Investigating the Efficiency of Machine Learning Algorithms on MapReduce Clusters with SSDs

Leonidas Akritidis, Athanasios Fevgas, Panagiota Tsompanopoulou, Panayiotis Bozanis
Data Structuring & Engineering Lab
Department of Electrical and Computer Engineering
University of Thessaly
Volos, Greece
Email: {leoakr,fevgas,yota,pbozanis}@e-ce.uth.gr

Abstract—**In the big data era, the efficient processing of large volumes of data has became a standard requirement for both organizations and enterprises. Since single workstations cannot sustain such tremendous workloads, MapReduce was introduced with the aim of providing a robust, easy, and fault-tolerant parallelization framework for the execution of applications on large clusters. One of the most representative examples of such applications is the machine learning algorithms which dominate the broad research area of data mining. Simultaneously, the recent advances in hardware technology led to the introduction of high-performing alternative devices for secondary storage, known as Solid State Drives (SSDs). In this paper we examine the performance of several parallel data mining algorithms on MapReduce clusters equipped with such modern hardware. More specifically, we investigate standard dataset preprocessing methods including vectorization and dimensionality reduction, and two supervised classifiers, Naive Bayes and Linear Regression. We compare the execution times of these algorithms on an experimental cluster equipped with both standard magnetic disks and SSDs, by employing two different datasets and by applying several different cluster configurations. Our experiments demonstrate that the usage of SSDs can accelerate the execution of machine learning methods by a margin which depends on the cluster setup and the nature of the applied algorithms.**

I. Introduction

During the past few years, the problem of managing and processing large volumes of data has became increasingly important. Presently, massive amounts of data are generated and processed not only by the popular Web information systems (e.g. search engines, social networks, portals, etc), but also by billions of mobile devices, wireless sensors and IoT systems. Apart from its huge size, this data also presents a high degree of diversity and it can represent countless types of information including posts, messages, user logs, scientific or news articles, temperature measurements, etc. Furthermore, it may be subject to frequent updates or removals, thus exhibiting high volatility. These characteristics (volume, variety, and velocity), along with some others (veracity and value) comprise the 5 Vs of what is now known as BigData.

Due to these overwhelming characteristics, single workstations cannot process BigData efficiently. MapReduce is one of the first general-purpose parallelization frameworks which allowed researchers to execute their algorithms on large clusters of interconnected machines. One of the most important features of the system lies on is simplicity; that is, the users do not deal with complex network programming issues such as socket I/O, buffering and connection protocols. On the contrary, it is required that they develop their solutions by implementing only two functions: *map* and *reduce* [1].

MapReduce operates on top of a distributed file system which transparently handles all data transfers among the nodes of the cluster, performs file management and provides fault tolerance. The programming model of the framework dictates that the parallel execution of a job is coordinated by a single machine, the *Master*. The Master assigns the processing of each input fragment to a *Mapper*, a node which applies the map function to every key/value pair of the input and generates a number of intermediate key/value pairs. In the sequel, this intermediate data is *shuffled* in a manner which dictates that all the values which are associated with the same intermediate key are transferred to the same *Reducer*. When the map and shuffle phases are completed, the data is sorted and the Reduce phase starts; during this phase the nodes apply the reduce function to all values associated with the same key. Finally, they write their final output to disk, also formatted in key/value pairs.

The aforementioned programming model of MapReduce has received a considerable amount of criticism in the relevant literature [2]–[4]. More specifically, the linear data flow and the process of transferring intermediate data among the nodes of the cluster, has rendered the parallel execution of some algorithms inefficient, mainly due to the expensive I/O operations of the traditional hard drives (HDDs). This drawback is magnified significantly in the case of iterative algorithms; since each iteration triggers a new cycle of Map and Reduce phases, multiple data shuffling phases are performed. Consequently, there are occasions where the running time of an iterative job is dominated by such disk I/O operations.

Nonetheless, the recent advances in hardware technology led to the introduction of Solid State Drives (SSDs), that is, Non-Volatile Memory (NVM) devices with much higher performance compared to the traditional hard drives. Nowadays, NAND Flash is the most popular type of non-volatile memory. Its characteristics include high read/write rates, large densities, low power consumption and shock resistance. On the other hand, successive deletions degrade the ability of Flash cells to retain their charge. Therefore, Flash blocks are able to sustain a limited number of program/erase cycles before wearing out.

In this article we investigate the performance of several data mining algorithms in MapReduce clusters equipped with SSDs. Motivated by the high throughput of these storage systems and the wide adoption of the machine learning approaches in BigData analytics, we study the potential benefits in the execution of parallel data mining jobs. More specifically, we evaluate the efficiency of multiple dataset preprocessing methods (such as split, vectorization and tf-idf vectors pruning), and two supervised classifiers, Naive Bayes and Linear Regression. We conduct exhaustive experiments on a 5-node cluster by employing two different datasets: i) a dump of the articles of the English version of Wikipedia, and ii) HIGGS, a dataset which originates from the scientific field of high-energy Physics. In short, the contributions of this research are:

- it focuses on machine learning and data engineering algorithms and it is not based on special data-intensive benchmarks such as Terasort and Teragen. Consequently, what is examined here is the usefulness of SSDs on popular, real-world parallel applications.
- it compares the performance of Hadoop with SSDs and HDDs by applying different degrees of parallelization with the aim of discovering potential relationships between external storage and cluster size. More specifically, the experiments are conducted for 5 different cluster sizes in terms of active processing nodes.
- it considers the impact of HDFS block size in the overall performance of the framework by examining two different settings, namely 128 MB and 512 MB.
- it evaluates the performance of both iterative and non-iterative machine learning algorithms to investigate the benefits of the usage of SSDs in different job types.

The rest of the paper is organized as follows: In Section II we present the current state-of-the-art in the area, whereas in Section III we describe our experimental cluster setup and the technical specifications of the utilized hardware. In Sections IV, V, and VI we discuss the performance of the examined data engineering methods and classification algorithms. Finally, Section VII summarizes the conclusions of the article and contains some insights of our planned future work.

II. Related Work

The evaluation of the parallel algorithms performance on MapReduce clusters with SSDs is currently a hot research topic and there is a considerable amount of relevant work in the literature. Nonetheless, to the best of our knowledge, the present work is the first to investigate the efficiency of some of the most popular data engineering and machine learning approaches on such environments.

MapReduce has been used extensively over the last decade for deploying data-intensive tasks such as data mining and knowledge engineering algorithms [5]–[9]. On the other hand, in the earlier literature we encounter robust implementations of the most wide-spread data structures on SSDs. Some of the most representative works include R-Trees [10], R*-Trees [11], generic spatial indexes [12], K-D-B-Trees [13] and Grid Files [14].

One of the first works which attempted to study the effects of SSDs utilization in Hadoop clusters is [15]. In this work the authors launch multiple virtual nodes on a single machine and report a remarkable tripling of the performance compared to the traditional HDDs. However, conducting experiments on a single-node cluster does not guarantee that the same improvements can also be achieved on multi-node clusters. This is because the job execution across multi-node clusters is also affected by multiple diverse parameters, including data transfer errors through the network, various latency types, network implementation, nodes interconnection method, nodes malfunction or death, and numerous others.

Indeed, according to [16] and [17], the available network bandwidth plays a crucial role in the performance of MapReduce jobs. In general, these studies demonstrate that the increase of bandwidth by improving the interconnection method, reduces the running times on both storage types. However, the gains are higher in the case of SSDs. These conclusions support our claim that the results on single-node clusters cannot be accurately generalized on multi-node environments.

Similar efforts which compare Hadoop performance on clusters equipped with magnetic disks and SSDs are presented in [18] and [19]. The former concludes that the usage of SSDs has a positive impact in the performance of MapReduce shuffle phase, whereas the latter reports that the gains depend on the type of the job executed on the cluster. The experiments in these works are based on specialized benchmarks such as Teraread and Teragen and do not study the performance of data engineering or iterative machine learning algorithms.

Another work with interesting findings is [20], where the authors analyze the performance improvement on datasets of different sizes. Their experiments on a five-node cluster with the Intel HiBench benchmark [21] indicate that the larger the input of a MapReduce algorithm is, the higher the benefits of utilizing SSDs are. In contrast to all other related works, the authors of [22] state that Hadoop performs nearly the same on clusters equipped with either HDDs, or SSDs. Additionally, in [23]–[25] the authors investigate various cost-effective techniques which enhance the performance of Hadoop clusters with SSDs, with respect to energy consumption.

Finally, a number of papers introduce extensions to the Hadoop framework for clusters equipped with modern SSDs. More specifically, in [26] the authors developed VENU, a MapReduce extension which employs the traditional HDDs for generic data storage, whereas it utilizes SSDs as a cache of the most frequently used data. Moreover, [27] introduces an extension which improves the Hadoop performance on streaming data tasks by employing SSDs.

III. Experimental Setup & Methodology

In this Section we describe the most important preliminary elements of our experimental evaluation. Initially we present the basic characteristics of the cluster and we refer to its configuration. In the sequel, we discuss the methodology which we applied in this paper to conduct the comparative experiments.

Characteristic	Samsung SSD	WD Blue HDD
Capacity (GB)	512	500
Buffer size (MB)	512	16
Sequential read (MB/s)	2300	113
Sequential write (MB/s)	1550	107
Avg. access time (msec)	0.07	16

TABLE I
EXPERIMENTAL STORAGE UNITS

A. Hardware Specifications and Cluster Configuration

The experiments were conducted on a laboratory cluster comprised of 6 workstations with identical hardware. One workstation served as the Master node of the cluster, whereas the other five were configured as processing nodes. Each node was equipped with a single Intel XEON E3-1220 CPU@3GHz with 4 processing cores (4 threads) and 8 GB of memory, and it was running Ubuntu Linux 18.04 LTS. The nodes were connected to a standard wired network with single 1 Gbps Ethernet controllers.

Regarding the utilized secondary storage units, each workstation had an identical pair of disks which included: i) a 512 GB Samsung SM951A NVMe2 Solid State Drive with 512 MB of buffer memory, and ii) a Western Digital Blue WD5000AAKX Hard Drive with 500 GB of storage space and a 16 MB buffer. The basic technical specifications of these two disks are given in Table I.

We installed the Apache Hadoop 2.9.0 parallelization framework which operates in conjunction with the Hadoop Distributed File System (HDFS) and employs Apache YARN for resource management and job scheduling. YARN executes a job by launching a number of *executors* across the cluster, which in case of MapReduce, perform Map and Reduce operations in parallel. The executors are run within *containers* on the slave nodes and are created by a special daemon called *NodeManager*. In addition, YARN deploys the *Application Master (AM)* for each submitted job, which is responsible for monitoring the operation of the executors. Notice that AM manifests itself on a random node in the cluster.

In Table II we report the basic configuration parameters of YARN, MapReduce and HDFS. Each container was allowed to allocate exactly one processing core, whereas the sum of memory allocated by all containers within a node was limited to 6 GB (2 GB were left for the needs of the operating system and other critical services). In addition, the maximum memory for Map and Reduce tasks was set to 1.5 and 3 GB respectively. The value of the `slowstart` parameter in the sixth row of Table II prevents early shuffling and dictates that Reducers will be launched only after all Map tasks have been completed[1]. In addition, we prevented speculative execution, since any backup Map or Reduce tasks would distort our comparative results[2].

[1]The Reduce phase includes three stages: shuffle, sort, and reduce; the former may be executed in parallel with the Mappers (early shuffling), whereas the other two start only after all Map tasks have been completed. By setting slowstart=1.0 we dictate that data shuffling also starts after the Map phase.

[2]In case the framework detects a slow worker, speculative execution shall launch the same Map or Reduce task in multiple faster nodes.

Parameter	Value
yarn.scheduler.maximum-allocation-vcores	1
yarn.nodemanager.resource.memory-mb	6144 MB
mapreduce.map.memory.mb	1536 MB
mapreduce.reduce.memory.mb	3072 MB
mapreduce.job.reduce.slowstart.completedmaps	1.0
mapreduce.map.speculative	false
mapreduce.reduce.speculative	false
dfs.block.size	128/512 MB
dfs.replication	1

TABLE II
CLUSTER CONFIGURATION

Given the mandatory presence of AM, the aforementioned configuration will allow the parallel execution of $(4N-1)$ Map or $(2N-1)$ Reduce tasks, where N is the number of nodes in the cluster. For instance, for 5 nodes ($N=5$) Hadoop will create 19 Map or 9 Reduce tasks plus the AM, across the nodes of the entire cluster.

B. Methodology

In the experiments which we present in the following sections we investigated how the usage of SSDs improves performance over the traditional hard drives. In contrast to other similar works, we attempt to examine this enhancement in combination with other characteristics such as the degree of parallelization and the selected HDFS block size.

For this reason, each experiment was conducted on 5 different cluster sizes by activating different number of nodes each time (i.e. 1–5 nodes). These experiments were repeated for both disks and with two settings of HDFS block size, namely 128 MB and 512 MB. The job execution times were retrieved from the corresponding log files; in total, a set of 20 measurements were recorded for each examined algorithm.

Before each new experimental phase, Hadoop and YARN were stopped and the new cluster parameters were set (i.e. slave nodes, utilized disk, block size). In the sequel, the required configuration files were copied to the slave nodes via SSH. Any old data was manually removed from the local file systems and HDFS was reformatted. To ensure fair and unbiased results, we also flushed the caches of the operating system to eliminate the undesired effect of transferring data from main memory instead of the attested storage unit. Finally, at the beginning of each job the dataset was redistributed to the active nodes to ensure that it was visible from the nodes which participated in the test.

IV. PARALLEL DATA PREPROCESSING METHODS

Most datasets are usually provided in a form which is not suitable for processing by the majority of machine learning algorithms. For instance, the pages which are retrieved by Web crawlers and spiders are stored in raw text format, whereas most machine learning algorithms are designed to operate on feature vectors. Consequently, in such occasions it is necessary to apply a series of preprocessing methods with the aim of converting the data to an appropriate form.

Here we examine the performance of various dataset preprocessing methods on the latest publicly available dump of the English version of Wikipedia, dated back to June of 2008. This dump consists of a single XML file which accommodates roughly 18.6 million Web pages. Each page contains a single Wikipedia article, and in total, all pages occupy approximately 64 GB in uncompressed form. Our goal is to prepare the dataset with the aim of discovering and classifying articles which are related to either United Kingdom, or United States. For this purpose we shall later employ the Naive Bayes classifier which accepts vectorized data on its input.

This task was achieved by utilizing Apache Mahout[3], a linear algebra distributed framework which implements a wide variety of parallel machine learning and data engineering algorithms for MapReduce and Spark. More specifically, we used the `seqwiki` and `seq2sparse` programs of this library; the former splits the aforementioned XML file into a set of *Sequence files*, whereas the latter performs a series of transformations to convert these Sequence files into a set of normalized vectors. In the following Subsections we describe each method in details and we compare their performance on our Hadoop cluster with HDDs and SSDs.

A. Creation of Sequence Files

The single XML file which accommodates the Wikipedia dataset is undoubtedly in an unsuitable form for parallel processing by multiple nodes. For this reason, it is required that we split it into a set of smaller files and distribute these files to the nodes of the cluster through HDFS. Furthermore, since we plan to use this dataset to identify documents which are relevant to United Kingdom or United States, it is necessary that we also create a small text file which contains the literals of these two categories.

The `seqwiki` program of Mahout accepts the text file with the two categories and starts a MapReduce job to parse each document in the XML file. This procedure identifies the documents with a category tag which matches a line in the category file. If no match is found, the document is assigned an unknown category. Finally, the documents are written to a Sequence file in a <Text,Text> format.

The top left diagram of Figure 1 illustrates the execution times of this procedure on our cluster with standard HDDs and SSDs. As stated earlier, we present measurements for two settings of the HDFS block size, namely 128 MB and 512 MB. The results indicate that the selected block size indeed affects performance; SSDs and HDDs with a block size of 512 MB are consistently faster than SSDs and HDDs with a block size of 128 MB for all cluster sizes. However, the performance gap decreases from 15% (for 2 nodes) to almost 1%, as the number of the active machines increases. Therefore, HDFS block size is important only for small degrees of parallelization.

For equally sized blocks (either 128 MB, or 512 MB), SSDs are faster than HDDs by an infinitesimal margin of 0.5-1%. In all cases, the degree of parallelization improves performance. The gain is initially super-linear (for 1–2 nodes), becomes linear and sub-linear as the number of nodes increases (2–5), and finally disappears for more than 5 nodes.

[3]https://mahout.apache.org/

B. tf-idf Vectorization

Vectorization is the process of transforming a dataset into a set of *feature vectors*. More specifically, the features of each record of the dataset are associated with integer identifiers and they are assigned a weight value. These (featureID, weight) pairs comprise the components of the feature vector of the record. This process is particularly important in the area of data mining, since the majority of machine learning approaches (including regression, classification, and clustering algorithms) are designed to operate on vectorized data.

One of the most popular data vectorization techniques is based on tf-idf, a well-established metric which originates from the information retrieval systems. In particular, tf-idf vectorization is widely used to transform textual datasets, like the Wikipedia dump which we utilize here. Mahout provides a special program, `seq2sparse`, which vectorizes a dataset according to tf-idf by performing the following steps:

- *Tokenization:* This process reads the input Sequence files and extracts the distinct words and/or the n-grams from each document. In the sequel, these words are stored within a dictionary data structure accompanied by their in-document frequency values.
- *WordCount:* A traditional MapReduce job which counts the number of the occurrences of each word of the dictionary in the entire document collection.
- *Vectorization:* Initially, each word in the dictionary is assigned a unique integer identifier. In the sequel, the Map phase creates and emits (document,feature ID) pairs which are then passed to the Reducers. The pairs are grouped by key and finally merged together to output the desired (document, list of feature IDs) vectors.
- *Dimensionality Reduction:* The vectors of the previous phase may consist of thousands of features of which some may be of small value for the effectiveness of a machine learning algorithm. More importantly, such large vectors require huge amounts of memory to be processed. For this reason, a dimensionality reduction (or vector pruning) method must be applied. In tf-idf scheme, this process is divided in two phases; in the first phase the tf-idf value of all features is computed (based on the aforementioned in-document and collection frequencies). In the second phase, a percentage of the features with the lowest tf-idf values are removed from the vectors.

Now let us elaborate in the performance measurements of these vectorization steps. Figure 1 illustrates all execution times on various cluster sizes for two sizes of HDFS blocks.

In the upper right diagram of Figure 1 we show the execution times of the document tokenization process. The results indicate that SSDs are considerably faster than HDDs for all cluster sizes, regardless of the selected block size. More specifically, on the 5-node cluster and for the default block size of 128 MB, the SSD outperformed both HDD-128 and

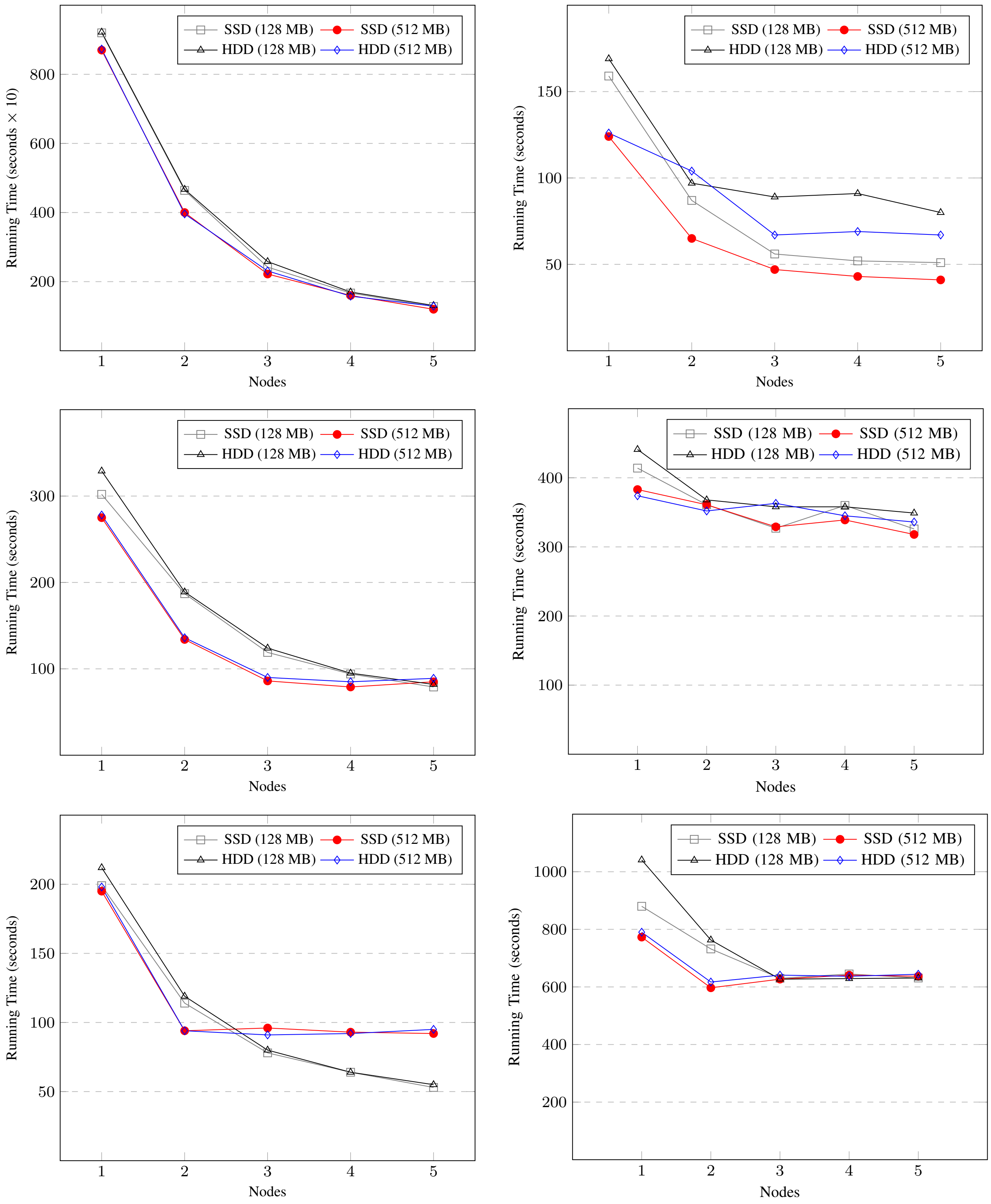


Fig. 1. Performance comparison of various dataset preprocessing methods on a 5-node cluster with the Wikipedia dataset: i) Creation of Sequence Files from Wikipedia XML dump (top left), ii) document tokezination (top right), iii) WordCount (middle left), iv) dataset vectorization (middle right), v) tf-idf calculation (bottom left), and vi) tf-idf pruning (bottom right).

HDD-512 by roughly 36% and 24%, respectively. In smaller clusters the benefits are similar, whereas for larger block sizes SSDs perform even better; the SSD-512 scheme was faster than HDD-128 and HDD-512 by approximately 50% and 41%, which is translated to an average speedup of about 2 times. In addition, in this experiment the selection of large block sizes led to enhanced performance: about 20% for SSDs and 16% for HDDs on the 5-node cluster.

Regarding the WordCount task (middle left diagram of Figure 1), the results are similar to those of Sequence files creation. In details, SSDs outperformed HDDs by a small margin of about 3% on our 5-node cluster and only when the comparison is made on equal block sizes. In only one case, that is, selected block size equal to 128 MB in a single-node cluster, the SSD was faster than the HDD by more than 8%. Furthermore, the larger block sizes exhibit better performance for small degrees of parallelization (28% in 2- and 3-node clusters and 16% in 4-node clusters for SSDs), however, in 5-node clusters the 128 MB setting leads to slightly improved efficiency of 3.5% for both storage units.

The fourth method, dataset vectorization, is an example algorithm which does not benefit significantly from parallelization. The middle right diagram of Figure 1 shows that performance is similar on clusters of 2–5 nodes. However, the selected block size and the type of storage media result in different efficiency measurements as the size of the cluster increases. For instance, on the 3-node cluster the HDD-128 combination was faster than HDD-512, whereas the opposite holds for clusters with 4 and 5 servers. In the greatest degree of parallelization, SSDs were faster than HDDs and the gains were larger when the HDFS block size was set equal to 512 MB. Hence, SSD-512 outperformed HDD-128 and HDD-512 by 9% and 5.5% respectively, whereas SSD-128 was faster than HDD-128 and HDD-512 by 6.5% and 3%, respectively.

The examination of the bottom left diagram of Figure 1 reveals a new pattern; in tf-idf calculation, the large block size of 512 MB outperforms the 128 MB setting for single and 2-node clusters. However, after this point the execution times do not improve further. On the contrary, the storage units with 128 MB block sizes continue to scale-up and on the 5-node cluster they perform nearly twice as fast as their 512 MB counterparts. In this occasion, SSDs are marginally faster than HDDs by 3.5%.

Similarly to the vectors creation process, the performance of the vectors pruning algorithm does not improve as the cluster size increases. The only notable difference is at the transition from one to two machines which leads to an enhancement which ranges from 16% to about 23.5%. For larger cluster sizes, both storage media exhibit equal execution times regardless of the selected HDFS block size. Notice that in this case, the increase in the cluster size results in slightly degraded performance of all storage media.

Finally, among these experiments there were some rare cases where HDDs achieved slightly better performance than SSDs for equally sized HDFS blocks. Although surprising, our work is not the first which mentions such results.

V. Naive Bayes Classifier

Naive Bayes is one of the most common classification algorithms mainly due to its simplicity and its efficiency on large-scale datasets. It is based on the Bayes theorem:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} \tag{1}$$

and assumes independence among the predictors (i.e. features). During the training phase, the algorithm builds a frequency table based on the observations of the training set. This table stores the frequency of each (feature x, class y) pair and it is used during the test phase to classify an unlabelled entry. In particular, the classifier initially extracts the features of the entry and computes three probabilities $P(y)$, $P(x)$ and $P(x|y)$. The two former represent the prior probabilities of a class y and a feature x respectively, whereas the latter is is the likelihood of feature x given the class y.

The left diagram of Figure 2 depicts the running times of the Naive Bayes model training process. The input of this task is the vectorized data which were generated by the preprocessing methods of Section IV. Here we encounter a pattern which is similar to that of tf-idf calculation (bottom left diagram of Figure 1); that is, for small levels of parallelization (1–2 nodes) the setting of 512 MB for the HDFS block size improves performance. However, the situation is reversed when more machines are added into the cluster. In general, the 128 MB setting exhibits a normal behavior against parallelization, that is, the running times monotonically decrease with the number of active nodes. In this algorithm SSDs outperformed HDDs by a small margin of 3.8% and 15.7% for a block size of 128 MB and 512 MB, respectively.

Regarding test phase (right part of Figure 2), the execution times are affected by the selected HDFS block size in a manner which is identical to the times of the training phase. SSDs in combination with a setting of 128 MB for the block size were faster than HDD-128 by a margin between 7.4% (on the 5-node cluster) and 19.5% (on the 3-node cluster).

VI. Linear Regression

Linear Regression is another popular classification method. It assigns a label y to a sample x with features $x_1, x_2, ..., x_n$, by plotting a regression line given by the following equation:

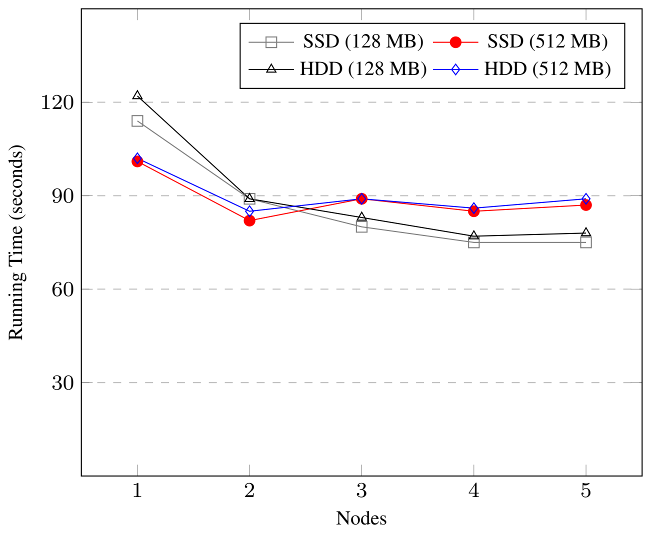
$$y = \theta_0 + \sum_{i=1}^{n} \theta_i x_i \tag{2}$$

where $\theta_0, \theta_1, ..., \theta_n$ are the features learning coefficients. There are multiple techniques for their computation; one of the the most common employs the iterative least squares method with the aim of minimizing the overall squared test error:

$$\min_{\theta} \sum_{i=1}^{n} (y - y')^2 \tag{3}$$

where y and y' are the real and the predicted classes.

In this experiment we decided to utilize a dataset with different characteristics than the Wikipedia dump. More specifically,



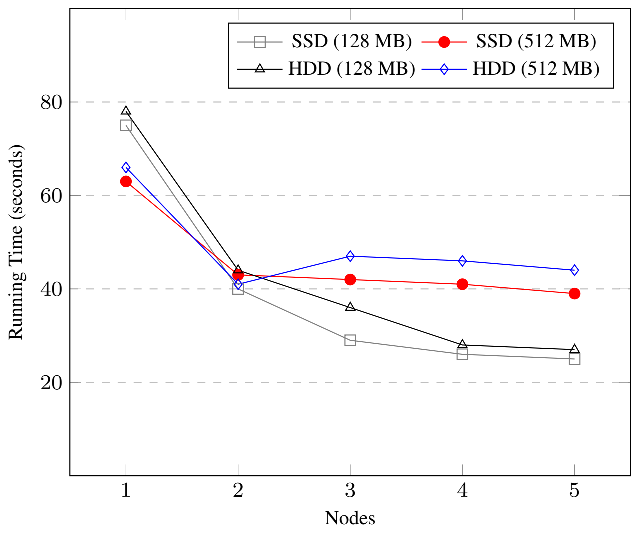
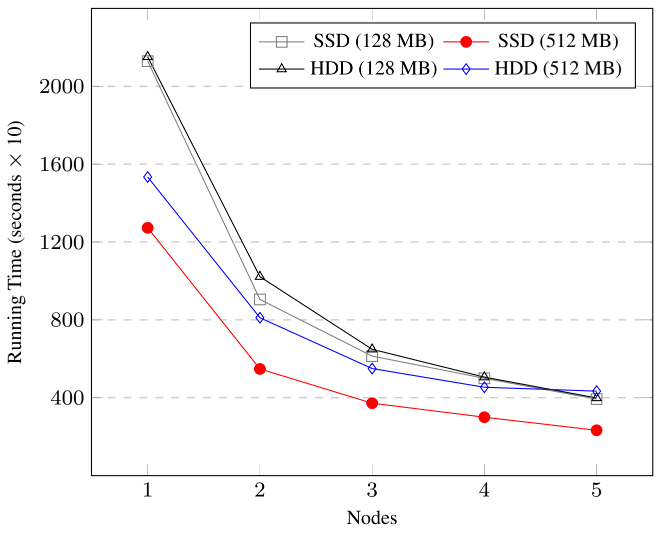


Fig. 2. Performance of the Naive Bayes classifier on a 5-node cluster with the Wikipedia dataset: i) Model training (left), and ii) testing (right).



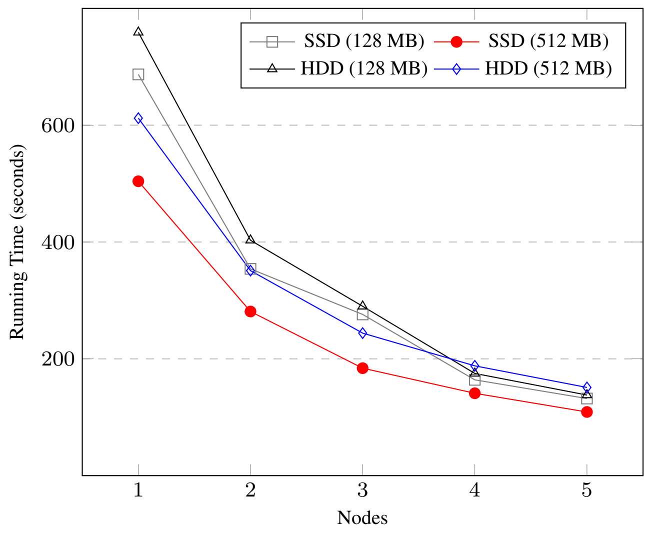


Fig. 3. Linear Regression performance on a 5-node cluster with the synthetic HIGGS dataset: i) Model training (10 iterations, left), and ii) testing (right).

we desired a large-scale non-textual repository of observations with low dimensionality. Therefore, we employed the HIGGS dataset[4], a collection of 11 million Monte Carlo simulations of high-energy particle collisions. Each sample is accompanied by its class label (which is binary) and a set of 28 features which represent kinematic and other properties of the particles.

Furthermore, we considered that the 8 GB of the dataset were an unacceptably small size for reliable large-scale processing analysis. For this reason, we generated a new collection by creating a random number of copies (between 10 and 20) of the records of the HIGGS dataset. This process led to a 127 GB synthetic dataset comprised of more than 236 million samples. The sizes of the training and test sets were equal to the 80% and 20% of the synthetic dataset, respectively.

In this experiment we used our own Java implementation of Linear Regression. The running times for the training and test phases are depicted in the left and right diagrams of Figure 3, respectively. Regarding the former, the measured times concern the accumulated durations of 10 iterations for the calculation of the θ values with the Batch Gradient Descent algorithm. The SSDs outperformed the HDDs by a substantial margin for block sizes equal to 512 MB. Notice that the percentage difference between SSD and HDD execution times increases as more nodes are added to the cluster; the difference grows from 17% (on the single-node cluster) to 46.3% on the 5-node cluster. On the other hand, for the default value of HDFS block size, Linear Regression on SSD outperformed HDD by only 1.5%. Both HDD-128 and SSD-128 were considerably slower than SSD-512; on the 5-node cluster, the performance gap was equal to 42% and 41%, respectively.

Regarding the test phase, the SSD was faster than the HDD and performed better for a block size equal to 512 MB. On the 5-node cluster, SSD-128 and SSD-512 outperformed HDD-128 and HDD-512 by 4.5% and 28%, respectively.

[4]https://archive.ics.uci.edu/ml/datasets/HIGGS

VII. Conclusions and Future Work

In this paper we investigated the performance of several machine learning and data engineering algorithms on MapReduce clusters equipped with SSDs. We performed exhaustive experiments on a 5-node laboratory cluster and we compared the efficiency of these algorithms on a standard magnetic disk and a high-performing SSD device. During these experiments we considered the impact of various parameters, including i) the degree of parallelization (i.e. the number of nodes in the cluster), ii) HDFS block size, iii) different datasets, and iv) different types of tasks such as CPU and disk intensive data processing methods, and iterative classifiers. The conclusions which derive from our research are:

- The benefits of utilizing SSDs on MapReduce clusters are diverse and depend on the nature of the executed task. In a portion of our examined algorithms, SSDs improved execution times by a small margin of 1–8%. Nevertheless, in two cases (tokenization and Linear Regression model training), SSDs were almost twice as fast as HDDs.
- Not all algorithms benefit from parallelization. There are cases where the addition of more machines to the cluster may hurt efficiency, even a little.
- The HDFS block size plays a significant role in the performance of parallel machine learning algorithms. In most cases its impact is greater for small clusters and degrades as the number of the nodes increases. However, there are two exceptions where HDFS block size is important on 5-node clusters: tokenization and tf-idf calculation. In the former case the efficiency is improved for large block sizes, whereas the opposite holds on the latter case.

Our future research includes an extended study of the performance of additional data mining algorithms. We aim at evaluating regression, recommendation, and clustering algorithms on MapReduce clusters not only with SSDs, but also with new types of storage devices such as Optane XPoint.

References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin *et al.*, "Apache Spark: a Unified Engine for Big Data Processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016.

[3] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A System for Large-Scale Graph Processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, 2010, pp. 135–146.

[4] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: Interactive Analysis of Web-Scale Datasets," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 330–339, 2010.

[5] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, K. Olukotun, and A. Y. Ng, "Map-Reduce for Machine Learning on Multicore," in *Advances in Neural Information Processing Systems*, 2007, pp. 281–288.

[6] W. Zhao, H. Ma, and Q. He, "Parallel k-Means Clustering Based on MapReduce," in *Proceedings of the IEEE International Conference on Cloud Computing (CloudCom 2009)*, 2009, pp. 674–679.

[7] L. Akritidis and P. Bozanis, "A Supervised Machine Learning Classification Algorithm for Research Articles," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC)*, 2013, pp. 115–120.

[8] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly *et al.*, "The Genome Analysis Toolkit: a MapReduce Framework for Analyzing Next-Generation DNA Sequencing Sata," *Genome Research*, 2010.

[9] A. Ghoting, R. Krishnamurthy, E. Pednault, B. Reinwald, V. Sindhwani, S. Tatikonda, Y. Tian, and S. Vaithyanathan, "SystemML: Declarative Machine Learning on MapReduce," in *Proceedings of the 27th IEEE International Conference on Data Engineering (ICDE)*, 2011, pp. 231–242.

[10] C.-H. Wu, L.-P. Chang, and T.-W. Kuo, "An Efficient R-Tree Implementation over Flash-Memory Storage Systems," in *Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems*, 2003, pp. 17–24.

[11] T. Emrich, F. Graf, H.-P. Kriegel, M. Schubert, and M. Thoma, "On the Impact of Flash SSDs on Spatial Indexing," in *Proceedings of the 6th International Workshop on Data Management on New Hardware*, 2010, pp. 3–8.

[12] M. Sarwat, M. F. Mokbel, X. Zhou, and S. Nath, "FAST: a Generic Framework for Flash-Aware Spatial Trees," in *Proceedings of the International Symposium on Spatial and Temporal Databases*, 2011, pp. 149–167.

[13] G. Li, P. Zhao, L. Yuan, and S. Gao, "Efficient implementation of a multi-dimensional index structure over flash memory storage systems," *The Journal of Supercomputing*, vol. 64, no. 3, pp. 1055–1074, 2013.

[14] A. Fevgas and P. Bozanis, "Grid-File: Towards to a Flash Efficient Multi-Dimensional Index," in *Proceedings of the 26th International Conference on Database and Expert Systems Applications*, 2015, pp. 285–294.

[15] S.-H. Kang, D.-H. Koo, W.-H. Kang, and S.-W. Lee, "A Case for Flash Memory SSD in Hadoop Applications," *International Journal of Control and Automation*, vol. 6, no. 1, pp. 201–210, 2013.

[16] S. Sur, H. Wang, J. Huang, X. Ouyang, and D. K. Panda, "Can High-Performance Interconnects Benefit Hadoop Distributed File System," in *Proceedings of the 2010 Workshop on Micro Architectural Support for Virtualization, Data Center Computing, and Clouds (MASVDC)*, 2010.

[17] N. S. Islam, M. W. Rahman, J. Jose, R. Rajachandrasekar, H. Wang, H. Subramoni, C. Murthy, and D. K. Panda, "High Performance RDMA-based Design of HDFS over InfiniBand," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2012.

[18] S. Moon, J. Lee, and Y. S. Kee, "Introducing SSDs to the Hadoop MapReduce Framework," in *Proceedings of the 7th IEEE International Conference on Cloud Computing (CLOUD)*, 2014, pp. 272–279.

[19] K. Kambatla and Y. Chen, "The Truth About MapReduce Performance on SSDs," in *Proceedings of the USENIX Large Installation System Administration Conference (LISA)*, 2014, pp. 109–118.

[20] D. Wu, W. Luo, W. Xie, X. Ji, J. He, and D. Wu, "Understanding the Impacts of Solid-state Storage on the Hadoop Performance," in *Proceedings of the 2013 International Conference on Advanced Cloud and Big Data (CBD)*, 2013, pp. 125–130.

[21] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The HiBench Benchmark Suite: Characterization of the MapReduce-based Data Analysis," in *Proceedings of the 26th IEEE International Conference on Data Engineering Workshops (ICDEW)*, 2010, pp. 41–51.

[22] P. Saxena and D. J. Chou, "How much Solid State Drive can Improve the Performance of Hadoop Cluster? Performance Evaluation of Hadoop on SSD and HDD," *International Journal of Modern Communication Technologies & Research*, vol. 2, no. 5, 2014.

[23] J. Hong, L. Li, C. Han, B. Jin, Q. Yang, and Z. Yang, "Optimizing Hadoop Framework for Solid State Drives," in *Proceedings of the 2016 IEEE International Congress on Big Data*, 2016, pp. 9–17.

[24] S. Moon, J. Lee, X. Sun, and Y.-s. Kee, "Optimizing the Hadoop MapReduce Framework with High-Performance Storage Devices," *The Journal of Supercomputing*, vol. 71, no. 9, pp. 3525–3548, 2015.

[25] S. Ahn and S. Park, "An Analytical Approach to Evaluation of SSD Effects under MapReduce Workloads," *Journal of Semiconductor Technology and Science*, vol. 15, no. 5, pp. 511–518, 2015.

[26] K. Krish, M. S. Iqbal, and A. R. Butt, "VENU: Orchestrating SSDs in Hadoop Storage," in *Proccedings of the 2014 IEEE International Conference on Big Data*, 2014, pp. 207–212.

[27] A. Kaitoua, H. Hajj, M. A. Saghir, H. Artail, H. Akkary, M. Awad, M. Sharafeddine, and K. Mershad, "Hadoop Extensions for Distributed Computing on Reconfigurable Active SSD Clusters," *ACM Transactions on Architecture and Code Optimization*, vol. 11, no. 2, pp. 1–26, 2014.