`

With the aforementioned hardware and software, we configure the system in three ways as shown in Figure 2 to make comparison. For the first setting, local, we let a single server in charge of the entire system: the NVMe disk is attached to the server locally, and the Hadoop Distributed File System (HDFS [10]) is configured to only use the local disk to store data. Then we introduce a remote disk using the NVMe-over-TCP protocol, and let the HDFS merely use the remote disk. This turns to be the second setting, namely, remote NVMe-over-TCP. Eventually, we move the datanode [1] of HDFS to a second server with a NVMe disk attached, so the tasks undertaken by the HDFS datanode can access the data locally on the second server, while providing the results to the master server (where the namenode [2] and the database application run). We name this setting as remote NVMe w/ HDFS. All the three system settings enumerated above configure HDFS with one datanode and the replication factor is one. A modern database application, RocksDB [16], runs on top of the HDFS setup.

B. Experiments With Micro-benchmarks

HDFS is designed to accommodate large quantities of data and be tolerant to hardware failures. In order to stress the file system, a HDFS distribution comes with built-in microbenchmarks. These benchmarks utilize the MapReduce framework [17]. Table II lists the three MapReduce microbenchmarks used in our experiments. The *wordcount* is the very typical example for MapReduce processing: the content

[1] A DataNode stores data in the HadoopFileSystem [15].

[2] The NameNode is the centerpiece of an HDFS file system. It keeps the directory tree of all files in the file system, and tracks where across the cluster the file data is kept. It does not store the data of these files itself [15].

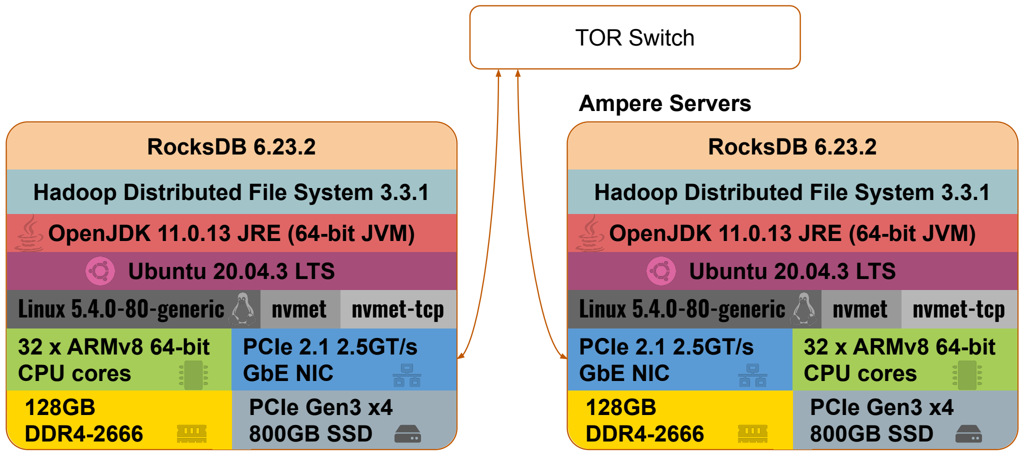


Fig. 1: System specifications

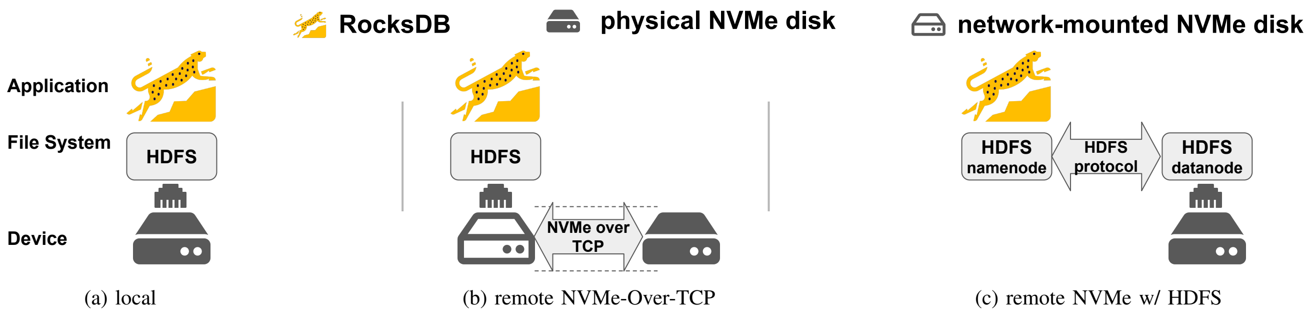


Fig. 2: The three configurations that are experimented in this paper

TABLE II: The built-in MapReduce microbenchmarks from HDFS installation that we use in the experiments.

microbenchmark	description
randomwriter	write randomly generated bytes into a list of files
wordcount	break file content into tokens and aggregates the occurrence of each token
sort	read values from a file then write them into a file in sorted order

of input files are cut by the word boundary and the workers create pairs of value 1 and token (mapping occurrences by word), then the reducers aggregate values according to the tokens. The *randomwriter* benchmark generates content byte by byte and dumps to a list of files. The *sort* benchmark exploits the shuffling and sort phase in the reducing process of MapReduce tasks to sort the values from the input file.

We measure the total execution time (i.e., latency) of these microbenchmarks in the three system settings (§ III-A, Figure 2). It is important to check whether the remote NVMe disks provide the bandwidth close to the local disk under varying levels of access intensity. Therefore, we increase the number of *wordcount* microbenchmark instances launched at the same time (similar to the approach that SPEC CPU2017 rate suite takes [18]) from 1 to 40 (the server memory capacity limits at most 40 tasks), and measure the overall duration (from the beginning to the last task finishes). All instances read the same input file but write to different output files.

Each measurement is repeated 40 times in order to reduce the randomness in the results. We discard 2 minimal values and 2 maximum values and use the average of the left 36 values.

In order to gain a deeper understanding, we further profile the execution of the *wordcount* micro-benchmark, and compare the function call stacks of two configurations: local NVMe and remote NVMe-over-TCP. We use the async-profiler [19] designed for Java workload to sample the stack every 10 milliseconds during the execution. The profiling covers all Java processes on the server (the third configuration remote NVMe w/ HDFS is skipped due to the difficulty of synchronizing profiler across machines), when the server is only loaded by the *wordcount* benchmark. With over twenty thousand stack records and more than sixty four thousand samples, we attribute them into five major categories and a few more sub-categories.

C. Experiments With Database Application

RocksDB [20] is a persistent key-value store developed and contributed to the open-source community by Facebook. Considering the advances in the field of storage, RocksDB is explicitly optimized for fast storage like flash-based disks, fully exploiting the potential of high read/write rates provided by flash in order to maximize the performance. There is also build-in support for HDFS in RocksDB. We run two sets of RocksDB benchmarks in the same three configurations outlined in Section III-A. This will show how the performance of NVMe-over-TCP extends to a full application as compared to within microbenchmarks.

In addition, RocksDB provides its adaption of the *db_bench* benchmarks from LevelDB [21], so we test the db_bench performance on our RocksDB setup as well.

IV. Evaluation

We present and analysis the experimental results in this section.

A. Micro-benchmarks

1) Latency and Bandwidth Tests:

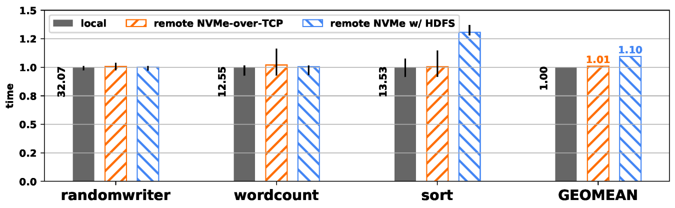


Fig. 3: Overhead of accessing remote storage with HDFS MapReduce microbenchmarks. The execution time of the two remote disk configurations are normalized to time of the local configuration for each benchmark. The absolute time of the local configuration are reported on the left of the grey solid bar in seconds. The geometric mean of the slowdowns over local configuration is shown on the right (GEOMEAN). Remote NVMe w/ HDFS has 10% overhead whereas remote NVMe-Over-TCP only has 1% overhead.

Figure 3 presnets the 9 sets of performance results we measured on the micro-benchamrks. We report the variation observed in the 40 experiments as the error bars in Figure 3.

For each microbenchmark, we use the local setting time as the baseline (the absolute values in seconds are reported beside the grey solid bar), and normalize the time in the remote settings to the baseline. We notice using a remote NVMe-over-TCP disk has almost the same performance as using a local disk across the microbenchmarks, and on average (geometrical mean) there is only 1% slowdown with NVMe-over-TCP. The second remote setting, remote NVMe w/ HDFS, also achieves comparable latency on randomwriter and worcount, but has notable time increase (more than 30%) on the sort microbenchmark. Overall, remote NVMe w/ HDFS is about 10% slower than local NVMe.

For the bandwidth test experiments, Figure 4 reflects the trend, where we can learn that the two remote NVMe disk configurations are affected similarly as the local disk configuration when the contention is increased: increasing the number of

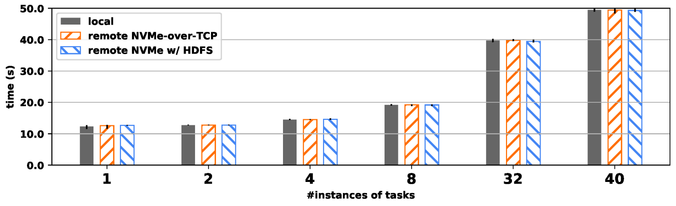


Fig. 4: Effect of increasing workload on the wordcount microbenchmark.

wordcount task instances only slightly rises the execution time initially, and by 32 (running out of CPU cores) the execution time has been tripled.

The takeaway from Figure 3 and Figure 4 is that there exists no adverse effects evident from using remote NVMe disks.

2) Function Stack Profiling:

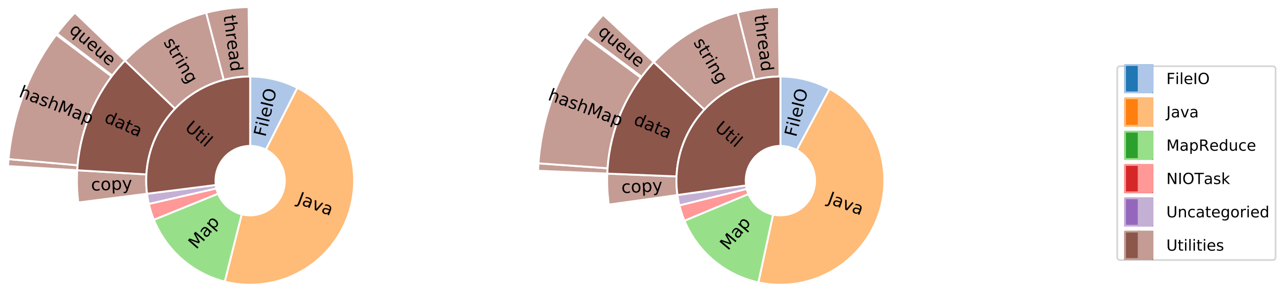
These categorization results on function-call stack profiling data are visualized in Figure 5. The two charts looks almost identical as expected, and we notice that network I/O (labeled as NIOTasks in Figure 5) only contributes around 2.5% of the overall samples for both configurations. The most of time are spent on Java-related libraries and data structure operations. These stack sample breakdown results could explain the nearly identical performance results shown in Figure 4. Since the NIOTasks are not the bottleneck for the entire system, the overall performance would not be affected too much on different configurations. The detailed categorization result is reported in Table III.

TABLE III: Function call stack categorization of NVMe-over-TCP and each sample represents 10ms interval.

Call stack category	local		remote NVMe-over-TCP	
	#samples	percentage	#samples	percentage
Java	30387	46.44%	29496	45.55%
MapReduce	9772	14.93%	9913	15.31%
FileIO	4906	7.50%	5081	7.85%
NIOTask	1662	2.54%	1602	2.47%
Utilities/string	5737	8.77%	5876	9.07%
Utilities/thread	2587	3.95%	2482	3.83%
Utilities/copy	1939	2.96%	1847	2.85%
Utilities/time	85	0.13%	87	0.13%
Utilities/datastruct/hashMap	5788	8.85%	5810	8.97%
Utilities/datastruct/hashSet	36	0.06%	47	0.07%
Utilities/datastruct/hashTable	22	0.03%	19	0.03%
Utilities/datastruct/queue	1180	1.80%	1132	1.75%
Utilities/datastruct/array	301	0.46%	317	0.49%
Uncategoried	1035	1.58%	1043	1.61%
Sum	65437	100%	64752	100%

B. Database Application

In Figure 6, we show the throughput of various RocksDB operations achieved in the three different settings. The results are measured by the benchmark script that comes with RocksDB and the execution as well as parameters follow the instructions on the RocksDB github wiki page [22]. We can see remote NVMe-over-TCP is able to obtain about the same throughput on all tested RocksDB operations, except



(a) local (b) remote NVMe-over-TCP

Fig. 5: Function call stack categorization from the wordcount microbenchmark.

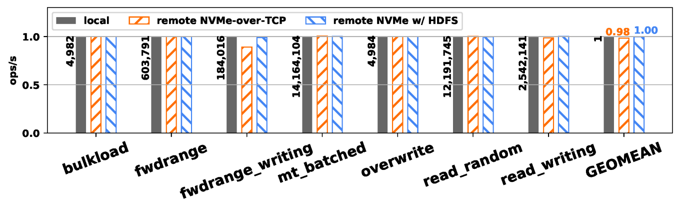


Fig. 6: RocksDB benchmark throughput. For each benchmark, the throughput of the two remote settings are normalized to the throughput of the local setting, which is labeled on the left of the grey solid bar.

fwdrange_writing, which randomly iterates over a range of keys with some background writings happening. Overall, remote NVMe-over-TCP gets 98% throughput of local on average (geometrical mean). The throughput of local NVMe and remote NVMe w/ HDFS remains mostly indistinguishable on fwdrange_writing, and other benchmarks. There several other metrics (such as percentile latency, bandwidth) measured by the benchmarks, and we summarize them in Table IV.

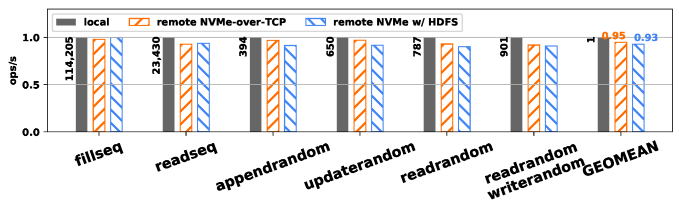


Fig. 7: The db_bench [21] benchmarks throughput (on RocksDB). For each benchmark, the throughput of the two remote settings are normalized to the throughput of the local setting, which is labeled on the left of the grey solid bar.

Figure 7 compares the throughput achieved under the three NVMe settings. Remote NVMe-over-TCP has small drops (within 9%) on the operation throughput of three benchmarks (i.e., *readseq*, *readrandom*, and *readrandomwriterandom*), but has very close performance on the other ones. Remote NVMe w/ HDFS has very limited (less than 10%) performance degradation on most benchmarks except that *fillseq* is almost the same as the performance of local NVMe.

The evaluation on the database application, RocksDB, further confirms our finding that the overhead with NVMe-over-TCP is not likely to be the performance bottleneck of datacenter workloads. Data center workloads can safely use NVMe-over-TCP and maintain performance while having access to large amounts of storage.

V. Conclusion

The NVMe-over-TCP protocol allows for a massive increase in the amount of storage accessible in datacenters. Further, it allows for the decoupling of compute and storage nodes (i.e., disaggregation), which is the current trend in datacenter resource management because of the much better hardware utilization of all available resources. Largely, the benefits that come from using the NVMe-over-TCP protocol outweigh the overheads. Our experiments with MapReduce kernel benchmarks as well as the RocksDB application illustrate that the performance overheads of accessing remote storage using the NVMe-over-TCP protocol is largely negligible. Performance degradation is minimal ranging from $1-2\%$ on microbenchmarks and $2-9\%$ on the RocksDB full database application. To accommodate the growing demand for storage, we show the NVMe-over-TCP protocol to be a viable option that minimizes the overhead of accessing storage over the network and provides a solution that has a low barrier to deployment.

Acknowledgements

This research was supported in part by NSF grant numbers 1745813, and 1763848, and funding from Meta. We also acknowledge the computing servers donated from Ampere Computing. Any opinions, findings, conclusions or recommendations are those of the authors and not of the National Science Foundation or other sponsors.

References

[1] A. A. Shvidkiy, A. A. Savelieva, and A. A. Zarubin, “Caching methods analysis for improving distributed storage systems performance,” in *2021 Systems of Signal Synchronization, Generating and Processing in Telecommunications (SYNCHROINFO*, 2021, pp. 1–5.

[2] G. Weikum, “Data partitioning and load balancing in parallel storage systems,” in *Proceedings Thirteenth IEEE Symposium on Mass Storage Systems. Toward Distributed Storage and Data Management Systems*, 1994, pp. 99–.

TABLE IV: Table Illustrating Performance metrics of RocksDB benchmarks on different NVMe system configurations.

configuration	**time**	**ops/s**	**MB/s**	**μs/op**	**latency percentile[*] (μs)**					**Uptime**
	(s)				**p50**	**p75**	**p99**	**p99.9**	**p99.99**	**(s)**
bulkload										
local NVMe disk	5706	4982	2.0	200.7	10.3	13.0	33	83	239	5344
remote NVMe over TCP	5711	4982	2.0	200.7	10.3	13.0	33	84	239	5344
remote NVMe w/ HDFS	5709	4981	2.0	200.7	10.1	12.9	33	80	238	5345
fwdrange										
local NVMe disk	5401	603791	2418.4	106.0	53.0	70.2	314	13812	23790	5344
remote NVMe over TCP	5402	603910	2418.9	106.0	53.1	70.3	321	13740	23437	5345
remote NVMe w/ HDFS	5401	603982	2419.2	106.0	53.1	70.3	311	13893	24224	5344
fwdrangewhilewriting										
local NVMe disk	5404	184016	737.1	347.8	143.5	196.7	7024	18755	28348	5369
remote NVMe over TCP	5406	164075	657.2	390.0	144.5	201.5	7544	18867	28933	5403
remote NVMe w/ HDFS	5405	182218	729.9	351.2	143.9	198.3	7022	18586	27967	5373
multiread_batched										
local NVMe disk	5402	14164104	167.1	4.5	12.9	23.7	104	10781	26173	5341
remote NVMe over TCP	5402	14220827	167.8	4.5	13.0	23.7	102	10696	25976	5341
remote NVMe w/ HDFS	5402	14197918	167.5	4.5	13.0	23.8	103	10747	26165	5341
overwrite										
local NVMe disk	5406	4984	2.0	12836.4	1406.3	1841.7	4261	6190	7878	5394
remote NVMe over TCP	5406	4984	2.0	12836.1	1403.9	1833.6	4240	6128	6594	5393
remote NVMe w/ HDFS	5406	4984	2.0	12836.2	1466.2	2063.8	4356	6355	10464	5396
readrandom										
local NVMe disk	5401	12191745	143.8	5.2	1.2	1.7	42	55	12497	5341
remote NVMe over TCP	5401	12212943	144.1	5.2	1.2	1.7	42	51	12471	5341
remote NVMe w/ HDFS	5401	12140446	143.2	5.3	1.2	1.7	42	53	12543	5341
readwhilewriting										
local NVMe disk	5404	2542141	64.5	25.2	8.3	9.9	73	827	23592	5365
remote NVMe over TCP	5403	2505848	63.6	25.5	8.4	10.0	74	841	23655	5366
remote NVMe w/ HDFS	5403	2544075	64.6	25.2	8.3	9.9	74	816	23613	5367

[*] latency percentile means that percentage of all the requests achieve equal or lower response latency, for example p50 means median.

[3] Y. Tanimura and H. Koie, "Operation-level performance control in the object store for distributed storage systems," in *2015 IEEE International Conference on Data Science and Data Intensive Systems*, 2015, pp. 111–112.

[4] X. Zhou and L. He, "A virtualized hybrid distributed file system," in *2013 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, 2013, pp. 202–205.

[5] J. Zhang, K. Zhou, P. Huang, X. He, M. Xie, B. Cheng, Y. Ji, and Y. Wang, "Minority disk failure prediction based on transfer learning in large data centers of heterogeneous disk systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 9, pp. 2155–2169, 2020.

[6] Q. Xu, H. Siyamwala, M. Ghosh, T. Suri, M. Awasthi, Z. Guz, A. Shayesteh, and V. Balakrishnan, "Performance analysis of nvme ssds and their implication on real world databases," in *Proceedings of the 8th ACM International Systems and Storage Conference*, ser. SYSTOR '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2757667.2757684

[7] C. Brett, J. Hands, and D. Jeanette. (2020) SAS vs. NVMe Interface Smackdown. [Online]. Available: https://www.scsita.org/wp-content/uploads/2020/08/SAS_vs_NVMe_Interface_Smackdown-final2.pdf

[8] N. Express. (2022) What is NVMe. [Online]. Available: https://nvmexpress.org/

[9] S. Grimberg and P. Onufryk. (2019) NVMe™/TCP: What You Need to Know About the Specification. [Online]. Available: https://nvmexpress.org/wp-content/uploads/March-2019-NVMe-TCP-What-You-Need-to-Know-About-the-Specification.pdf

[10] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010, pp. 1–10.

[11] D. Han and B. Nam, "Improving access to hdfs using nvmeof," in *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, 2019, pp. 1–2.

[12] 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*, ser. SYSTOR '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3078468.3078483

[13] Z. Guz, H. Li, A. Shayesteh, and V. Balakrishnan, "Performance characterization of nvme-over-fabrics storage disaggregation," *ACM Trans. Storage*, vol. 14, no. 4, dec 2018. [Online]. Available: https://doi.org/10.1145/3239563

[14] Y. Jia, E. Anger, and F. Chen, "When nvme over fabrics meets arm: Performance and implications," in *2019 35th Symposium on Mass Storage Systems and Technologies (MSST)*, 2019, pp. 134–140.

[15] apache/hadoop. (2022) Hadoop Wiki. [Online]. Available: https://cwiki.apache.org/confluence/display/ HADOOP2

[16] S. Dong, A. Kryczka, Y. Jin, and M. Stumm, "Rocksdb: Evolution of development priorities in a key-value store serving large-scale applications," *ACM Trans. Storage*, vol. 17, no. 4, oct 2021. [Online]. Available: https://doi.org/10.1145/3483840

[17] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, p. 107–113, Jan 2008. [Online]. Available: https://doi.org/10.1145/1327452.1327492

[18] SPEC. (2017) What are "SPECspeed" and "SPECrate" metrics? [Online]. Available: https://www.spec.org/cpu2017/Docs/overview.html#Q15

[19] jvm-profiling tools. (2022) jvm-profiling-tools/asynch-profiler. [Online]. Available: https://github.com/jvm-profiling-tools/async-profiler

[20] S. Dong, M. D. Callaghan, L. Galanis, D. Borthakur, T. Savor, and M. Strum, "Optimizing space amplification in rocksdb," in *CIDR*, 2017.

[21] google. (2022) google/leveldb. [Online]. Available: https://github.com/google/leveldb

[22] facebook/rocksdb. (2022) Performance Benchmarks. [Online]. Available: https://github.com/facebook/rocksdb/wiki/performance-benchmarks