Measuring Scale-up and Scale-out Hadoop with Remote and Local File Systems and Selecting the Best Platform

Zhuozhao Li, *Student Member* and Haiying Shen, *Senior Member, IEEE,*

Abstract—MapReduce is a popular computing model for parallel data processing on large-scale datasets, which can vary from gigabytes to terabytes and petabytes. Though Hadoop MapReduce normally uses Hadoop Distributed File System (HDFS) local file system, it can be configured to use a remote file system. Then, an interesting question is raised: for a given application, which is the best running platform among the different combinations of scale-up and scale-out Hadoop with remote and local file systems. However, there has been no previous research on how different types of applications (e.g., CPU-intensive, data-intensive) with different characteristics (e.g., input data size) can benefit from the different platforms. Thus, in this paper, we conduct a comprehensive performance measurement of different applications on scale-up and scale-out clusters configured with HDFS and a remote file system (i.e., OFS), respectively. We identify and study how different job characteristics (e.g., input data size, the number of file reads/writes, and the amount of computations) affect the performance of different applications on the different platforms. Based on the measurement results, we also propose a performance prediction model to help users select the best platforms that lead to the minimum latency. Our evaluation using a Facebook workload trace demonstrates the effectiveness of our prediction model. This study is expected to provide a guidance for users to choose the best platform to run different applications with different characteristics in the environment that provides both remote and local storage, such as HPC cluster and cloud environment.

Index Terms—MapReduce, Hadoop, scale-up, scale-out, remote file system, local file system, job characteristics

1 Introduction

MapReduce [19] is a framework designed to process a large amount of data in the parallel and distributed manner on a cluster of computing nodes. Hadoop, as a popular open source implementation of MapReduce, has been deployed in many large companies such as Yahoo! [18] and Facebook [45]. Also, many high-performance computing (HPC) sites [1] extended their clusters to support Hadoop MapReduce. HPC differs from Hadoop on the configuration of file systems. In Hadoop Distributed File System (HDFS), data is stored in the compute nodes, while in HPC, data is usually stored on remote storage servers. The Clemson Palmetto HPC cluster successfully configured Hadoop by replacing the local HDFS with the remote Orange File System (OFS) [1], as shown in Figures 1 and 2.

In the last decade, the volumes of computation and data have increased exponentially [12], [40]. Real-world applications may process data size up to the gigabytes, terabytes, petabytes, or exabytes level. This trend poses a formidable challenge of providing high performance on MapReduce and motivates many researchers to explore to improve the performance. While scale-out is a normal method to improve the processing capability of a Hadoop cluster, scale-up appears as a better alternative for a certain workload with a median data size (e.g., MB and GB) [14], [30], [33]. Scale-up is vertical scaling, which refers to adding more resources (typically processors and RAM) to the nodes in a system. Scale-out is horizontal scaling, which refers to adding more nodes with few processors and RAM to a system.

Considering the different combinations of scale-up and scale-out Hadoop with a remote file system (OFS) and a local file system (HDFS), we can create four platforms as shown in Table 1: scale-up cluster with OFS (denoted as up-OFS), scale-up cluster with HDFS (denoted as up-HDFS), scale-out cluster with OFS (denoted as out-OFS), and scale-out cluster with HDFS (denoted as out-HDFS). Then, an interesting question is raised: for a given application, which is the best running platform.

To answer this question, it is important to understand the performance of different types of applications (e.g., data-intensive, CPU-intensive, and I/O-intensive) with different characteristics (e.g., input data size, the number of file reads/writes, and the amount of computations) on these four platforms, since a big data workload generally consists of different types of jobs, with input data size ranging from KB to PB [18], [43]. However, there have been no previous works that conduct such a thorough analysis. CPU-intensive applications include a large amount of computations and devote most of the time on computing. Data-intensive and I/O-intensive applications have large input data size and require large amount of data read/write operations. Data-intensive applications contain certain amount of computations such as counting, while I/O-intensive applications do not or have only few computations. Different characteristics of applications may lead to different performance and gain different benefits in the scale-up and scale-out systems. For example, data-intensive applications have large input and shuffle data size and may benefit more from a large size of memory and hence from the scale-up machines.

In this paper, we have conducted comprehensive experiments

- *Haiying Shen is the corresponding author. Email: hs6ms@virginia.edu; Phone: (434) 924-8271, Fax: (434) 982-2214.*
- *Z. Li and H. Shen are with the Department of Computer Science, University of Virginia, Charlottesville, VA 22904. E-mail: {zl5uq, hs6ms}@virginia.edu*

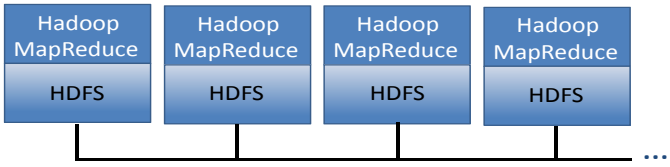


Fig. 1. Typical Hadoop with HDFS local storage (HDFS in short).

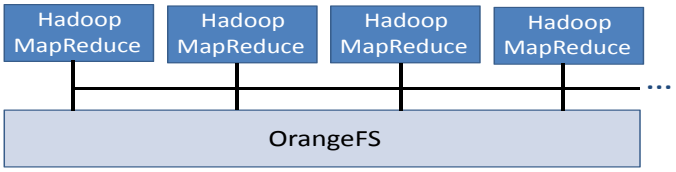


Fig. 2. Hadoop with the OrangeFS remote storage (OFS in short).

TABLE 1
Different platforms.

	Scale-up	Scale-out
OFS	up-OFS	out-OFS
HDFS	up-HDFS	out-HDFS

for different types of applications (including data-intensive, CPU-intensive, and I/O-intensive applications) on the four platforms with different input data sizes and provide an insightful analysis on their performance. We also have analyzed how different application characteristics affect the application performance and system overheads on the four platforms and determine the best platform for an application with certain characteristics. Our measurement results provide a guidance on how to select the best platform to run different types of applications with different characteristics.

The contributions of our paper are as follows:

1. We have conducted thorough experiments for different types of applications (including data-intensive, CPU-intensive and I/O-intensive) on the four platforms. We have analyzed how different application features (e.g., input data size, the number of reading/writing files and the amount of computations) affect the application performance on the four platforms and determine the best platform for an application with certain features. We confirm that replacing HDFS with OFS for Hadoop is feasible when data size is relatively large.
2. Our measurement results provide a guidance on how to select the best platform that leads to minimum latency to run different type of jobs with different job characteristics.
3. Based on the measurement results, we also propose a performance prediction model to help users select the best platforms. Our evaluation using a Facebook workload trace demonstrates the effectiveness of our prediction model.

The remainder of this paper is organized as follows. Section 2 describes the configurations of scale-up and scale-out machines for Hadoop with OFS and HDFS on a HPC-based cluster. Section 3 presents the measurement results of performance for different types of Hadoop applications and provides an in-depth analysis of the results. Section 4 summarizes the observations and further discusses the guidance to cloud environment. Section 5 presents a performance prediction model to help users select the best platforms and evaluates the prediction accuracy of our prediction model. Section 6 gives an overview of the related work. Section 7 concludes this paper with remarks on our future work.

2 Configurations on HPC-Based Hadoop

In this section, we introduce the details on how to configure Hadoop MapReduce on a HPC cluster. We do our experiments on HPC cluster because HPC clusters generally have machines with different CPU and memory, which allows us to deploy scale-up and scale-out machines easily without any further cost. In our experimental measurement, we use Clemson Palmetto HPC cluster, which ranks the top five fastest supercomputers at public universities in United States and the 66^{th} fastest supercomputers globally [5].

2.1 Introduction of Hadoop MapReduce

MapReduce [19] is a scalable and parallel processing framework to handle large datasets. HDFS is a highly fault tolerant and self-healing distributed file system to cooperate with Hadoop MapReduce. HDFS has a master/slave architecture, which generally consists of a namenode and multiple datanodes. Namenode manages the metadata of the cluster and provides the access to files to clients, while datanodes are used to store the data blocks. HDFS stores the input data of each job into several blocks. The number of blocks is calculated by $\frac{input\ data\ size}{block\ size}$. In a MapReduce job, there are generally three phases: map, shuffle and reduce. In the map phase, the job tracker assigns each mapper to process one data block. Note that the data block may locate at the same nodes with the mapper, which is called data locality. Hadoop MapReduce prefers high data locality to reduce network consumption for data movement to improve performance. All the mappers generate the output, called intermediate data (i.e., shuffle data). In the shuffle phase, each mapper's output is then partitioned and sorted. Different partitions are shuffled to corresponding reducers. Once the reducers are scheduled on specific nodes by the job tracker, the shuffle data is copied to the reduce nodes' memory first. If the shuffle data size is larger than the size of in-memory buffer, the shuffle data will be spilled to local disks, which results in extra overheads. In the reduce phase, the reducers aggregate the shuffle data and produce the final output of the jobs.

2.2 Experiment Environment

In the experiments, we use the Hadoop MapReduce version 1.2.1. We use four machines for scale-up Hadoop. Each scale-up machine is equipped with four 6-core 2.66GHZ Intel Xeon 7542 processors, 505GB RAM, and 91GB hard disk and 10Gbps Myrinet interconnections. To achieve fair performance comparison, we require the scale-up and scale-out machines have similar cost. We investigated the cost information from [9] and found that one scale-up machine matches similar price with 6 scale-out machines. Therefore, the scale-out cluster consists of twenty-four machines, each of which has two 4-core 2.3GHZ AMD Opteron 2356 processors, 16GB RAM, and 193GB hard disk and 10Gbps Myrinet interconnections. Note that Myrinet is a high-speed local area networking system. It has much lower protocol overheads than Ethernet and hence can provide better throughput. With Myrinet, the data can be accessed with less latency and the communication overheads between each node are reduced.

2.3 Configurations on HDFS and OFS

As we mentioned in Section 1, while traditional Hadoop is deployed with the distributed local file system HDFS, conventional HPC architecture relies on the remote file system. On HPC cluster, compute and data are separated and connected with high speed interconnects, such as Ethernet and Myrinet. However, we can still deploy Hadoop MapReduce framework with HDFS on HPC cluster. Under the help of myHadoop [28], we easily configure Hadoop with HDFS on the HPC cluster in our university.

Recently, in order to achieve better performance, a Java Native Interface (JNI) shim layer has been successfully implemented on the HPC cluster in our university, which allows Hadoop to work directly with remote file system OFS. Both the input and output data can be stored in the remote file system, while the shuffle data is still required to store in local file system of each node. OFS is a parallel file system (PVFS) that distributes data across multiple servers. Moreover, OFS is demonstrated to be able to offer much better I/O performance [1] than HDFS on processing large amount of data.

In order to achieve fair comparisons between the remote file system and the local file system, a couple of parameters are required to be set consistently in OFS and HDFS. In HDFS, we set the HDFS block size to 128MB to match the setting in the current industry clusters [45]. Similarly, OFS stores data in simple stripes (i.e., similar as blocks in HDFS) across multiple storage servers in order to facilitate parallel access. In order to compare OFS fairly with HDFS, we also set the stripe size to 128MB. Typically, in current commercial MapReduce cluster [14], the total number of map and reduce slots is set to the number of cores. Therefore, in our experiments, each scale-up machine has 24 map and reduce slots, while each scale-out machine has 8 map and reduce slots in total.

For HDFS, the replication factor is set to 3 by default, which means that each file block has three replicas. For OFS, it currently does not support build-in replications. However, it does not affect our measurement results since data loss never occurs in OFS during our experiments.

2.4 Configurations for Best Performance

The scale-out architecture deploys many scale-out machines with less powerful CPU and small RAM size. On the other hand, the scale-up architecture has a few machines with high performance CPU and large RAM size. In order to fully utilize the CPU and RAM size advantages of scale-up machines, several parameters of the scale-up Hadoop clusters are configured differently from the conventional Hadoop clusters.

Heap size In Hadoop, each map and reduce task runs in a JVM. The heap size is the memory allocated to each JVM for buffering data. The map outputs are written to a circular buffer in memory, which is determined by the heap size [14]. When the circular buffer is closed to full, the data is spilled to the local disk, which introduces overheads. Therefore, by increasing the heap size, it is less likely for the data to be spilled to local disk if the heap size is larger, leading to better performance in the shuffle phase.

The heap size is 200MB for each JVM by default in Hadoop. In the experiments, the machines for scale-up and scale-out machines allow us to set the heap size to a much larger value than 200MB. We tune the heap size through trial and error on both scale-up and scale-out machines. To achieve the best performance and also avoid the out of memory error [14], we set the heap size to 8GB per task on scale-up machines, and to 1.5GB on scale-out machines, respectively, through trial and error.

RAM drive to place shuffle data After setting the heap size to 8GB, we find that there is still much memory left (more than 300GB) on scale-up machines. In Hadoop, the shuffle data of the jobs is required to store on local file system. On the HPC cluster in our university, it enables us to use half of the total memory size as *tmpfs*, which serves the same functions as RAMdisk. Therefore, we use half of the RAM (253GB) as RAMdisk to place the shuffle data on scale-up machines. If the shuffle data size is larger than the available RAMdisk size, the rest of the shuffle data is stored on the local disks. On the other hand, since the memory size is not large on the scale-out machines (i.e., 16GB), the shuffle data is placed on the local disks only.

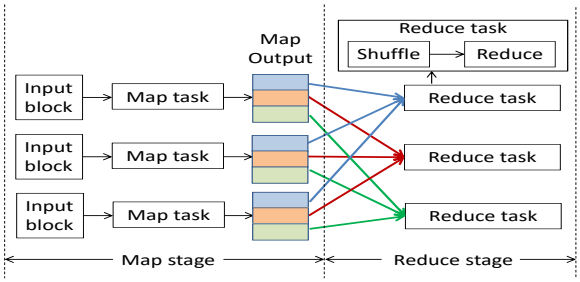


Fig. 3. Map, shuffle and reduce phases in MapReduce [19].

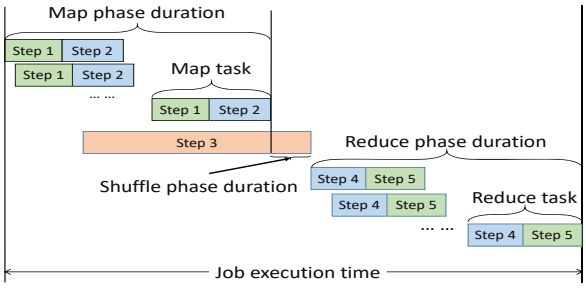


Fig. 4. Timeline of each step in a MapReduce job.

Other configurations Besides the key configurations above, there are also some other Hadoop configuration parameters (e.g., io.sort.mb and io.sort.spill.percent) that have significantly impact on the performance of the applications. To select the best platforms for applications, we need to compare the best performance of each platform in the measurement study. Hence, we assume that users can leverage the tools in several previous studies such as Starfish [22] and iTuned [20], which help configure Hadoop clusters to achieve the best performance for different applications on different platforms. In the measurement study in Section 3, for each application, we tune the Hadoop configuration parameters on different platforms to achieve the best performance, based on the instructions in Starfish [22]. As a result, we can focus on how job characteristics of the applications affect the performance on different platforms.

Next, let us discuss the details in a Hadoop MapReduce job. Generally, a MapReduce job consists of map stage and reduce stage, as shown in Figure 3. However, many researchers actually consider that the MapReduce job has three phases by further splitting the reduce stage to shuffle phase and reduce phase. In this paper, we consider that there are three phases in a MapReduce job – map phase, shuffle phase and reduce phase.

2.5 Factors of Job Execution Time in Hadoop MapRuduce

According to [44], in the following, we break down the Hadoop MapReduce execution flow and analyze the factors of each step in a job, as shown in Figure 4.

• **Factors for the Time Duration of Map Task**

Step 1. The map tasks of a job need to read the input data for the MapReduce execution. The time duration of this data reading process depends on two main factors. The first one is the amount of input data that a map task needs to process. The second is the I/O speed of the file system.

Step 2. Each map task processes one input data block and generates the intermediate $< key, value >$ pairs (called map output

data or shuffle data or reduce input data), according to the user-defined map function. The time duration of this step depends on two main factors. The first one is the amount of data each map task needs to process. The second one is the speed of each node to process the data.

For the second factor, as each node processes different jobs at a different speed, it is difficult to generalize a speed model for every job. However, since the map task's function is to process the input data and generate intermediate data, we can approximately convert the second factor to the map output data size, i.e., shuffle data size.

• **Factors for the Time Duration of Shuffle Task**

Step 3. In Hadoop, only when a certain portion (called map completion threshold) of map tasks for a job have completed, the reduce tasks of this job are allowed to scheduled. Only after a reduce task is scheduled, the shuffle task starts. The map outputs are first written to the memory buffer, and then spilled to the disk. The map outputs are partitioned corresponding to the number of reduce tasks. All the partitions are then transmitted to the corresponding nodes that run the reduce tasks. The time duration of this step depends on three main factors. The first one is the amount of data each node needs to process. The second one is the speed of the network to transmit the data. The third one is the buffer size.

• **Factors for the Time Duration of Reduce Task**

Step 4. Note that the reduce tasks begin only after the shuffle phase completes. Each reduce task takes the transmitted data from the shuffle phase and processes the shuffle data according to the user-defined reduce function. As all the reduce tasks in the reduce phase are run by the worker nodes in parallel, the time duration of this step depends on two main factors. The first one is the amount of data each reduce task needs to process. The second one is the speed of each node to process the data. Similar to Step 2, we can actually convert the second factor to the generated output data size.

Step 5. The output data of the reduce tasks is written to the file system. The time duration of this step depends on two main factors. The first one is the amount of output data that a reduce task generates. The second one is the I/O speed of the file system.

3 Performance Measurements

In this section, we will compare the performance of data-intensive, CPU-intensive, and I/O-intensive jobs with different input data sizes on the four platforms as mentioned previously. The four configurations include scale-up machines with OrangeFS , scale-up machines with HDFS, scale-out machines with OrangeFS, and scale-out machines with HDFS, denoted by up-OFS, up-HDFS, out-OFS, and out-HDFS, respectively. We expect to provide a guidance for users on how different applications benefit from different platforms.

3.1 Measured Applications and Metrics

We classify the representative Hadoop benchmarks into three types: data-intensive, I/O-intensive and CPU-intensive in our performance measurement. We can roughly infer the types of applications by the size of the input data, shuffle data and output data. In general, data-intensive applications have large input and shuffle data sizes and devote much processing time to I/O requests, while I/O-intensive applications generally conduct only read/write operations on the file system. CPU-intensive applications include a large amount of computations such as iterative computations. The representative Hadoop applications we measure in this section include *Wordcount*, *Grep*, *Terasort*, table cross join [10], write and read test of *TestDFSIO*, *PiEstimator*, and matrix multiplication [4].

Among them, *Wordcount*, *Grep*, *Terasort* and *TCJ* are typical data-intensive applications since they need to read/write and process a large amount of data. *Wordcount* and *Grep* have relatively large input and shuffle sizes but small output size, while *Terasort* generally has relatively large input, shuffle and output sizes. We generated the input data for *Wordcount*, *Grep*, *Terasort* from a big data benchamrk BigDataBench [43], which is based on the Wikipedia datasets.

Table cross join (*TCJ*) is an application to cross join two tables. It is developed in Apache PIG, which is a high-level platform used with Hadoop. The application joins and sorts all the same key-value pairs in two tables to a much larger table. The mappers complete most of the cross join jobs since the mappers need to list and sort out all the key-value pairs in the two tables. The reducers aggregate the same key-value pairs in the map output and generate the final output.

The write and read test of *TestDFSIO* are typical I/O-intensive applications. They complete a large number of read/write operations during the map tasks and only do some calculations like calculating the I/O rate in the reduce tasks. In *TestDFSIO*, each mapper reads/writes one file. It allows us to set the number of mappers (i.e., the number of files) and the read/write size of file, regardless of the block size. For example, if we want to read/write 80GB data in total, we can either read/write eighty 1GB files or forty 2GB files, though the block size is 128MB. Note that in Hadoop, the number of reading/writing files in a job actually affects the number of disks the job reads/writes to, that is, the number of I/O operations on the cluster. More reading/writing files means more I/O operations and vice versa.

The CPU-intensive applications we use in the experiments are *PiEstimator* and matrix multiplication. *PiEstimator* uses a statistical (quasi-Monte Carlo) method [15] to estimate the value of Pi. Points placed at random inside of a unit square also fall within a circle inscribed within that square with a probability equal to the area of the circle, Pi/4. The value of Pi can be estimated from the value of 4R where R is the ratio of the number of points that are inside the circle to the total number of points that are within the square. The larger the sample of points used, the better the estimate is. The mappers generate a specified number of sample points placed at random inside of a unit square and then counts the number of those points that are inside a unit circle. The reducers accumulate points counted by the mappers and then estimates the value of Pi. Matrix multiplication (*MM*) in the experiments calculates the multiplication of two square matrices. The two matrices are decomposed to a large number of small blocks and hence each mapper processes one block multiplication, while the reducers aggregate all the output block results generated in the mappers. The majority computations of the jobs are also completed during the map phase.

We measure these metrics for different applications:

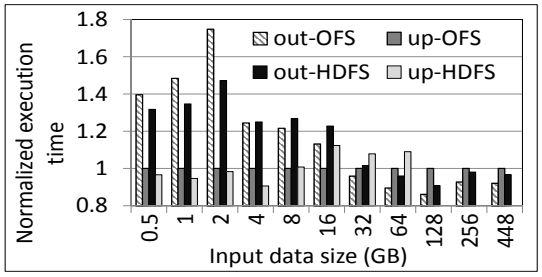
• *Execution time*, which is the job running time and calculated by the job ending time minus job starting time.

• *Map phase duration*, which is calculated by the last mapper's ending time minus the first mapper's starting time.

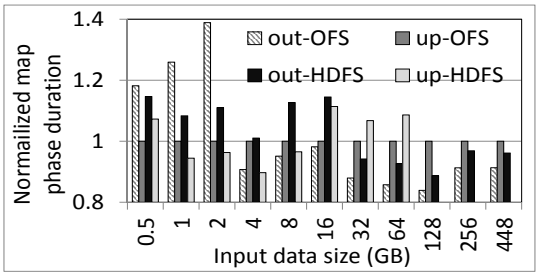
• *Shuffle phase duration*, which is defined as last shuffle task's ending time minus the last mapper's ending time.

• *Reduce phase duration*, which is from the ending time of the last shuffle task to the end of the job.

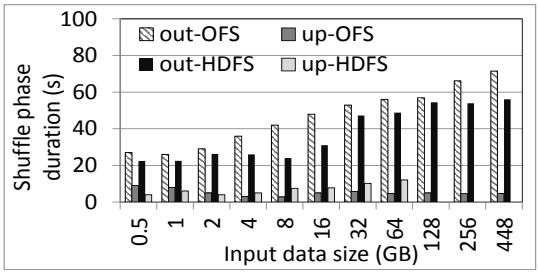
In the experiments, we normalize the execution time and map phase duration by the results of up-OFS. For example, if a job running on up-OFS and up-HDFS has an execution time of 10



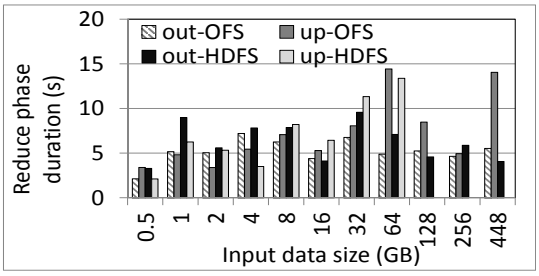
(a) Execution time.



(b) Map phase duration.



(c) Shuffle phase duration.



(d) Reduce phase duration.

Fig. 5. Measurement results of data-intensive jobs of *Wordcount*.

and 15 seconds, respectively, then up-OFS on the figure is shown as 1, while up-HDFS on the figure is shown as 1.5. Due to the limit of local disk size, we cannot process data more than 80GB on up-HDFS platform. Therefore, in the following measurement results, we do not show the up-HDFS for input data size more than 80GB. On one hand, this limitation does not have any impacts on our measurement analysis, as we are able to have meaningful observations before the input data size is increased to 80GB. On the other hand, we can take a first sight of the drawback of scale-up machines from this limitation. That is, as the input data size increases, it finally exceeds the capability of scale-up machines. Thus, scale-up machines are not as scalable as the scale-out machines. The number of *map (reduce) waves* of a job is calculated by the number of distinct start times from all mappers (reducers) of the job. If the number of mappers (reducers) of a job is larger than the number of map (reduce) slots in a node, partial mappers (reducers) are scheduled to all the slots first, forming the first wave. After the tasks complete and some slots are available, the second, third and subsequent waves are scheduled in sequence. When all the mappers (reducers) have the same execution time, the number of map (reduce) waves is equal to $\frac{the\ number\ of\ tasks}{the\ number\ of\ task\ slots}$.

3.2 Data-Intensive Applications

In this section, we show the performance evaluation of data-intensive applications including *Wordcount*, *Grep*, *Terasort*, and *TCJ*. Figures 5(a), 6(a), 7(a), and 8(a) show the normalized execution time of *Wordcount*, *Grep*, *Terasort*, and *TCJ* versus different input data size, respectively. Note that in all these applications, the number of mappers is determined by the input data size, which is calculated by $\lceil \frac{Input\ data\ size}{Block\ size} \rceil$. Since the block size is fixed in the experiments, the number of mappers is proportional to the input data size. From the figures, we have several meaningful observations.

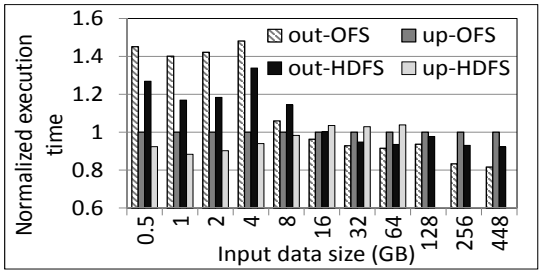
We observe that when the input data size is small (*WordCount*: 0.5-16GB, *Grep and TeraSort*: 0.5-8GB, *TCJ*: 1-16 mappers), the performance of *Wordcount*, *Grep*, *Terasort*, and *TCJ* is better on the scale-up machines than the scale-out machines. On the contrary, when the input data size is large (*WordCount*: $\geq$32GB, *Grep and TeraSort*: $\geq$16GB, *TCJ*: $\geq$ 32 mappers), the performance of *Wordcount*, *Grep*, *Terasort*, and *TCJ* is better on the scale-out machines than on the scale-up machines. This result is caused by the following reasons.

First, when the input data size is small, the number of mappers that needs to process is also small. As we mentioned, the number of task waves is related to the total number of mappers and the slots available on the nodes. Though the scale-out machines have more CPU cores, small jobs (i.e., jobs that process small input data size) on the scale-up machines can also be completed in only one wave or a few task waves. As a result, the small jobs benefit from the more powerful CPU resources of the scale-up machines and hence better performance. Second, recall that the shuffle data is copied to the reduce nodes' memory, which is determined by the JVM heap size. Since the scale-up machines have larger heap sizes, it is less likely for the shuffle data to be spilled to local disks, leading to better performance than the scale-out machines. Third, the utilization of RAMdisk on the scale-up machines provides a much faster shuffle data placement than the scale-out machines. In summary, the more powerful CPU, larger heap size, and utilization of RAMdisks guarantee the better performance on scale-up machines than on scale-out machines, when the input data size is small.

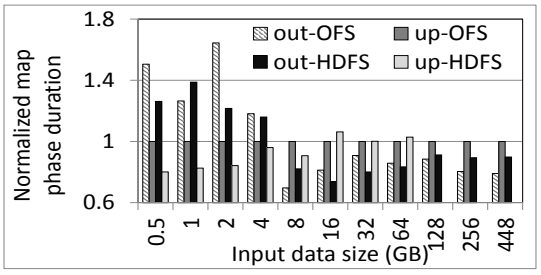
When the input data size is large, there are more mappers/reducers in the jobs. In this situation, the scale-out machines benefit from more task slots than the scale-up machines. Therefore, the scale-out machines complete jobs in fewer task waves than the scale-up machines do. Note that the more task waves will lead to a significant longer phase duration. Therefore, even though the scale-up machines are configured with larger heap size and utilization of RAMdisk, the scale-out cluster still outperforms the scale-up cluster.

Comparing HDFS and OFS, when the input data size is large, OFS outperforms the HDFS. However, when the input data size is small, surprisingly, the performance of HDFS is 20% (calculated by $\frac{|OFS-HDFS|}{OFS}$) better than OFS, although OFS can provide better I/O performance than HDFS [1] as we mentioned. This is because of the following reasons.

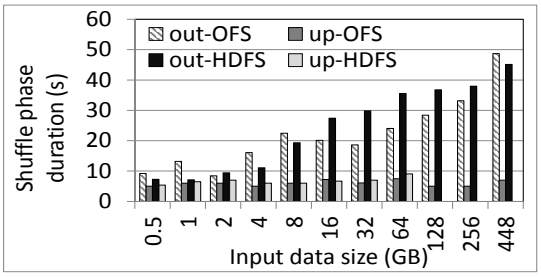
(1) The remote file system is required to be accessed through network. Although Myrinet provides a very fast local area in-



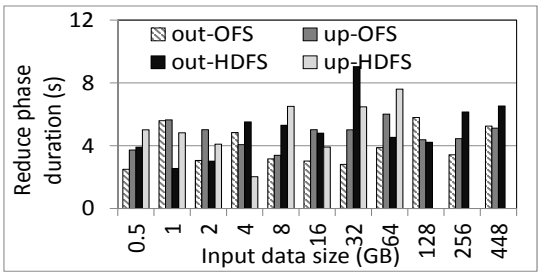
(a) Execution time.



(b) Map phase duration.

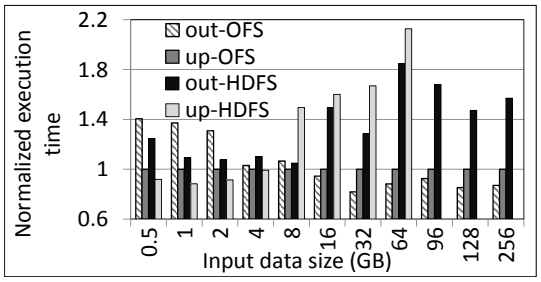


(c) Shuffle phase duration.

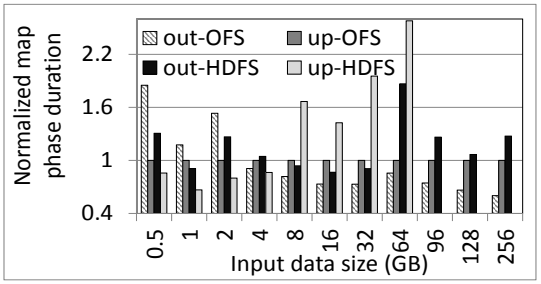


(d) Reduce phase duration.

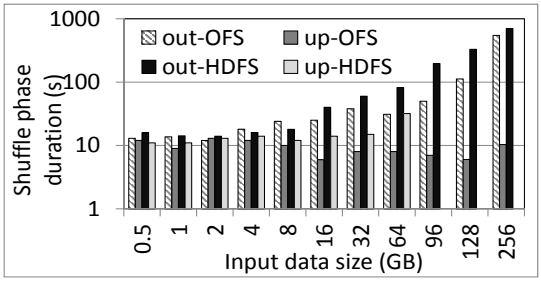
Fig. 6. Measurement results of data-intensive jobs of *Grep*.



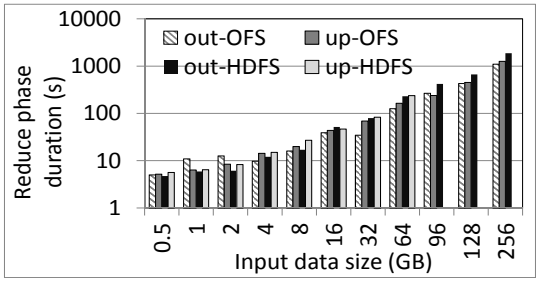
(a) Execution time.



(b) Map phase duration.



(c) Shuffle phase duration.



(d) Reduce phase duration.

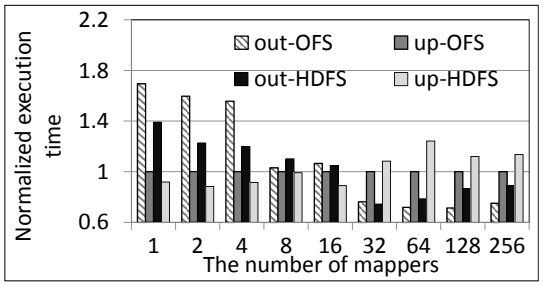
Fig. 7. Measurement results of data-intensive jobs of *Terasort*.

terconnect, there is still network latency in OFS, while HDFS benefits from data locality and hence avoids network latency. The network latency includes the latency generated by the overhead, the latency used to establish a connection with the remote file system and the latency of round-trip delay time. When the input data size is small, the execution time is relatively small. In this case, the network latency is not negligible comparing to the small execution time and the performance of small size jobs is degraded by this network latency in OFS [32].

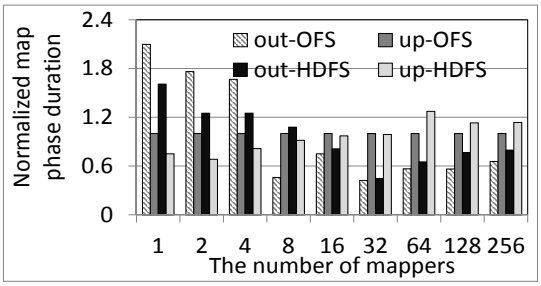
(2) On the other hand, when the input data size is large, the execution time becomes large and hence the network latency is gradually amortized by the large file. In this situation, since OFS has better I/O performance than HDFS as aforementioned, the execution time is shorter on OFS than on HDFS.

Therefore, we observe that when the input data size is small, the performance follows up-HDFS $>$ up-OFS $>$ out-HDFS $>$ out-OFS ($>$ means better). When the input data size is large, the performance of *Wordcount* and *Grep* follows out-OFS $>$ out-HDFS $>$ up-OFS $>$ up-HDFS, while *Terasort* follows out-OFS$>$up-OFS$>$out-HDFS$>$up-HDFS. *Terasort* performs a little bit different from *Wordcount* and *Grep* on up-OFS and out-HDFS when the input data size is large. This is because the sorting program not only has relatively a large amount of shuffle data but also a large amount of output data, while *Wordcount* and *Grep* have a negligible output data size compared to *Terasort*. It means that with OFS, *Terasort* reads input data for map tasks from OFS and writes the output of reduce task to OFS, while *Wordcount* and *Grep* only take advantage of OFS during reading input data for map tasks. Therefore, *Terasort* benefits twice from the higher I/O rate of OFS, which results in better performance on up-OFS than out-HDFS.

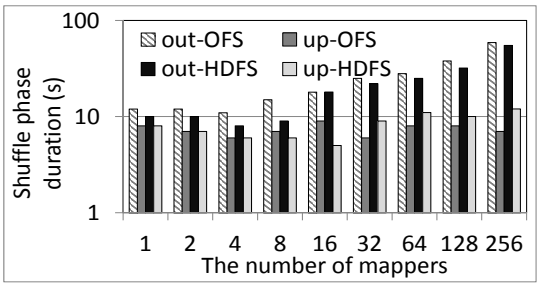
Furthermore, from Figures 5(a), 6(a), and 7(a), we observe that



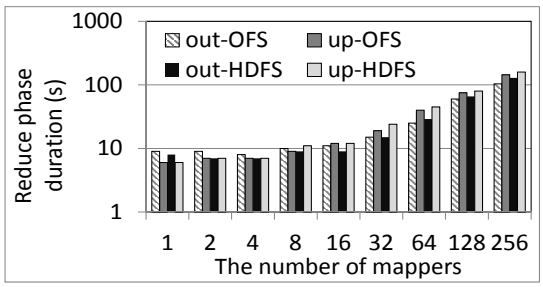
(a) Execution time.



(b) Map phase duration.



(c) Shuffle phase duration.



(d) Reduce phase duration.

Fig. 8. Measurement results of data-intensive jobs of *TCJ*.

the performance of WordCount is better on scale-up machine until the input data size of WordCount reaches 32GB (namely cross point), while the performance of Grep and TeraSort is better on scale-up machine until the input data size reaches 16GB. This indicates that these three applications have different degrading speed on scale-up and scale-out machines as the input data size increases, although they are all data-intensive applications. This difference for the applications is caused by the different ratio of $\frac{shuffle\ data\ size}{input\ data\ size}$. In our experiments, no matter how much input data size the jobs have, the shuffle/input ratio of *Wordcount*, *Terasort* and *Grep* are around 1.6, 1.0 and 0.4, respectively. Given an input data size, with a higher shuffle to input ratio, WordCount tends to have more shuffle data than *Terasort* and *Grep*. Hence, WordCount can achieve more benefits from the larger heap size and RAMdisk of scale-up machines, resulting in a slower application performance degradation on the scale-up machines.

Since the execution time of a job consists of the durations in the map, shuffle and reduce phases, we then study these broken-down durations. Figures 5(b), 6(b), 7(b), and 8(b) show the normalized map phase duration of *Wordcount*, *Grep*, *Terasort*, and *TCJ*, respectively. We observe a similar relationship of the map phase duration with the job execution time due to the same reasons. When the input data size is small (0.5-8GB), the map phase duration is shorter on scale-up than on scale-out; when the input data size is large (>16GB), the map phase duration is shorter on scale-out than on scale-up. As to the comparison between OFS and HDFS, it is also similar with the relationship of the job execution time due to the same reasons. We see that when the input data size range is 0.5-8GB, the map phase duration of these jobs are 10-50% shorter on HDFS than on OFS. When the input data size is larger than 16GB, the map phase duration is 10-40% shorter on OFS than on HDFS, no matter if they are configured with the scale-up or scale-out cluster.

Figures 5(c), 6(c), 7(c) and 8(c) show the shuffle phase duration of *Wordcount*, *Grep*, *Terasort*, and *TCJ*, respectively. We see that the shuffle phase duration is always much shorter on scale-up machines than on scale-out machines. This is because of the larger heap size and RAMdisk of scale-up machines as aforementioned.

Figures 5(d), 6(d), 7(d), and 8(d) show the reduce phase duration of *Wordcount*, *Grep*, *Terasort*, and *TCJ*, respectively. In *Wordcount* and *Grep*, the reduce phase aggregates the map outputs which have small size and hence the reduce phase duration is very short. Therefore, the reduce phase duration of *Wordcount* and *Grep* is around a few seconds and there is not any specific relationship of the reduce phase duration. On the other hand, the reduce phase of *Terasort* needs to sort the map outputs which has the same size as the input data, resulting in a long reduce phase duration (increasing from 5 to 1800 seconds as the input data size increases. The number of task waves of the reduce slots on scale-out machines is fewer than on scale-up machines and hence the reduce phase duration of *Terasort* performs similarly as the execution time and the map phase duration. That is, when the input data size is small (0.5-8GB), the reduce phase duration is shorter on scale-up than on scale-out; when the input data size is large (>16GB), the reduce phase duration is shorter on scale-out than on scale-up. As to *TCJ*, its reduce phase is similar to the reduce phase of *TeraSort*. When the number of mappers in *TCJ* increases, the output data size becomes larger. Therefore, we see from Figure 8(d) that the reduce phase duration of *TCJ* on scale-out machines is smaller when the number of mappers is large; when the number of mappers is small, the reduce phase duration is similar on the scale-up and scale-out clusters.

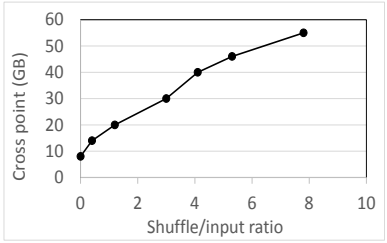
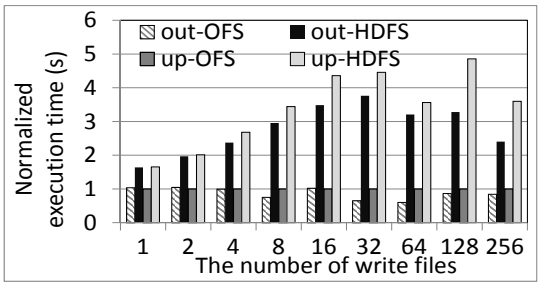
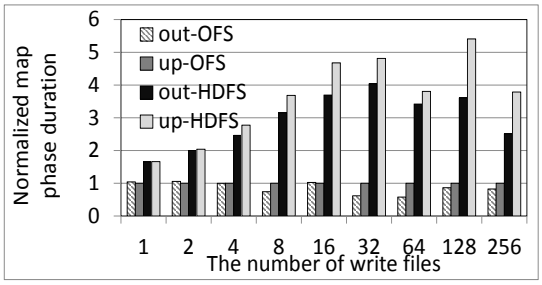


Fig. 9. Cross point versus shuffle/input ratio.

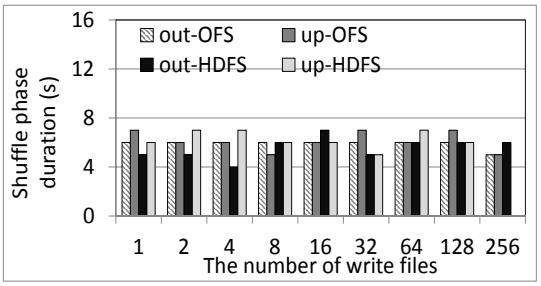
We see neither OFS nor HDFS affects the reduce phase duration of *Wordcount* and *Grep*. This is because the reduce phase duration of these two applications only lasts for a short time, which is hardly affected by the file system. For *Terasort*, the reduce phase duration is 20-65% shorter on HDFS than on OFS when the input data size is small (0.5-8GB) on either the scale-up or scale-out



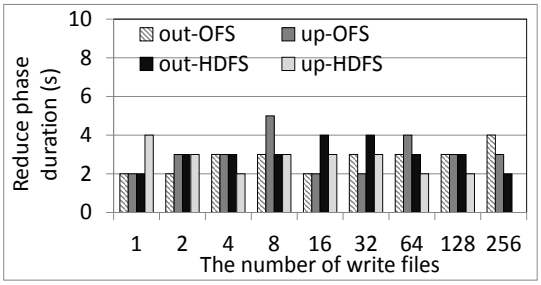
(a) Execution time.



(b) Map phase duration.

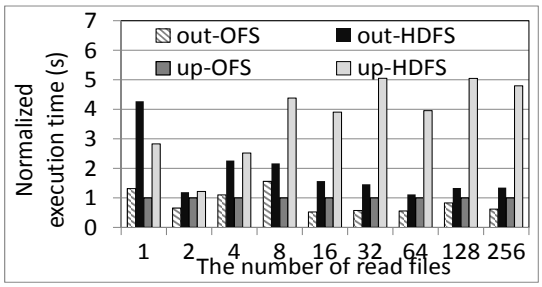


(c) Shuffle phase duration.

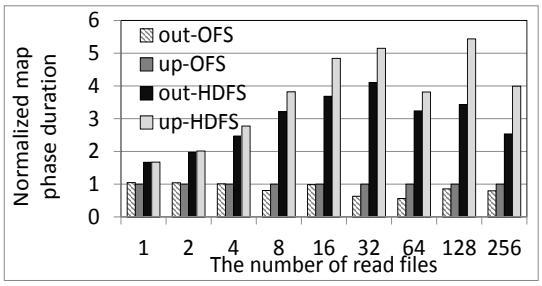


(d) Reduce phase duration.

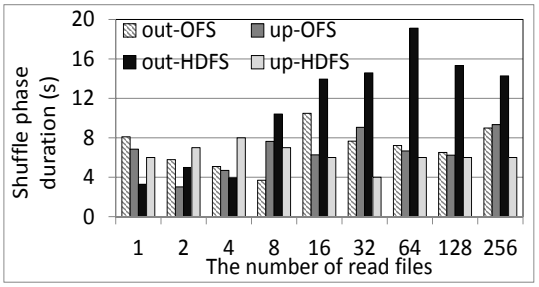
Fig. 10. Measurement results of I/O-intensive write test (80GB) of *TestDFSIO*.



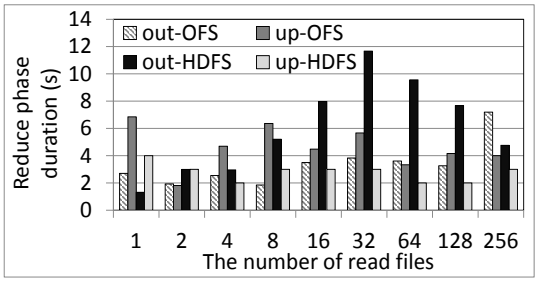
(a) Execution time.



(b) Map phase duration.



(c) Shuffle phase duration.



(d) Reduce phase duration.

Fig. 11. Measurement results of I/O-intensive read test (80GB) of *TestDFSIO*.

cluster. When the input data size is large (>16GB), it is 20-70% shorter on OFS than on HDFS, no matter if they are configured with the scale-up or scale-out cluster. This is because *Terasort* has the same output data size as the input data size. When the input data size is large, the output data size is also large. Therefore, as aforementioned, when the input data size is large, OFS provides better I/O performance than HDFS and can complete the writing of output data faster; when the input data size is small, HDFS outperforms OFS because of the network latency on OFS.

In this section, *WordCount*, *Grep*, and *Terasort* prove that the shuffle data size does impact the cross point (and hence platform selection). Next, we varied the shuffle data size of the same application to further demonstrate this observation. We ran a set of *Grep* measurement experiments by varying the regular expressions to grep. In this way, the shuffle/input ratio of Grep ranged from 0.0 to 8.0. We aim to explore the cross points between scale-up and scale-out for *Grep* with different shuffle/input ratios. Figure 9 shows the cross point versus the shuffle/input ratio for *Grep*. We see that as the shuffle/input ratio increases, the cross point also increases. *Grep* can achieve more benefits from scale-up machines as the shuffle data size increases, which demonstrates that the shuffle data size is one of the key factors in platform selection.

3.3 I/O-Intensive Applications

In this section, we measure the performance of a relatively large data size (80GB). Figures 10(a) and 11(a) show the normalized execution time of *TestDFSIO* reading/writing 80GB data versus different number of files, respectively. In these reading/writing tests, the size of each file is equal to $\frac{80GB}{the\ number\ of\ files}$. More reading/writing files means more I/O operations and vice versa. We see that when the number of files is large, the performance

of I/O-intensive applications (both read and write) is better on scale-out machines than on scale-up machines, no matter if they use HDFS or OFS. For HDFS, the I/O rate of local disks is similar on both scale-up and scale-out machines. However, scale-out machines read/write data from/to twelve datanodes simultaneously, while scale-up machines read/write data from/to only two datanodes. Therefore, scale-up machines read/write to fewer disks in parallel, which limits their performance. As to OFS, it allows CPU to build up multiple communications with remote storage servers simultaneously and hence read/write files in parallel. As a result, the scale-out machines that have more CPU cores can read/write more files from/to OFS at the same time. On the other hand, since the scale-up machines have fewer CPU cores, they build up fewer communications with OFS and hence read/write fewer files from/to OFS. Therefore, for both HDFS and OFS, the scale-out cluster outperforms the scale-up cluster.

When the number of files is small, the performance is similar on scale-up machines and scale-out machines. This is because when there are only a small number of files, both scale-up and scale-out machines read/write files from/to only a small number of disk devices simultaneously, which cannot take advantage of the larger number of disk devices in the scale-out machines. In this case, the factor that affects the performance is mainly the disk rate. Since the disk rate is similar on the scale-up and scale-out machines, no matter if they use HDFS or OFS, the execution times on scale-up and scale-out machines are similar to each other.

Figures 10(b), 10(c) and 10(d) and Figures 11(b), 11(c) and 11(d) show the map, shuffle and reduce phase durations of the write test and the read test of 80GB data, respectively. we see that in both the write and read tests, the map phase duration exhibits a similar performance trends as the execution time. The shuffle and reduce phase durations of both tests are quite small ($<$8s), and hence they exhibit no specific relationship. Comparing OFS and HDFS in the scale-up and scale-out clusters, the map phase duration is shorter on OFS than on HDFS for reading/writing 80GB. Since the shuffle and reduce phase durations are very small, they are not affected by using either OFS or HDFS.

3.4 CPU-Intensive Applications

Figure 12(a) shows the execution time of *PiEstimator* versus the number of sample points. *PiEstimator* has 80 mappers in this experiment. Note that as the number of sample points increases, there are more calculations in the experiments because the application needs to calculate the location of each point to determine whether it is in the unit circle or not, which results in an increase of the execution time of each mapper. Moreover, as the number of sample points increases, each mapper needs to process an input file size ranging from (2-20000)KB.

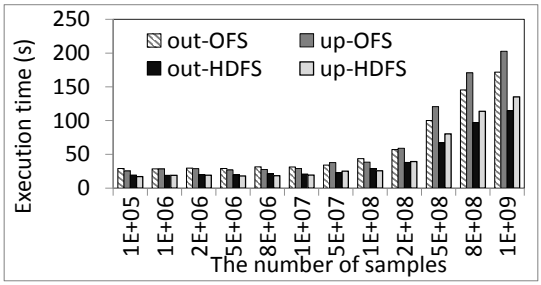
When the number of sample points is small (10^5-10^7) (i.e., each mapper conducts fewer computations and can be completed in a shorter time), we see that the scale-up machines outperform the scale-out machines. However, when the number of sample points is large ($> 10^7$) (i.e., each mapper conducts more computations and requires more time to complete), we see that the scale-out machines perform better than the scale-up machines. Although the scale-out machines benefit from more CPU cores to handle the mappers, the scale-up machines still outperform the scale-out machines when the number of sample points is small. This is because of the L1 cache size difference of CPUs on the scale-up and scale-out machines. When the number of sample points is small, the input data size (2-200KB) is as small as the CPU L1 cache size and hence the CPUs can process all the data within the fastest cache. When all the data is placed in L1 cache, the CPUs on scale-up machines can be fully utilized. Therefore, the full utilization of CPUs on scale-up machines compensates the disadvantage of fewer CPU cores. When the amount of computations is large, the input data size is much larger than the L1 cache size, which means that CPU cannot maintain the fastest speed, resulting in lower performance. Then, the disadvantage of fewer map slots on scale-up machines cannot be compensated. As a result, the execution time on scale-up machines is higher than on scale-out machines when the amount of computations is large.

Comparing the performance of OFS and HDFS, we see that OFS always performs worse than HDFS. This is because each mapper handles a small file size in *PiEstimator*. As we mentioned previously, when the input data size is small, the network latency is non-negligible in OFS. In contrast, HDFS benefits from high data locality and avoids the network latency. Therefore, HDFS outperforms OFS for small input data sizes.

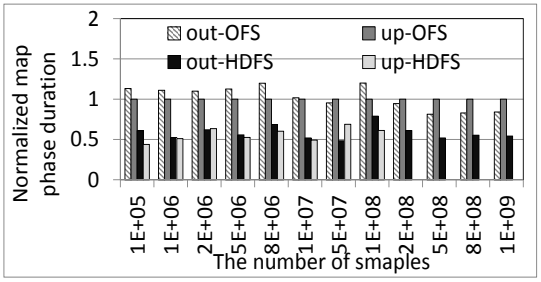
Figures 12(b), 12(c) and 12(d) show the map, shuffle and reduce phase durations of *PiEstimator*, respectively. Since the map phase of *PiEstimator* completes the majority of the work in the jobs (determining whether the sample points are in the unit circle or not), while the shuffle phase only collects the statistics and the reduce phase simply derives Pi from the map results, we see that the map phase duration exhibits a similar performance trend as the execution time. The shuffle and reduce phase durations of *PiEstimator* are quite small ($<$5s), and they exhibit no specific relationships on either scale-up or scale-out machines. Comparing OFS and HDFS, OFS leads to $50-80\%$ longer map phase duration. This is caused by the non-negligible network latency for processing a small data size. As to the shuffle and reduce phases, since their durations are very small, whether using OFS or HDFS does not affect the durations of these two phases much.

Figure 13(a) shows the normalized execution time of *MM* versus different number of mappers. Note that more mappers means that the matrix's size is larger. When the number of mappers is small (1-16), we see that scale-up machines perform better because of their better CPUs as indicated previously. When the number of mappers is large ($>$16), scale-out machines perform better since there are fewer map slots on scale-up machines and hence more task waves. In spite of the better CPU on scale-up machines, the performance is degraded because of the more task waves of *MM*.

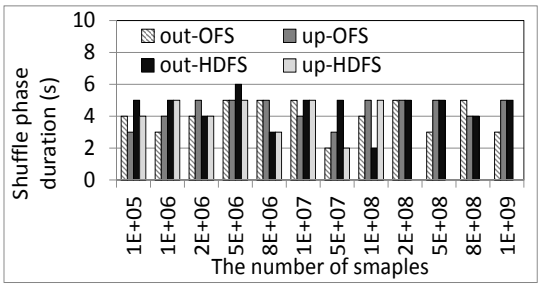
Figures 13(b), 13(c), and 13(d) show the map, shuffle, and reduce phase durations of *MM* versus the number of mappers. We see that the map phase duration has similar results as the execution time because the majority of the work of *MM* is completed in the map phase. The shuffle phase duration is shorter on scale-up machines than on scale-out machines because the scale-up machines handle shuffle data more quickly as explained previously for the data-intensive jobs. The reduce phase of *MM* aggregates the results generated in map phase. As the size of matrix (hence the number of mappers) increases, the output data size of *MM* becomes larger. We see that when the number of mappers is large, the reduce phase duration on scale-out machines becomes smaller because the scale-out machines can write the output data to more disk devices simultaneously, as explained previously for the I/O-intensive jobs. When the number of mappers is small, we see that the reduce phase duration is similar on the scale-up and scale-out clusters. This is because the output data size is small and hence



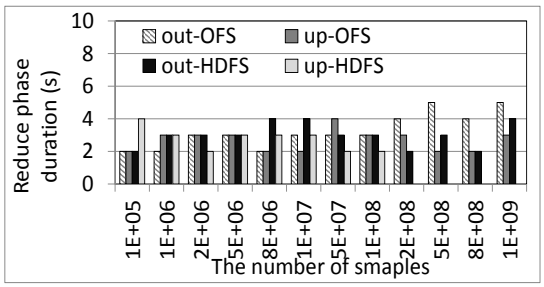
(a) Execution time.



(b) Map phase duration.

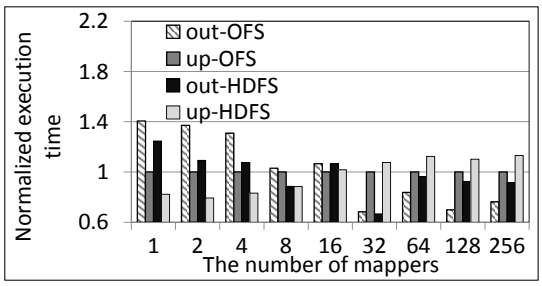


(c) Shuffle phase duration.

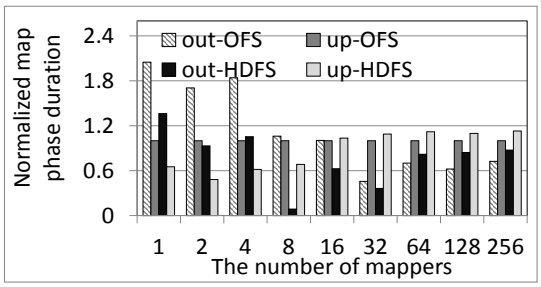


(d) Reduce phase duration.

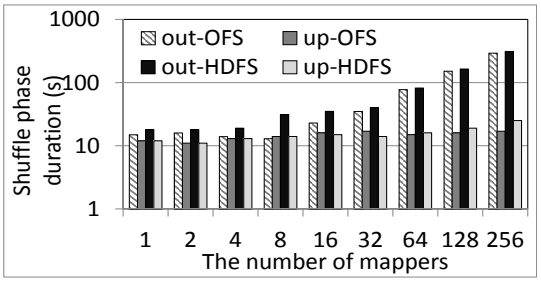
Fig. 12. Measurement results of CPU-intensive jobs of *PiEstimator*.



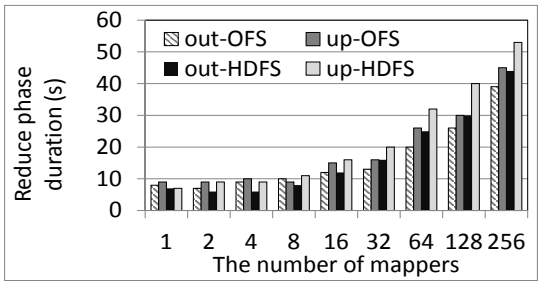
(a) Execution time.



(b) Map phase duration.



(c) Shuffle phase duration.



(d) Reduce phase duration.

Fig. 13. Measurement results of CPU-intensive jobs of *MM*.

it can be written in a few blocks, which cannot take advantage of the large number of disk devices of the scale-out machines.

Comparing OFS and HDFS for *MM*, when the number of mappers is small, it is better to use HDFS. On the contrary, as the number of mappers increases, it becomes better to use OFS rather than HDFS. The reason is that OFS can provide more powerful I/O performance than HDFS, as explained for the data-intensive applications previously. However, when the input data size is small, the network latency is non-negligible and degrades the performance of OFS.

4 Discussions

In this section, we first summarize the measurement results and analysis in Section 3. Then we discuss the potential implications of our measurement results on guiding the users who would like to configure Hadoop in the cloud environments.

4.1 Summary of Results

We can make the following conclusions for data-intensive, CPU-intensive and I/O-intensive applications.

Data-intensive applications:

(1) When the input data size is small, the performance relationship is up-HDFS>up-OFS>out-HDFS>out-OFS.

(2) When the input data size is large, the performance of applications with a small output data size (e.g., *Wordcount* and *Grep*) follows out-OFS>out-HDFS>up-OFS>up-HDFS, while the performance of applications with a large output data size (e.g., *Terasort*) follows out-OFS>up-OFS>out-HDFS>up-HDFS.

I/O-intensive applications:

(1) When the number of reading/writing files is small, the performance relationship is up-OFS>out-OFS>up-HDFS > out-HDFS.
(2) When the number of reading/writing files is large and the total file size is large (e.g., 80GB), the performance relationship is out-OFS>up-OFS>out-HDFS>up-HDFS.

CPU-intensive applications:

(1) When both the amount of computations and the input file size are small, the performance relationship is up-HDFS > out-HDFS > up-OFS>out-OFS.
(2) When both the amount of computations and the input file size are large (e.g., *MM*), the performance relationship is out-OFS>out-HDFS>up-OFS>up-HDFS. On the other hand, when the amount of computations is large but the input file size is small (e.g., *PiEstimator*), the performance relationship is out-HDFS>up-HDFS>out-OFS>up-OFS.

Therefore, for a specific type of applications, users can determine which platform should be use to execute the applications to achieve the best performance. For example, when a data-intensive job with input data size 100GB is submitted, based on our conclusions, the job should run on the out-OFS platform. We expect that our measurement results can help users to select the most appropriate platforms for different applications with different characteristics on HPC clusters.

4.2 The Guidance of Hadoop Configurations in Cloud Environments

Although the the performance measurements in our paper were conducted on the Hadoop configurations on a HPC cluster, our performance measurement results may also provide guidance for the users who would like to configure Hadoop in the cloud environments.

Take Amazon EC2 [2] as an example. Users are able to configure their Hadoop MapReduce clusters with different types of instances on Amazon EC2. Our performance measurement results can first guide the users on how to select the instances for their Hadoop MapReduce cluster for better performance based on the workloads they have. For example, if the workloads are data-intensive and most of the jobs in the workloads have large input data size, it is better for them to select a bunch of scale-out machines. For some CPU-intensive workloads with small input data size, selecting several scale-up instances instead of many scale-out instances may be a better choice.

Another guidance of our paper is on the file system configuration of Hadoop. We can guide the users on whether Hadoop with HDFS or Hadoop with remote file system can provide better performance. Data is generally stored in Amazon Simple Storage Service (Amazon S3) [3], which is a highly-scalable remote storage. In addition to Amazon S3, each instance on Amazon EC2 also comes with a storage. However, the storage on the instances is small, which may not be suitable for large-size applications. Besides, in the cloud environment, users need to first transfer the data from Amazon S3 to the storage on each instance, which is costly and takes time. In this paper, our performance measurement results show that the users may not need to use the traditional Hadoop with HDFS configuration. For example, users may achieve better performance on their Hadoop clusters by directly configuring Hadoop with S3 for workloads that are data-intensive and have large input data size. For some CPU-intensive workloads with small input data size, configuring the traditional Hadoop with HDFS may be a better choice. We leave the exploration of configuring Hadoop with remote storage in cloud environments as the future work.

5 Performance Prediction Model for Best Platform Selection

In this section, we aim to develop a performance prediction model to predict the performance of different applications based on their characteristics, which helps users select the best platforms for their applications.

Recall in Section 2 that a MapReduce job consists of map, shuffle, and reduce phases. In this section, in order to predict the performance, we revisit the basic principle of MapReduce. First, we would like to elaborate the factors that impact the performance of a MapReduce job (i.e., the job execution time). Then we analyze the impacts from different machines (i.e., scale-up and scale-out machines) and different storages (i.e., HDFS and OFS) on the job execution time.

5.1 Proposed Prediction Model

The execution time of a MapReduce job is impacted by the time duration of the five steps in Section 2.5, as shown in Figure 4. Due to the parallel feature of MapReduce, shuffle phase runs highly overlap with map phase. Therefore, Step 3 only contributes part of its time duration to the job execution time.

Figure 4 indicates which parts the four metrics (i.e., execution time, map phase duration, shuffle phase duration and reduce phase duration in Section 3) represent. We see that the shuffle phase duration is actually slightly different from the time duration of Step 3. Let us denote *MPD*, *SPD*, and *RPD* as the map phase duration, shuffle phase duration, and reduce phase duration, respectively. We can express the job execution time ET as follows,

$$ET = MPD + SPD + RPD. \tag{1}$$

Let us denote $t_r, t_{map}, t_{shu}, t_{red}, t_w$ as the time durations of input data reading, execution of map tasks, shuffle data transferring, execution of reduce tasks, and output data writing, respectively. A simple "rule-of-thumb" [24], [25], [38], [44] states that the time duration of each step is proportional to the amount of processed data. Let us denote the input data size, shuffle data size, and output data size as *IS*, *SS*, and *OS*, respectively. Suppose the job has M map tasks and R reduce tasks. Then, based on the factors that affect each part, we can derive the time duration of each step as follows,

$$\begin{aligned} t_r &= \alpha * \frac{IS}{M} + d_1, \\ t_{shu} &= p * SS, \\ t_w &= \beta * \frac{OS}{R} + d_2, \end{aligned} \tag{2}$$

where α and β are the speed coefficients of the disk I/O. d_1 and d_2 are constants, which could be the network setup delay for remote file transfer. p is the speed coefficient of the network. The time durations of Step 2 and Step 4 are given by

$$\begin{aligned} t_{map} &= a * \frac{IS}{M}, \\ t_{red} &= b * \frac{SS}{R}, \end{aligned} \tag{3}$$

where a and b are the coefficients that reflect the speed of each node to process the data. As mentioned above, to generalize the

model for different jobs, we can approximately convert a and b to shuffle data size. Therefore, we have

$$t_{map} = \delta * \frac{SS}{M} * \frac{IS}{M}, \\ t_{red} = \theta * \frac{OS}{R} * \frac{SS}{R}, \tag{4}$$

where δ and θ are the speed coefficients of each node to process the data.

In additional to the data size, we also need to consider the number of waves (mentioned in Section 3.1) to estimate the job execution time [24]. Suppose the entire cluster has N machines, each of which contains m map slots and r reduce slots. Therefore, the map tasks run in $\lceil \frac{M}{N*m} \rceil$ waves, while the reduce tasks run in $\lceil \frac{R}{N*r} \rceil$ waves. We then can derive the job execution time ET as follows,

$$ET = MPD + SPD + RPD \\ = (t_r + t_{map}) * \lceil \frac{M}{N*m} \rceil + \eta * t_{shu} + (t_{red} + t_w) * \lceil \frac{R}{N*r} \rceil, \tag{5}$$

where η is the coefficients that describes how much time duration Step 3 contributes to the execution time. Substituting Equations (2) and (4) into (5), we have the execution time ET prediction model as follows,

$$ET = \lceil \frac{M}{N*m} \rceil * (\alpha * \frac{IS}{M} + d_1 + \delta * \frac{SS}{M} * \frac{IS}{M}) \\ + \eta * p * SS + \lceil \frac{R}{N*r} \rceil * (\theta * \frac{OS}{R} * \frac{SS}{R} + \beta * \frac{OS}{R} + d_2). \tag{6}$$

Now let us analyze the prediction model in the above. In a given cluster, the number of machines (i.e., N), the number map and reduce slots (i.e., m and r) are constant. For a given job with certain characteristics running on this cluster, IS, SS, OS, M, and R are all known parameters. Therefore, in Equation (6), the unknown coefficients are actually d_1, d_2, δ, θ, η, p, α and β. Notice that different platforms result in different sets of coefficients. To get the coefficients for different platforms, we can use linear regression [7] to train the model.

Next, we briefly discuss how different machines (i.e., scale-up and scale-out machines) and different storages (i.e., HDFS and OFS) impact the coefficients in the prediction model.

- Scale-up machines and scale-out machines. One difference between scale-up machines and scale-out machines is the difference of CPU and memory capacity. This difference between different machines certainly results in different δ and θ. As scale-up machines have more powerful CPU and memory, we would expect that with scale-up machines, δ and θ are smaller.

Besides, due to the memory and network configuration differences, scale-up machines and scale-out machines also lead to different η and p. Similarly, it is expected to have smaller η and p with scale-up machines.

Moreover, since the scale-up machines and scale-out machines have different number of CPU cores, the number of map and reduce slots m and r are different on different machines.

- HDFS and OFS. As we stated above in Section 2.5, the file system affects the speed of input data reading and the output data writing. Therefore, α and β are impacted by the selection of HDFS and OFS. From the measurement results in Section 3, it is expected that with OFS, α and β are smaller.

In summary, we present a performance prediction model to help users select their best platforms based on the job characteristics. Specifically, we derive a simple linear model based on the job characteristics to predict the performance. In this model, we leverage two widely used assumptions in many previous studies. The first assumption is that the time duration of each step is proportional to the amount of processed data [24], [25], [38], [44]. The second assumption is that the tasks tends to finish in waves [25], [27], [46]. Several previous studies have presented more complex performance models, such as Starfish [22], MRPerf [42], iTuned [20] and the analytic model in [34]. However, in this paper, our goal is not to accurately predict the job execution time of each job. Instead, we aim to propose a simple mathematical model to let users quickly estimate the performance difference of their jobs between different platforms and select the best platforms. In other words, we expect to propose a model to find out which platforms are the best for different jobs.

5.2 Performance Evaluation

In this section, we used the Facebook synthesized workload FB-2009 [18] to evaluate the performance of our prediction model, focusing on whether users can utilize our prediction model to select the best platforms. FB-2009 is a workload trace that records the characteristics of jobs running in production clusters. The trace contains the job characteristics such as job submission time, input data size, shuffle data size, and output data size. According to previous studies [17], [18], The jobs in FB-2009 are very diverse and have input data size ranging from KB to TB, which ensures sufficient datasets for every platform to evaluate the performance of our prediction model.

Based on FB-2009, we generated the jobs correspondingly [18] and reran all the jobs in FB-2009 one by one on each of the four platforms. The configurations of the four platforms are the same as the configurations in Section 2. After running the jobs, we collected the Hadoop logs for all the jobs, which include the job execution time, the number of map and reduce tasks, and input, shuffle and output data sizes. We used the logs as the datasets and the features (job characteristics) of every job include $\lceil \frac{M}{N*m} \rceil * \frac{IS}{M}$, $\lceil \frac{M}{N*m} \rceil$, $\lceil \frac{M}{N*m} \rceil * \frac{SS}{M} * \frac{IS}{M}$, SS, $\lceil \frac{R}{N*r} \rceil * \frac{OS}{R} * \frac{SS}{R}$, $\lceil \frac{R}{N*r} \rceil * \frac{OS}{R}$, and $\lceil \frac{R}{N*r} \rceil$, as shown in Equation 6.

We can formulate the platform selection problem as two machine learning problems, regression and classification [39].

TABLE 2
Coefficients of different platforms.

Platforms	λ
up-OFS	1.00
up-HDFS	5e10
out-OFS	93690.87
out-HDFS	5e10

In regression, we leverage the model in Equation (6) to predict the execution times of the application on every platform and then select the one with the smallest execution time. Specifically, we utilize the Ridge Regression (RR) [23], which is a widely used linear regression model with regularization when the features have high collinearity. The ridge regression uses a parameter λ to control the regularization to avoid overfitting problem. The ridge regression is used when the features have high collinearity. In this case, $\lceil \frac{M}{N*m} \rceil * \frac{IS}{M}$ and $\lceil \frac{M}{N*m} \rceil$ are highly collinear, as $\frac{IS}{M}$ represents the block size of the file systems (i.e., 128MB). It is worth mentioning that each λ is corresponding to one linear model. To choose the best λ value (the best model), we used 10-fold-cross-validation approach [6] to train the FB-2009 trace. Specifically, we first randomly divided the FB-2009 trace into 10 folds, among which 9 folds are the training dataset and the remaining 1 fold

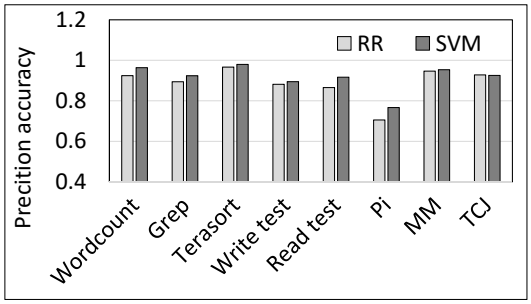


Fig. 14. Prediction accuracy in selecting the best platforms for different applications.

is the testing dataset. For a λ, we repeated this process for 10 times, with each fold of data used once as the testing dataset. Each time we computed the mean square error (MSE) [8] of the trained model on the one fold of testing dataset (in total 10 MSE results for 10 folds). We then took the average of 10 MSEs as the MSE of the λ and we chose the best λ that gives the smallest MSE. The regularization parameter λ for each platform are listed in Table 2. We then applied the trained model for each platform on the FB-2009 trace. We computed the execution time of each job on each platform and selected the platform with the smallest execution time. The prediction accuracy of the best platform is *89.23%*.

In classification, instead of using Equation (6) to calculate the execution time, we use the equation as a reference to imply that what features significantly affect the final performance of an application and hence the platform selection. Then, we use the Support Vector Machine (SVM) classifier with the nonlinear RBF kernel [26], which takes the features of each application in Equation (6) as inputs and predicts which class (i.e., which platform) the application should run on. The RBF model have two parameters γ and C to control the regularization to avoid over-fitting problem. As the RR, we use the 10-fold-cross-validation approach as introduced above to select the best γ and C with the highest prediction accuracy. Using SVM, we can get a prediction accuracy of 94.16%, which is higher than the prediction accuracy in the regression model.

Further, we applied the trained models to the measurement results in Section 3 to see how the models perform and whether the models overfit the FB-2009 dataset. Figure 14 shows the accuracy of our models in selecting the best platforms for different applications. We see from the figure that *Wordcount*, *Grep*, *Terasort*, *TestDFSIO*, *MM* and *TCJ* all achieve high accuracies in selecting the best platforms (more than 87% using RR and more than 89% using SVM), while the CPU-intensive application *PiEstimator* only achieves an accuracy around 70%. This is because of the current artifact of OFS – when transferring extremely small files (e.g., smaller than 10KB), OFS suffers from abnormal latency, as explained in Section 3.4.

6 Related Work

File system. MapReduce [19] is a popular framework that performs parallel computations on big data. Many HPC sites [1] have extended their clusters to support Hadoop MapReduce. File systems are an essential component in the MapReduce and HPC clusters. Tantisiriroj *et al.* [41] integrated PVFS into Hadoop and compared its performance with HDFS. Other works [11], [13] successfully implement HPC file systems (GPFS and Lustre) in Hadoop. Meza *et al.* [35] provides a large scale study of flash memory based solid state drives (SSD) in data center. Chen *et al.* [16] evaluated the performance of network file system version 4 with different features. Our work is different from the above work in that we combine HDFS and OFS with scale-up and scale-out machines and measure the application performance on different platforms in order to provide guidance on selecting the most appropriate platform to run a job based on its characteristics.

Workload characterization. In order to improve the performance of MapReduce clusters, characterizing the workload features is important since cluster provisioning, configuring and managing is essential for a cluster. Studying the workloads can provide general insights about the performance of clusters. Chen *et al.* [17] characterized new MapReduce workloads, which are driven in part by interactive analysis and with heavy use of query-like programming frameworks such as Hive on top of MapReduce. Elmeleegy [21] studied the workloads from Yahoo! Hadoop cluster and revealed that the majority of jobs are short and have only small number of tasks. Ren *et al.* [37] conducted a case study of the jobs and tasks of the workload from a commodity Hadoop cluster Taobao. Kavulya *et al.* [27] analyzed MapReduce logs from the *M*45 supercomputing cluster. Appuswamy *et al.* [14] measured the performance of a set of representative Hadoop applications on scale-up and scale-out machines. All of these works provide guidance on how to characterize different applications. Our work is different from the above works since we configure scale-up and scale-out machines for Hadoop with HDFS and a remote file system and measure the performance difference among all these platforms. From the results, we can select the best platform for different jobs with different characteristics.

Scale-up or scale-out. Whether scale-up or scale-out architecture is not only an attracted research area in MapReduce, but also exists in some other research ares. GraphChi [29], a work focusing on graph computations, advocates processing big data in a single machine. Michael et al. [36] investigated scale-up versus scale-out in an emerging search application. However, they found that the scale-out solutions provide better price/performance ratio, although at the cost of increasing management complexity.

7 Conclusion

In this paper, we have conducted performance measurement study of data-intensive, I/O-intensive and CPU-intensive applications on four HPC-based Hadoop platforms: scale-up cluster with OFS, scale-up cluster with HDFS, scale-out cluster with OFS and scale-out cluster with HDFS. We have conducted a thorough analysis on the measurement results and identified the best platform for each type of applications with certain characteristics, which provides a guidance on selecting platforms to run different applications. We confirm that replacing HDFS with OFS for Hadoop is feasible and we found that OFS outperforms HDFS when an application processes a large data size. Also, an application processing a small data size should be considered to be executed on the scale-up cluster. We expect that our measurement results can help users to select the most appropriate platforms for different applications with different characteristics. We also propose a performance prediction model to help users select the best platforms for different applications. Our evaluation using a Facebook workload trace demonstrates the effectiveness of our prediction model. Additionally, our results can also provide potential guidance on instance selection and file system selection for the users who would like to configure Hadoop in the cloud environments that provide similar file system architectures as HPC cluster. In the future, we plan to study the in-memory computing systems such as Spark on HPC clusters. Moreover, based on the conclusions in

this paper, we plan to develop an adaptive hybrid platform that contains both scale-up and scale-out machines, and HDFS and OFS. It can automatically determine the best platform for a given application with certain characteristics.

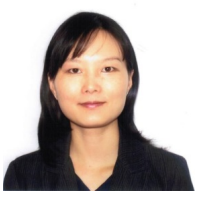
ACKNOWLEDGEMENTS

We would like to thank Dr. Walter Ligon, Mr. Jeffrey Denton, and Mr. Matthew Cook for their insightful comments. This research was supported in part by U.S. NSF grants ACI-1719397 and CNS-1733596, and Microsoft Research Faculty Fellowship 8300751. An early version of this work was presented in the Proceedings of Cloud 2016 [31].

REFERENCES

[1] Accelerate Hadoop MapReduce Performance using Dedicated OrangeFS Servers. http://www.datanami.com/2013/09/09/accelerate_hadoop_mapreduce_performance_using_dedicated_orangefs_servers.html.
[2] Amazon EC2. https://aws.amazon.com/ec2/.
[3] Amazon S3. https://aws.amazon.com/cn/s3/.
[4] Apache Hadoop. http://hadoop.apache.org/.
[5] Clemson Palmetto HPC. http://newsstand.clemson.edu/media relations/clemson-supercomputers-ranked-in-top-5-fastest-at-public-universities/.
[6] Cross validation. https://en.wikipedia.org/wiki/Cross-validation_(statistics).
[7] Linear regression. https://en.wikipedia.org/wiki/Linear_regression.
[8] Mean squared error. https://en.wikipedia.org/wiki/Mean_squared_error.
[9] Newegg. http://www.newegg.com/.
[10] Pig progress indicator. http://hadoop.apache.org/pig/.
[11] Using Lustre with Apache Hadoop. http://wiki.lustre.org/images/1/1b/Hadoop_wp_v0.4.2.pdf.
[12] S. Agrawal. The Next Generation of Apache Hadoop MapReduce. Apache Hadoop Summit India, 2011.
[13] R. Ananthanarayanan, K. Gupta, P. Pandey, H. Pucha, P. Sarkar, M. Shah, and R. Tewari. Cloud analytics: Do we really need to reinvent the storage stack? In *Proc. of HOTCLOUD*, 2009.
[14] R. Appuswamy, C. Gkantsidis, D. Narayanan, O. Hodson, and A. Rowstron. Scale-up vs scale-out for hadoop: Time to rethink? In *Proc. of SOCC*, 2013.
[15] R. E. Caflisch. Monte carlo and quasi-monte carlo methods. *Acta numerica*, 7:1–49, 1998.
[16] M. Chen, D. Hildebrand, G. Kuenning, S. Shankaranarayana, B. Singh, and E. Zadok. Newer is sometimes better: An evaluation of nfsv4.1. In *Proc. of SIGMETRICS*, 2015.
[17] Y. Chen, S. Alspaugh, and R. Katz. Interactive Analytical Processing in Big Data Systems: A Cross Industry Study of MapReduce Workloads. In *Proc. of VLDB*, 2012.
[18] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz. The Case for Evaluating MapReduce Performance Using Workload Suites. In *Proc. of MASCOTS*, 2011.
[19] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proc. of OSDI*, 2004.
[20] S. Duan, V. Thummala, and S. Babu. Tuning database configuration parameters with ituned. *Proceedings of the VLDB Endowment*, 2(1):1246–1257, 2009.
[21] K. Elmeleegy. Piranha: Optimizing Short Jobs in Hadoop. In *Proc. of VLDB Endow*, 2013.
[22] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. Cetin, and S. Babu. Starfish: A self-tuning system for big data analytics. In *Proc. of CIDR*, 2011.
[23] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.
[24] V. Jalaparti, H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron. Bridging the tenant-provider gap in cloud services. In *Proc. of SOCC*, page 10. ACM, 2012.
[25] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar. Network-aware scheduling for data-parallel jobs: plan when you can. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 407–420. ACM, 2015.
[26] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. *Machine learning: ECML-98*, pages 137–142, 1998.
[27] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan. An analysis of traces from a production MapReduce cluster. In *Proc. of CCGRID*, 2010.
[28] S. Krishnan, M. Tatineni, and C. Baru. myHadoop-Hadoop-on-Demand on Traditional HPC Resources. Technical report, 2011.
[29] A. Kyrola, G. Blelloch, and C. Guestrin. GraphChi: Large-Scale Graph Computation on Just a PC. In *Proc. of OSDI*, 2012.
[30] Z. Li and H. Shen. Designing a hybrid scale-up/out hadoop architecture based on performance measurements for high application performance. In *Proc. of ICPP*, 2015.
[31] Z. Li and H. Shen. Performance Measurement on Scale-up and Scale-out Hadoop with Remote and Local File Systems. In *Proc. of Cloud*, 2016.
[32] Z. Li, H. Shen, J. Denton, and W. Ligon. Comparing application performance on hpc-based hadoop platforms with local storage and dedicated storage. In *Proc. of BigData*, 2016.
[33] Z. Li, H. Shen, W. B. L. III, and J. Denton. An exploration of designing a hybrid scale-up/out hadoop architecture based on performance measurements. *IEEE Transactions on Parallel and Distributed Systems*, PP(99):1–1, 2016.
[34] H. Lim, D. G. Andersen, and M. Kaminsky. Towards accurate and fast evaluation of multi-stage log-structured designs. In *Proc. of FAST*, 2016.
[35] J. Meza, Q. Wu, S. Kumar, and O. Mutlu. A large-scale study of flash memory failures in the field. In *Proc. of SIGMETRICS*, pages 177–190, 2015.
[36] M. Michael, J. Moreira, D. Shiloach, and R. Wisniewski. Scale-up x scale-out: A case study using nutch/lucene. In *Proc. of IPDPS*, 2007.
[37] Z. Ren, X. Xu, J. Wan, W. Shi, and M. Zhou. Workload characterization on a production Hadoop cluster: A case study on Taobao. In *Proc. of IISWC*, 2012.
[38] S. Rosen, P. Salemi, B. Wickham, A. Williams, C. Harvey, E. Catlett, S. Taghiyeh, and J. Xu. Parallel empirical stochastic branch and bound for large-scale discrete optimization via simulation. In *Proc. of Winter Simulation Conference (WSC)*, pages 626–637, 2016.
[39] F. Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.
[40] I. Stonica. A Berkeley view of big data: Algorithms, Machines and People. UC Berkeley EECS Annual Research Symposium, 2011.
[41] W. Tantisiriroj, S. Patil, G. Gibson, S. W. Son, S. J. Lang, and R. B. Ross. On the Duality of Data-intensive File System Design: Reconciling HDFS and PVFS. In *Proc. of SC*, 2011.
[42] G. Wang, A. R. Butt, P. Pandey, and K. Gupta. A simulation approach to evaluating design decisions in mapreduce setups. In *Proc. of MASCOTS*, pages 1–11. IEEE, 2009.
[43] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, et al. Bigdatabench: A big data benchmark suite from internet services. In *Proc. of HPCA*, 2014.
[44] T. White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.
[45] M. Zaharia, D. Borthakur, S. Sen, K. Elmeleegy, S. Shenker, and I. Stoica. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *Proc. of EuroSys*, 2010.
[46] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica. Improving MapReduce Performance in Heterogeneous Environments. In *Proc. of OSDI*, 2008.

Zhuozhao Li received the B.S. degree in Optical Engineering from Zhejiang University, China in 2010, and the M.S. degree in Electrical Engineering from University of Southern California in 2012. He is currently a Ph.D. student in Department of Computer Science at University of Virginia. His research interests include distributed computer systems, with an emphasis on cloud computing, resource management in cloud networks, and big data processing.



Haiying Shen received the B.S. degree in Computer Science and Engineering from Tongji University, China in 2000, and the M.S. and Ph.D. degrees in Computer Engineering from Wayne State University in 2004 and 2006, respectively. She is currently an Associate Professor in the Department of Computer Science at University of Virginia. Her research interests include distributed computer systems and computer networks, with an emphasis on P2P and content delivery networks, mobile computing, wireless sensor networks, and cloud computing. She is a Microsoft Faculty Fellow of 2010, a senior member of the IEEE and a member of the ACM.