Designing A Hybrid Scale-Up/Out Hadoop Architecture Based on Performance Measurements for High Application Performance

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

***Abstract*—Since scale-up machines perform better for jobs with small and median (KB, MB) data sizes while scale-out machines perform better for jobs with large (GB, TB) data size, and a workload usually consists of jobs with different data size levels, we propose building a hybrid Hadoop architecture that includes both scale-up and scale-out machines, which however is not trivial. The first challenge is workload data storage. Thousands of small data size jobs in a workload may overload the limited local disks of scale-up machines. Jobs from scale-up and scale-out machines may both request the same set of data, which leads to data transmission between the machines. The second challenge is to automatically schedule jobs to either scale-up or scale-out cluster to achieve the best performance. We conduct a thorough performance measurement of different applications on scale-up and scale-out clusters, configured with Hadoop Distributed File System (HDFS) and a remote file system (i.e., OFS), respectively. We find that using OFS rather than HDFS can solve the data storage challenge. Also, we identify the factors that determine the performance differences on the scale-up and scale-out clusters and their cross points to make the choice. Accordingly, we design and implement the hybrid scale-up/out Hadoop architecture. Our trace-driven experimental results show that our hybrid architecture outperforms both the traditional Hadoop architecture with HDFS and with OFS in terms of job completion time.**

***Keywords*-Hadoop; scale-up; scale-out; remote file system;**

I. Introduction

MapReduce [13] 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 Facebook, Google and Yahoo!. In the last decade, the amount of computation and data increases exponentially [7]. This trend poses a formidable challenge of high performance on MapReduce and motivates many researchers to explore to improve the performance from different aspects such as job scheduling [2], [4], [15], [16], short jobs performance optimization [14] and intermediate data shuffling [6], [11], [12], [22].

A common sense in the IT community is that a larger Hadoop cluster of machines is always better for processing big data, i.e., a very large volume of data in terabytes, petabytes or exabytes. Recent studies indicate that most jobs in the production workloads (e.g., at Facebook [10] and Microsoft [20] clusters) usually have input/shuffle/output sizes in the MB to GB range. These production clusters with many small jobs suffer from poor performance because the existing Hadoop MapReduce clusters were not originally designed for short and latency-sensitive jobs [14]. Therefore, optimizing the performance of short jobs is important. To process the current production workloads, *Appuswamy et al.* [8] claimed that scale-up machines is a better option than scale-out machines. Scale-up is vertical scaling, which means adding more resources to the nodes of a system, typically the processors and RAM. Scale-out is horizontal scaling, which refers to adding more nodes with few processors and RAM to a system.

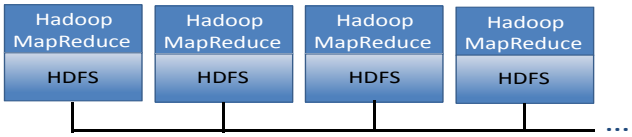


Figure 1: Typical Hadoop with HDFS local storage.

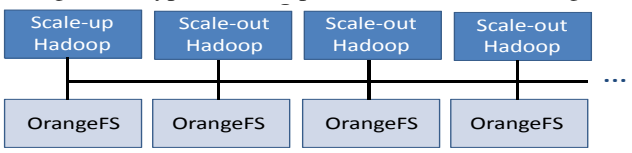


Figure 2: Hybrid scale-up/out Hadoop with a remote storage.

A real-world workload usually consists many jobs handling diverse data size levels and computations. Also, in this big data era, the data size handled by jobs has been increasingly larger. We calculated the Cumulative Distribution Function (CDF) of the data sizes of more than 6000 jobs in the Facebook synthesized workload trace FB-2009 [9] and show the results in Figure 3. We see that the input data size ranges from KB to TB. Specifically, 40% of the jobs process less than 1MB small datasets, 49% of the jobs process 1MB to 30GB median datasets, and the rest 11% of the jobs process more than 30GB large datasets. Such a workload requires both scale-up and scale-out machines to handle datasets with different size levels. Therefore, it is not practical to simply decide to use scale-up machines or scale-out

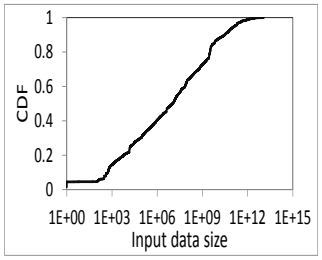


Figure 3: CDF of input data size in the Facebook synthesized trace.

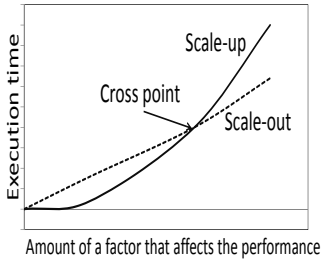


Figure 4: Cross point.

machines for a workload. Motivated by these observations, we see the great potential to design a Hadoop architecture with coexistence of both scale-up and scale-out machines to improve the performance of real-world workloads. However, this task is non-trivial due to two main challenges.

• Proper data storage to enable both scale-up machines and scale-out machines efficiently access data needed. The Facebook trace shows that thousands of jobs (85%) handle small or median data size levels. Since majority jobs are better to run in scale-up machines, these jobs may overload the local disks of scale-up machines. Also, both scale-up and scale-out machines may request the same workload dataset, which leads to data transmission between the machines and may degrade the application performance.

• Adaptively scheduling a job to either scale-up cluster or scale-out cluster that benefits the job the most. Simply referring to job input data size may not be sufficient and there may exist other factors that determine the performance difference between scale-up and scale-out clusters.

In this paper, we aim to investigate the feasibility of designing such a hybrid architecture and start the initial exploration. Hadoop Distributed File System (HDFS) is the file system designed to closely work with Hadoop MapReduce (Figure 1). To handle the first challenge, we could let HDFS consider both scale-out and scale-up machines equally as datanodes for data distribution. However, in this case, scale-up machines must frequently access data in scale-out machines, which not only degrades their job performance but also consumes their bandwidth. We then explore handling this challenge by using a remote dedicated storage system (e.g., OrangeFS [5] (Figure 2). The dedicated storage offloads I/O load from the compute nodes and enables the data sharing between scale-up machines and scale-out machines easily. To handle the second challenge, we need to decide not only the factors that affect the performance difference between scale-up cluster and scale-out cluster but also the *cross points* of the factors for the selection. As shown in Figure 4, the *cross point* is the amount of a factor, at which the scale-up and scale-out clusters provide similar performance and if the actual amount is higher or lower than the cross point, one cluster provides better performance.

Table I: Four architectures in our measurement.

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

To this end, by taking advantage of the Clemson University Palmetto HPC cluster that was successfully configured Hadoop by replacing the local HDFS with the remote OrangeFS (OFS), we configured four architectures as shown in Table I: scale-up machines with OFS (denoted by up-OFS), scale-up machines with HDFS (denoted by up-HDFS), scale-out machines with OFS (denoted by out-OFS), and scale-out machines with HDFS (denoted by out-HDFS). We then measure the performance of representative Hadoop applications (i.e., shuffle-intensive and map-intensive) on these architectures. Through the measurement, we aim to see if the use of a remote file system can provide efficient data storage as we expected and whether it brings about any side-effect to scale-up cluster or scale-out cluster. More importantly, we study the benefits gained from scale-up cluster and scale-out cluster, respectively, for different jobs, based on which we can decide where to run a given job.

Through our performance measurement, we confirm the benefits of the remote file system, identify the factors (i.e., input data size and shuffle/input data size ratio) that determine the performance differences on the scale-up and scale-out clusters and their cross points to make the choice. Accordingly, we design a hybrid scale-up/out Hadoop architecture. In this architecture, different jobs in a workload can be executed on either scale-up or scale-out cluster that benefits them the most, thus achieving higher workload performance. In this paper, we use execution time to evaluate application performance. Our contributions are summarized below:

1. We have identified comparable scale-up cluster and scale-out cluster, built four architectures shown in Table I and optimized their configurations to achieve the best performance by trial of experiments.
2. We have conducted thorough experiments of different applications on the four architectures with different input data sizes and provided an insightful analysis on the performance.
3. Based on our measurement analysis, we design a scheduler, which helps decide whether to execute a job on the scale-up or scale-out cluster to gain the most benefit. We then design a hybrid scale-up/out Hadoop architecture that incorporates this scheduler and uses a remote file system.
4. We have conducted experiments driven by the Facebook synthesized workload trace, which show that our hybrid architecture outperforms both the traditional Hadoop architecture with HDFS and with OFS in terms of the execution time.

As far as we know, our work is the first that i) studies the application performance on the four architectures in Table I, ii) proposes the idea of a hybrid scale-up/out Hadoop

architecture to better serve a real-world workload with jobs handling diverse data size levels and computations, and iii) introduces a method to build the hybrid scale-up/out Hadoop architecture. Our new architecture is only an initial design and has many aspects to improve but we expect it can stimulate many researches on this topic.

The remainder of this paper is organized as follows. Section II describes the configurations of scale-up and scale-out machines for the HPC-based Hadoop. Section III presents the performance measurements for different types of applications on the four architectures. Section IV presents our proposed hybrid scale-up/out Hadoop architecture. Section V presents the trace-driven experimental results of our architecture compared with the traditional Hadoop. Section VI gives an overview of the related work. Section VII concludes the paper with remarks on our future work.

II. Optimization of the HPC-Based Hadoop MapReduce Configurations

In this section, we introduce the details on how we configured Hadoop MapReduce on the Clemson Palmetto HPC cluster, which is ranked as the top five fastest supercomputers at public universities in United States and the 66^{th} fastest supercomputers globally [3]. The HPC cluster makes it easy to build the hybrid scale-up/out Hadoop architecture due to two reasons. First, a HPC center have different kinds of machines with different number of CPU cores and RAM size, which allows us to build the architecture without any further cost to buy new machines.

Second, the configuration of Hadoop with a remote storage makes the coexistence of scale-up and scale-out machines in Hadoop possible. These machines can share the same datasets and access their required data easily without frequent data transmission between machines.

A. Introduction of Hadoop MapReduce

HDFS generally consists of a namenode that manages the metadata of the cluster and multiple datanodes used to store the data blocks. The running process of a MapReduce job is composed of three phases: map, shuffle and reduce. Each node has a specific number of map and reduce slots. Given the input data, HDFS divides it to $\frac{input\ data\ size}{block\ size}$ number of data blocks and stores the blocks into datanodes. In the map phase, the job tracker assigns each mapper to process one data block in a datanode. The output of all the mappers is intermediate data (i.e., shuffle data). In the shuffle phase, the shuffle data is then partitioned and shuffled to corresponding reduce nodes. 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 (which is determined by the heap size), the shuffle data will be spilled to local disk. In the reduce phase, the reducers aggregate the shuffle data and generate the final output.

B. Hadoop MapReduce on HPC Cluster

We use myHadoop [18] to automatically configure Hadoop on the Clemson Palmetto HPC cluster. Recently, a Java Native Interface (JNI) shim layer has been implemented on the Clemson Palmetto HPC cluster that allows Hadoop MapReduce to store input/output on a remote storage file system (i.e., OFS) directly. OFS is a parallel file system that distributes data across multiple servers. The remote storage in general has much faster I/O performance than local disks [1]. Moreover, because of the centralization of remote storage, it is much easier to manage and maintain than the local storage. With OFS, no modifications to the Hadoop source code and MapReduce jobs are required.

C. Experiment Environment

In the experiments, we use Hadoop version 1.2.1. We use two machines for scale-up Hadoop MapReduce. Each scale-up machine is equipped with four 6-core 2.66GHZ Intel Xeon 7542 processors, 505GB RAM, 91GB hard disk, and 10Gbps Myrinet interconnections. The scale-out cluster consists of twelve machines, each of which has two 4-core 2.3GHZ AMD Opteron 2356 processors, 16GB RAM, 193GB hard disk, and 10Gbps Myrinet interconnections.

The reason that we select two scale-up machines and twelve scale-out machines is because it makes the scale-up and scale-out clusters have the same price cost (according to the investigation of market), thus makes the performance measurements comparable. Previous research [8] used only one scale-up machine and hence did not consider the network performance between scale-up machines in the performance study. In order not to exclude the network factor in the performance study, we use two scale-up machines and comparably 12 scale-out machines.

We compare the performance of different types of applications on four architectures in Table I. For the HDFS configuration, if one of the machines acts as both namenode and datanode, it will degrade the performance of Hadoop. Since OFS itself has metadata servers, in order to achieve fair comparison, we use an additional machine to serve as namenode in HDFS.

D. Hadoop Configurations

It is important for us to optimize the configurations of both scale-up and scale-out machines, either with OFS or HDFS, to achieve the best application performance of the four architectures.

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 overhead. By default, the heap size is 200MB for each JVM. Current modern servers (regardless of scale-out or scale-up machines) always provide sufficient memory for us to increase the heap size to reduce the overhead and

improve the JVM performance. However, if the heap size is too large, the memory used for heap is wasted and the out of memory error may occur. To achieve the best performance and also avoid the out of memory error [8] in our experiments, through trial and error, we set the heap size to 8GB per task on scale-up machines, and to 1.5GB and 1GB for shuffle-intensive and map-intensive applications on scale-out machines, respectively.

Map and reduce slots To ensure the best performance, the total number of map and reduce slots is set to the number of cores for both scale-up and scale-out machines. For example, if we use machines with 4-core CPUs, the sum of map and reduce slots is equal to 4. 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.

Remote file system strip size In HDFS, a file is broken into small blocks and each data block is processed by one map task. It is important to set the block size properly, which cannot be too small or too large. We set the HDFS block size to 128MB to match the setting in the current industry clusters. Similarly, OFS stores data in simple stripes (i.e., similar as blocks in HDFS) across multiple storage servers in order to facilitate parallel access. The stripe size is 4KB in default. In order to compare OFS fairly with HDFS, we also set the stripe size to 128MB.

The number of remote storage servers There are 32 remote storage servers in OFS in the Clemson HPC cluster. Each remote storage server consists of 2 SSD disks with a RAID-1 configuration and 5 SATA disks (ST32000644NS) with a RAID-5 configuration. The two SSD disks are used for storing metadata and the operating system for the remote storage servers, while the 5 SATA disks are used to store data. These 32 remote storage servers are connected with high-speed interconnection Myrinet, which is a high-speed local area network and has much lower protocol overhead than standard Ethernet. Currently, these 32 remote storage servers are sufficient to provide low latency for around 2000 machines on Palmetto and hence we do not need to worry that this remote file system is not scalable to large Hadoop MapReduce clusters. In our experiments, since each file in the input data is not large (maximum 1GB) and the stripe size is set to 128MB, we do not need to use the full 32 remote storage servers. Thus, in our experiments, we use 8 (1GB/128MB) remote servers to store each file in parallel.

Replication factor of HDFS In traditional large cluster Hadoop MapReduce with HDFS, the replication factor is set to 3, which means that each file block has three replicas. In general, the first replica is in one node; the second replica is placed in a different rack from the node and the third replica is placed in the same rack as the second replica. A large HDFS runs on a cluster of computers that commonly spread across many racks. Therefore, setting the replication factor to 3 for large cluster helps spread the data to two different racks. However, because of the relatively small scale of Hadoop cluster in our experiments (i.e., 12 machines), the machines of our cluster are in the same rack, which means it is not necessary to set the replication factor to 3. To reduce the bandwidth consumption and time to replicate data and also spread replicas among nodes to help achieve high data locality, we set the replication factor of HDFS to 2. For the remote file system OFS, it currently does not support build-in replications. However, it does not affect our measurement performance since data loss never occurs in OFS during our experiments.

RAM drive of scale-up machines The scale-up machines provide a large amount of memory size (i.e., 505GB in the experiments), which is an advantage of the scale-up machines. Even though we set the heap size to 8GB, there is much memory space left. To fully take advantage of the unused memory in the scale-up machines, Palmetto enables to use half of the total memory size as *tmpfs*, which serves the same functions as RAMdisk. On the other hand, since the memory size is limited on the scale-out machines (i.e., 16GB), we do not use memory as RAMdisk.

Shuffle data placement Although the Clemson Palmetto HPC cluster allows us to configure Hadoop with remote file system OFS, we only can place input and output data to OFS, but cannot place shuffle data to OFS, which requires the modification of Hadoop MapReduce code. Then, we still need to utilize the local file system to store the shuffle data. For the scale-up machines in our experiments (either with HDFS or OFS), we place the shuffle data on RAMdisks, which improves the shuffle data I/O performance greatly. For scale-out machines, we store the shuffle data in the local disks (i.e., HDD).

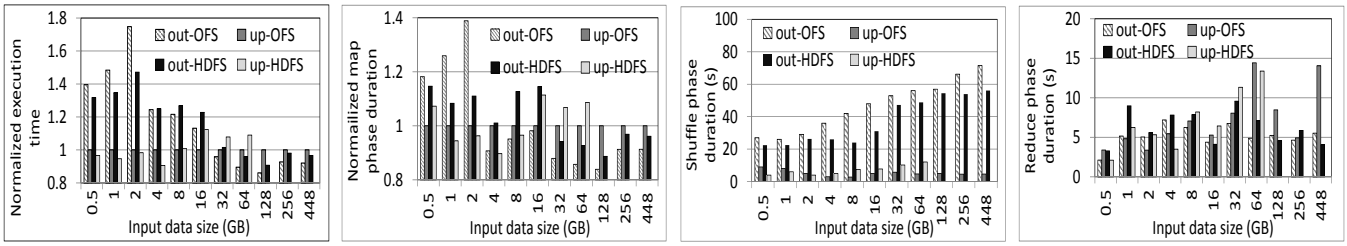
III. Performance Measurement of Applications

In this section, we compare the performance of shuffle-intensive and map-intensive jobs with different input data sizes on the four architectures in Table I. Shuffle-intensive applications have large shuffle data size, while map-intensive applications generally do not contain large shuffle data size. We expect to provide a thorough analysis on how different applications benefit from scale-up and scale-out clusters, and on remote storage and local storage respectively.

A. Types of Applications

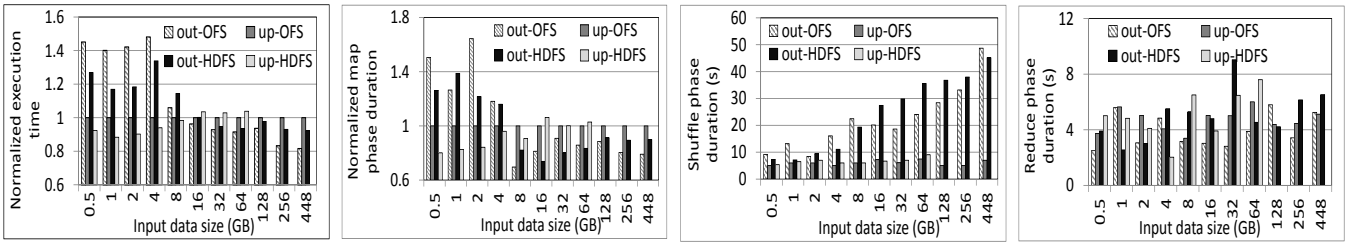
The applications we use include *Wordcount*, *Grep*, and the write test of *TestDFSIO*. Among them, *Wordcount* and *Grep* are typical shuffle-intensive applications. Specifically, *Wordcount* and *Grep* have only relatively large input and shuffle size but small output size. We generated the input data by BigDataBench [21] based on the Wikipedia datasets for *Wordcount* and *Grep*. The write test of *TestDFSIO* is typical map-intensive applications. We measure the following metrics of each job:

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



(a) Execution time. (b) Map phase duration. (c) Shuffle phase duration. (d) Reduce phase duration.

Figure 5: Measurement results of shuffle-intensive *Wordcount*.



(a) Execution time. (b) Map phase duration. (c) Shuffle phase duration. (d) Reduce phase duration.

Figure 6: Measurement results of shuffle-intensive *Grep*.

• *Map phase duration* calculated by the last map task's ending time minus the first map task's starting time.
• *Shuffle phase duration* calculated by the last shuffle task's ending time minus the last map task's ending time.
• *Reduce phase duration*, which is the time elapsed from the ending time of the last shuffle task to the end of the job.

Note that due to the limitation of local disk size, up-HDFS cannot process the jobs with input data size greater than 80GB.

Since the execution time and map phase duration of jobs with different input data sizes differs greatly, it is difficult to see the experimental results of small data sizes in the figures. Therefore, we normalize execution time and map phase duration results by the results of up-OFS, since we only focus on the performance comparison among up-OFS, up-HDFS, out-OFS, and out-HDFS, rather than the exact execution time or map phase duration. For example, if a job running on up-OFS and up-HDFS has an execution time of 10 and 20 seconds, respectively, then up-OFS on the figure is shown as 1, while up-HDFS on the figure is shown as 2. And we only need to know from the figure that up-OFS has better performance than up-HDFS.

B. Performance of Shuffle-Intensive Applications

Figures 5(a) and 6(a) show the execution time of *Wordcount* and *Grep* versus different input data sizes, respectively. We see that when the input data size is small (0.5-8GB), the performance of *Wordcount* and *Grep* all follows: up-HDFS>up-OFS>out-HDFS>out-OFS. Recall that the number of required map slots equals $\lceil \frac{input\ data\ size}{block\ size} \rceil$. The scale-up machines have better performance when the input data size is small because of three reasons. First, the two scale-up machines can provide the majority of required map slots of the small jobs (defined as jobs with small input data size), which means that the jobs can be completed in only a few task waves. 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. Thus, small jobs can benefit more from more powerful CPU resources of the scale-up machines than from scale-out machines. Second, these jobs are all shuffle-intensive and their performance is very related to the memory resource. The map outputs are copied to the reduce nodes' memory, which is limited by the heap size. A larger heap size makes it less likely to spill the map outputs to the local disks. The more memory resource of scale-up machines provides larger heap size and hence enhances the performance of these shuffle-intensive applications. Third, for the shuffle data placement, the RAMdisks in the scale-up machines are much faster than the local disks in the scale-out machines.

When the input data size is small, the performance of out-HDFS is around 20% (calculated by $\frac{OFS-HDFS}{HDFS}$) better than out-OFS, and up-HDFS is around 10% better than up-OFS. Although a remote file system has better I/O performance than HDFS [1], its advantage cannot be shown for small jobs. This is caused by the network latency in the communication with the remote file storage, which is independent on the data size. When the data size is small, the execution time is also small and the network latency occupies a relatively high portion of the total execution time. Then, the performance of the remote file system becomes

slightly worse than the local file system. However, we see that up-OFS performs around 10-25% better than out-HDFS, which means that scale-up Hadoop with remote file system outperforms the traditional scale-out Hadoop with HDFS.

We also see from the figures that when the input data size is large (>16GB), the performance of *Wordcount* and *Grep* follows out-OFS>out-HDFS>up-OFS>up-HDFS. It means that scale-out machines are better for the shuffle-intensive jobs with large input data size (i.e., large jobs) than scale-up machines. The reason is that a large input data size usually requires a large number of map slots, which however is the primary bottleneck of scale-up machines though they have more powerful CPU resources. The requirements of more map slots and less task waves of large jobs make them benefit more from the scale-out machines. OFS performs better than HDFS because the more powerful dedicated remote servers in OFS and the high speed HPC interconnections (i.e., 10Gbps Myrinet) can provide a higher I/O performance than HDFS.

Furthermore, we see from the figures that as the input data size increases, the performance on scale-up machines decreases while the performance on scale-out machines increases. The two applications have different performance degrading speed on scale-up machines as the input data size increases though they are all shuffle-intensive applications. The cross points of input data size of *Wordcount* and *Grep* are close to 32GB and 16GB, respectively. That is, when the input data size of a *Wordcount* (or *Grep*) is smaller than 32GB (or 16GB), then it performs better in scale-up machines, otherwise, it performs better in scale-out machines. This cross point difference between the jobs is caused by the different shuffle/input ratio calculated by $\frac{shuffle\ data\ size}{input\ data\ size}$. Given the same input data size, if a job's shuffle data size is large, it can benefit more from the large memory and fast RAMdisk of the scale-up machines in the shuffle phrase, thus reduces the shuffle phase duration. In our experiments, regardless of the input data size of the jobs, the shuffle/input ratio of *Wordcount* and *Grep* are always around 1.6 and 0.4, respectively. Therefore, the larger shuffle data size of *Wordcount* leads to slower application performance degradation on the scale-up machines when the input data size increases and a larger cross point than other applications.

To illustrate the cross points of *Wordcount* and *Grep*, we draw Figure 7, which shows the normalized execution time of each job on the scale-out cluster by its execution time on the scale-up cluster (e.g., $\frac{execution\ time\ on\ scale-out(Grep)}{execution\ time\ on\ scale-up(Grep)}$), denoted by out-OFS-Wordcount and out-OFS-Grep, respectively. *Wordcount* with a larger shuffle/input ratio has a higher cross point (i.e., near 32GB) than the cross point (i.e., near 16GB) of *Grep* with smaller shuffle/input ratios. A higher shuffle/input ratio leads to a higher cross point, and vice versa. Near the cross point, the benefit from scale-out cluster due to large input data size equals the benefit from

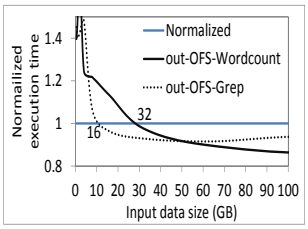


Figure 7: Cross points of *Wordcount* and *Grep*.

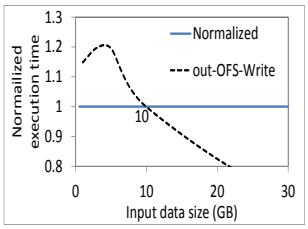


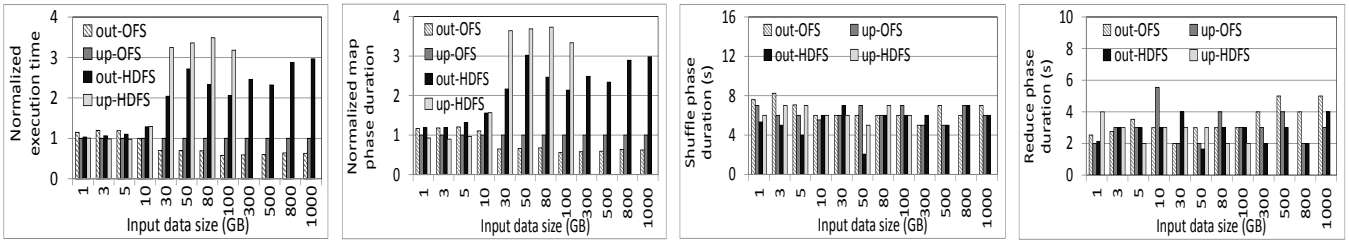
Figure 8: Cross point of write test of *TestDFSIO*.

scale-up cluster due to large shuffle data size. When the input data size is smaller than the cross point, scale-up cluster is a better choice, otherwise, scale-out cluster is a better choice.

Since the execution time is determined by the durations in the map, shuffle and reduce phases, we then study these broken-down durations. Figures 5(b) and 6(b) show the map phase duration of *Wordcount* and *Grep*, respectively. We see that the map phase duration of these jobs has similar performance trends and order as the job execution time. 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. This is mainly because the map phase duration consists of more waves of map tasks on scale-up machines than on scale-out machines. Comparing OFS and HDFS in either scale-up or scale-out machines, we see that when the input data size is between 0.5 and 8GB, the map phase duration of these jobs are 10-50% shorter on HDFS. However, up-OFS still outperforms out-HDFS by 10-25% because of the higher benefits from scale-up machines for small jobs. When the input data size is larger than 16GB, the map phase duration is 10-40% shorter on OFS than on HDFS, no matter on the scale-up or scale-out cluster.

Figures 5(c) and 6(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 the shuffle phase benefits from the larger memory resource and the RAMdisk of scale-up machines. We then can conclude from the map phase and shuffle phase figures: the cross point appears when the benefit of shuffle phase from scale-up machines is not able to compensate the drawback of scale-up machines due to fewer map and reduce slots.

Figures 5(d) and 6(d) show the reduce phase duration of *Wordcount* and *Grep*, respectively. The reduce phase of *Wordcount* and *Grep* just aggregates the map outputs, which have small size. Therefore, *Wordcount* and *Grep* use only a few seconds during reduce phase after the shuffle phase. We see neither OFS nor HDFS affects the reduce phase duration of *Wordcount* and *Grep*, since the reduce phase duration is small.



(a) Execution time. (b) Map phase duration. (c) Shuffle phase duration. (d) Reduce phase duration.

Figure 9: Measurement results of map-intensive write test of *TestDFSIO*.

C. Performance of Map-Intensive Applications

In this section, we show the performance evaluation of the write test of *TestDFSIO*. In the write test, each map task is responsible for writing a file. Therefore, the number of mappers is equal to the number of files. There is only one reduce task, which collects and aggregates the statistics of the map tasks, such as completion time of write and file size.

Figure 9(a) shows the normalized execution time of *TestDFSIO* write test versus input data size, respectively. Again, scale-up machines are the best for small jobs (1-5GB) because small jobs do not require many mappers and scale-up machines can provide the required map slots and better CPU than scale-out machines. However, the execution time difference of *TestDFSIO* between scale-up and scale-out is not as significant as the shuffle-intensive applications. This is because *TestDFSIO* is map-intensive application, which do not have large shuffle data. Therefore, the large memory benefit of scale-up machines for improving shuffle phase duration is not exhibited in map-intensive applications.

On the other hand, for large input data size ($\geq$10GB) (i.e., a large number of mappers), the performance follows out-OFS>up-OFS>out-HDFS. For large jobs, running on scale-out machines with remote storage is better than scale-up machines with remote storage because scale-out machines have more map slots and large jobs can be completed in fewer task waves. Running on scale-out machines with local storage results in the worst performance because the local disks are slower than the remote storage and the remote file system OFS can provide higher I/O performance than HDFS.

Figures 9(b), 9(c) and 9(d) show the map, shuffle and reduce phase durations of the write test, respectively. Since the map phase of *TestDFSIO* completes the majority work in the jobs, while the shuffle phase only collects the statistics and the reduce phase simply aggregates the results, we see that in the write 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 they exhibit no specific relationships and are not affected by either OFS or HDFS. Comparing OFS and HDFS in either the scale-up or scale-out cluster, when the input data size is small (1-5GB), HDFS leads to 10-20% shorter map phase duration. However, up-OFS still generates 5-15% shorter map phase duration than out-HDFS. When the input data size is large ($\geq$10GB), OFS leads to $50-80\%$ shorter map phase duration, a significant improvement.

We conclude that for map-intensive jobs in our experiment environment, if the input data size is small (1-5GB), the scale-up machines are the better choice because of better CPU. On the other hand, if the input data size is large ($\geq$10GB), scale-out machines can achieve better performance because of more map and reduce slots and better I/O performance of OFS over HDFS.

Figure 8 shows the normalized execution time of the write (denoted by out-OFS-Write) of *TestDFSIO* on the scale-out cluster by its execution time on the scale-up cluster, respectively. We see that the cross point is around 10GB for both tests. Since the shuffle size (in KB) is negligible (which makes the shuffle/input ratio close to 0) in both tests, these map-intensive jobs benefit little from the scale-up machines during the shuffle phase. We conclude that the cross point for map-intensive applications is smaller than shuffle-intensive applications.

Conclusions

- Input data size and shuffle/input ratio affect the benefits gained from scale-up and scale-out clusters. When the input data size is small, the scale-up cluster outperforms the scale-out cluster, and when the input data size is large, the scale-out cluster outperforms scale-up machines. The cross point of the input data size depends on the shuffle data size; a larger shuffle size leads to more benefits from the scale-up machines and vice versa.
- Hadoop with a remote file system improves the performance of Hadoop with HDFS when jobs are scheduled to run on the scale-up or scale-out cluster to gain more benefits.

IV. A Hybrid Scale-up/out Hadoop Architecture

A traditional Hadoop cluster only has scale-out machines in general. However, we have demonstrated in Section III that different jobs perform better on scale-up machines or on scale-out machines. In current real-world workloads, the jobs are increasingly diverse mix of computations and data size levels. Recall that in Section I improving the performance

of small jobs is important for production clusters. Therefore, we propose a new Hadoop architecture including both scale-up and scale-out machines to achieve better performance for real-world workloads.

As indicated in Section I, there exist two main challenges for building such a new architecture: data storage and scale-up or scale-out selection. For the data storage challenge, HDFS and storing all data blocks needed by scale-up machines in themselves are not efficient methods. Besides, this latter solution requires the modification of the function of the namenode to decide where to store the data blocks based on the jobs handling them. Since usually data is distributed before jobs are launched, this solution introduces not only extra overhead and but also complexity to the namenode's function. We then use a remote file system (e.g., OFS) for the hybrid Hadoop architecture, the feasibility and advantage of which are shown in Section III. Since both scale-up and scale-out machines can be mounted with the same remote file system on HPC, jobs can read/write data no matter they are scheduled to the scale-up or scale-out clusters without data transmission between machines. Moreover, using a remote file system allows us to implement the hybrid architecture without modifying the namenode code for data distribution. Intuitively, it seems that this hybrid architecture improves the performance of small jobs at the cost of the performance of large jobs because the traditional Hadoop cluster has more map and reduce slots than the hybrid architecture. However, we demonstrate in Section V that even with the hybrid architecture, the performance of large jobs is improved due to better I/O performance of OFS over HDFS and less slot competition for large jobs.

To handle the second challenge, we leverage our observations in Section III to make the decisions based on job characteristics. Our experiments show that when a job has shuffle/input ratio between 0.4 and 1, if the input data size is smaller than 16GB, scale-up is a better choice, otherwise, scale-out is a better choice. When a job has shuffle/input ratio equals 1.6, if the input data size is smaller than 32GB, scale-up is a better choice, otherwise, scale-out is a better choice. We generalize 1.6 to ratios greater 1. We consider jobs with shuffle/input ratios less than 0.4 as map-intensive jobs, for which when the input data size is smaller than 10GB, scale-up is a better choice, otherwise, scale-out is a better choice. Based on these observations, we design an algorithm to decide whether scale-up or scale-out is a better choice for a given job based on its shuffle/input ratio and input data size.

We assume that the shuffle/input ratio is input by the users, which means that either the users once ran the jobs before or the jobs are well-known as map-intensive. If the users do not know the shuffle/input ratio of the jobs anyway, we treat the jobs as map-intensive (i.e., shuffle/input ratio less than 0.4) and hence the cross points of the jobs are smaller (i.e., 10GB). This is because we need to avoid scheduling any large jobs to the scale-up machines. Otherwise, it would result in performance degradation of small jobs. The pseudo-code of this algorithm is shown in Algorithm 1. We implemented this scheduler in Linux bash script.

Note the cross points in our algorithm are from the measurement results from our architecture configurations. A fine-grained ratio partition can be conducted from more experiments with other different jobs to make the algorithm more accurate. Also, different configurations of scale-up and scale-out machines will lead to different cross point results. In this paper, we only attempt to show the factors affecting the scale-up and scale-out selection and how they affect the selection decision, and provide a method to design the selection algorithm for the hybrid architecture. Other designers can follow the same method to measure the cross points in their systems and develop the hybrid architecture.

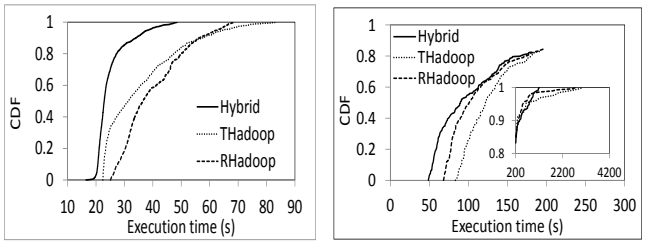
Algorithm 1 Selecting scale-up or scale-out for a given job.

Inputs: $NextJob$: next job in the queue
$Input$: input data size of the job
S/I: shuffle/input ratio

while $NextJob$ exists in the job queue **do**
  if $shuffle/input\ ratio > 1$ **then**
    if $InputDataSize < 32GB$ **then**
      Scale-up $\leftarrow NextJob$
    else
      Scale-out $\leftarrow NextJob$
    end if
  else if $0.4 \leq shuffle/input\ ratio \leq 1$ **then**
    if $InputDataSize < 16GB$ **then**
      Scale-up $\leftarrow NextJob$
    else
      Scale-out $\leftarrow NextJob$
    end if
  else
    if $InputDataSize < 10GB$ **then**
      Scale-up $\leftarrow NextJob$
    else
      Scale-out $\leftarrow NextJob$
    end if
  end if
end while

V. Performance Evaluation

In this section, we use the Facebook synthesized workload FB-2009 [10] to evaluate the performance of our hybrid scale-up/out Hadoop architecture compared with traditional Hadoop architecture. The CDF of input data size of this workload is shown in Figure 3. We see that more than 80% of jobs have an input data size less than 10GB. Due to the limitation of space, please refer the other characteristics of this workload to [10]. We used 2 scale-up machines and 12 scale-out machines to deploy the hybrid scale-up/out architecture with the OFS remote file system (Hybrid in short), and the hardware configurations of these machines



(a) Execution time of scale-up jobs. (b) Execution time of scale-out jobs.

Figure 10: Measurement results of the Facebook workload experiment.

are the same as explained in Section III. As a baseline, we deployed a traditional Hadoop cluster (THadoop in short) with HDFS and a Hadoop cluster with remote file system OFS (RHadoop in short) using 24 scale-out machines (which have comparably the same total cost as the machines in the hybrid architecture) and one additional namenode. Since the trace workload is synthesized from a 600-machine cluster and we did our experiments on 24 machines, we shrank the input/shuffle/output data size of the workload by a factor of 5 to avoid disk insufficiency. We ran the Facebook workload consecutively on these two architectures based on the job arrival time in the traces. We refer to the jobs that are scheduled to scale-up cluster and scale-out cluster by our scheduler as *scale-up jobs* and *scale-out jobs*, respectively.

Figure 10(a) shows the CDF of the execution time of the scale-up jobs in the workload. We see that the execution time distribution of Hybrid is much broader than THadoop and RHadoop. Their maximum execution time for scale-up jobs is 48.53s, 83.37s and 68.17s, respectively. This result demonstrates the effectiveness of the hybrid architecture in improving the performance of small jobs. RHadoop has the worst performance because OFS performs worse than HDFS for small input data sizes on the scale-out Hadoop.

Figure 10(b) shows the CDF of the execution time of the scale-out jobs in the workload. In order to illustrate the figure clearly, we only show the scale-out jobs with execution time less than 200s in the main figure, and use the embedded small figure to show those with execution time greater than 200s. For scale-out jobs, the maximum execution time is 1207s, 3087s and 2734s on Hybrid, THadoop and RHadoop, respectively. The percent of jobs completed after 1207s on THadoop and RHadoop are 4.4% and 1.4%, respectively. Benefitting from the higher I/O performance of OFS, RHadoop outperforms THadoop for large input sizes.

Intuitively, it seems that we improve the performance of scale-up jobs at the cost of the performance of scale-out jobs and the scale-out jobs should have better performance on THadoop because THadoop has more map and reduce slots than Hybrid. However, Figure 10(b) shows that Hybrid still outperforms THadoop and RHadoop even for scale-out jobs, indicating that Hybrid improves not only the performance of scale-up jobs but also the scale-out jobs. There are two main reasons. The first reason is because Hybrid is configured with the remote file system, which provides better performance for jobs with large input sizes than HDFS, as shown in Section III. The second reason is that although there are more map and reduce slots in THadoop, a large amount of scale-up jobs in the workload occupy the slots and have poor performance due to less powerful CPU, thus resulting in a long time before releasing the occupied slots to scale-out jobs. On the contrary, in Hybrid, all the scale-up jobs are run on the scale-up cluster, while scale-out jobs run on the scale-out cluster. Therefore, scale-out jobs in Hybrid do not need to compete with scale-up jobs for slots or with other scale-out jobs because only 15% of the jobs in the workload are scale-out jobs. Similarly, this is also the reason that Hybrid outperforms RHadoop for scale-out jobs. Therefore, scale-out jobs in Hybrid can always be provided with more map and reduce slots than THadoop or RHadoop, resulting in the best performance in Hybrid.

VI. Related Work

MapReduce [13] has been a popular framework that performs parallel computations on big data. Cluster provisioning, configuring and managing for the Hadoop clusters is essential, which requires thorough understanding of the workloads. Recently, there are many efforts devoted to characterizing the workloads in real-world cluster. Chen *et al.* [10] analyzed two production MapReduce traces from Yahoo! and Facebook in order to establish a vocabulary for describing MapReduce workloads. Ren *et al.* [19] characterized the workload of Taobao production Hadoop cluster to provide an understanding of the performance and the job characteristics of Hadoop in the production environment. Kavulya *et al.* [17] analyzed MapReduce logs from the $M45$ supercomputing cluster. Appuswamy *et al.* [8] conducted an evaluation of representative Hadoop jobs on scale-up and scale-out machines, respectively. They found that scale-up machines achieve better performance for jobs with data size at the range of MB and GB. Our work is different from the above workload characterization works is that our work is the first that compare the performance of the Hadoop workloads in the four architectures shown in Table I. Many works focus on improving the performance of the MapReduce clusters from different aspects such as job scheduling [2], [4], [15], [16], intermediate data shuffling [6], [11], [12], [22] and improving small job performance [14]. Unlike these previous works that focus on improving the performance of traditional Hadoop, our work focuses on designing a new hybrid scale-up/out Hadoop architecture that fully utilizes both the advantages of scale-up and scale-out machines for different jobs in a workload to improve its performance.

VII. Conclusion

Since a real-world workload usually has many jobs with increasingly diverse of computations and data size, solely using either scale-up or scale-out cluster to run a workload cannot achieve high performance. In this paper, we explore building a hybrid scale-up/out Hadoop architecture. However, building such an architecture faces two main challenges. First, how to distribute data blocks of a workload dataset to avoid degrading node performance caused by limited local disk or data transmission between nodes. Second, how to decide whether to use scale-up or scale-out cluster for a given job. To handle these challenges, we have conducted performance measurement of different applications on four HPC-based Hadoop platforms: scale-up machines with OFS, scale-out machines with OFS, scale-up machines with HDFS, and scale-out machines with HDFS. Based on our measurement results, we design a hybrid scale-up/out Hadoop architecture, which uses a remote file system rather than HDFS and has a scheduler to determine using scale-up or scale-out for a given job to achieve better performance. We conducted experiments driven by the Facebook synthesize workload to demonstrate that this new architecture outperforms both the traditional Hadoop with HDFS and with OFS. This is the first work that proposes the hybrid architecture. We consider our work as a starting point and expect it will stimulate many other works on this topic. In future work, we will try to optimize the scheduler such as the load balancing between the scale-up machines and scale-out machines. For example, if many small jobs arrive at the same time without any large jobs, all the jobs will be scheduled to the scale-up machines, resulting in imbalance allocation of resources between the scale-up and scale-out machines.

Acknowledgements

We would like to thank Mr. Jeffrey Denton, Dr. Walter Ligon, and Mr. Matthew Cook for their insightful comments. This research was supported in part by U.S. NSF grants NSF-1404981, IIS-1354123, CNS-1254006, CNS-1249603, 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. [Accessed in Dec. 2014].

[2] Capacity Scheduler. http://hadoop.apache.org/docs/r1.2.1/capacity_scheduler.html. [Accessed in Dec. 2014].

[3] Clemson Palmetto. http://newsstand.clemson.edu/media relations/clemson-supercomputers-ranked-in-top-5-fastest-at-public-universities. [Accessed in Dec. 2014].

[4] Fair Scheduler. http://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html. [Accessed in Dec. 2014].

[5] OrangeFS. http://www.orangefs.org. [Accessed in Dec. 2014].

[6] Using Lustre with Apache Hadoop. http://wiki.lustre.org/images/1/1b/Hadoop_wp_v0.4.2.pdf. [Accessed in Dec. 2014].

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

[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] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. Mapreduce online. In *Proc. of NSDI*, 2010.

[12] P. Costa, A. Donnelly, A. Rowstron, and G. O'Shea. Camdoop: Exploiting in-network aggregation for big data applications. In *Proc. of NSDI*, 2012.

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

[14] K. Elmeleegy. Piranha: Optimizing Short Jobs in Hadoop. In *Proc. of VLDB Endow*, 2013.

[15] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource Packing for Cluster Schedulers. In *Proc. of SIGCOMM*, 2014.

[16] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proc. of NSDI*, 2011.

[17] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan. An analysis of traces from a production MapReduce cluster. In *Proc. of CCGRID*, 2010.

[18] S. Krishnan, M. Tatineni, and C. Baru. myHadoop-Hadoop-on-Demand on Traditional HPC Resources. Technical report, 2004.

[19] 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.

[20] A. Rowstron, D. Narayanan, A. Donnelly, G. O'Shea, and A. Douglas. Nobody ever got fired for using hadoop on a cluster. In *Proc. of HotCDP*, 2012.

[21] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang, et al. Bigdatabench: A big data benchmark suite from internet services. In *Proc. of HPCA*, 2014.

[22] Y. Wang, C. Xu, X. Li, and W. Yu. Jvm-bypass for efficient hadoop shuffling. In *Proc. of IPDPS*, 2013.