Analysis and Optimization of Massive Data Processing on High Performance Computing Architecture

He Huang, Shanshan Li, Xiaodong Yi, Feng Zhang, Xiangke Liao and Pan Dong
School of Computer Science
National Univ. of Defense Technology, Changsha, China
{*huanghe, shanshanli, xdyi, zhangfeng, xkliao, pdong*}*@nudt.edu.cn*

***Abstract*—MapReduce has emerged as a popular and easy-to-use programming model for numerous organizations to deal with massive data processing. Present works about improving MapReduce are mostly done under commercial clusters, while little work has been done under HPC architecture. With high capability computing node, networking and storage system, it might be promising to build massive data processing paradigm on HPCs. Instead of DFS storage systems, HPCs use dedicated storage subsystem. We first analyze the performance of MapReduce on dedicated storage subsystem. Results show that the performance of DFS scales better when the number of nodes increases; but, when the scale is fixed and the I/O capability is equal, the centralized storage subsystem can do a better job in processing large amount of data. Based on the analysis, two strategies for reducing the network transmitting data and distributing the storage I/O are presented, so as to solve the problem of limited data I/O capability of HPCs. The optimizations for storage localization and network levitation in HPC environment respectively improve the MapReduce performance by 32.5% and 16.9%.**

Keywords-high-performance computer; massive data processing; MapReduce paradigm.

I. Introduction

MapReduce [1] has emerged as a popular and easy-to-use programming model for numerous organizations to process explosive amounts of data and deal with data-intensive problems. Meanwhile, data-intensive applications, such as huge amount of web pages indexing and data mining in business intelligence nowadays have become very popular and are among the most important classes of applications. At the same time, High Performance Computers (HPCs) often deal with traditional computation-intensive problems. Though HPCs are very powerful when dealing with scientific computation problems, the architecture currently is not very suitable for running MapReduce paradigm and processing data-intensive problems.

There have been works done by improving MapReduce performance under HPC architecture. Yandong Wang et al. [3] improves Hadoop performance through optimizing its networking and several stages of MapReduce on HPC architecture. Wittawat et al. [4] integrates PVFS (Parallel Virtual File System) into Hadoop and compare its performance to HDFS and studies how HDFS-specific optimizations can be matched using PVFS and how consistency, durability, and persistence tradeoffs made by these file systems affect application performance. However, specific issues related to HPC architecture, especially the dedicated storage subsystem, are seldom taken into consideration in these former works. In the storage aspect these works are oriented to distributed file system (DFS) which uses local disks of each node to store data blocks, and it is not relevant to the centralized storage subsystem of the HPC architecture.

In this paper, alternatively, we consider every aspect of the HPC architecture, including processor, networking and especially the storage subsystem. In our work, the differences of DFS and centralized storage subsystem are analyzed in detail, and optimizations are proposed for the storage subsystem specifically in HPC environment. The prior concern of this paper is the deploying of MapReduce paradigm on HPCs and its overall performance. First of all, the difficulty and the significance of the Massive Data Processing problem on HPCs is described, and the necessity, feasibility, and problems that may be encountered of deploying MapReduce Paradigm on HPCs are analyzed. Secondly, the performance of MapReduce Paradigm on HPCs, especially the I/O capability of the dedicated storage subsystem and the DFS is analyzed and evaluated. Following that, two optimization strategies for relieving the I/O burden of the system and improving the performance of MapReduce on HPCs are presented, due to the limited data I/O capability of HPCs, which probably cannot meet the requirements of data-intensive applications.

Several challenges exist for deploying MapReduce paradigm and dealing with data-intensive problems effectively on HPCs. Firstly, data blocks are distributed and stored on DFS but centrally stored on the storage subsystem of HPCs. Therefore, how to decrease data transmission in advantage of the centralized storage in order to improve performance is a great challenge. Secondly, the IO and buffering capability of centralized storage is not as good as DFS. How to relieve the burden of storage I/O and improve the overall performance is another challenge.

This paper explores the possibility of building Massive Data Processing Paradigm on HPCs, and discusses how to deal with Massive Data Processing applications efficiently on HPCs and how to improve its performance. The main contributions are:

(1) The performance of MapReduce Paradigm on HPCs, especially the I/O capability of the dedicated storage subsystem specific to HPCs is analyzed. Results show that the performance of DFS scales better when the number of nodes increases, but when the scale is fixed, the centralized storage subsystem can do better in processing large amount of data.

(2) Two strategies for improving the performance of the MapReduce paradigm on HPCs are presented, so as to solve the problem of limited data I/O capability of HPCs. The optimizations for storage localization and network levitation in HPC environment respectively improve the MapReduce performance by 32.5% and 16.9%.

The paper is organized as follows: related works of HPCs and data-intensive applications are discussed in Section II. In Section III, the specific issues of running MapReduce paradigm on HPCs are analyzed. Following that, in Section IV, two optimization strategies for improving the performance of the MapReduce Paradigm on HPCs are presented, in order to solve the problem of limited data I/O capability of HPCs, which probably cannot meet the requirements of data-intensive applications. Then, the effectiveness of these two optimization strategies is demonstrated respectively through experiments in Section V. Finally, we give a conclusion in Section VI.

II. Related Work

MapReduce is a programming model for large-scale arbitrary data processing. The model popularized by Google provides very simple but powerful interfaces, while hiding complex details of parallelizing computation, fault-tolerance, distributing data and load balancing. Its open-source implementation, Hadoop [2], provides a software framework for distributed processing of large datasets.

A rich set of research has been published on improving the performance of MapReduce recently. Originally, the Hadoop scheduler assumed that all nodes in a cluster were homogeneous and made progress with the same speed. Jiang et al. [5] conducted a comprehensive performance study of MapReduce (Hadoop), concluding that the total performance could be improved by a factor of 2.5 to 3.5 by carefully tuning the factors, including: I/O mode, indexing, data parsing, grouping schemes and block-level scheduling. Zaharia et al. [6] designed a new scheduling algorithm, Longest Approximate Time to End (LATE), for heterogeneous environments where ideal application environment might not be available.

Ananthanarayanan et al. [7] proposed the Mantri system which manages resources and schedules tasks on the MapReduce system of Microsoft. Mantri monitors tasks and culls outliers using cause- and resource-aware techniques and Mantri improves job completion times by 32%. Y. Chen et al. [8] proposed a strategy called Covering Set (CS) to improve the energy efficiency of Hadoop. It keeps only a small fraction of the nodes powered up during periods of low utilization, as long as all nodes in the Covering Set are running. The strategy should ensure that there is at least one copy of all data blocks in the Covering Set. On the other hand, Willis Lang et al. [9] proposed All-In Strategy (AIS). AIS uses all the nodes in the cluster to run a workload and then powers down the entire cluster. Both CS and AIS are efficient energy saving strategies.

The closest work to ours is Hadoop-A as proposed by Yandong Wang et al. [3] and Reconciling HDFS and PVFS by Wittawat et al. [4] The former paper improves Hadoop performance through optimizing its networking and several stages of MapReduce on HPC architecture. It introduces an acceleration framework that optimizes Hadoop and describes a novel network-levitated merge algorithm to merge data without repetition and disk access. Taking advantage of the InfiniBand network and RDMA protocol of HPCs, Hadoop-A doubles the data processing throughput of Hadoop, and reduces CPU utilization by more than 36%.

The second one, Reconciling HDFS and PVFS, explores the similarities and differences between PVFS, a parallel file system used in HPC at large scale, and HDFS, the primary storage system used in cloud computing with Hadoop. It integrates PVFS into Hadoop and compare its performance to HDFS using a set of data-intensive computing benchmarks. It also studies how HDFS-specific optimizations can be matched using PVFS and how consistency, durability, and persistence tradeoffs made by these file systems affect application performance.

Nonetheless, not every aspect of the HPC architecture is taken into consideration. For example, previous works claim that due to the price and the poor scalability of the centralized storage subsystem, disk arrays are not suitable for massive data processing. So in these works they simply use the DFS built upon local disks of each node. Consequently, in this paper a thorough study of dealing with massive data processing on the HPC architecture is given. The impact of every aspect on performance is checked, including processor, networking, and especially the storage subsystem.

III. Specific Issues of MapReduce on HPCs

This section mainly evaluates performance of MapReduce paradigm on HPCs through experiments and analyzes the problems of running MapReduce paradigm on both HPCs and clusters of commercial machines, especially the differences caused by the centralized storage subsystem and the DFS.

When running MapReduce paradigm on HPCs, the input and output data are stored in a dedicated storage subsystem (mainly composed of disk arrays and a parallel file system). Meanwhile, when running MapReduce paradigm on clusters of commercial machines, the Distributed File System (DFS) is responsible for managing the input and output data, and the data is actually stored on local disks of each node.

Table I
CLUSTER I/O PERFORMANCE UNDER DIFFERENT SIZES (MB/S)

#nodes	DFS		Storage Subsystem	
	Read	Write	Read	Write
4	2,960	408	13,300	1,024
7	4,690	630	15,500	1,010
10	6,400	860	19,000	1,020
20	12,120	1,680	29,044	1,022
40	23,760	3,280	31,100	1,200
80	44,000	6,400	31,000	1,190
100	62,200	8,080	31,093	1,160

Note that the read & write throughput of the cluster is evaluated by the Hadoop benchmark TestDFSIO. The throughput of 2,960, e.g., denotes that the cluster has a throughput of 2,960 MB/s for read.

Table II
MAPREDUCE PERFORMANCE UNDER DIFF. AMOUNT OF DATA

		60G	80G	100G	120G	140G
DFS	1	2'42	2'57	6'45	10'22	18'22
	2	3'16	6'06	7'55	12'17	15'34
	3	2'43	6'22	11'49	17'38	15'49
Disk Array	1	4'17	6'30	7'27	8'47	10'32
	2	4'49	6'51	8'08	9'52	11'40
	3	5'02	6'43	8'09	9'40	11'25

Note that the job finish time is evaluated by the Hadoop benchmark Sort with different amount of data. The finish time of 2'42, e.g., denotes that the job was finished in 2 min 42 sec. The size of both clusters is fixed with 10 nodes. Every job is evaluated for three times.

In order to analyze the performance of MapReduce paradigm on HPCs, it is needed to compare its performance with the performance of MapReduce paradigm on cluster of commercial machines under the same scale. So experiments in this section are divided into two groups: the first group store input and output data on dedicated storage subsystem of HPCs, representing the HPC computing environment. The second group uses the same scale of nodes, and differences are that the data is actually stored on local disks of each node and the DFS is responsible for managing data storage.

Data-intensive applications are different from traditional scientific computation applications. They need much more data accessing I/O bandwidth (i.e., disk accessing I/O bandwidth and network accessing I/O bandwidth) than computation resources. For I/O bandwidth is so important, the I/O capacity of the cluster should be evaluated first.

First of all, cluster I/O performance under different sizes is evaluated. Evaluation is done respectively in a commercial cluster and under the HPC environment. Nodes in these two clusters are the same, but during each assessment the number of nodes increases. The size of the DFS increases as the cluster scales, but the size of the centralized storage subsystem stays all the same: the dedicated storage system is composed of 167 600 GB fiber channel disks, and managed by Lustre [11] parallel file system. The I/O capacity of the cluster under different scales is listed in Table I.

From Table I, we can see that the I/O performance of DFS can improve linearly as the number of nodes increases. This is because as the number of nodes increases, the number of local disks in the DFS also increases. Under the management of the DFS, the I/O performance of the cluster can take advantage of all local disks to achieve aggregation I/O bandwidth, bringing linear performance improvement.

On the other hand, for dedicated storage subsystem, its I/O performance depends on the scale of disk arrays in the storage subsystem, and is almost not relevant with the number of computing nodes. As in the current experiment the size of the storage subsystem is fixed, the I/O bandwidth it can provide for all compute nodes is limited. Therefore, we get *Analysis 1: the scalability of DFS is better than that of the centralized storage system. The performance of DFS improves when the number of nodes increases, and the performance of centralized storage improves only when new disk arrays are added.*

Secondly, the performance of these two clusters under the same scale is evaluated. Table II describes MapReduce performance under different amount of data when the system is composed of 10 nodes and the I/O capability of DFS and centralized storage is nearly equal. From Table II, we can see that when the amount of data is small (less than 100 GB), the MapReduce performance of the DFS is better than the performance of the centralized storage subsystem. On the contrary, When the amount of data is large (more than 100 GB), the MapReduce performance based on centralized storage subsystem becomes much better. Besides the disadvantage in scalability of the centralized storage, this phenomenon reveals an advantage of the disk arrays and we get *Analysis 2: when the DFS and the centralized storage subsystem can provide equal I/O capability, the disk arrays of the centralized storage subsystem are better at dealing with huge amount of data, compared to disks of the DFS.*

In general, the experiments show that compared to the computation resources, the I/O accessing bandwidth is a valuable resource under HPC architecture and may has great impact on the performance of MapReduce. Firstly, the scalability of DFS is better than that of the centralized storage system. Meanwhile, the cost of enlarging the scale of centralized storage is much higher than that of DFS. Secondly, an advantage of the centralized storage is, the disk arrays are better at handling large amount of data, compared to the DFS. Based on the above analysis, optimizations for relieving the burden of I/O system and improving the overall performance are proposed in the next section.

IV. Optimizations of MapReduce Paradigm on HPCs

Based on the analysis of Section III, designs for improving the performance of the MapReduce Paradigm on HPCs are proposed, in order to solve the problem of limited data I/O capability of HPCs, which probably cannot meet the requirements of data-intensive applications. Therefore, two optimizations for relieving the burden of I/O system and improving the overall performance, Intermediate Results Network Transfer Optimization and Intermediate Results Localized Storage Optimization, are proposed in this section.

A. Intermediate Results Network Transfer Optimization

Data blocks are distributed and stored on DFS but centrally stored on the storage subsystem of HPCs. The main idea of Intermediate Results Network Transfer Optimization is to decrease data transmission in advantage of the centralized storage in order to reduce the time cost on networking and improve performance.

HPCs use dedicated storage subsystem and parallel file system to provide storage services for all computing nodes. DFS handle data blocks that is physically distributed on disks of each node, and logically organize these data blocks into a unified name space. On the other hand, when using dedicated storage subsystem, the difference is that data blocks are stored in disk arrays that are physically centralized. Therefore, the Map Output Files (MOFs) in the MapReduce paradigm are centrally stored in the storage subsystem, not distributed on every disk of each node, which brings possibility of optimizing network transmission of MOFs.

On clusters of commercial machines, after the Map phase of a MapReduce job finishes, the intermediate results (MOFs) are stored locally on the disk of the Map task execution node. At the Shuffle phase, all nodes that execute Reduce tasks must get the MOFs from the Map task execution node. So, these data blocks (MOFs) must be transmitted over the network between nodes. At this time, the MOFs are stored on distributed nodes, and their network transmission is inevitable.

Different from MapReduce jobs on clusters of commercial machines, these jobs on HPCs store intermediate results in the dedicated storage subsystem. Therefore, in this case, Map tasks just need to transmit the division and storage information of MOFs to all Reduce tasks, and then the Reduce tasks themselves are responsible for reading the corresponding intermediate results directly from the dedicated storage subsystem. Intermediate Results Network Transfer Optimization is illustrated in Figure 1.

From Figure 1, we can see that after Intermediate Results Network Transfer Optimization, Map tasks just transmit the division and storage information of MOFs to all Reduce tasks, and then the Reduce tasks themselves go for reading the corresponding intermediate results directly from the dedicated storage subsystem. This eliminates the process of transmitting the intermediate results to Reduce tasks over network and can relieve the networking I/O burden of the system. If the network is the performance bottleneck of the system, this optimization can improve the overall system performance.

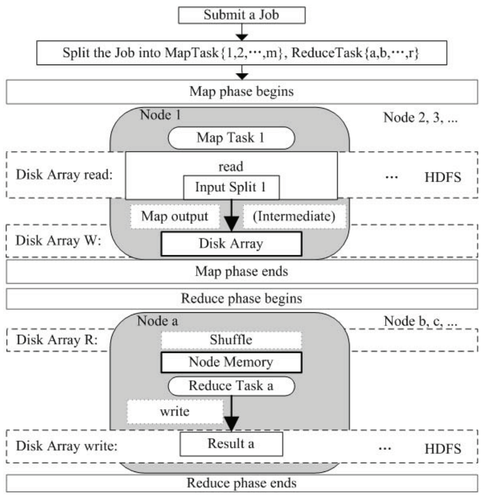

Figure 1. Intermediate Results Network Transfer Optimization

B. Intermediate Results Localized Storage Optimization

Compared to the networking I/O resources, the storage I/O resources provided by the centralized storage system are more likely to become the performance bottleneck of the system. The main idea of Intermediate Results Localized Storage Optimization is to distribute the storage I/O to both the centralized storage and local disks of each computing node. By storing temporary data files on local disks of computing nodes, the I/O pressure of centralized storage can be reduced greatly.

The I/O capability which the dedicated storage subsystem can provide is limited by the size of the storage subsystem itself. On the other side, when the MapReduce paradigm is initially designed, the intermediate results (MOFs) are not written to DFS, but stored temporarily on the local disk of each node. We can learn from this design, and store the intermediate results temporarily on the local disk of each node to reduce data I/O pressure of the centralized storage system.

Furthermore, as the dedicated storage subsystem of HPCs is expensive and limited in scale, the capacity and I/O capability of the storage subsystem often become a kind of scarcer resources, rather than network I/O bandwidth or computation resources. Then it is more urgent to relieve the I/O pressure of the storage system, rather than to optimize

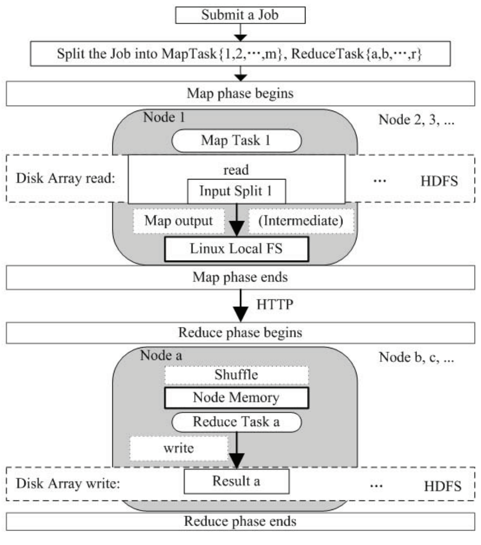

Figure 2. Intermediate Results Localized Storage Optimization

the data transmission over the network to improve the performance of MapReduce paradigm on HPCs.

In view of this, we can learn from the practice of distributed file systems, and buffer the intermediate results locally. As MOFs are temporary files and belong to specific jobs, and are usually deleted after the completion of their corresponding job, buffering the MOFs locally does not affect the correct execution of MapReduce jobs. At the same time, buffering intermediate results locally can relieve the burden of the centralized storage greatly. Intermediate Results Localized Storage Optimization is illustrated in Figure 2.

From Figure 2, we can see that the job input and output data are read and written to the centralized storage, but the intermediate results are no longer written to the centralized storage. Instead, they are buffered locally on disks of each computing node. Therefore, the I/O of the whole system is distributed and the burden of centralized storage is relieved greatly.

V. Evaluation

Experiments are done respectively in a commercial cluster and the HPC environment. The commercial cluster and HPC environment both have 100 compute nodes, each node has dual-way six-core 2.93 GHz Intel Xeon processors and 50GB memory. In the commercial cluster nodes are connected by 1 GB Ethernet. In HPC environment nodes are connected by 40 Gbps optical fiber channel and InfiniBand [10] network. The dedicated storage system is composed of 167 600 GB fiber channel disks, and managed by Lustre [11] parallel file system.

Three groups of evaluation are done in this section. First of all, the scalability of the DFS and the centralized storage

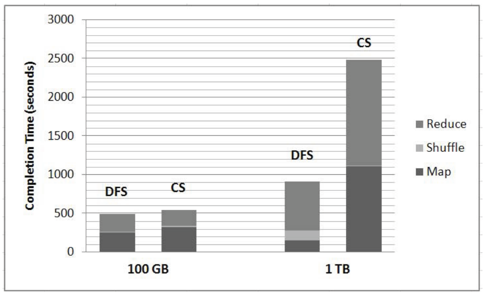

Figure 3. The Performance Scalability of DFS and CS

is evaluated and compared. Then, the effectiveness of the Intermediate Results Network Transfer Optimization and the Intermediate Results Localized Storage Optimization is demonstrated respectively, when the networking or storage I/O becomes the performance bottleneck of the overall system.

Firstly, the Terasort benchmark of Hadoop is run to evaluate the performance scalability of DFS and centralized storage. 100 GB data is sorted on 10 nodes and 1TB data is sorted on 100 nodes respectively. And the networking of both groups is 40 Gbps InfiniBand. The results are illustrated in Figure 3. In Figure 3, DFS represents the distributed file system and CS represents the centralized storage. The total time cost is composed of the time cost of three MapReduce phases: Map, Shuffle and Reduce.

From Figure 3, we can see that, when 100 GB data is sorted on 10 nodes, the performance of DFS and CS is nearly equal. But when the system scales, that is, when 1 TB data is sorted by 100 nodes, the performance of CS is worse than that of DFS. As the computation hardware and networking are the same, the differences come from distinctive storage system. This validates our analysis in Section 3: the scalability of DFS is better than that of the centralized storage system. In fact, the cost of enlarging the scale of centralized storage is much higher than that of DFS, as disk arrays are much more expensive than simple disks attached to computing nodes.

Secondly, the effectiveness of Intermediate Results Localized Storage Optimization is demonstrated. The same from above, 1 TB data is sorted on 100 nodes with centralized storage, for the first time without optimization and the second time with storage localization optimization. The networking of both tests is 40 Gbps InfiniBand. From the former experiments we can see that the performance bottleneck of the overall system lies on the storage I/O. After Intermediate Results Localized Storage Optimization, the MOFs are not written to the centralized storage system anymore, and it greatly relieves the pressure of the disk arrays of the storage subsystem. The results are illustrated in Figure 4.

From Figure 4, we can see that after storage localization

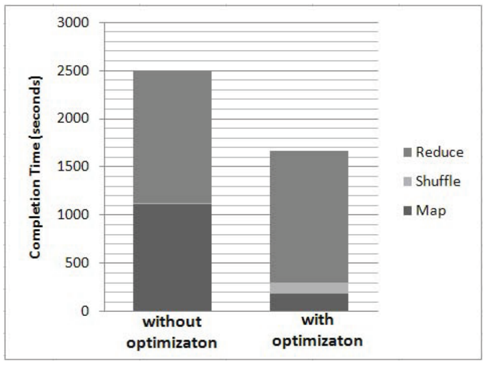

Figure 4. Validation of Intermediate Results Localized Storage Optimization

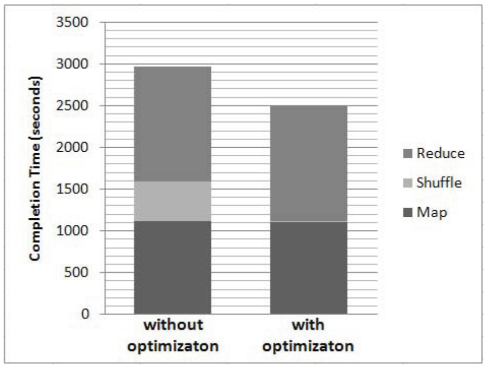

Figure 5. Validation of Intermediate Results Network Transfer Optimization

optimization, the time cost decreases greatly, especially the time cost of Map phase. Because during Map phase, it is not needed any more to write intermediate results to the centralized storage, the performance of Map phase improves a lot. In fact, the Intermediate Results Localized Storage Optimization in HPC environment can improve the MapReduce performance by 32.5%.

Thirdly, the effectiveness of Intermediate Results Network Transfer Optimization is demonstrated. The same from above, 1 TB data is sorted on 100 nodes with centralized storage, for the first time without optimization and the second time with network levitation optimization. But the networking changes to 1 GB/s Ethernet this time, in order to see the networking as performance bottleneck. The results are illustrated in Figure 5.

Different from the former experiments, we can see from Figure 5 that the performance bottleneck of the overall system this time lies on both the networking and the storage I/O. Note that if the 40 Gbps InfiniBand is used this time, the networking would not be the bottleneck and the Intermediate Results Network Transfer Optimization would become useless.

Figure 5 shows that when the networking capability turns into the bottleneck of the system, the Intermediate Results Network Transfer Optimization in HPC environment can improve the MapReduce performance by 16.9%. In fact, only the Shuffle phase consumes the networking resources, and the Intermediate Results Network Transfer Optimization improves the performance of the Shuffle phase a lot. Most data transmitted over network is MOFs, and after the networking levitation optimization most network flow of the Shuffle phase is eliminated.

VI. Conclusion

This paper aimed at exploring the possibility of building Massive Data Processing Paradigm on HPCs. The performance of MapReduce Paradigm on HPCs, especially the I/O capability of the dedicated storage subsystem specific to HPCs is analyzed. Two optimizations for storage localization and network levitation in HPC environment respectively improve the MapReduce performance by 32.5% and 16.9%. The conclusion is that when the corresponding I/O capability is the performance bottleneck of the overall system, these optimizations can help improve MapReduce paradigm under HPC architecture.

Acknowledgment

The authors are thankful to the anonymous reviewers. This work was supported by NSF China grant 61133005.

References

[1] J. Dean, and S. Ghemawat, “Mapreduce: Simplified Data Processing on Large Clusters,” In Proc. of OSDI, 2004, pp. 137-150.

[2] Hadoop, http://hadoop.apache.org/, retrieved: May, 2012.

[3] Y. Wang, X. Que, W. Yu, D. Goldenberg, and D. Sehgal, “Hadoop Acceleration through Network Levitated Merging,” In Proc. of Super Computing, 2011, pp. 57-66.

[4] W. Tantisiriroj, S. Son, S. Patil, S. Lang, G. Gibson, and R. Ross, “On the Duality of Data-Intensive File System Design: Reconciling HDFS and PVFS,” In Proc. of Super Computing, 2011, pp. 67-78.

[5] D. Jiang, B. Ooi, L. Shi, and S. Wu, “The Performance of Mapreduce: An In-Depth Study,” In Proc. of VLDB, 2010, pp. 472-483.

[6] M. Zaharia, A. Konwinski, A. Joseph, R. Katz, and I. Stoica, “Improving Mapreduce Performance in Heterogeneous Environments,” In Proc. of OSDI, 2008, pp. 29-42.

[7] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, Y. Lu, and B. Saha, “Reining in the Outliers in Map-Reduce Clusters using Mantri,” In Proc. of OSDI, 2010, pp. 265-278.

[8] Y. Chen, L. Keys, and R. Katz, “Towards Energy Efficient Hadoop,” In UC Berkeley Technical Report, number UCB/EECS-2009-109, 2009.

[9] W. Lang, and J. Patel, “Energy Management for MapReduce Clusters,” In Proc. of VLDB, 2010, pp. 129-139.

[10] IBTA, “InfiniBand Architecture Specification, Release 1.0,” http://www.infinibandta.org/specs/, retrieved: May, 2012.

[11] Lustre Parallel Filesystem, “The Lustre Storage Architecture,” http://www.lustre.org/, retrieved: May, 2012.