Performance Benefits of DataMPI: A Case Study with BigDataBench

Fan Liang[1,2](✉), Chen Feng[1,2], Xiaoyi Lu[3], and Zhiwei Xu[1]

[1] Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
{liangfan,fengchen,zxu}@ict.ac.cn
[2] University of Chinese Academy of Sciences, Beijing, China
[3] Department of Computer Science and Engineering,
The Ohio State University, Columbus, USA
luxi@cse.ohio-state.edu

Abstract. Apache Hadoop and Spark are gaining prominence in Big Data processing and analytics. Both of them are widely deployed in Internet companies. On the other hand, high-performance data analysis requirements are causing academical and industrial communities to adopt state-of-the-art technologies in HPC to solve Big Data problems. Recently, we have proposed a key-value pair based communication library, DataMPI, which is extending MPI to support Hadoop/Spark-like Big Data Computing jobs. In this paper, we use BigDataBench, a Big Data benchmark suite, to do comprehensive studies on performance and resource utilization characterizations of Hadoop, Spark and DataMPI. From our experiments, we observe that the job execution time of DataMPI has up to 57 % and 50 % speedups compared with those of Hadoop and Spark, respectively. Most of the benefits come from the high-efficiency communication mechanisms in DataMPI. We also notice that the resource (CPU, memory, disk and network I/O) utilizations of DataMPI are also more efficient than those of the other two frameworks.

Keywords: DataMPI · Hadoop · Spark · BigDataBench

1 Introduction

Data explosion is becoming an irresistible trend with the development of Internet, social network, e-commerce, etc. Over the last decade, there have been emerging a lot of systems and frameworks for Big Data, such as Hadoop [1], Dyrad [8], Yahoo! S4 [14] and so on. Apache Hadoop has become as the defacto standard for Big Data processing and analytics. Many clusters in the production environment already contain thousands of nodes to dedicatedly run Hadoop jobs everyday. Beyond the success of Hadoop, Spark [19] provides another feasible way to process large amount of data by introducing the in-memory computing techniques. Nowadays, both of them have attracted more and more attention from academical and industrial areas.

However, the performance of current commonly used Big Data systems is still in a sub-optimal level. Many studies [9,11,15,17] have been trying to adopt state-of-the-art technologies in the High Performance Computing (HPC) area to accelerate the performance of Big Data processing. As one example of these attempts, our previous work [12,13,18] shows the performance of Hadoop communication primitives still have huge performance improvement potentials, and Message Passing Interface (MPI), which is widely used in the field of HPC, can help to optimize communication performance of Hadoop. Furthermore, the key-value pair based communication library, DataMPI [3,12], has been proposed to efficiently execute Hadoop/Spark-like Big Data Computing jobs by extending MPI. Since the open-source nature of these systems, it will be very interesting for users to know the performance characteristics of the emerging Big Data systems by doing a systematical performance evaluation over different aspects.

In this paper, we use BigDataBench [16], one of the benchmark suites for Big Data Computing systems, to evaluate the performance of Hadoop, Spark and DataMPI. By tracing the resource utilization, we analyse the execution behavior of each system. Our contributions in this paper include

– We propose a seven-pronged approach to evaluate Big Data Computing systems, which can help researchers to understand the performance of those systems systematically.
– Evaluation results show DataMPI can achieve up to 57 % and 50 % speedups compared to Hadoop and Spark, respectively, for the high-efficiency communication mechanisms and lightweight software design.

The rest of this paper is organized as follows. Section 2 discusses background and related work. Section 3 states our experiments methodology. The evaluation results and analysis are given in Sect. 4. Section 5 concludes the paper.

2 Background and Related Work

2.1 Big Data Systems

MapReduce programming model is pivotal in Big Data Computing. Hadoop [1], one of the open-source implementations of MapReduce, is becoming the defacto standard. It has been widely used in various areas and applications, such as log analysis, machine learning, search engine, etc., and achieves success for its high scalability, built-in fault-tolerance and simplicity of programming. Spark [19], one of the emerging Big Data Computing systems, processes task-parallel jobs with in-memory techniques. It implements resilient distributed datasets (RDDs), the distributed memory abstraction which builds on the lineage concept, and performs efficiently in iterative algorithms of machine learning and interactive data mining. DataMPI [12] is a key-value pair based communication library which extends MPI for Hadoop/Spark-like Big Data Computing systems. It implements a bipartite communication model and leverages the state-of-the-art technologies of MPI in HPC to accelerate the execution performance of Big Data applications.

Those three Big Data systems are respectively typical for the different implementation technologies based on their particular execution models. This motivates us to comprehensively evaluate them with Big Data benchmarks. Besides, many other Big Data systems using HPC technologies have been emerging. The authors in [9,11,15] implement Hadoop-RDMA, which uses RDMA-capable (Remote Direct Memory Access) interconnects to enhance the design of Hadoop. Wang et al. [17] propose the network-levitated merge algorithm and implement Hadoop-A which overlaps data merge and reduce operations for Hadoop Reduce tasks.

2.2 Big Data Benchmarks

Researchers have proposed several benchmarks for evaluating Big Data Computing systems. MRBench [10] is designed for evaluating MapReduce frameworks using TPC-H workloads. HiBench [7] is designed for Hadoop-based systems based on micro-benchmarks, web search, machine learning and HDFS benchmarks. BigBench [6], an end-to-end benchmark proposal based on product retailer, is designed for parallel DBMS and MapReduce systems. BigDataBench [16] is a benchmark suite for different Big Data Computing systems, such as Hadoop, Spark, etc. It covers six typical application scenarios which include fundamental workloads and Internet service applications. BigDataBench also provides a data generator, Big Data Generator Suite (BDGS) which extracts the characteristics of real-world data, to create synthetic data sets.

As BigDataBench contains various workloads, and synthetic data generator, we choose it as our benchmark suite. According to the specification of BigDataBench, we use DataMPI to implement the benchmarks and evaluate the performance of Hadoop, Spark and DataMPI fairly.

3 Benchmarking Methodology

3.1 Chosen Workloads

We choose five typical workloads in BigDataBench as our benchmarks, which include three micro-benchmarks and two application benchmarks.

- **Micro-benchmarks** include *WordCount*, *Grep* and *Sort*, which are fundamental and widely used operations in broad analysis processes. *WordCount* counts the number of each word occurrence in a collection of documents. *Grep* searches strings conforming to a certain pattern in the input documents and counts the number of the occurrence of each matched string. *Sort* reads each record of the input files as a key-value pair and sorts the records based on the keys.
- **Application benchmarks** include *K-means* and *Naive Bayes*, which are typical applications in social network and e-commerce scenarios. *K-means* is a classical clustering algorithm in data mining which aims to partition the input objects to k clusters by calculating the nearest mean cluster of each

object belongs to. *Naive Bayes* is a probabilistic algorithm for classification. It is based on Bayes' theorem with strong independence assumption, which means the features of the model are independent with each other.

3.2 Evaluation Methodology

We record the execution time of each workload over one system, which reflects the system performance. To understand the runtime status, we monitor the systems with resource metrics which include:

- CPU utilization: it is recorded as a percentage of CPU usage and shows the time the total CPU has spent on running a workload. The CPU utilization will be recorded each second. We calculate the average CPU utilization during one workload execution.
- Network I/O throughput: it is defined as the average amount of data transmitted (send/receive) per second over the network.
- Disk I/O throughput: it is defined as the average amount of data transmitted (read/write) per second through the hard disks.
- Memory footprint: it refers to the memory used when running a workload. The memory footprint will dynamically change when system allocates and releases memory during workload execution. We calculate the average memory footprint during one workload execution to compare memory utilization.

To evaluate Hadoop, Spark and DataMPI, we follow a seven-pronged approach as shown in Fig. 1. To show the performance, we calculate the average execution time of each kind of workloads over each system. The small job is based on the micro-benchmarks, while data size of each workload is 128 MB. The data sizes of normal micro-benchmarks and application benchmarks vary

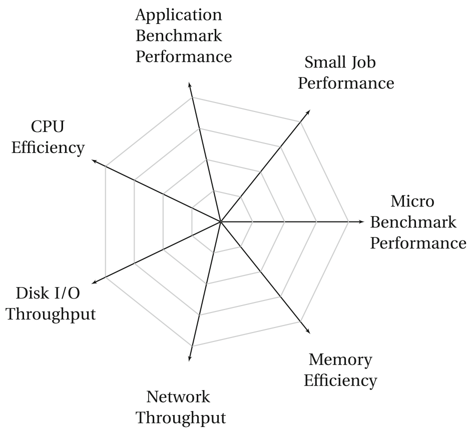


Fig. 1. Evaluation methodology

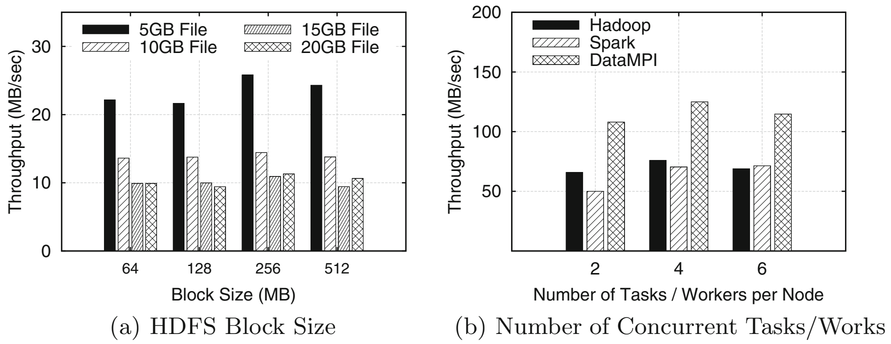


Fig. 2. Parameter tuning

from 8 GB to 64 GB. The CPU efficiency is defined as the CPU usage for the workload. The memory efficiency reflects the average amount of memory allocated to the system. The less memory is used, the better memory efficiency is achieved. We summarize the results with these seven dimensions in Sect. 4.7. To better understand the results, we use the execution time and resource utilization of Hadoop as the baseline and normalize the corresponding values of Spark and DataMPI.

4 Experimental Evaluation

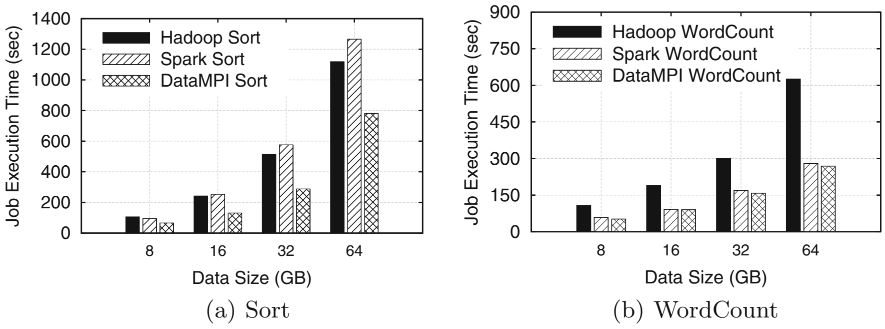
4.1 Experiment Setup

We use a cluster composed of 8 nodes interconnected by a 1 Gigabit Ethernet switch as our testbed. Each node is equipped with two Intel Xeon E5620 CPU processors (2.4 GHz) with disabling the hyper-threads. Each processor has four physical cores. Each node has 16 GB DDR3 RAM with 1333 MHz and one 150 GB free space SATA disk.

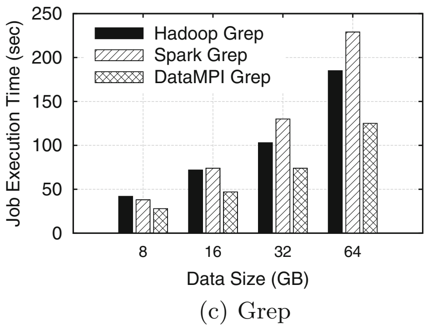
The operation system used is CentOS release 6.5 (Final) with kernel version 2.6.32-431.el6.x86_64. The software stack is comprised with JDK 1.7.0_25, MVAPICH2-2.0b, Scala 2.9.3, BigDataBench 2.1, Hadoop 1.2.1, Mahout 0.8 [2], Spark 0.8.1 and DataMPI 0.6. For all evaluations, we report results that are averaged across three executions.

4.2 Chosen Parameters

Hadoop, Spark and DataMPI have abundant parameters to set to achieve better performance. In this section, we tune the parameters for fair evaluations. We mainly focus on the HDFS block size and the number of tasks or workers, because the disk and network will easily become the bottlenecks in our testbed, and the concurrent execution instances have a great influence on performance.



(a) Sort (b) WordCount



(c) Grep

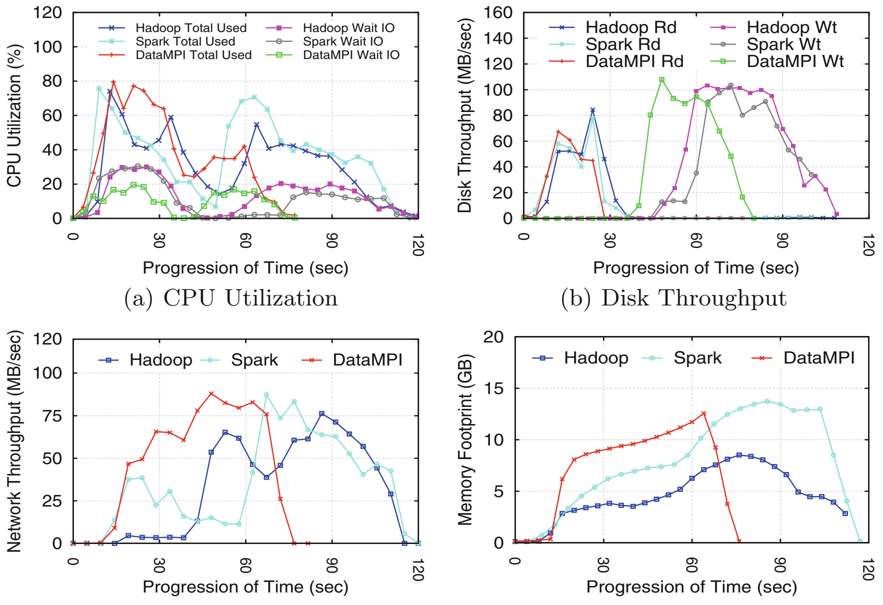
Fig. 3. Performance comparison of different micro-benchmarks

We use DFSIO program, a file system level benchmark of Hadoop, as the workload for tuning HDFS block size. We measure the throughput by varying the HDFS block size and input data size. Figure 2(a) shows the throughput achieves the best, when block size is 256 MB. When tuning the number of concurrent tasks or workers, we execute Sort benchmark and measure the throughput by processing 1 GB data per Hadoop/DataMPI task and Spark worker with increasing the number of concurrent tasks or workers from 2 to 6 per node. Figure 2(b) shows the systems can get the best throughput when the number of concurrent tasks or workers on each node is 4.

Based on the two tests, we run our following evaluations based on 256 MB HDFS block size and 4 concurrent tasks or workers per node. The replication of each block in HDFS is set to three for the high data availability and flexible data locality.

4.3 Micro-benchmark Performance

In this section, we evaluate the performance of micro-benchmarks among Hadoop, Spark and DataMPI. We use BDGS in BigDataBench to produce the data sets and upload them on the HDFS with the uncompressed text format. The seed model used in BDGS is *lda_wiki1w* which is trained from wikipedia entries corpus.



(a) CPU Utilization (b) Disk Throughput

(c) Network Throughput (d) Memory Footprint

Fig. 4. Resource utilization of Sort benchmark with 8 GB data

We vary the input data size from 8 GB to 64 GB. Figure 3(a) shows DataMPI has 30 %–44 % (averagely 39 %) improvement compared to Hadoop and 38 %–50 % (averagely 44 %) improvement compared to Spark, when running Sort. Figure 3(b) shows DataMPI and Spark have similar performance and achieve 47 %–57 % (averagely 52 %) performance improvement compared to Hadoop, when running WordCount. The Grep evaluation results in Fig. 3(c) show that DataMPI cuts down the execution time by 29 %–34 % (averagely 32 %) compared to Hadoop, and by 26 %–45 % (averagely 38 %) compared to Spark.

DataMPI can achieve the best performance for those benchmarks, while Spark is not performing better than Hadoop in Sort and Grep cases, which means for batch jobs, Hadoop is still relatively good.

4.4 Profile of Resource Utilization

We profile the resource utilization of Hadoop, Spark and DataMPI based on the workloads of 8 GB Sort and 32 GB WordCount from four aspects, i.e. CPU utilization, disk throughput, network throughput and memory footprint. We record the total CPU usage percentage and the CPU wait I/O percentage. A higher CPU wait I/O percentage means CPU costs more time to wait for I/O operations to complete. For the sake of page limitation, we only show the figures of Sort benchmark with 8 GB case.

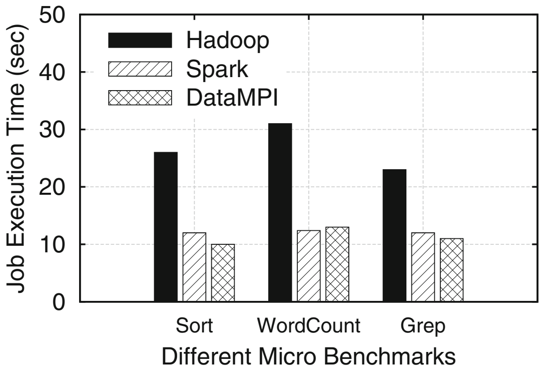


Fig. 5. Performance comparison based on small jobs

For the Sort benchmark, DataMPI costs 69 s while Hadoop and Spark cost 117 s and 114 s, respectively. Hadoop has Map/Reduce phases, DataMPI has O/A phases, and Spark has map phase (Stage0), sort-by-key phase (Stage1) and save phase (Stage2). The Map phase of Hadoop costs 36 s, the Stage 0 of Spark costs 38 s, and the O phase of DataMPI costs 28 s. As shown in Fig. 4(a), the average CPU utilizations of Hadoop, Spark and DataMPI are 37 %, 38 % and 45 %. The average CPU wait I/O percentages of Hadoop, Spark and DataMPI are 15 %, 12 % and 10 %. This means Hadoop costs more time to wait I/O operations. Figure 4(b) shows the disk throughput. The average disk read throughputs of Hadoop Map phase, Spark Stage 0 and DataMPI O phase are 49 MB/s, 46 MB/s and 50 MB/s. The average disk write throughputs of Hadoop Shuffle-Reduce phase, Spark Stage 2 and DataMPI A phase are 67 MB/s, 66 MB/s and 69 MB/s. Figure 4(c) shows the network throughput of DataMPI is averagely 62 MB/s, which is 59 % higher than that of Hadoop (39 MB/s) and 55 % higher than that of Spark (40 MB/s). This means MPI-based communication mechanism can use network resource more efficiently. Figure 4(d) shows the average memory footprints of Hadoop, Spark and DataMPI are 5 GB, 9 GB and 8 GB.

When running WordCount, Spark and DataMPI cost 169 s and 158 s, and have 47 %, 43 % speedups compared to Hadoop (301 s), respectively. The CPU utilizations of Hadoop, Spark and DataMPI are 82 %, 58 % and 84 %, respectively. The average read throughput of DataMPI is 44 MB/s, which is approximately equal to that of Spark and is higher than that of Hadoop (20 MB/s). We observe that DataMPI and Hadoop have few network transmissions, while Spark transmits more intermediate data for the RDDs creation. The average memory footprints of Hadoop, Spark and DataMPI are 8 GB, 6 GB, and 5 GB.

From the above two cases, we observe that DataMPI can leverage the resources to run jobs more effectively than Hadoop and Spark for the high efficient data communication and computation.

4.5 Small Jobs

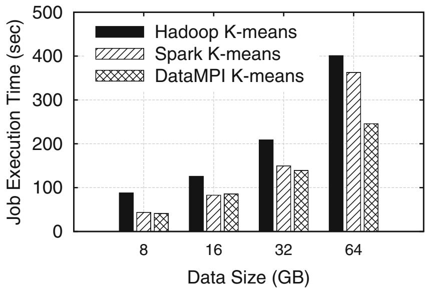
According to the recent study [5], more than 90 % of MapReduce jobs in Facebook and Yahoo! are small jobs, which means the input data sizes of the jobs are usually kilo or mega bytes. The system overheads of the initialization and the finalization have serious impacts on the performance. In this section, we compare the performance of Hadoop, Spark and DataMPI when they run the micro-benchmarks of Sort, WordCount and Grep with smaller input data sets. The input data size of each workload is 128 MB. The number of the concurrent tasks or works is set to one per node. Figure 5 shows that DataMPI has similar performance with Spark, and performs averagely 54 % more efficiently than Hadoop. The benefits of Spark and DataMPI are contributed by the lightweight software designs.

4.6 Application Benchmark Performance

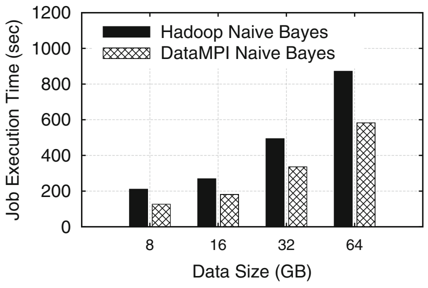
In this section, we present the results of the application evaluations. The Hadoop implementations of K-means and Naive Bayes in BigDataBench are based on Mahout, while the Spark implementation of K-means is based on Spark MLlib [4]. We implement K-means and Naive Bayes over DataMPI according to the BigDataBench specification. Because BigDataBench 2.1 lacks the implementation of Naive Bayes in Spark, we only compare the performance of this benchmark between DataMPI and Hadoop. We first explain the processing characteristics of the applications from the implementation-level and then give the performance results.

K-means: We use BDGS to generate the input data sets based on five seed models, amazon1-amazon5. Using *genData_Kmeans* of BDGS, text files are converted to sequence files from directories, then transformed to the sparse vectors as the input data of training clusters. Our evaluations are based on the sparse vectors and mainly focus on the performance of training execution. As stated in Sect. 3.1, K-means trains the cluster centroids iteratively. Each iteration of K-means is a MapReduce job. In one job, Map tasks read the initial or previous cluster centroids from HDFS, afterwards, assign the input vectors to appropriate clusters according to the distance calculation and train the new centroids independently. At the end of Map tasks, new centroids will be sent to the Reduce tasks according to the cluster indexes. Reduce tasks receive and update the centroids for the next iterative execution. We observe that most of K-means calculation happens in Map phase, and few intermediate data is generated.

Our tests show Spark has outstanding performance when running the iterative computations based on the RDDs. Since Hadoop is not designed for iterative jobs, for fair comparison, we record the execution time of the first iteration from the job start, which considering the overheads of loading data, computation and communication, and outputing results. Figure 6(a) shows that DataMPI has at most 39 % improvement than Hadoop and 33 % improvement than Spark when the input data size varies from 8 GB to 64 GB.



(a) K-means



(b) Naive Bayes

Fig. 6. Performance comparison of application benchmarks

Naive Bayes: The input document data sets are generated by BDGS, and are classified into five categories. The procedure of Naive Bayes mainly contains two steps, including converting sequence files to sparse vectors and training the Naive Bayes model. Mahout runs several MapReduce jobs to create the sparse vectors. Firstly, one document is converted to a token array. After that, some MapReduce jobs are launched to count the term frequency in one document and document frequency of all terms. The sparse vector of one document is calculated according to the term frequency and document frequency. The main operation in above steps is counting, including term counting and document counting, which means that the behavior of Naive Bayes is similar to WordCount. In our evaluation cases, the data sizes of sparse vector and term-counting dictionary are within several mega bytes. The model training contains two MapReduce jobs to execute the probabilistic computations. The two jobs cost less time than the sparse vectors generation because of the simple calculating operations and small input data sizes. Figure 6(b) shows DataMPI has 33 % improvement than Hadoop averagely.

4.7 Discussion of Performance Results

We summarize the performance comparisons with different benchmarks using seven-pronged diagram, depicted in Fig. 7. We normalize the values of Spark and DataMPI according to the corresponding Hadoop values. Besides, we only take the K-means results to calculate the values of the application benchmarks. Compared to Hadoop, DataMPI can averagely achieve 41 %, 54 % and 38 % performance improvements when running micro-benchmarks, small jobs and application benchmarks, respectively, while Spark has 10 %, 54 % and 31 % performance improvements, respectively. From the Sort and WordCount cases, the average CPU utilizations of Hadoop, Spark and DataMPI are 60 %, 48 % and 64 %, which means DataMPI has similar CPU efficiency with Hadoop, and leverages the CPU resource 33 % more efficiently than Spark. The average disk I/O throughputs of Spark and DataMPI are 15 %, 20 % higher than that of Hadoop, respectively. DataMPI achieves 56 % and 55 % network throughput

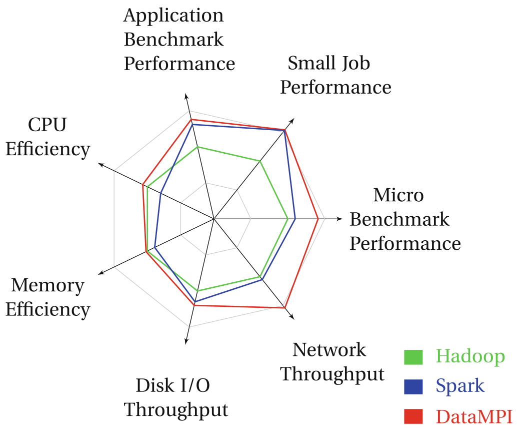


Fig. 7. Evaluation results

improvements than those of Spark and Hadoop, respectively. The average memory footprints of Hadoop, Spark and DataMPI are 6.5 GB, 7.5 GB and 6.5 GB, which means Hadoop and DataMPI can efficiently utilize memory when running the workloads. The benefits of DataMPI come from the lightweight software design and the high performance communication design which is able to leverage system resources to pipeline the computation and communication operations efficiently [12].

5 Conclusion

In this paper, we provide a systematical performance evaluation of Hadoop, Spark and DataMPI based on BigDataBench. We choose three micro benchmarks (Sort, WordCount and Grep) and two application benchmarks (K-means and Naive Bayes) as our testing experimental workloads. Based on the Sort, WordCount benchmark cases, we present a detailed resource utilization analysis of the three systems. Our evaluation shows that with the mirco-benchmarks, DataMPI can achieve 29 %–57 % performance improvement compared to Hadoop, and up to 50 % performance improvement compared to Spark. The small job evalutions show the low overheads of DataMPI and Spark make them gain 54 % performance improvement compared to Hadoop. Evaluations of K-means and Naive Bayes benchmarks show DataMPI can achieve 33 %–39 % application-level performance improvement compared to Hadoop and Spark.

Acknowledgments. This work is supported in part by the Strategic Priority Program of Chinese Academy of Sciences (Grant No. XDA06010401). We thank the anonymous referees for their comments. We also thank Prof. Li Zha for his help to support this research.

References

1. Apache Hadoop Project. http://hadoop.apache.org
2. Apache Mahout Project. https://mahout.apache.org
3. DataMPI Project. http://datampi.org
4. MLlib Project. https://spark.apache.org/mllib
5. Chen, Y., Ganapathi, A., Griffith, R., Katz, R.: The case for evaluating MapReduce performance using workload suites. In: Proceedings of the 19th International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS '11), Singapore (2011)
6. Ghazal, A., Rabl, T., Hu, M., Raab, F., Poess, M., Crolotte, A., Jacobsen, H.A.: BigBench: towards an industry standard benchmark for big data analytics. In: Proceedings of the 32nd ACM SIGMOD International Conference on Management of Data (SIGMOD '13), New York, NY, USA (2013)
7. Huang, S., Huang, J., Dai, J., Xie, T., Huang, B.: The HiBench benchmark suite: characterization of the MapReduce-based data analysis. In: Proceedings of the 26th International Conference on Data Engineering Workshops (ICDEW '10), Long Beach, CA, USA (2010)
8. Isard, M., Budiu, M., Yu, Y., Birrell, A., Fetterly, D.: Dryad: distributed data-parallel programs from sequential building blocks. ACM SIGOPS Oper. Syst. Rev. **41**(3), 59–72 (2007)
9. Islam, N.S., Rahman, M.W., Jose, J., Rajachandrasekar, R., Wang, H., Subramoni, H., Murthy, C., Panda, D.K.: High performance RDMA-based design of HDFS over InfiniBand. In: Proceedings of the 25th International Conference on High Performance Computing, Networking, Storage and Analysis (SC '12), Salt Lake City, UT, USA (2012)
10. Kim, K., Jeon, K., Han, H., Kim, S., Jung, H., Yeom, H.: MRBench: a benchmark for MapReduce framework. In: Proceedings of the 14th International Conference on Parallel and Distributed Systems (ICPADS '08), Melbourne, Victoria, Australia (2008)
11. Lu, X., Islam, N.S., Wasi-ur-Rahman, M., Jose, J., Subramoni, H., Wang, H., Panda, D.K.: High-performance design of Hadoop RPC with RDMA over InfiniBand. In: Proceedings of the 42nd International Conference on Parallel Processing (ICPP '13), Lyon, France (2013)
12. Lu, X., Liang, F., Wang, B., Zha, L., Xu, Z.: DataMPI: extending MPI to Hadoop-like big data computing. In: Proceedings of the 28th International Parallel and Distributed Processing Symposium (IPDPS '14), Phoenix, AZ, USA (2014)
13. Lu, X., Wang, B., Zha, L., Xu, Z.: Can MPI benefit Hadoop and MapReduce applications? In: Proceedings of the 40th International Conference on Parallel Processing Workshops (ICPPW '11), Taipei, China (2011)
14. Neumeyer, L., Robbins, B., Nair, A., Kesari, A.: S4: distributed stream computing platform. In: Proceedings of the 10th IEEE International Conference on Data Mining Workshops (ICDMW '10), Sydney, Australia (2010)
15. Wasi-ur Rahman, M., Islam, N., Lu, X., Jose, J., Subramoni, H., Wang, H., Panda, D.: High-performance RDMA-based design of Hadoop MapReduce over InfiniBand. In: Proceedings of the 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW '13), Cambridge, MA, USA (2013)

16. Wang, L., Zhan, J., Luo, C., Zhu, Y., Yang, Q., He, Y., Gao, W., Jia, Z., Shi, Y., Zhang, S., Zheng, C., Lu, G., Zhan, K., Li, X., Qiu, B.: BigDataBench: a big data benchmark suite from internet services. In: Proceedings of the 20th IEEE International Symposium on High Performance Computer Architecture (HPCA '14), Orlando, FL, USA (2014)
17. Wang, Y., Que, X., Yu, W., Goldenberg, D., Sehgal, D.: Hadoop acceleration through network levitated merge. In: Proceedings of the 24th International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11), New York, NY, USA (2011)
18. Xu, Z.: High-performance techniques for big data computing in internet services. In: Proceeding of the 2012 SC Companion: High Performance Computing (SC '12), Salt Lake City, UT, USA (2012)
19. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M., Shenker, S., Stoica, I.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI '12), San Jose, CA, USA (2012)