Comparing Application Performance on HPC-based Hadoop Platforms with Local Storage and Dedicated Storage

Zhuozhao Li, Haiying Shen
Department of Computer Science
University of Virginia, Charlottesville, 22904
Email: {zl5uq, hs6ms}@virginia.edu

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

***Abstract*—Many high-performance computing (HPC) sites extend their clusters to support Hadoop MapReduce for a variety of applications. However, HPC cluster differs from Hadoop cluster on the configurations of storage resources. In the Hadoop Distributed File System (HDFS), data resides on the compute nodes, while in the HPC cluster, data is stored on separate nodes dedicated to storage. Dedicated storage offloads I/O load from the compute nodes and provides more powerful storage. Local storage provides better locality and avoids contention for shared storage resources. To gain an insight of the two platforms, in this paper, we investigate the performance and resource utilization of different types (i.e., I/O-intensive, data-intensive and CPU-intensive) of applications on the HPC-based Hadoop platforms with local storage and dedicated storage. We find that the I/O-intensive and data-intensive applications with large input file size can benefit more from the dedicated storage, while these applications with small input file size can benefit more from the local storage. CPU-intensive applications with a large number of small-size input files benefit more from the local storage, while these applications with large-size input files benefit approximately equally from the two platforms. We verify our findings by trace-driven experiments on different types of jobs from the Facebook synthesized trace. This work provides guidance on choosing the best platform to optimize the performance of different types of applications and reduce system overhead.**

I. Introduction

In this "big data" era, enormous datasets abound in all facets of our lives and in many realms (e.g., science and engineering, commerce) and data volume continues to grow at an astonishing rate. MapReduce [9] is a computing model for parallel data processing in high-performance cluster for data-intensive applications. Hadoop [1], as a popular open-source implementation of MapReduce, has been widely adopted in many companies and organizations nowadays (e.g., Yahoo, Facebook, Amazon, Twitter) to process big data and compute tens to hundreds terabytes of data per day. Through parallel computing, MapReduce can significantly reduce the time to complete a job on large datasets by running small tasks on multiple machines in a large-scale cluster. High performance computing cluster is an important infrastructure to provide high-performance and data-parallel processing for applications utilizing big data. Many HPC sites extend their clusters to support Hadoop MapReduce in order to meet the needs from a variety of applications.

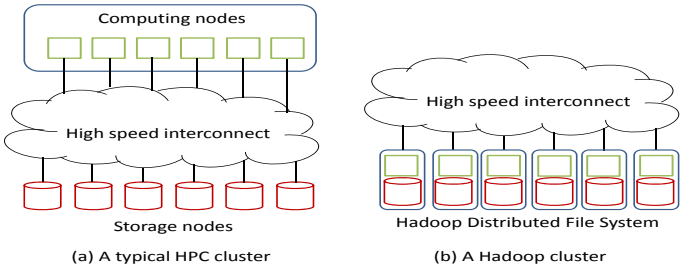


Fig. 1: A typical HPC and Hadoop cluster [20].

However, the Hadoop cluster differs from HPC cluster in the configuration of storage resources, as shown in Figure 1. In Hadoop Distributed File System (HDFS), data resides on the compute nodes, while in HPC cluster, data is usually stored on separate nodes dedicated to storage. Examples of the dedicated storage subsystems include OrangeFS [2], PVFS [6] and Lustre [4]. On HPC cluster, the dedicated storage subsystem offloads I/O workload from the compute nodes, and its more powerful storage can provide a better I/O performance in many cases. In Hadoop, localized storage nodes provide better data locality and avoid contention for the global shared storage subsystem. Recently, our institution deployed a new shim layer on Clemson Palmetto HPC cluster (ranked in the top five of U.S. public universities) that allows Hadoop MapReduce to access dedicated storage (i.e., OrangeFS). It also allows users to configure a Hadoop cluster with HDFS on the HPC cluster using myHadoop [12]. Figure 2 and Figure 3 show the two HPC-based Hadoop platforms. For simplicity, in the following, we use OFS to denote the platform where Hadoop MapReduce accesses remote OrangeFS, and use HDFS to denote the platform where Hadoop MapReduce accesses local storage.

Current MapReduce workload consists of different types of applications (e.g., I/O-intensive, data-intensive and CPU-intensive) [7, 8, 15] and different types of applications may benefit differently from these two platforms [15]. CPU-intensive applications devote most execution time to com-

putational requirements and typically require small volumes of data. Data-intensive and I/O-intensive applications both require large volumes of data and devote most processing time to I/O. Data-intensive applications contain certain amount of computation, while I/O-intensive applications do not or have only few computations [30]. Additionally, data-intensive applications generate a relatively large amount of intermediate data, while I/O-intensive applications contain only pure write/read operations. It is important to gain a deep understanding of each type of applications on these two platforms, which can provide guidance to decide the best platform for a given application to maximize performance and minimize the system overhead. However, there has been no previous effort that conducted this investigation despite significant research on HPC and MapReduce. Thus, in this paper, we mainly focus on investigating the performance and resource utilization of different types of applications on these two platforms. We summarize our contributions below.

(1) We have conducted extensive experiments with different types of Hadoop applications (i.e., data-intensive, compute-intensive, I/O-intensive) on the two platforms and measured their performance and resource utilization.

(2) We have analyzed how different application features (e.g., I/O rate, data size) affect the application performance and system overhead on the two platforms and determine the best platform for an application with certain features.

(3) We have used a Facebook synthesized trace to conduct extensive experiments to verify our investigation results of the platform selection for different types of applications.

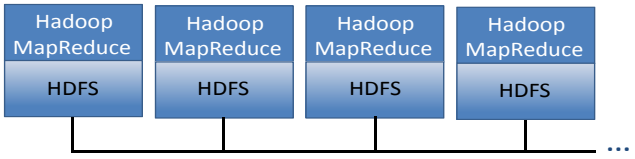


Fig. 2: Typical Hadoop with HDFS local storage (HDFS in short).

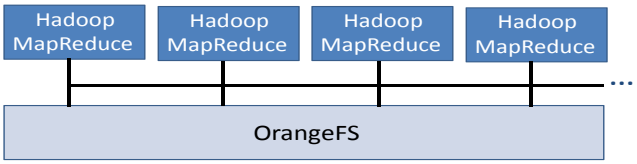


Fig. 3: Hadoop with the OrangeFS dedicated storage (OFS in short).

The remainder of this paper is organized as follows. Section II explains the two HPC-based Hadoop platforms. Section III presents the measurement results of performance for different types of Hadoop applications. Section IV presents the experimental results based on the Facebook synthesized trace to verify our conclusion of platform selection based on application features. Section VI gives an overview of the related work. Section VII concludes this paper with remarks on our future work.

II. Two HPC-based Hadoop Platforms

To facilitate the understanding of application performance analysis, we first give an overview of Hadoop MapReduce and then introduce the two different HPC-based Hadoop platforms with dedicated and local storage, respectively.

Hadoop [1] is an open source implementation of the MapReduce [9]. In Hadoop, HDFS is a very fundamental component. Figure 2 shows the typical Hadoop with HDFS on local storage. HDFS is an implementation of the Google File System [10] used to store both the input and output data for the map and reduce tasks. HDFS consists of NameNode and DataNodes. NameNode is a master server that manages the file system and the access to files by clients. DataNodes are responsible for serving read and write requests from the clients. Generally, the TaskTraker and DataNodes run on the same groups of nodes. JobTracker always tries to schedule the tasks on the nodes that store the data needed by the tasks in order to decrease the data movement and bandwidth utilization. Figure 3 shows the extension of a HPC cluster to support Hadoop MapReduce, in which compute nodes access existing HPC dedicated storage system such as OrangeFS [2, 13, 14]. OrangeFS is an open-source parallel file system for compute and storage clusters. It stores files as objects (i.e., metadata and file data) across multiple servers and accesses these objects in parallel, which offers better I/O performance. Each file or directory consists of two or more objects: one primarily containing file metadata, and the other(s) primarily containing file data. Each file is partitioned to stripes for storage. To access a file, the metadata object of the file is firstly fetched, which provides the information that which servers store the data objects.

We used Hadoop 1.2.1 in our experiments. We used the OFS platform on HPC cluster in our institution. In the OFS platform, no modifications of Hadoop were required, and no modification of MapReduce jobs are required to utilize OrangeFS. The key component of this structure is a Java Native Interface (JNI) to OrangeFS client. This is because Hadoop MapReduce is written in Java, but OrangeFS's client libraries are written in C. The JNI shim allows Java code to execute functions present in the OrangeFS Direct Client Interface. We configured the HDFS platform on the same HPC cluster by ourselves using myHadoop [12], which is a framework for configuring Hadoop on traditional HPC. All nodes and OrangeFS servers in the two platforms are interconnected with 10Gb/s Myrinet.

We used 8 storage servers in the OFS file system, each of which includes two SSD disks for metadata only and 5 SATA disks for data storage. OFS combines the storage across the 8 servers. Therefore, there are in total 40 (i.e., 8*5) hard disks for data storage in the OFS platform. To achieve fair comparisons between the OFS and HDFS platforms, we chose 40 nodes (i.e., 8*5) as compute nodes in the two platforms and hence HDFS also has 40 storage disks as OFS does. Each node has 16GB memory, 8*2.3GHZ CPU capacity, 193GB storage capacity. Additionally, OFS has its own metadata servers and we expect that the 40 machines are all used as datanodes. Therefore, we use an additional machine to serve as namenode for the HDFS platform. Some detail configurations of the Hadoop cluster with OFS and HDFS are described as

follow. In the OFS platform, input and output data is placed in dedicated storage servers, while the shuffle data is still stored in the local storage, since currently Palmetto does not support to place the shuffle data on the dedicated storages. In the HDFS platform, all the data is stored in local storage. We set the replication factor to 3 as default [16, 18]. The block (stripe) size of both file system is 128MB, which is a recommended default block size of Hadoop [1]. The total number of map and reduce slots on each node is set to the number of CPU cores on each node (a common setting for Hadoop Cluster [8]). For example, if a node has 4 CPU cores, we can set the numbers of map and reduce slots to 2, respectively.

III. Performance Measurement of Different Applications On the Two Platforms

In this section, we measure the performance and resource utilization of different types of applications on the OFS and HDFS platforms introduced above. Based on the measurement results, we provide guidance in deciding the best platform for a given application based on its characteristics. We ran a number of typical Hadoop applications including *TestDFSIO*, *Wordcount*, *Grep*, *PiEstimator* and *PageRank* on the OFS and HDFS platforms.

TestDFSIO is an I/O-intensive benchmark, which tests the I/O performance of the network and file systems. *Grep* and *Wordcount* are data-intensive benchmarks. They count the matches of a regular expression and the words in the input files, respectively. If *Grep* and *Wordcount* process the same datasets, then *Grep* generally does not have as much calculations as *Wordcount* does, since *Grep* only counts the number of specified regular expression, while *Wordcount* counts all the words appeared in the text. We generated the input files from BigDataBench [26], which is based on the Wikipedia datasets. For *Grep* and *Wordcount*, each file size is fixed to 500MB. In the measurements, the number of files is up to 4096, which means the input file size is up to 2TB.

PiEstimator is a typical CPU-intensive job and it uses a statistical (quasi-Monte Carlo) method to estimate the value of Pi. *PageRank* is another CPU-intensive job. We used the *PageRank* program from the CMU PEGASUS [3] and generated the input file from BigDataBench based on Google web graph datasets. A big difference between *PageRank* and *PiEstimator* is that *PageRank* has one large-size input file while *PiEstimator* has a large number of small input files.

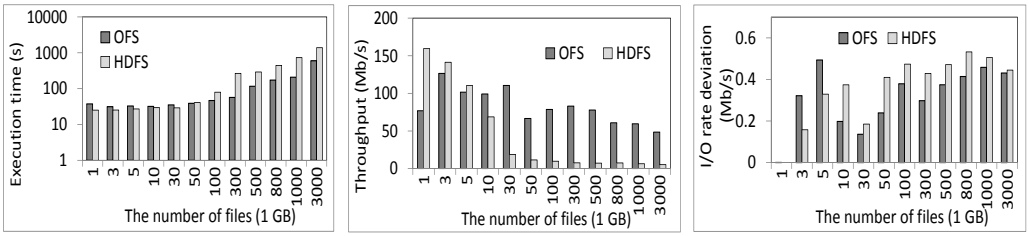
In our measurements, the input data size of I/O-intensive, data-intensive and CPU-intensive applications reaches terabyte at most, which is large enough to evaluate the performance of running big data applications on the two platforms.

We measured different metrics for the performance and resource utilization for each application including the job execution time and average map/reduce task completion time (i.e., the sum of all the map/reduce task completion time divided by the number of map/reduce tasks). All the metric information is parsed out directly from the Hadoop logs. Note that in Hadoop, the reduce tasks include data shuffling from map tasks to reduce tasks. Hadoop always tries to overlap this data shuffling with the map tasks and let the shuffling start right after some map tasks are completed. Therefore, the average reduce task completion time contains the shuffling time, which results in long average reduce task completion time. To monitor the resource utilization [23, 28], we used SYSSTAT utilities *mpstat* to monitor the CPU time every second. We developed a bash script using *ifconfig* to monitor bandwidth utilization every second. Note that these tools introduce a low overhead during measurement [27].

A. I/O-Intensive Applications

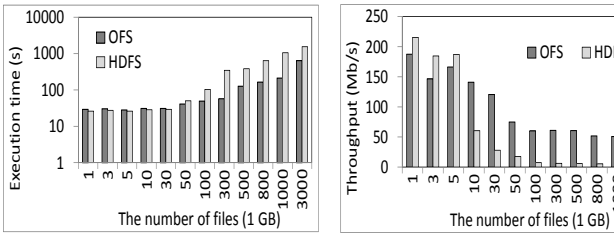
In this section, we measure the performance and resource utilization of *TestDFSIO* (I/O-intensive). *TestDFSIO* provides two tests: write and read. In the write test, each map task writes a file, and in the read test, each map task reads a file and hence the number of mappers equals the number of input files. Therefore, more input files mean more mappers located in one node, which leads to higher competition on I/O accessing the local disk in one node. In this experiment, we varied the number of input files and set file size to 1GB. We first introduce the metrics we measured. The *execution time* of a job is calculated by the ending time of the job minus its starting time. In *TestDFSIO*, the reduce task only collects the statistics of the map tasks, such as the number of completed tasks, execution time and I/O rate, so the throughput only occurs in the map tasks. Therefore, we define *throughput* as the total read/write file size divided by the map phase duration time. Each file's *I/O rate* is calculated by the file size divided by its corresponding map task time. We define *I/O rate deviation* as the standard deviation of I/O rates of all files. This metric reflects the stability of I/O performance for each file since I/O congestion leads to instability and hence a large deviation. *Throughput* and *I/O rate deviation* reflect the storage I/O performance.

Figures 4(a), 4(b), 4(c) show the execution time, throughput and I/O rate deviation of OFS and HDFS in the write test versus the number of input files, respectively. Figures 5(a), 5(b), 5(c) show the execution time, throughput and I/O rate deviation of OFS and HDFS in the read test versus the number of input files, respectively. We see that in both read and write tests, when the number of files is no greater than five, OFS leads to slightly higher execution time and lower throughput than HDFS. However, when the number of files becomes larger, OFS needs much less execution time and has much higher throughput than HDFS. Figure 4(c) indicates that the I/O rate deviation has an opposite pattern of throughput. When the number of files is no greater than five, the deviation of HDFS is smaller than OFS; when it is more than five, the deviation of OFS is smaller than HDFS. Figures 7(a) and 7(b) show the average map and reduce task completion time in the write test versus the number of input files. We see that the results are consistent with the execution time and throughput in Figures 4(a) and 4(b). Figures 7(c) and 7(d) show the average map and reduce task completion time in the read test, which are also consistent with the execution time and throughput in Figures 5(a) and 5(b).



(a) Execution time. (b) Throughput. (c) I/O rate deviation.

Fig. 4: Performance of I/O-intensive application *TestDFSIO* write test.



(a) Execution time. (b) Throughput. (c) I/O rate deviation.

Fig. 5: Performance of I/O-intensive application *TestDFSIO* read test.

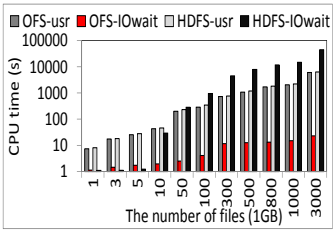
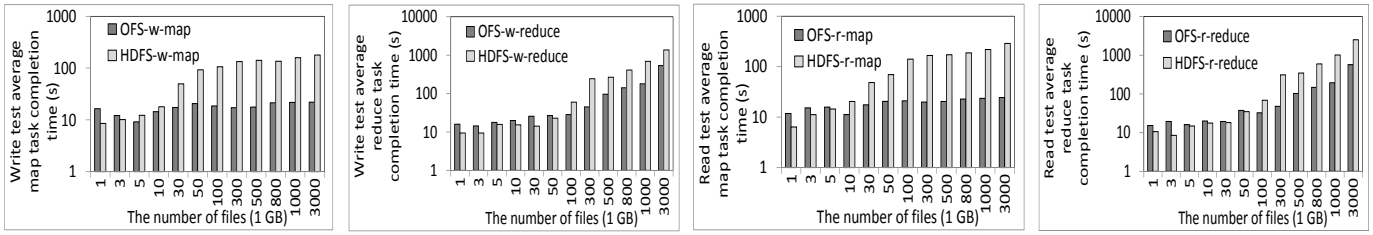


Fig. 6: CPU time of I/O-intensive application *TestDFSIO*.



(a) Write test average map task completion time. (b) Write test average reduce task completion time. (c) Read test average map task completion time. (d) Read test average reduce task completion time.

Fig. 7: Average task completion time of I/O-intensive application *TestDFSIO* write/read test.

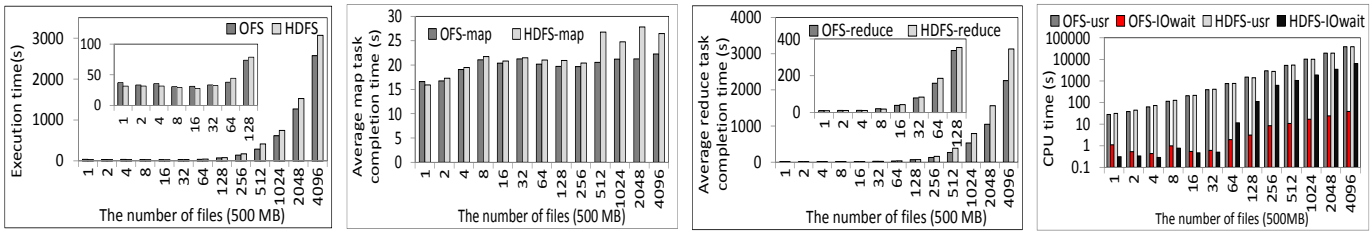
In order to find out the reasons for the performance differences on the two platforms shown in the above figures, we measured the user-level and I/O-waiting CPU utilizations on both platforms. The user-level CPU time means the CPU time used for executing the user-level application, while the I/O-waiting CPU time is the time period that the CPU(s) are idle during which the system has an outstanding I/O request.

Figure 6 shows the user-level and I/O waiting CPU times in both write and read tests on OFS and HDFS. The figure indicates that the user-level CPU times are approximately the same on both platforms. It means that except I/O-waiting, *TestDFSIO* consumes similar CPU time on both platforms, which implies that the major factor that affects the CPU time difference on both platforms is the I/O-waiting CPU time. On HDFS, when the number of files increases, the I/O-waiting CPU time becomes extremely high, which almost occupies more than 70% of the total CPU time (i.e., user-level+I/O-waiting). On the contrary, the I/O-waiting CPU time on OFS is much smaller, which constitutes 0.1%-6% of the I/O-waiting CPU time of HDFS. It indicates that the applications spend much longer CPU time on waiting the I/O completion on HDFS, which results in higher average map task completion time. This is because the local disks in HDFS are slow and cannot quickly handle large I/O amounts. In OFS, the OrangeFS dedicated storage offloads I/O operations from the compute nodes and its more powerful storage and OrangeFS infrastructure can provide better I/O performance than HDFS. However, for a small amount of data size (1-5 files), the I/O-waiting CPU time on OFS is slightly higher than HDFS due to the network latency (almost constant) necessary for the remote communication regardless of the data size. With HDFS, the tasks are more likely to be data local, which can eliminate possible network latency. However, with OFS, the tasks always need to use the network to read/write the input/output data. Since the execution time is relatively small when the data size is small, the network latency affects the performance greatly. On the other hand, the network latency can be neglected when the data size is large, since the execution time is also large.

From the above experimental results, we can make the conclusions for I/O-intensive applications.

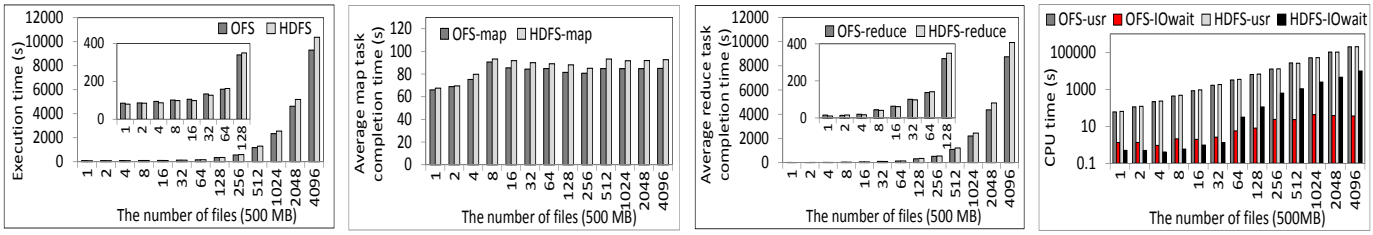
(1) I/O performance is a major factor for execution time and throughput.

(2) To achieve better I/O performance, if a job has a large amount of data size to read/write, OFS is a better platform;



(a) Execution time. (b) Average map task completion time. (c) Average reduce task completion time. (d) The CPU time.

Fig. 8: Performance of data-intensive application *Grep*.



(a) Execution time. (b) Average map task completion time. (c) Average reduce task completion time. (d) The CPU time.

Fig. 9: Performance of data-intensive application *Wordcount*.

if a job has a very small amount of data size to read/write, HDFS is a better platform to avoid network latency.

B. Data-Intensive Applications

Data-intensive applications need to process a large amount of data. This kind of jobs devote most of the processing time on I/O and data manipulations, which are quite similar as I/O-intensive applications. However, data-intensive applications contain certain amount of calculations, while I/O-intensive applications have no or only a few computations. We measure the performance and resource utilization of *Grep* and *Wordcount* data-intensive applications in this section. For both *Grep* and *Wordcount*, recall that the size of each file we generated is 500MB.

Figures 8(a) and 9(a) show the job execution time versus the number of input files for *Grep* and *Wordcount*, respectively. We see that when the number of input files is relatively small (1-64 input files), *Grep* and *Wordcount* do not have much difference on execution time between HDFS and OFS. As the number of input files becomes greater, the execution time of both applications is shorter on OFS than on HDFS. This result is consistent with the results of the I/O-intensive jobs due to the same reasons, since data-intensive jobs also involve a large amount of I/O requests.

We calculated the execution time difference between OFS and HDFS by $\frac{|HDFS.exe - OFS.exe|}{HDFS.exe}$. This metric value is 2%-30% for *Grep* and is 1%-10% for *Wordcount*, which are smaller than this metric value for *TestDFSIO* (20%-70%). Also, this metric for *Wordcount* is smaller than *Grep*. This is because data-intensive *Grep* and *Wordcount* contain more computations than I/O-intensive *TestDFSIO* and *Wordcount* contains even more calculations than *Grep* since *Wordcount* counts all the words appeared in a file while *Grep* only counts the matches of regular expression in the file. As computation also contributes to the execution time, when there are more computations, the I/O performance plays a less important role in determining the execution time and the performance difference between the two platforms becomes smaller. Figures 8(b) and 8(c) show the average map and reduce task completion time versus the number of input files for *Grep*. We see that these two figures match the pattern of execution time of *Grep*. Similarly, Figures 9(b) and 9(c), which show the average map and reduce task completion time versus the number of input files for *Wordcount*, also have the same pattern as the execution time result of *Wordcount*.

Figures 8(d) and 9(d) show user-level CPU time and I/O-waiting CPU time for *Grep* and *Wordcount* on both platforms. We see that when the number of files is large (greater than 64), I/O-waiting CPU time with HDFS is significant, which occupies 3%-19% of total CPU time for *Grep* and 0.8% - 5% for *Wordcount*. On the other hand, I/O-waiting CPU time with OFS occupies 0.1%-0.3% of the total CPU time for *Grep* and 0.08%-0.1% for *Wordcount*. The user-level CPU times are nearly the same in both HDFS and OFS for each application, which means each application's computations take nearly the same CPU times in both platforms. The data-intensive jobs involve a large amount of I/O operations on the data, and hence their performance is influenced by the I/O performance of the file systems. Since OFS has better I/O performance when the total input data size is large, it produces less I/O-waiting CPU time.

We also notice that for a given number of input files, *Grep* and *Wordcount* have similar I/O-waiting CPU time on each platform. However, *Wordcount* has much higher user-level CPU time than *Grep* since *Wordcount* contains more computations than *Grep* as aforementioned previously. On the other hand, the user-level CPU time of *Grep* is much higher than *TestDFSIO* at the same input data size (file size * the

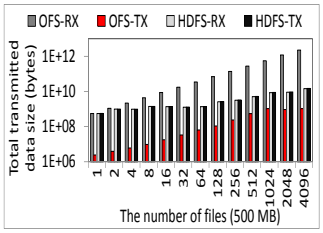


Fig. 10: Total transmitted data size of data-intensive *Grep*.

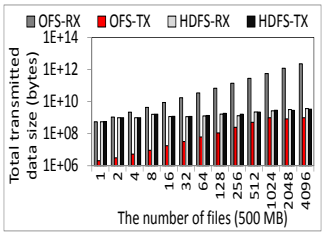


Fig. 11: Total transmitted data size of data-intensive *Wordcount*.

number of files) because *Grep* involves certain computations besides I/O requests while *TestDFSIO* only has I/O requests. Consequently, the I/O-waiting CPU time of *Wordcount* occupies less percentage of total CPU time than *Grep*, which makes the delay caused by I/O performance difference less significant for *Wordcount*. Hence, the execution time difference between OFS and HDFS for *Wordcount* is not as large as the difference of *Grep*. Similarly, the execution time difference between OFS and HDFS for *Grep* and *Wordcount* is not as large as *TestDFSIO*. From these results, we can conclude that the more computations an application has, the less influence from the I/O performance on the execution time and the less performance difference between the two platforms.

Figures 10 and 11 show the total transmitted data size of all the compute nodes for *Grep* and *Wordcount* on both platforms, respectively. The received data (RX) of the nodes contains input, shuffle and output data that all compute nodes read from other nodes. The transmitted data (TX) contains input files, shuffle data, output data transmitted out by all compute nodes, and also task status that is needed to report to JobTracker. We see that for both applications on HDFS, the transmitted data size is much smaller than the input data size. It indicates that Hadoop can achieve very high data locality to avoid the data movement between nodes on HDFS. Therefore, the total transmitted data size on HDFS is small. On the other hand, on OFS, the data must be input and output to remote storage. The input data for map tasks on OFS is always needed to read through network. Therefore, the size of received data of all the nodes is very close to the input data size. For the output data, it is also stored in the remote storage through network. Then, the size of transmitted data of all the nodes is close to the output data size of reduce tasks. The transmitted data size in OFS is slightly smaller than that in HDFS. This is because in OFS, the nodes only need to send output and shuffle data, while the nodes in HDFS also need to send the input data blocks needed by the mappers allocated on other nodes besides output and shuffle data. The result implies that a dedicated storage can offload I/O load from the compute nodes to avoid the I/O-waiting CPU time.

We can make the conclusions for data-intensive jobs.

(1) If an application has a large input data size, OFS is a better platform to achieve better I/O performance and reduce the execution time.

(2) If an application has a small input data size, HDFS is a better platform to avoid network latency and reduce the total transmitted data size.

(3) The more computations an application has, the less influence from the I/O performance on the execution time and the less performance difference between the two platforms.

(4) Since the I/O-waiting CPU time occupies less percentage of total CPU time for data-intensive applications, the performance difference between OFS and HDFS for data-intensive applications is not as large as I/O-intensive applications.

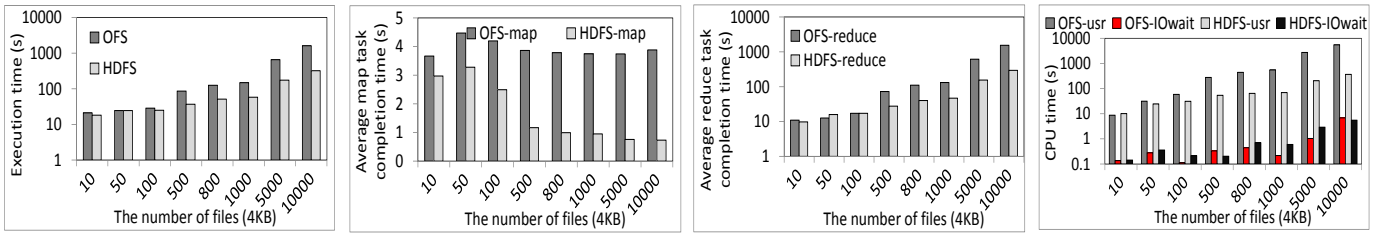
C. CPU-Intensive Applications

In this section, we first measure the performance and resource utilization of the *PiEstimator* application. It has a large number of small-size input files and each file is processed by one mapper. Each file in *PiEstimator* has around 4KB. Then, we measure the *PageRank* application, which has only one large-size input file. The file is processed by $\frac{filesize}{blocksize}$ mappers. For both applications, the input files are stored into the local storage in HDFS and the dedicated storage in OFS, then the applications start to read files and process the files.

PiEstimator Figure 12(a) shows the execution time of *PiEstimator* versus the number of input files on OFS and HDFS. HDFS always has a shorter execution time than OFS regardless of the number of input files. The execution time difference between HDFS and OFS is not significant when the number of input files is small, and it becomes greater when the number of input files increases.

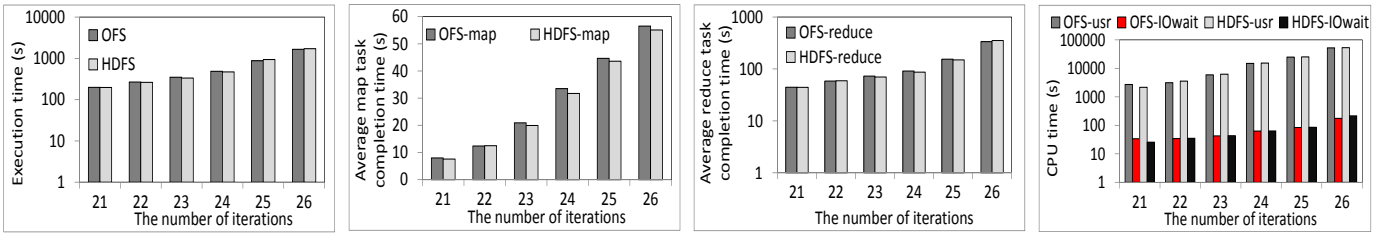
Recall that each file is very small (4KB) in *PiEstimator*. In OFS, when OrangeFS reads/writes a file, the setup of the network communication from the client to the remote servers must occur. Therefore, each file read generates a certain network latency even though it has a small size. More small-size file reads from the dedicated storage through network lead to higher total file access delay. In HDFS, files are read directly from the local disk without network connection latency and the files' small size makes the I/O congestion less likely to occur. As a result, the application needs longer time to fetch small-size files in OFS than in HDFS. Therefore, when the file size is small, the network latency for each file read offsets the better I/O performance of the dedicated storage. As shown in the pervious experiments, if the file size is large, the network communication setup time is negligible compared to the total data transmission time, and then the better I/O performance of the dedicated storage in OFS can be clearly shown.

Figure 12(d) shows the CPU time of *PiEstimator* on two platforms on user level and I/O-waiting. We see that though *PiEstimator* runs the same computing task in both platforms, it consumes much more user-level CPU time on OFS than on HDFS. Also, the increase rate of user-level CPU time on OFS is much faster than that on HDFS as the number of input files increases. This is because when OrangeFS reads/writes a file, the setup of communication with the remote servers is all conducted by the OrangeFS client library which runs in user space whereas the HDFS uses kernel virtual file system (VFS). For communications of large-size files in the previous *TestDFSIO* experiment, the CPU time used to set up the communications is negligible compared to the network latency



(a) Execution time. (b) Average map task completion time. (c) Average reduce task completion time. (d) The CPU time.

Fig. 12: Performance of CPU-intensive application *PiEstimator*.



(a) Execution time. (b) Average map task completion time. (c) Average reduce task completion time. (d) The CPU time.

Fig. 13: Performance of CPU-intensive application *PageRank*.

for transferring large-size files. However, for a large number of small files, OrangeFS client library needs to set up a large number of communications. The CPU time is not negligible compared to the small network latency for transferring small-size files. More small-size file reads from remote storage generates a much larger total CPU time, which degrades the performance of *PiEstimator* jobs on OFS. We also see that the OFS and HDFS produce similar I/O waiting CPU time, which are both quite small. On both platforms, the user-level CPU time occupies more than 99% of the total CPU time, while I/O-waiting CPU time is almost negligible. It is because this application is CPU-intensive and the file size is small, CPU is less likely to be idle to wait for the I/O requests due to outstanding I/O in both platforms.

Figures 12(b) and 12(c) show the map and reduce task duration of *PiEstimator*, respectively. We see that *PiEstimator* always has higher average map and reduce task completion time on OFS than on HDFS, especially when there are many input files. This is because of a certain latency for network communication even if the file size is small. Many times of network communication to access small-size data from the remove storage lead to a long total network latency. On the other hand, on HDFS, small-size file read generates shorter access time in the local storage, hence leads to short average map and reduce task completion time. For Hadoop with HDFS configurations, we use an additional machine to serve as namenode only. This is because OFS itself has own metadata servers. To achieve fair comparisons between HDFS and OFS, we expect the 8 machines are purely datanodes, otherwise, one of the 8 machines will have to act as both namenode and datanode, which probably degrades the performance of HDFS and leads to unfair results.

TABLE I: I/O-intensive *TestDFSIO* read/write test of ten thousands 1MB files.

Platforms	read execution time	write execution time	CPU-usr	CPU-IOwait	RX	TX
OFS	1380s	1182s	13190s	662s	29 GB	10 GB
HDFS	194s	211s	798s	247s	1 GB	1 GB

Figure 14 shows the total transmitted data size of all the nodes for *PiEstimator* on both platforms. We see that with OFS, the nodes have a large amount of received data, which is much larger than the size of the input files. This is because of the communication overhead between the nodes and the remote storage servers when the nodes try to connect with the dedicated storage. The overhead may include the package header added to user-transmitted data package for carrying routing information and error correcting and operational instructions. The transmitted data size of all the nodes with OFS is slightly smaller than with HDFS due to the same reason in Figures 10 and 11.

In order to confirm the larger user-level CPU time of OFS than HDFS in the case of a large number of small-size files, we conducted another *TestDFSIO* read/write experiment for ten thousands 1MB files on both platforms. We set the file size to 1MB, the smallest available file size for *TestDFSIO*. The results are shown in Table I. We see that the execution time of write/read test on OFS is much greater than on HDFS. It is caused by the same reason as mentioned above. *TestDFSIO* consumes much more user-level CPU time to transfer a large number of small-size files on OFS than on HDFS. Both the size of received data (RX) and transmitted data (TX) on HDFS is less than on OFS. This is because HDFS benefits from data locality, which can help HDFS avoid too many data movement through network while OFS produces extra communication overhead for each small-size file. Unlike the previous experiments, the TX value of OFS is much larger than HDFS because

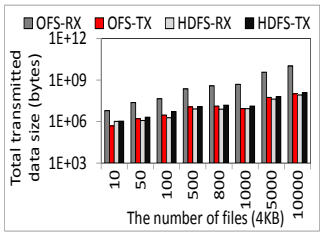


Fig. 14: Total transmitted data size of CPU-intensive *PiEstimator*.

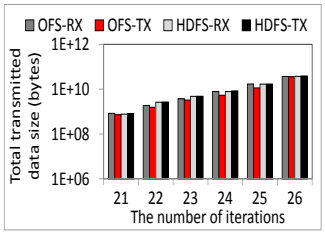


Fig. 15: Total transmitted data size of CPU-intensive *PageRank*.

there are considerably more write operations in *TestDFSIO*. The table also indicates that the nodes have a greater amount of received data than the input data size (10GB) on OFS because of communication overhead. Actually, neither HDFS nor OFS are designed for small files. If we must deal with applications with a large number of small-size files, HDFS with local storage is a better platform than OFS with dedicated storage.

PageRank We then measure the performance of *PageRank*, which has one file with a much larger size. Based on the specified number of iterations n, we generated a file of edges between 2^n webpages. One edge means one webpage is linked to another webpage. The input file is stored in the first compute node in HDFS. The input data of *PageRank* is partitioned to 128MB-size blocks and each mapper processes one block. Figures 13(a), 13(b) and 13(c) show the execution time, the average map and reduce task completion time of *PageRank* versus different number of iterations on both platforms. We see that both platforms generate similar experimental results. This is because for the applications that contain a large amount of computations, the performance mainly depends on the CPU resources of the nodes. In both platforms, the CPU resources are from the same nodes despite the different storages.

Figure 13(d) shows that the two platforms generate similar user-level CPU time and I/O-waiting CPU time in different number of iterations, which indicates that there is not any difference between OFS and HDFS for *PageRank*. Recall that *PageRank* contains a large amount of computations, which is mainly related to CPU computing capability rather than the storage and the two platforms use the same compute nodes. We also see that the user-level CPU time on two platforms is very high, while the I/O-waiting CPU time is comparatively very low (0.3%-1%), which indicates that there is little CPU time spent on waiting the I/O requests although the input data size is large. The data size of input can reach 40GB when the number of iterations increases. However, it does not consume as much CPU time on waiting the I/O requests as the data-intensive applications. This is because the recursively computing dominates the CPU time of *PageRank* and each input data needs a large amount of recursive computation. It makes the I/O performance, which affects the I/O-waiting CPU time, play a much less significant role in determining the application performance in *PageRank*. Figure 15 shows the total transmitted data size on all the nodes on both platforms. We see that the amount of received and transmitted data is close to the input and output on both platforms.

From the above experimental results, we can make the conclusions for CPU-intensive applications.

(1) Although CPU-intensive applications can contain large-size input files, a large amount of calculations dominate the CPU time for this kind of applications, which makes the I/O performance play a much less important role in determining the application performance.

(2) If CPU-intensive applications have a large number of small-size input files, HDFS is better platform that can avoid high user-level CPU time for communication setup with the remote storage in OFS.

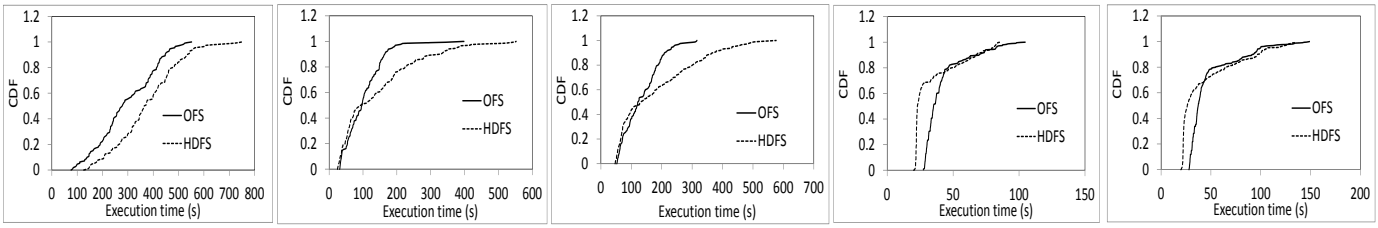
(3) If CPU-intensive applications have large-size input files, both HDFS and OFS produce comparable performance.

IV. Performance Evaluation

In this section, we verify our conclusions on the platform selection for different types of applications by using workload from 2009 Facebook synthesized trace [8]. In the experiments, we used 40 compute nodes and 40 storage nodes. We selected the first 1500 jobs from the total 6638 jobs in the trace and ran the jobs in both OFS and HDFS platforms to measure the performance. Since the trace is collected from 600-machine cluster and we have only 40 machines, we shrank the data size by a factor of 5 based on the instructions of the workload [8].

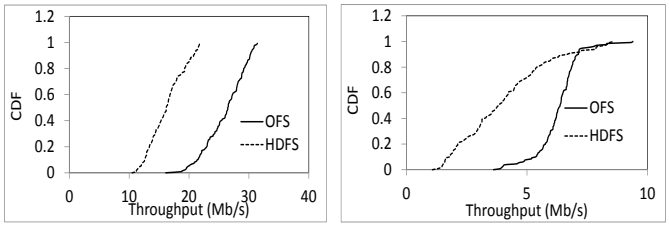
According to the application analysis, we approximately characterize the types of jobs based on input/shuffle/output data size. I/O-intensive jobs include write and read jobs. The write jobs have large output size but small input and shuffle size, while the read jobs have large input size and small shuffle and output size. Data-intensive applications have large input and shuffle size (in the same order), while the output size can be either large or small. From the 1500 jobs, we picked out the jobs with input size greater than 1GB and shuffle and output size smaller than 100MB as I/O intensive read jobs. For I/O-intensive write jobs, the input and shuffle sizes are smaller than 100MB and output size is greater than 1GB. For data-intensive jobs, the input size is greater than 1GB, the shuffle size has a similar order to input, and output size is not limited. We then approximately considered the remaining jobs as CPU-intensive jobs and classified them to a small-size file group with each file having less than 256MB and non-small-size file group.

Figure 16(a) shows the execution time distribution of I/O-intensive write jobs. It shows that the performance of write jobs are always better on OFS than on HDFS since the write jobs from the trace all have large sizes. Figures 16(b) and 16(c) show the execution time distribution of read jobs and data-intensive jobs, respectively. We see that the CDF of the execution time of the read jobs and data-intensive jobs have similar patterns. When the execution time is small ($< 100s$), which indicates the input size is small, the execution time on HDFS has a broader distribution than on OFS. When the execution time is large, on the contrary, the execution time has a broader distribution on OFS than on HDFS. The result means that when the input size is small, HDFS generates smaller execution time, while when the input size is larger,



(a) I/O-intensive write. (b) I/O-intensive read. (c) Data-intensive jobs. (d) CPU-intensive jobs with small-size files. (e) CPU-intensive jobs with non-small size files.

Fig. 16: Execution time distribution for different jobs in trace-driven experiment.



(a) Throughput of write jobs. (b) Throughput of read jobs.

Fig. 17: I/O intensive jobs in trace-driven experiment.

OFS generates smaller execution time. It is consistent with our conclusions that for I/O-intensive and data-intensive jobs with small input size, it is better to use HDFS, while OFS is the best choice when the input size is large.

Figures 16(d) and 16(e) show the execution time distribution of small-size file group and non-small-size file group in the CPU-intensive jobs, respectively. In Figure 16(d), we see that the execution time on HDFS has a broader distribution than on OFS, which indicates that HDFS is a better platform for CPU-intensive jobs with small-size files to achieve high performance. In Figure 16(e), the execution time on HDFS has less broader distribution, which means the benefits of HDFS diminish for larger size file jobs and both HDFS and OFS can be the choice of these jobs. The results are consistent with our investigation results for CPU-intensive jobs. We see that at the tail of the curves, the distributions of OFS and HDFS are overlapped. It means that for CPU-intensive jobs, in which the I/O performance does not play an important role in the application, since the compute nodes are the same, the performance in the two platforms is similar. Note that there is some overlap at the tails of the curves for the two platforms. This is because the jobs appeared at the tail may be the boundary between CPU-intensive and I/O-intensive jobs or data-intensive jobs.

Figures 17(a) and 17(b) show the throughput of I/O intensive write and read jobs, respectively. We see that HDFS has broader write and read throughput distribution than OFS. It indicates that OFS has higher throughput and hence higher I/O performance than HDFS, which is consistent with our analysis conclusion for I/O-intensive jobs with large input size. The read throughput distributions of OFS and HDFS are overlapped at the tail. When the read jobs input size decreases, the throughput on HDFS becomes higher than on OFS as stated before, leading to the overlapped tail. The throughput distribution and execution time distribution for the write jobs in Figure 17(a) do not have this pattern because there may not exist small-size write jobs in our I/O-intensive write job group from the trace.

In summary, our trace-driven experiments verify our investigation results on how to select the best platform for different applications.

V. Discussions

In summary, our performance measurements in this paper aim to help the users to determine which platform should be used to run the applications to achieve the best performance. However, our performance measurements were conducted on only one HPC cluster. We do not expect the specific results can be generalized to any HPC clusters. We aim to provide a basic idea that configuring Hadoop with a remote file system is feasible for some types of applications. The users from different HPC clusters may utilize the metrics in this paper to measure which applications are better to run on Hadoop with remote file system on their HPC clusters. Based on the measurement results, users are able to select the best platform to run the applications to achieve best performance. For example, when a data-intensive job with very large input data size is submitted, based on our conclusions, the job should run on the Hadoop with remote file system in a HPC cluster.

VI. Related Work

In both MapReduce and HPC cluster, the underlying cluster file system is a crucial component. Much recent research has focused on the integration of Hadoop and HPC cluster [24], particularly in the use of the HPC file system in lieu of HDFS in the Hadoop framework, e.g., [5, 19, 25]. Researchers also tried to seek for substitution for HDFS. Examples of modern distributed file systems include GoogleFS [10], PVFS [6] and OrangeFS [29]. Tantisiriroj *et al.* [25] tried to demonstrate how PVFS can replace HDFS in the Hadoop framework. Later works such as GPFS [5] and Lustre [4] also demonstrate the using of their respective file system with Hadoop. Our work differs from the above works in that we investigates the performance of HPC-based Hadoop platforms with dedicated storage and local storage, respectively. Our results can give guidance on which applications are more suitable to run on a Hadoop platform with a dedicated storage substitution for HDFS.

Many efforts have been devoted to characterizing the workloads on MapReduce and cloud platforms [17, 22]. Chen *et al.*

[8] analyzed and compared two production MapReduce traces from Yahoo and Facebook in order to develop a vocabulary for describing MapReduce workloads. Their another work [7] 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.* [21] characterized a workload from Taobao at the granularity of job and task, respectively, which provides an understanding of the performance and the job characteristics of Hadoop in the production environment. Kavulya *et al.* [11] analyzed the characteristics of the workload based on MapReduce logs from the $M45$ supercomputing cluster. Although these studies instruct us on how to study the applications' features, performance and resource utilization, we have novelty that we study on the different storage platform. In addition to workload analysis, we have also investigated whether storing data in dedicated storage or local storage can achieve better performance. Moreover, we intuitively guide the clients to select the best storage structures to achieve better performance. It is useful for guiding both the cloud provider and clients to maximize performance and minimize the system consumption.

VII. Conclusions and Future Work

In this paper, we have presented a thorough measurement analysis of application performance and resource utilization on the two HPC-based Hadoop platforms: Hadoop configured with a local storage and Hadoop configured with a dedicated storage. We did our measurements and evaluations using different types of MapReduce applications: I/O-intensive, data-intensive and CPU-intensive applications for the analysis. We have concluded from the measurements which storage platform can benefit each type of applications with certain characteristics (e.g., input file size, I/O workload, CPU workload). We have used the Facebook synthesized trace workloads to verify our measurement conclusions. We demonstrated the validations of our measurement analysis, which provides guidance in determining which platform can best maximize the performance and reduce system overhead for a given application to the users. This study can also help HPC site staff who wishes to extend HPC capabilities with Hadoop by mixing workloads and then select a file system in the two file systems to allocate workloads accordingly. In the future, we plan to work on studying how to optimally arrange data (e.g., shuffle data, input data) placement between local storage and dedicated storage to improve application performance.

Acknowledgements

We would like to thank Mr. Matthew Cook for his insightful comments. 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] Apache Hadoop. http://hadoop.apache.org/.
[2] OrangeFS. http://www.orangefs.org.
[3] PEGASUS. http://www.cs.cmu.edu/~pegasus/.
[4] Using Lustre with Apache Hadoop. http://wiki.lustre.org/images/1/1b/Hadoop_wp_v0.4.2.pdf.
[5] 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.
[6] P. H. Carsn, W. B. Ligon, R. B. Ross, and R. Thakur. PVFS: A Parallel File System for Linux Clusters. 2000.
[7] 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.
[8] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz. The Case for Evaluating MapReduce Performance Using Workload Suites. In *Proc. of MASCOTS*, 2011.
[9] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proc. of OSDI*, 2004.
[10] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google File System. In *Proc. of ACM SOSP*, 2003.
[11] S. Kavulya, J. Tan, R. Gandhi, and P. Narasimhan. An analysis of traces from a production MapReduce cluster. In *Proc. of CCGRID*, 2010.
[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. Li and H. Shen. Performance measurement on scale-up and scale-out hadoop with remote and local file systems. In *Proc. of CLOUD*, 2016.
[15] Z. Li, H. Shen, W. B. Ligon, and J. Denton. An exploration of designing a hybrid scale-up/out hadoop architecture based on performance measurements. *IEEE Transactions on Parallel and Distributed Systems*, PP(99):1–1, 2016.
[16] Y. Lin and H. Shen. Eafr: An energy-efficient adaptive file replication system in data-intensive clusters. In *Proc. of ICCCN*, 2015.
[17] G. Liu, H. Shen, and H. Wang. Computing load aware and long-view load balancing for cluster storage systems. 2015.
[18] J. Liu and H. Shen. A popularity-aware cost-effective replication scheme for high data durability in cloud storage. In *Proc. of IEEE BigData*, Washington D.C., 2016.
[19] C. Maltzahn, E. Molina-Estolano, A. Khurana, A. Nelson, S. Brandt, and S. Weil. Ceph as a scalable alternative to the Hadoop Distributed File System). *The USENIX MAGAZINE*, 4(35):518–529, 2010.
[20] W. C. Moody, L. B. Ngo, E. Duffy, and A. Apon. Jummp: Job uninterrupted maneuverable mapreduce platform. In *Proc. of Cluster*, 2013.
[21] 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.
[22] A. Sarker, C. Qiu, H. Shen, A. Gil, J. Taiber, M. Chowdhury, J. Martin, M. Devine, and A. Rindos. An efficient wireless power transfer system to balance the state of charge of electric vehicles. In *Proc. of ICPP*. IEEE, 2016.
[23] H. Shen, L. Yu, L. Chen, and Z. Li. Goodbye to fixed bandwidth reservation: Job scheduling with elastic bandwidth reservation in clouds. In *Proc. of CloudCom*, 2016.
[24] 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.
[25] W. Tantisiroj, S. Patil, and G. Gibson. Data intensive file systems for internet services: A rose by any other name. Technical report cmu-pdl-08-114, 2008.
[26] 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. *arXiv preprint arXiv:1401.1406*, 2014.
[27] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy. Profiling and modeling resource usage of virtualized applications. In *Proc. of Middleware*, 2008.
[28] L. Yan and H. Shen. TOP: vehicle trajectory based driving speed optimization strategy for travel time minimization and road congestion avoidance. In *Proc. of MASS*, 2016.
[29] S. Yang, W. Ligon, and E. Quarles. Scalable Distributed Directory Implementation on Orange File System. In *Proc. of SNAPI*, 2011.
[30] 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.