Performance Measurement on Scale-up and Scale-out Hadoop with Remote and Local File Systems

Zhuozhao Li, Haiying Shen, Walter Ligon, and Jeffrey Denton
Department of Electrical and Computer Engineering
Clemson University, Clemson, SC 29631
Email: {zhuozhl, shenh, walt, denton}@clemson.edu

***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. 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.**

I. Introduction

MapReduce [11] 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! [10] and Facebook [18]. 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. Hence, for traditional Hadoop in HPC clusters, we need to first transfer the data from the remote storages to the local storages, which is costly and not time-efficient. This inspires us to configure Hadoop with remote file systems on HPC clusters. For example, 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 [6, 15]. 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) [8, 13]. 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 I: 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.

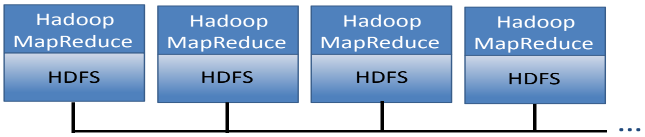


Figure 1. Typical Hadoop with HDFS local storage (HDFS in short).

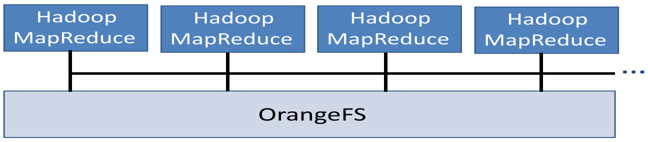


Figure 2. Hadoop with the OrangeFS remote storage (OFS in short).

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. [10, 17]. 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

Table I
DIFFERENT PLATFORMS.

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

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 aim to provide a basic idea that different platforms do provide different performance for different types of applications. In addition, we provide the metrics on how to decide the platforms for different applications. The contributions of this paper are summarized as follows:

(1) We have conducted comprehensive experiments 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.

(2) 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.

(3) Although the performance measurements in this paper are conducted on only one HPC cluster, the users on other HPC clusters also encounter similar situations on selecting different platforms. They can follow the same metrics used in this paper to first characterize the best platforms for different applications. Therefore, our measurement results provide a guidance on how to select the best platform to run different types of applications with different characteristics.

The remainder of this paper is organized as follows. Section II describes the configurations of the four platforms. Section III presents the measurement results of different types of applications and provides an in-depth analysis of the results. Section IV summarizes the observations and further discusses the guidance to cloud environment. Section V gives an overview of the related work. Section VI concludes this paper with remarks on our future work.

II. 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 [3].

A. Introduction of Hadoop MapReduce

MapReduce [11] 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 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.

B. Experiment Environment

In the experiments, we use Hadoop MapReduce version 1.2.1. We use four machines for scale-up Hadoop, each of which is equipped with four 6-core 2.66GHZ Intel Xeon 7542 processors, 505GB RAM, and 91GB hard disk. 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. To achieve fair performance comparison, we require the scale-up and scale-out machines have similar cost. We investigated the cost information from [4] and found that one scale-up machine matches similar price with 6 scale-out machines.

C. Configurations on HDFS and OFS

As we mentioned in Section I, 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 [12], we easily configure Hadoop with HDFS on Palmetto. To achieve fair comparison, for both OFS and HDFS file systems, we use the 10GB Myrinet as the interconnections.

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 HDFS, we set the HDFS block size to 128MB, which is the same as the current production clusters [18]. OFS stores data in simple stripes (i.e., similar as blocks in HDFS) across multiple storage servers in order to facilitate parallel access. To compare OFS fairly with HDFS, we set the stripe size to 128MB. Typically, in current commercial MapReduce cluster [8], 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. 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.

D. Configurations on Scale-up Machines

The scale-out architecture deploys many scale-out machines with poor 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 exert 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. If the memory is full, the data in memory is spilled to the local disk, which introduces overheads. Therefore, it is less likely for the data to be spilled to local disk if the heap size is larger, leading to fewer overheads and better performance. 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 [8], 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 Palmetto, 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.

III. Performance Measurements

In this section, we will compare the performance of data-intensive, CPU-intensive, and I/O-intensive jobs with different input data size on the four platforms as mentioned previously. We expect to provide a guidance for users on how different applications benefit from different platforms.

A. Measured Applications and Metrics

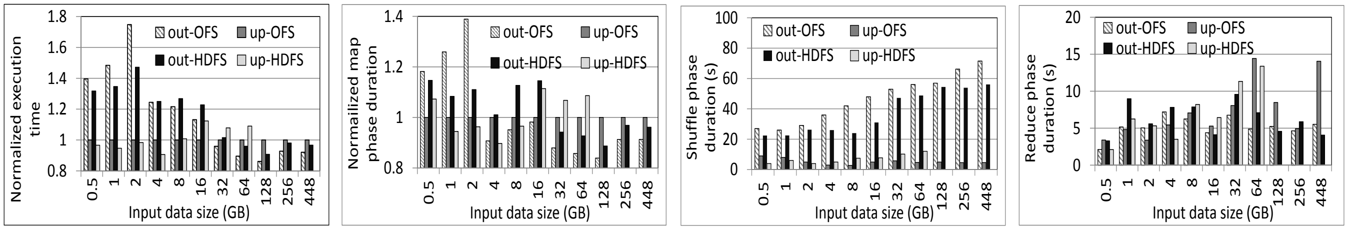
We classify the representative Hadoop benchmarks into three types: data-intensive, I/O-intensive and CPU-intensive in our performance measurement. We can roughly infer the types of applications by the size of the input data, shuffle data and output data. In general, data-intensive applications have large 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*, write and read test of *TestDFSIO*, *PiEstimator*, and matrix multiplication [2].

Among them, *Wordcount* and *Grep* 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. We generated the input data for *Wordcount* and *Grep* from a big data benchmark BigDataBench [17].

The write and read test of *TestDFSIO* are typical I/O-intensive applications. They complete a large amount 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.

The CPU-intensive applications we use in the experiments is *PiEstimator* and matrix multiplication. *PiEstimator* is an application to estimate the value of Pi. The mappers generate a specified number of sample points 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*) calculates the multiplication of two 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:



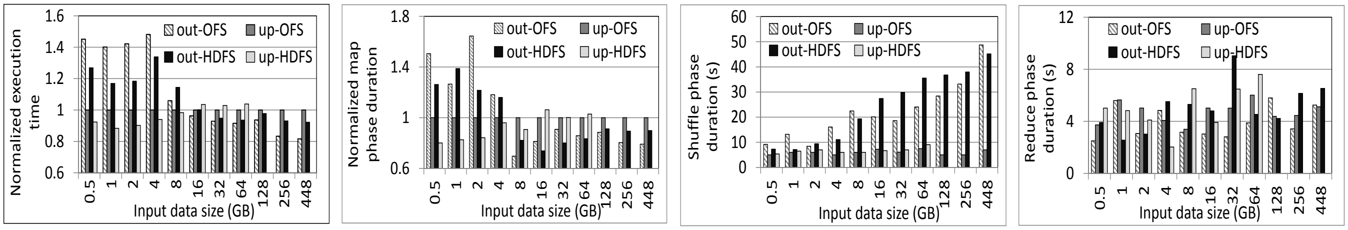
(a) Execution time.

(b) Map phase duration.

(c) Shuffle phase duration.

(d) Reduce phase duration.

Figure 3. Measurement results of data-intensive jobs of *Wordcount*.



(a) Execution time.

(b) Map phase duration.

(c) Shuffle phase duration.

(d) Reduce phase duration.

Figure 4. Measurement results of data-intensive jobs of *Grep*.

- *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 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.

A main advantage of the scale-out machines comparing to the scale-up machines is the more task slots and hence fewer task waves for a job. 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.

B. Data-Intensive Applications

In this section, we show the performance evaluation of data-intensive applications including *Wordcount* and *Grep*. Figures 3(a) and 4(a) show the normalized execution time of *Wordcount* and *Grep* 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 observe several meaningful observations.

We observe that when the input data size is small (0.5-8GB), the performance of *Wordcount* and *Grep* is better on the scale-up machines than the scale-out machines. On the contrary, when the input data size is large ($\geq$16GB), the performance of *Wordcount* and *Grep* 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 $\frac{|OFS-HDFS|}{HDFS}$) 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 interconnect, there is still network latency in OFS, while HDFS benefits from data locality and hence avoids network latency. When the input data size is small, the execution time is relatively small. Therefore, 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.

(2) On the other hand, when the input data size is large, the execution time becomes large and hence the network latency is negligible. 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.

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 3(b) and 4(b) show the normalized map phase duration of *Wordcount* and *Grep*, 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 3(c) and 4(c) show the shuffle phase duration of *Wordcount* and *Grep*, respectively. We see that the shuffle phase duration is always 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.

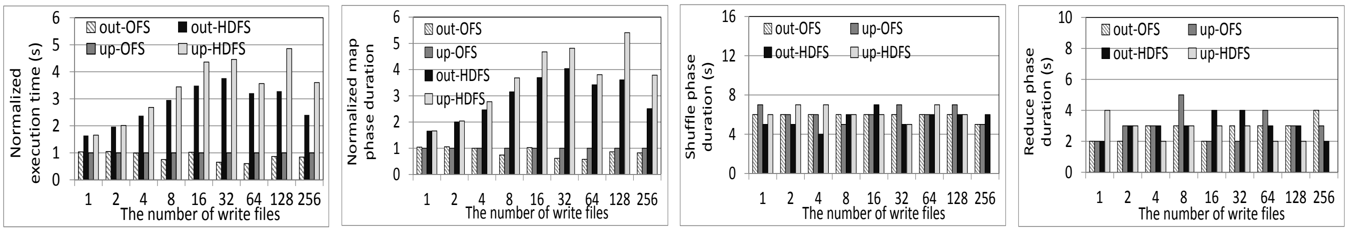
Figures 3(d) and 4(d) show the reduce phase duration of *Wordcount* and *Grep*, respectively. In *Wordcount* and *Grep*, the reduce phase aggregates the map outputs which have small size and hence the reduce phase duration is very short. Therefore, the reduce phase duration of *Wordcount* and *Grep* is around a few seconds and there is not any specific relationship of the reduce phase duration. 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.

C. I/O-Intensive Applications

In this section, we measure the performance of a relatively large data size (80GB). Figures 5(a) and 6(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 5(b), 5(c) and 5(d) and Figures 6(b), 6(c) and 6(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



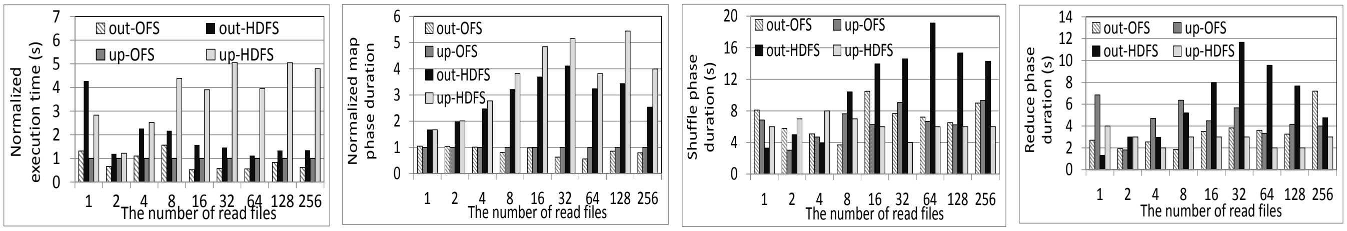
(a) Execution time.

(b) Map phase duration.

(c) Shuffle phase duration.

(d) Reduce phase duration.

Figure 5. 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.

Figure 6. Measurement results of I/O-intensive read test (80GB) of *TestDFSIO*.

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.

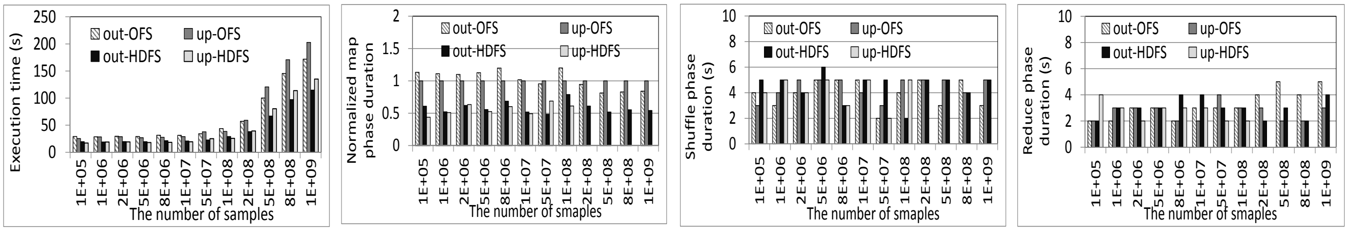
D. CPU-Intensive Applications

Figure 7(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. Therefore, HDFS outperforms OFS for small input data sizes.

Figures 7(b), 7(c) and 7(d) show the map, shuffle and reduce phase durations of *PiEstimator*, respectively. Since the map phase of *PiEstimator* completes the majority work in the jobs (that determine 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



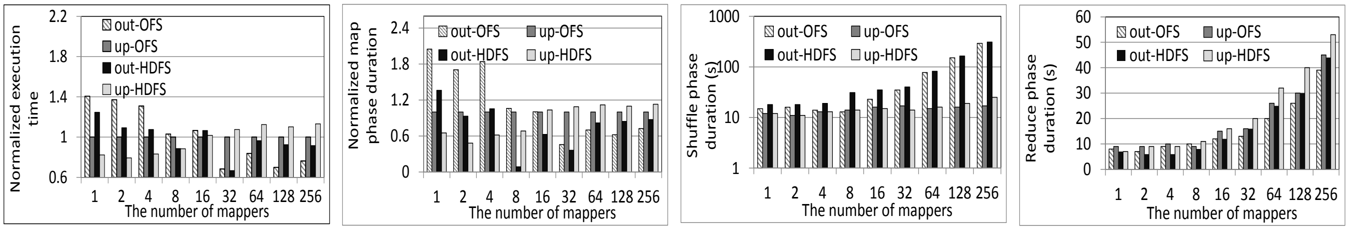
(a) Execution time.

(b) Map phase duration.

(c) Shuffle phase duration.

(d) Reduce phase duration.

Figure 7. Measurement results of CPU-intensive jobs of *PiEstimator*.



(a) Execution time.

(b) Map phase duration.

(c) Shuffle phase duration.

(d) Reduce phase duration.

Figure 8. Measurement results of CPU-intensive jobs of *MM*.

does not affect the durations of these two phases much.

Figure 8(a) shows the normalized execution time of *MM* versus different number of mappers. Note that more mappers mean 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 8(b), 8(c), and 8(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 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, when the number of mappers is small, it is better to use HDFS. On the contrast, 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.

IV. Discussions

A. 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.

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.

V. Related Work

MapReduce [11] 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.* [16] integrated PVFS into Hadoop and compared its performance with HDFS. Other works [5, 7] successfully implement HPC file systems (GPFS and Lustre) in Hadoop. 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.

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 how the performance of clusters. Chen *et al.* [9] 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. Ren *et al.* [14] conducted a case study of the jobs and tasks of the workload from a commodity Hadoop cluster Taobao. Appuswamy *et al.* [8] 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.

VI. 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 expect that our measurement results can help users to select the most appropriate platforms for different applications with different characteristics. Additionally, our results can also provide 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, using 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.

Acknowledgements

This research was supported in part by U.S. NSF grants NSF-1404981, IIS-1354123, CNS-1254006, IBM Faculty Award 5501145 and Microsoft Research Faculty Fellowship 8300751.

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] Apache Hadoop. http://hadoop.apache.org/.

[3] Clemson Palmetto HPC cluster. http://citi.clemson.edu/palmetto/.

[4] Newegg. http://www.newegg.com/.

[5] Using Lustre with Apache Hadoop. http://wiki.lustre.org/images/1/1b/Hadoop_wp_v0.4.2.pdf.

[6] S. Agrawal. The Next Generation of Apache Hadoop MapReduce. Apache Hadoop Summit India, 2011.

[7] 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.

[8] 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.

[9] Y. Chen, S. Alspaugh, and R. Katz. Interactive Analytical Processing in Big Data Systems: A CrossIndustry Study of MapReduce Workloads. In *Proc. of VLDB*, 2012.

[10] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz. The Case for Evaluating MapReduce Performance Using Workload Suites. In *Proc. of MASCOTS*, 2011.

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

[12] S. Krishnan, M. Tatineni, and C. Baru. myHadoop-Hadoop-on-Demand on Traditional HPC Resources. Technical report, 2011.

[13] 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.

[14] 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.

[15] I. Stonica. A Berkeley view of big data: Algorithms, Machines and People. UC Berkeley EECS Annual Research Symposium, 2011.

[16] 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.

[17] 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.

[18] 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.