2016 1st Joint International Workshop on Parallel Data Storage and data Intensive Scalable Computing Systems

Can Non-Volatile Memory Benefit MapReduce Applications on HPC Clusters?*

Md. Wasi-ur-Rahman, Nusrat Sharmin Islam, Xiaoyi Lu, and Dhabaleswar K. (DK) Panda
Department of Computer Science and Engineering, The Ohio State University
Email: {rahmanmd, islamn, luxi, panda}@cse.ohio-state.edu

***Abstract*—Modern High-Performance Computing (HPC) clusters are equipped with advanced technological resources that need to be properly utilized to achieve supreme performance for end applications. One such example, Non-Volatile Memory (NVM), provides the opportunity for fast scalable performance through its DRAM-like performance characteristics. On the other hand, distributed processing engines, such as MapReduce, are continuously being enhanced with features enabling high-performance technologies. In this paper, we present a novel MapReduce framework with NVRAM-assisted map output spill approach. We have designed our framework on top of the existing RDMA-enhanced Hadoop MapReduce to ensure both map and reduce phase performance enhancements to be present for end applications. Our proposed approach significantly enhances map phase performance proven by a wide variety of MapReduce benchmarks and workloads from Intel HiBench [9] and PUMA [18] suites. Our performance evaluation illustrates that NVRAM-based spill approach can improve map execution performance by 2.73x which contributes to the overall execution improvement of 55% for Sort. Our design also guarantees significant performance benefits for other workloads: 54% for TeraSort, 21% for PageRank, 58% for SelfJoin, etc. To the best of our knowledge, this is the first approach towards leveraging NVRAM in MapReduce execution frameworks for applications on HPC clusters.**

I. Introduction

High-Performance Computing (HPC) is an active area of research for many years now, providing ground-breaking solutions to many scientific and engineering problems with advanced technological solutions. Modern HPC clusters are equipped with high-performance hardware and software resources that need to be properly utilized to bring performance benefits to end applications. High-Performance interconnects, parallel file systems, modern accelerators, and coprocessors are few of the examples that are bringing fast effective solutions for both scientific and business domains. Non-Volatile Memory (NVM) is also one such resource that provides DRAM-like performance characteristics in addition to persistence which makes it an ideal choice for data-intensive computing. Because of its byte-addressable property, NVM can also be used as Non-Volatile Random Access Memory (NVRAM) which opens up opportunities to make NVRAM the second best choice after DRAM for performance sensitive applications.

On the other hand, recent studies [10–12, 15, 20, 21, 23, 24] have brought the two broad dimensions of modern computing, Big Data and HPC, to a convergent trajectory. Distributed processing engines, such as MapReduce [7], are continuously being enhanced with features enabling the usage of high-performance resources on HPC systems. High-performance RDMA (Remote Direct Memory Access)-capable designs of MapReduce framework [23, 24] brings down the performance bottlenecks in vanilla Hadoop MapReduce [4] significantly. With the availability of byte-addressable storage technologies, such as NVRAM, new performance enhancing opportunities become available for the RDMA-enhanced MapReduce frameworks. Although authors in [13] have already proposed NVRAM-based Hadoop Distributed File System (NVFS) that enhances the I/O and replication performance for HDFS, such investigations for the execution engines (e.g. MapReduce, Spark [28]) are yet to be done. All these lead us to the following broad challenges:

1) What are the possible choices for using NVRAM in the MapReduce execution pipeline?
2) How can MapReduce execution frameworks take advantage of NVRAM in such use cases?
3) Can MapReduce benchmarks and applications be benefitted through the usage of NVRAM in terms of performance and scalability?

In this paper, we have presented a novel MapReduce framework with NVRAM-assisted map output spill approach. We have designed our framework on top of the RDMA-enhanced Hadoop MapReduce [21, 23] to ensure both map and reduce phase performance enhancements to be present for end applications. Our proposed approach significantly enhances map phase performance proven by a wide variety of MapReduce benchmarks and workloads chosen from Intel HiBench [9] and PUMA [18]. Our performance evaluations illustrate that NVRAM-based spill approach can improve map phase execution performance by 2.73x which contributes to the overall execution improvement of 55% for Sort [2]. Our design also guarantees significant performance benefits for other workloads: 54% for TeraSort [3], 21% for PageRank [9], 58% for SelfJoin [18], etc.

The rest of the paper is organized as follows. Section II presents background and related work for this paper. We present the design in Section III. Section IV describes our detailed evaluation. We conclude in Section V with possible future works.

II. Background and Related Work

A. Advanced MapReduce Frameworks

Apache Hadoop [4] provides the implementation of MapReduce [7] programming model by executing user defined

*This research is supported in part by National Science Foundation grants #CNS-1419123, #IIS-1447804, and #CNS-1513120. It used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575.

978-1-5090-5216-5/16 $31.00 © 2016 IEEE
DOI 10.1109/PDSW-DISCS.2016.7

`map()` and `reduce()` functions with an execution framework managed by resource manager, YARN [25]. Because of performance limitations in the default Hadoop framework, advanced MapReduce designs [20, 21, 23, 24, 27] have been proposed in the literature that enhance the execution performance by utilizing advanced resources in HPC clusters. HOMR [23], a Hybrid approach of obtaining maximum possible Overlapping in MapReduce, enhances the default MapReduce through several of these new design features, such as RDMA-based shuffle engine and dynamic adjustment in shuffle data volume etc. HOMR also supports running MapReduce workloads efficiently on top of Lustre [20, 24].

B. Related Studies

There has been a number of studies in the literature to improve the MapReduce execution performance by improving the intermediate data storage and associated policies. Our earlier work [21] presents a pre-fetch cache that reduces the shuffle request response time by pre-fetching the intermediate data in the NodeManager. Different studies [20, 24] explore hybrid intermediate data directories with priority based selection scheme for MapReduce over Lustre environment. Authors in [26] have proposed different optimization techniques for MapReduce by considering workload characterization and storage architectures on HPC clusters. However, none of these works have considered utilizing NVM as intermediate data storage for MapReduce.

On the other hand, there has been much research to leverage the performance benefits of NVM for database and file systems. Authors in [6] present a file system and a hardware architecture to provide persistent, byte-addressable memory that provides strong reliability guarantees and better performance than traditional file systems. Fusion-IO's NVMFS [14] uses the virtualized flash storage layer for performance. In [19], the authors propose a hybrid file system to resolve the random write issues of SSDs by leveraging NVRAM. In [13], authors propose an advanced HDFS design that can leverage the byte-addressability of NVM for HDFS I/O operations and RDMA communications. However, none of the aforementioned studies explore the possibility of boosting the performance of MapReduce execution frameworks.

III. Opportunities and Design Details

In this section, we first discuss different opportunities to utilize NVRAM in the MapReduce execution pipeline. After that, we describe our design choices in detail.

A. Optimization Opportunities with NVRAM

The default MapReduce execution pipeline employs a number of disk operations during job execution. For example, the Map tasks spill the generated map output to the intermediate data directories. After spilling the data to disk, maps perform a merge operation that reads the spills from disks, merges them, and then writes the merged output back to disk. On the Reduce task, shuffle, merge, and reduce phases write the data to disk whenever the memory is not available. However, with the advanced MapReduce framework [23] discussed in Section II-A, the Reduce tasks can perform all the operations in memory (except the final write to the file system) without invoking any disk operations at all. Although this minimizes significant I/O bottlenecks in the reduce phase, the map phase still encounters I/O bottlenecks while spilling the map output to disks. Also, the map output data needs to be persisted so that the fault-tolerance of the Hadoop system remains unchanged. This gives us the motivation to utilize NVM for map output data.

Utilizing NVMs as PCIe devices (NVM SSD) for intermediate data directory is straightforward; but it cannot alleviate the I/O bottlenecks entirely. Figure 1 presents one such scenario where we evaluate Sort (20 GB on 4 node cluster) with intermediate directories configured using different storage devices (experimental setup is described in Section IV-A) for vanilla Hadoop. As we can see, different storage devices do not provide major performance benefits (RAMDisk improves 16% only over HDD for shuffle-intensive Sort) unless the execution pipeline is enhanced through other measures (e.g. RDMA with phase overlapping [21, 23]). This illustrates that to minimize the I/O bottlenecks in spill and merge operation, in-memory based spill is the only viable choice.

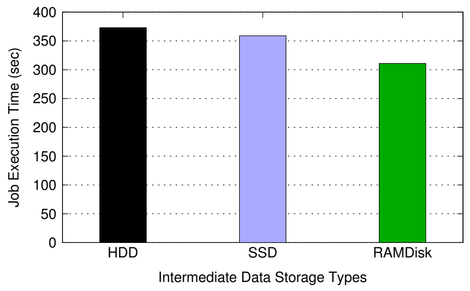


Figure 1. Evaluation of Sort with different intermediate data directory

To prove that spill and merge provide the major performance bottleneck in the map execution pipeline, we perform profiling analysis. According to [8, 22], the map execution performance, t_{Map}, can be defined using the equation:

$$t_{Map} = t_{read} + t_{map} + t_{collect} + t_{spill} + t_{merge}$$

Here, $read$, map, $collect$, $spill$, and $merge$ define different execution stages in the Map task. To identify the bottleneck region, we profile the Map tasks to find the break-down time for different stages. We use the micro-benchmarks, Sort and TeraSort, for these profiling experiments. Table I summarizes our observations.

Table I
Average break-down times for different stages in Map

Benchmark	Read + Map + Collect	Spill + Merge
Sort	1.96 s	3.36 s
TeraSort	3.14 s	11.79 s

As we can see, for both the benchmarks, spill and merge operations consume most of the execution time for the Map tasks. Read, map, and collect stages take less than 40% of the overall execution. Moreover, the map stage performance depends on the user-defined `map()` function which provides little opportunity to improve further. Thus, we concentrate on optimizing the spill and merge stages in map phase by leveraging NVRAM.

B. *Design of NVRAM-assisted Spill*

Figure 2 presents the architecture and design details of our NVRAM-assisted map output spill approach.

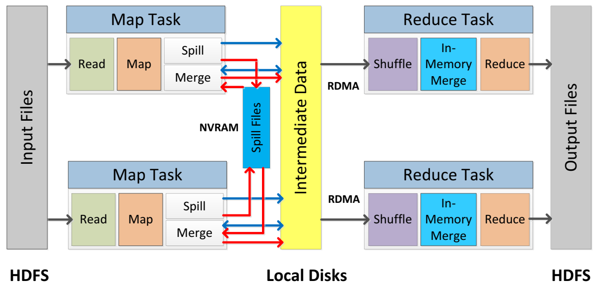


Figure 2. NVRAM-assisted spill with RDMA-enhanced MapReduce

In the default MapReduce framework, each Map task starts its execution by reading the split files that are assigned to it. It applies the user defined `map()` function to the read data and generates intermediate key-value pairs. This intermediate map output is spilled to the local disks before a final merge creates the sorted output for all the reducers (indicated by blue arrows in Figure 2). In our approach, instead of writing the map output to the local disks, we create an in-memory based data structure to hold the map output in memory. The data structure contains the map spill data, the partition info, and spill id. Since these map outputs may become really large based on the application characteristics, we use NVRAM instead of DRAM to store these data. The map output index information for the reducers are also kept in NVRAM through indexCache for future access during merge operations.

While performing the merge operation of these spill files, Map tasks read the spill bytes from NVRAM rather than from local disks, thus minimizing the disk access (indicated by red arrows in Figure 2). In the default MapReduce framework, merge operation just renames the spill file residing in local disk to the appropriate merged map output if the `map()` produces only one spill throughout its execution. However, in our design, with a single spill file, merge reads the file and then writes it to the local disk to generate the final intermediate data. Since in most big data applications, maps generate more than one spill files, our design can minimize the disk operations considerably in both spill and merge stages for most of the use cases.

Simulating NVRAM performance using DRAM: Since our experimental setup (described in Section IV-A) does not consist of any NVM devices, we use simulation approaches to mimic the NVRAM performance characteristics by using DRAM and adding a small delay. To do this, the map output data is written to and read from a hash map, an approach similar to other research studies [13, 17] on NVRAM performance simulation. We have assumed that NVRAM write is 10x slower compared to DRAM write and NVRAM read performs similarly compared to DRAM read [1, 17]. Using this assumption, we have added a small delay (δ) during spill write while kept as it is during spill read. To measure δ, we measure the spill write time (using `System.nanoTime()`) in DRAM for each spill file and add a sleep of 10 times of the measured time.

IV. Performance Evaluation

In this section, we divide the evaluations in three categories: (1) Evaluation of benefits in map phase, (2) Evaluation of benefits in the overall execution, and (3) Evaluation of macro benchmarks.

A. *Experimental Setup*

We have used SDSC Comet [5] cluster as our experimental setup. We have used nine nodes in this cluster. Each compute node in this cluster has two twelve-core Intel Xeon E5-2680 v3 (Haswell) processors, 128GB DDR4 DRAM, and 320GB of local SATA-SSD with CentOS operating system. The network topology in this cluster is 56Gbps FDR InfiniBand with rack-level full bisection bandwidth and 4:1 over-subscription cross-rack bandwidth.

We have used Hadoop-2.6.0 and JDK 1.7.0 throughout our experiments. Our RDMA-enhanced Hadoop MapReduce implementation is based on the RDMA-based Apache Hadoop [16] release 0.9.7. In the rest of the section, we refer Apache Hadoop MapReduce as **MR** and RDMA-enhanced Hadoop MapReduce as **RMR**. Our NVRAM-based design on top of RDMA-enhanced MapReduce is referred as **RMR-NVM**. In all our experiments for MR, RMR, and RMR-NVM, we have used SSD for both intermediate as well as HDFS data directory. For all the experiments, we have configured YARN to run 12 concurrent map and reduce containers with an HDFS block size of 256 MB.

B. *Evaluation of Benefits in Map Phase*

First, we present the performance improvement through NVRAM-based spill approach presented in Section III. Similar to Table I, we present the average break-down times for map stages and compare the performance among MR, RMR, and RMR-NVM. For this experiment, we use Sort and TeraSort benchmark with a data size of 20 GB each on 8 worker nodes.

Table II
Comparison of average break-down times for different stages in map

Benchmark	Read + Map + Collect (s)			Spill + Merge (s)		
	MR	RMR	RMR-NVM	MR	RMR	RMR-NVM
Sort	1.96	1.87	1.86	3.36	3.33	1.64
TeraSort	3.14	3.14	3.13	11.79	11.79	5.42

As shown in Table II, with our new NVRAM-based design, RMR-NVM achieves significant benefits in Spill + Merge stages compared to both MR and RMR. For both the benchmarks, the benefit is almost 2x in Spill + Merge, whereas Read + Map + Collect is performing similarly across MR and RMR. The NVRAM-based spill approach presented in this paper helps to achieve this benefit over default MR and RDMA-enhanced MR.

To take a closer look at the spill cost (t_{spill}), we further profile the spill operation running both of these benchmarks and compare t_{spill} between MR, RMR, and RMR-NVM. For space limitation, we present the evaluation for Sort benchmark only. Figure 3 presents this result. We run a total of 96 maps on 8 nodes for this experiment with 6 concurrent containers per node. We present the sorted spill costs for all the frameworks. The performance variations obtained in the spill cost are related to the resource contention among multiple concurrent mappers and reducers.

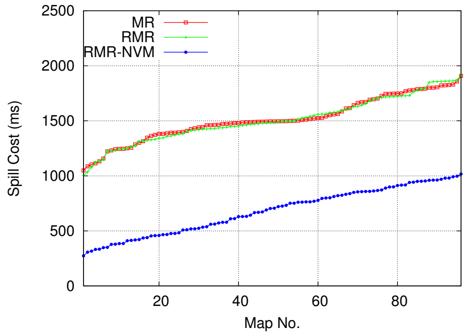


Figure 3. Performance evaluation of NVRAM-based spilling

As shown in Figure 3, RMR-NVM reduces the spill cost for all maps and achieves a maximum of 3.83x improvement compared to MR and RMR. An average benefit of 2.39x is obtained across all maps for this experiment. For TeraSort also, we observe similar performance benefits in (t_{spill}) and an average improvement of 2.73x is obtained for RMR-NVM compared to MR and RMR.

C. Evaluation of Benefits in the Overall Execution

In this section, we present the overall performance benefits of RMR-NVM compared to MR and RMR. We also isolate the performance benefits obtained from map phase enhancements and reduce phase enhancements. We choose the two popular shuffle-intensive benchmarks, Sort and TeraSort, for this evaluation. Figure 4 presents these results.

For both Sort and TeraSort experiments, we vary the data size from 20 GB to 60 GB on 8 worker nodes. Due to several design features in the reduce phase, RMR achieves significant performance benefits compared to MR running over IPoIB. For 60 GB data size, RMR achieves 37% performance benefit compared to MR with the reduce phase enhancements. However, as shown in Figure 4(a), the map phase performs similarly in both MR and RMR. With NVRAM-based design enhancements presented in this paper, RMR-NVM can achieve significant benefits in the map phase as well. For 60 GB Sort experiment, RMR-NVM achieves 28% performance benefits compared to RMR and 55% performance benefits compared to MR over IPoIB. The map phase in RMR-NVM achieves 2.37x performance benefits compared to both MR and RMR.

We observe a similar trend in TeraSort experiment as shown in Figure 4(b). However, the map phase executes longer compared to Sort because of the small key-value (100 bytes) size granularity in read and write operations compared to that of Sort. Similar to Sort experiment, RMR achieves 30% performance benefit compared to MR based on the reduce phase enhancements for the data size of 60 GB. However, RMR-NVM improves the map phase as well and obtains an improvement of 31% and 51% compared to RMR and MR, respectively.

D. Evaluation of Macro-benchmarks

In this section, we evaluate different benchmarks from Intel HiBench repository [9] and PUMA (PUrdue MApreduce benchmark suite) [18] repository.

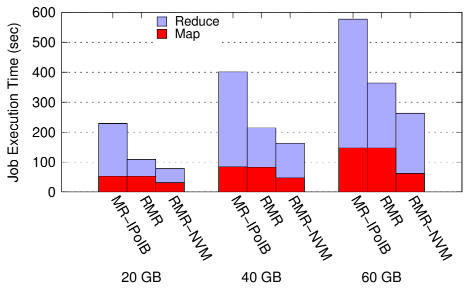
1) Intel HiBench: From Intel HiBench [9] repository, we choose six benchmarks for our evaluation. We choose both shuffle-intensive and other workloads for this evaluation to make sure that our design over NVRAM performs similar or better compared to the previous best case, RMR.

Sort: We evaluate HiBench Sort and compare the performance benefits obtained over MR and RMR. Although this benchmark is similar to the Hadoop in-built (evaluated in Section IV-C) Sort, we use different benchmarks for the input data generation phase. For HiBench Sort, we use RandomTextWriter benchmark to produce text input data, whereas in Section IV-C we evaluated with binary input data. We present here the performance evaluation with HiBench Gigantic data set (around 25 GB) and we see that RMR-NVM achieves 42% and 14% performance benefits compared to MR and RMR, respectively.

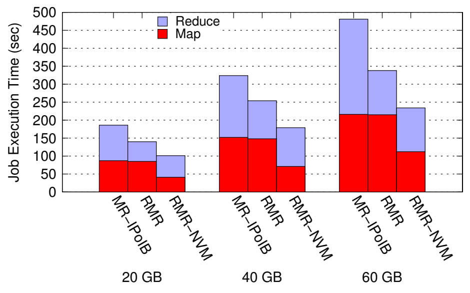
TeraSort: For HiBench TeraSort, we run the Huge (32 GB) data set from Intel HiBench with 32 mappers and 32 reducers. Here, RMR-NVM outperforms MR by 54% and RMR by 39%, similar to our previous observation from the default Hadoop in-built TeraSort benchmark.

PageRank: For PageRank, we choose the Huge dataset which has an input of 5,000,000 web pages. The data source for this workload is generated from Web data whose hyperlinks follow the Zipfian distribution. We set the number of iterations for the algorithm as 3 and choose 32 mappers and 16 reducers in each iteration. For this workload, we observe 21% performance benefit for RMR-NVM compared to MR.

Enhanced DFSIO: Although DFSIO is intended to stress the file system performance in terms of read and write throughput, we highlight the benefits that we observe in job execution time for write operation. We run with 256 maps,

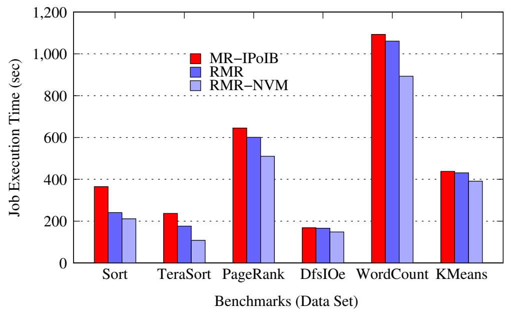


(a) Sort

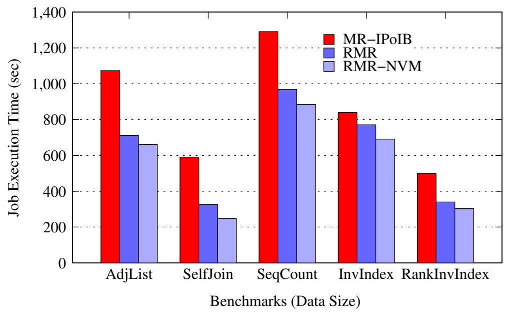


(b) TeraSort

Figure 4. Comparison of Map and Reduce time in the overall execution



(a) Intel HiBench [9]



(b) PUMA [18]

Figure 5. Performance benefits for Intel HiBench and PUMA workloads

each writing a file of 100 MB size. Since, this benchmark is not shuffle-intensive, we use only one reducer. RMR-NVM achieves 12% performance benefit compared to MR in terms of execution time.

WordCount: For WordCount, we use the Huge data set (25 GB) as well. We keep the combiner disabled for this benchmark and run with 32 maps and 1 reduce. Here, we observe 18% benefit compared to MR over IPoIB. WordCount is a compute-intensive benchmark and thus, the performance benefit is less compared to other workloads.

Kmeans: Intel HiBench has an implementation for the K-Means clustering algorithm. We choose Huge data set as well here which is generated based on Uniform Distribution and Gaussian Distribution. The input data set has 100,000,000 samples residing in 20 different files. We choose 5 clusters and 5 iterations for our evaluation. As shown in Figure 5(a), we observe 11% performance benefit compared to MR.

2) PUMA: From PUMA repository, we choose five different workloads for our evaluation. We evaluate each with a 30 GB data size on 8 nodes (32 mappers and 16 reducers) and compare the results among MR, RMR, and RMR-NVM. Figure 5(b) presents these results.

AdjList: This benchmark generates adjacency and reverse adjacency lists of nodes of a graph. We use the data set from the PUMA repository and observe a benefit of 39% compared to the default framework. The map phase in RMR-NVM performs almost 2x faster compared to that in MR.

SelfJoin: This benchmark is similar to the candidate generation part of the a-priori algorithm [18]. Here also, we use the data set from PUMA repository and observe 58% performance benefit compared to MR. Compared to RMR, we observe 24% performance gain.

SeqCount: This benchmark generates a count of all unique sets of three consecutive words in the input data set. To generate the input, we use RandomTextWriter benchmark. With SeqCount, RMR-NVM can achieve 32% performance benefit compared to MR.

InvIndex: InvIndex is used to generate word to document indices based on the input documents. RMR-NVM outperforms MR and RMR marginally here as this benchmark is not shuffle-intensive. Here, we observe a benefit of 18% for RMR-NVM compared to MR.

RankInvIndex: RankInvIndex is a shuffle-intensive workload which is used to generate a word list with the frequency. We use the output of SeqCount benchmark as the input and observe 39% benefit compared to MR.

V. Conclusion

In this paper, we present an enhanced design for MapReduce with NVRAM. Our proposed approach significantly enhances map phase performance proven by a wide variety of MapReduce benchmarks and workloads chosen from Intel HiBench [9] and PUMA [18]. Performance evaluations show that NVRAM-based spill approach can improve map phase execution performance by an average of 2.73x which contributes to the overall execution improvement of 55% for Sort micro-benchmark and 58% for SelfJoin workload. In the future, we plan to extend different MapReduce execution frameworks (e.g. Spark, Tez) by leveraging the performance benefits from NVRAM.

References

[1] NVRAM. http://www.enterprisetech.com/2014/08/06/flashtec-nvram-15-million-iops-sub-microsecond-latency/.

[2] Sort. http://wiki.apache.org/hadoop/Sort.

[3] TeraSort. http://hadoop.apache.org/docs/r0.20.0/api/org/apache/hadoop/examples/terasort/TeraSort.html.

[4] The Apache Hadoop Project. http://hadoop.apache.org/.

[5] Comet at San Diego Supercomputer Center. http://www.sdsc.edu/services/hpc/hpc_systems.html#comet.

[6] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, D. Burger, B. Lee, and D. Coetzee. Better I/O Through Byte-Addressable, Persistent Memory. In *Symposium on Operating Systems Principles (SOSP)*, 2009.

[7] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the OSDI*, 2004.

[8] H. Herodotou. Hadoop Performance Models. Technical Report CS-2011-05, Computer Science Department, Duke University.

[9] Intel. HiBench Suite. https://github.com/intel-hadoop/HiBench.

[10] N. S. Islam, X. Lu, M. W. Rahman, D. Shankar, and D. K. Panda. Triple-H: A Hybrid Approach to Accelerate HDFS on HPC Clusters with Heterogeneous Storage Architecture. In *15th IEEE/ACM Intl. Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2015.

[11] N. S. Islam, M. W. Rahman, J. Jose, R. Rajachandrasekar, H. Wang, H. Subramoni, C. Murthy, and D. K. Panda. High Performance RDMA-based Design of HDFS over InfiniBand. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2012.

[12] N. S. Islam, M. W. Rahman, X. Lu, D. Shankar, and D. K. Panda. Performance Characterization and Acceleration of In-Memory File Systems for Hadoop and Spark Applications on HPC Clusters. In *2015 IEEE International Conference on Big Data (IEEE BigData)*, 2015.

[13] N. S. Islam, M. Wasi-ur Rahman, X. Lu, and D. K. Panda. High Performance Design for HDFS with Byte-Addressability of NVM and RDMA. In *Proceedings of International Conference on Supercomputing*, ICS '16, 2016.

[14] W. K. Josephson, L. A. Bongo, K. Li, and D. Flynn. DFS: A File System for Virtualized Flash Storage. *Trans. Storage*, 2010.

[15] X. Lu, N. S. Islam, M. W. Rahman, J. Jose, H. Subramoni, H. Wang, and D. K. Panda. High-Performance Design of Hadoop RPC with RDMA over InfiniBand. In *ICPP*, France, 2013.

[16] OSU NBC Lab. High-Performance Big Data (HiBD). http://hibd.cse.ohio-state.edu.

[17] S. Pelley, T. F. Wenisch, B. T. Gold, and B. Bridge. Storage Management in the NVRAM Era. *Proc. VLDB Endow.*, 2013.

[18] Purdue MapReduce Benchmarks Suite (PUMA). https://sites.google.com/site/farazahmad/pumabenchmarks.

[19] S. Qiu and A. L. N. Reddy. NVMFS: A Hybrid File System for Improving Random Write in Nand-Flash SSD. In *IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*, 2013.

[20] M. W. Rahman, N. Islam, X. Lu, and D. Panda. A Comprehensive Study of MapReduce over Lustre for Intermediate Data Placement and Shuffle Strategies on HPC Clusters. *IEEE Transactions on Parallel and Distributed Systems*, 2016.

[21] M. W. Rahman, N. S. Islam, X. Lu, J. Jose, H. Subramoni, H. Wang, and D. K. Panda. High-Performance RDMA-based Design of Hadoop MapReduce over InfiniBand. In *Proceedings of HPDIC, in conjunction with IPDPS*, 2013.

[22] M. W. Rahman, X. Lu, N. Islam, and D. Panda. Performance Modeling for RDMA-Enhanced Hadoop MapReduce. In *Proceedings of ICPP*, 2014.

[23] M. W. Rahman, X. Lu, N. S. Islam, and D. K. Panda. HOMR: A Hybrid Approach to Exploit Maximum Overlapping in MapReduce over High Performance Interconnects. In *Proceedings of ICS*, 2014.

[24] M. W. Rahman, X. Lu, N. S. Islam, R. Rajachadrasekar, and D. K. Panda. High-Performance Design of YARN MapReduce on Modern HPC Clusters with Lustre and RDMA. In *Proceedings of IPDPS*, 2015.

[25] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler. Apache Hadoop YARN: Yet Another Resource Negotiator. In *Proceedings of SoCC*, 2013.

[26] Y. Wang, R. Goldstone, W. Yu, and T. Wang. Characterization and Optimization of Memory-Resident MapReduce on HPC Systems. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, May 2014.

[27] Y. Wang, X. Que, W. Yu, D. Goldenberg, and D. Sehgal. Hadoop Acceleration through Network Levitated Merge. In *Proceedings of SC*, 2011.

[28] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster Computing with Working Sets. In *Proceedings of HotCloud*, 2010.