International Journal on Artificial Intelligence Tools

Evaluating the Effects of Modern Storage Devices on the Efficiency of Parallel Machine Learning Algorithms

Leonidas Akritidis, Athanasios Fevgas, Panagiota Tsompanopoulou, Panayiotis Bozanis
Data Structuring & Engineering Lab
Department of Electrical and Computer Engineering
University of Thessaly
37 Glavani & 28th October Str, 382 21 Volos, Greece
{*leoakr, fevgas, yota, pbozanis*}*@e-ce.uth.gr*

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

BigData analytics is presently one of the most emerging areas of research for both organizations and enterprises. The requirement for deployment of efficient machine learning algorithms over huge amounts of data led to the development of parallelization frameworks and of specialized libraries (like Mahout and MLlib) which implement the most important among these algorithms. Moreover, the recent advances in storage technology resulted in the introduction of high-performing devices, broadly known as Solid State Drives (SSDs). Compared to the traditional Hard Drives (HDDs), SSDs offer considerably higher performance and lower power consumption. Motivated by these appealing features and the growing necessity for efficient large-scale data processing, we compared the performance of several machine learning algorithms on MapReduce clusters whose nodes are equipped with HDDs, SSDs, and devices which implement the latest 3D XPoint technology. In particular, we evaluate several dataset preprocessing methods like vectorization and dimensionality reduction, two supervised classifiers, Naive Bayes and Linear Regression, and the popular k-Means clustering algorithm. We use an experimental cluster equipped with the three aforementioned storage devices under different configurations, and two large datasets, Wikipedia and HIGGS. The experiments showed that the benefits which derive from the usage of SSDs depend on the cluster setup and the nature of the applied algorithms.

Keywords: Machine Learning; Data Mining; MapReduce; Performance; Parallel Algorithms; Flash Memory; SSDs; 3D XPoint; Distributed Storage.

1. Introduction

The continuous improvement of the effectiveness of the machine learning algorithms led to their enormous adoption in a broad spectrum of applications. In particular, such algorithms are presently utilized to address numerous problems including the analysis of medical data, the intelligent processing of astrophysical observations, numerous image and pattern recognition tasks, the prediction of economical models, the deployment of data mining algorithms in large text corpora, natural language processing, and countless others.

The usefulness of the machine learning methods has attracted a great amount of research towards the improvement of their performance, which in turn resulted in a wealth of state-of-the-art solutions in a variety of domains. By employing sophisticated techniques from intimate and foreign scientific areas (such as statistics and biology), these solutions have evolved from specialized research projects to popular real-world products.

The aforementioned techniques constitute the basic criterion for the categorization of the current machine learning methods. In particular, the supervised learning approaches construct (i.e. train) problem-solving models by exploiting preprocessed sets of data with known attributes and features (called training sets). After the training phase, the constructed models are subsequently used to solve problems such as classification, regression, entity recognition, pattern matching, and so on. On the other hand, the unsupervised learning methods do not rely on training sets, but on other properties such as distances and similarities of the involved data. The semi-supervised approaches combine the characteristics of both of these methods, whereas the re-enforced learning approaches adapt the learnt models according to their performance on a portion of the training examples. In addition, the recent advances on the effectiveness and efficiency of neural networks resulted in an explosion of another branch of research, regarding the deep learning methods. These methods constitute one of the best studied research fields in the machine learning and artificial intelligence area.

Furthermore, the blossoming of machine learning algorithms coincided with a huge increase in the volume of data on which they operate. Nowadays, tremendous amounts of data are generated and processed not only by the well-established Web information systems and social networks, but also by billions of networked devices, including mobile phones, wireless sensors, and IoT systems. Apart from its huge size, this data is also highly diverse, since it may represent numerous types of information including articles, reports, documents, posts, messages, user logs, media content, images, locations, measurements, and so on. In addition, it may be subject to frequent changes, thus exhibiting high volatility. These three basic attributes (volume, variety, and velocity), along with some others (veracity and value) comprise the 5 Vs of what is now known as BigData.

The effective and efficient management and processing of BigData is another hot topic of research. More importantly, the combination of machine learning algorithms and BigData is a topic which quickly gained the attention of the researchers. The relevant work can be divided into two primary categories: The first one regards the design of robust machine learning methods with the aim of sustaining the required heavy workloads; the second concerns the development of parallel algorithms which can be executed across large clusters of interconnected machines.

This article investigates the performance of various popular machine learning algorithms on MapReduce clusters. MapReduce is a distributed, fault-tolerant framework which was designed to facilitate the deployment of parallel algorithms across large clusters. More specifically, we are primarily motivated by the improved I/O

performance of the modern non volatile memory (NVM) storage devices, such as NAND Flash SSDs and the most recent devices based on the 3D XPoint technology. Our work focuses on the examination of the execution times of these algorithms on interconnected workstations equipped with such devices, and evaluates the potential benefits in the execution of parallel data mining jobs.

Our experiments were conducted on a 5-node cluster by utilizing two large-scale datasets: i) a complete dump of the articles of the English version of Wikipedia, and ii) HIGGS, a collection which derives from the scientific area of high-energy Physics and contains observations of particle collisions. In this environment, we measure the running times of numerous dataset preprocessing methods (such as vectorization and tf-idf vectors pruning), one of the most popular clustering algorithms, k-Means, and two supervised classifiers, Naive Bayes and Linear Regression. The contributions of this research are summarized in the following list:

- it is not based on specialized data-intensive benchmarks such as Terasort and Teragen. In contrast, it evaluates the running performance of several machine learning and data engineering algorithms; consequently, the usefulness of SSDs is measured through popular, real-world parallel applications.
- it compares the performance of MapReduce with HDDs, SSDs and 3D XPoint devices by applying different degrees of parallelization with the aim of discovering potential relationships between the external storage and the size of the cluster. In particular, the experiments are conducted on clusters of 5 different 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, i.e, 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 this paper is organized as follows: Section 2 refers to the most significant relevant work on the MapReduce framework, the SSD storage devices, and their combination on large clusters. Section 3 contains a brief description of the hardware and software systems which were used in this study, whereas Section 4 describes the different setups of the experimental cluster. The results of the experimental evaluation of three storage media with multiple machine learning algorithms are presented in Section 5. Finally, Section 6 summarizes the conclusions of the article.

2. Related Work

Although numerous novel parallelization frameworks (such as Spark[1] and Flink[2]) have been introduced during the last years, MapReduce is still one of the most widespread solutions for processing BigData in large clusters. This is due to its reliability, its capability of handling really massive amounts of data, and the large support base which still counts tens of thousands of installations.

A large number of algorithms based on MapReduce have been developed during the past few years for numerous data-intensive problems. Examples include data mining, knowledge engineering, and machine learning tasks.[3–8] Furthermore, a portion of the recent relevant works focus on the design of distributed deep learning algorithms on MapReduce. In[9] the authors present a distributed learning paradigm for the Restricted Boltzmann Machines and the backpropagation algorithm using MapReduce. A study of the problem of parallelizing the L-BFGS algorithm in large clusters of shared-nothing commodity machines is presented in.[10]

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,[11] R*-Trees,[12] generic spatial indexes,[13] K-D-B-Trees[14] and Grid Files.[15] In[16] the authors provide a complete survey of the most significant data structures and algorithms for Flash memories.

The improvement in the execution times of the parallel algorithms on MapReduce clusters equipped with SSDs is presently one of the most emerging relevant research topics. This study is builds on the preliminary work of[17] and extends it in two ways. At first, it examines modern storage devices which implement the most recent 3D XPoint technology, and second, it includes experiments with additional machine learning algorithms.

The first work which studied the effects of the SSD devices in Hadoop clusters was.[18] More specifically, the authors launched multiple virtual nodes on a single machine and reported a considerable tripling of the performance compared to the traditional HDDs. Nonetheless, the experiments on a single-node cluster cannot accurately represent the multi-node environment, since the execution of a job in multi-node clusters is affected by several independent factors, such as data transfer errors, nodes interconnection method, various latencies, and nodes malfunction or death. Two works which support this claim are[19] and;[20] both of them demonstrate that the performance of MapReduce jobs depends heavily on the available network bandwidth. Consequently, the enhancement of the nodes interconnection method, reduces the running times on both storage types. However, the gains are reported to be higher in the case of SSDs.

Similar efforts which compare Hadoop performance on clusters equipped with magnetic disks and SSDs are presented in[21] and.[22] 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. Additionally, in[23] the authors analyze the performance improvement on datasets of variable sizes. They conduct experiments on a five-node cluster by employing the Intel HiBench benchmark[24] and conclude that the benefits of utilizing SSDs are magnified when the input of a MapReduce algorithm becomes larger.

In contrast to all other related works, the authors of[25] state that Hadoop per-

forms nearly the same on clusters equipped with either HDDs, or SSDs. In[26–28] the authors investigate various cost-effective techniques which enhance the performance of Hadoop clusters with SSDs, with respect to energy consumption.

Finally, a significant part of the relevant research is dedicated to the development of specialized MapReduce extensions which aim to improve the performance of the framework on clusters equipped with modern storage devices. One such extension is VENU,[29] a system which is designed to utilize the traditional HDDs for storing generic types of data, whereas it employs SSDs to form a cache of the most frequently used (i.e. the hottest) data. Moreover,[30] introduces another extension which improves the Hadoop performance on streaming data tasks by using SSDs.

3. Preliminary Elements

This Section briefly describes the basic characteristics of the various systems which were employed in this study. Initially, an overview of the MapReduce framework, its underlying distributed file system, and the YARN resource management system is presented in Subsection 3.1. Moreover, Subsection 3.2 contains the most significant elements of the NVM technology.

3.1. *MapReduce, DFS & YARN*

The requirement for effortless and efficient deployment of parallel algorithms on large clusters led to the development of various parallelization frameworks. One of the most successful among them is *MapReduce*, a programming model which was introduced by Google in 2008.[31] Its primary contributions was scalability and fault tolerance, combined with the simplicity of implementation of parallel algorithms. More specifically, the framework was designed to allow the researchers focus on the task of parallel algorithm development, instead of dealing with complex network programming details such as message passing, socket handling, protocol-wise communications, and thread-safe server implementations. An open-source version of MapReduce, called *Hadoop*, is provided by the Apache foundation[a].

Apart from the distribution libraries, Hadoop also includes a distributed file system (broadly known as HDFS), which supports fault-tolerance through the replication of data across the nodes of the cluster. The objective of simplicity is achieved by the adoption of a programming model which requires from the researchers to develop their solutions by implementing only two functions, *map* and *reduce*. More specifically, the model dictates that the input data is represented by sequences of key/value pairs; these pairs are then split, distributed and replicated by HDFS to the nodes of the cluster.

Figure 1 depicts a high-level schematic diagram of the MapReduce architecture. The parallel execution of a job is coordinated by a single machine, the *Master*.

[a]https://hadoop.apache.org/

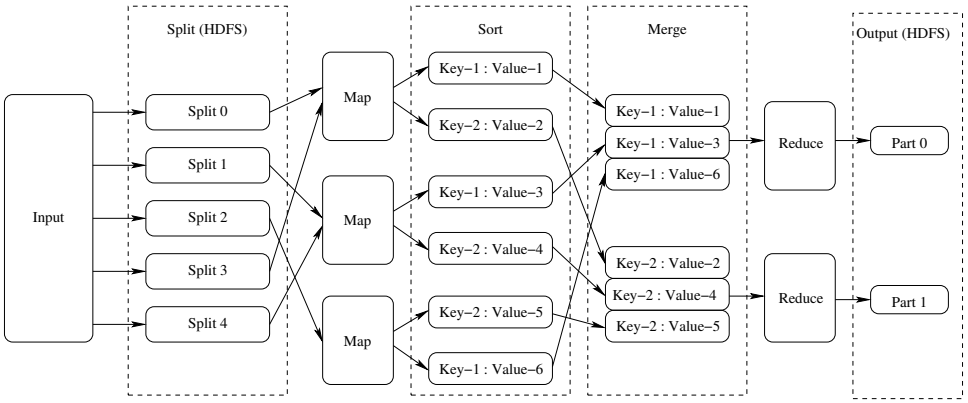

Fig. 1. A high-level diagram of MapReduce architecture

Initially, the Master assigns the processing of each part of the input 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 so the values which are associated with the same intermediate key are transferred to the same *Reducer*. After 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.

Two additional key points which affect the execution of a parallel job within a cluster are *scheduling* and *resource management*. YARN is another open-source cluster management technology designed to address these two issues. It is based on a module called the *Resource Manager* which monitors, allocates and frees the resources required to perform an operation. In particular, the system 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 the *Node Manager*. Furthermore, 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.

3.2. *Non Volatile Memory Basics*

Solid State Drives is a relatively new type of storage medium which is gradually replacing the traditional magnetic Hard Disk Drives in both consumer computer systems and the big data centers. They employ non-volatile memories (usually NAND Flash) to store data, instead of spinning magnetic plates used by HDDs.

Nowadays, NAND Flash is the most popular type of non-volatile memory. The information is stored within the Flash cells by trapping electrical charges. The first generation of Flash was capable of storing only one bit per cell (SLC), using a

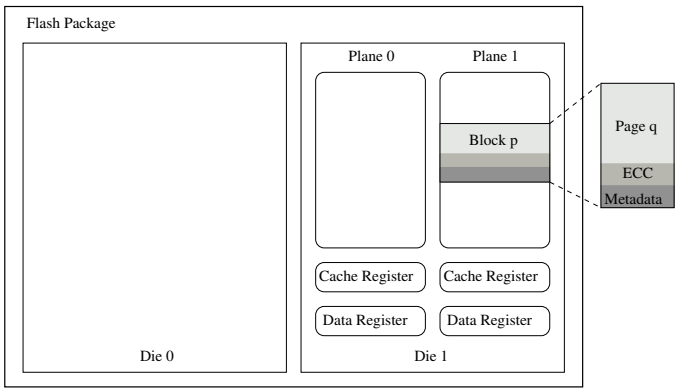

Fig. 2. The main components of a Flash package with 2 dies.

single voltage threshold to distinguish between '0' and '1'. In the following years, the number of thresholds was increased, enabling MLC (multi level), TLC (triple level), and recently QLC (quadruple level) cells, which are able to store two, three, and four bits, respectively. A Flash package may include a few NAND Flash memory chips (dies). The dies within a package are able to execute different read, program or erase commands simultaneously.

Fig. 2 depicts a typical NAND Flash package with two dies. Each die is further decomposed to multiple planes. The planes in a die are able to execute the same command concurrently. Planes are assembled by blocks and registers. Each block consists of a number of pages. Pages are the smallest programmable units in NAND Flash, while blocks are the smallest erasable ones.

Flash cells gradually lose their ability to retain electric charge due to continuous program/erase (PE) operations (wear-out). Thus, they usually present increased bit error rates after a certain number of PE cycles. Other types of errors are: i) charge retention errors, ii) read or write disturbance errors or iii) failures during erase operations. The most common reasons of malfunctioning are wearing-out, aging or exposure to high temperature conditions. Blocks that exhibit either unrecoverable errors or error rates above a tolerance, are marked as bad and are excluded from future usage.

3D XPoint is a new non-volatile memory technology introduced by Intel and Micron in 2017. It provides up to 10^3X better access time than NAND Flash and 10X higher density than DRAM.[32] 3D XPoint is based on a layered cross-point architecture that enables access at bit level. Moreover, in opposition to Flash, it supports writes-in-place. This simplifies the design of the storage device, eliminating the necessity of out-of-place updates and garbage collection mechanisms.[33]

According to[33] there are three possible uses of 3D XPoint memory within the existing computer systems: i) as a storage medium to persist data and applications by utilizing a block device abstraction, ii) as a mass-volume and low-cost extension

Table 1. Experimental storage units

Attribute	Intel Optane SSD	Samsung SSD	WD Blue HDD
Sequential read (MB/s)	1500	2300	113
Sequential write (MB/s)	640	1550	107
Avg. access time (msec)	< 0.01	0.07	16.55
Capacity (GB)	58	512	500
Buffer size (MB)		512	16

of DRAM where persistence is not a prerequisite, and, iii) as a persistent main memory which is directly accessed by the CPU. Currently, 3D XPoint has been deployed within block storage devices (SSD) by Intel under Optane brand name. However, the first products for the memory bus are expected in the middle of 2019. Recent studies[33,34] unveil that Optane SSDs can deliver high IOPS throughput even at small queue depths, opposite to NAND SSDs which provide their best performance at high queue depths, exploiting their internal parallelization. The average latency measured for a 4KB I/O is 16μs which is 10X lower compared to the latency of NAND based SSDs.

4. Experimental Setup & Methodology

In this Section we refer to the most important preliminary elements of our experimental evaluation. Subsection 4.1 presents the hardware and software components of the cluster and discusses its basic configuration options. Moreover, Subsection 4.2 describes the methodology that was applied in this paper to conduct the comparative experiments.

4.1. *Cluster Components & Configuration*

The experiments were conducted on a cluster of 6 commodity machines which were installed in a laboratory environment. The hardware specifications were identical for all 6 workstations and included a single Intel XEON E3-1220 CPU@3GHz with 4 processing cores (4 threads), and 8 GB of memory. The interconnection interface was comprised of a single gigabit Ethernet controller per node, which was attached to a 1 Gbps switch.

Each workstation was equipped with an identical array of 3 disks which included:

- a 58 GB Intel SSDPEK1W060GA Optane SSD storage unit,
- a 512 GB Samsung SM951A NVMe2 Solid State Drive with 512 MB of buffer memory, and
- a Western Digital Blue WD5000AAKX Hard Drive with 500 GB of storage space and a 16 MB buffer.

The basic attributes of these three storage units are reported in Table 1.

Similarly to the hardware, the installed software was also identical in all 6 workstations. In particular, the selected parallelization framework was Apache Hadoop

Table 2. Cluster configuration parameters

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

2.9.0, accompanied by the Hadoop Distributed File System (HDFS) for distributed storage. As stated earlier, the tasks of resource management and job scheduling within the cluster were assigned to Apache YARN. The operating system in each machine was Ubuntu Linux 18.04 LTS.

There are hundreds of parameters which control the behavior of the entire framework and determine the execution flow of a job across the cluster. The most important of them are reported in Table 2. The first two concern YARN and reveal that 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 this study we applied two different setups for the maximum allowed memory of the Map and Reduce tasks. The first one, which we shall call *Setup A*, permitted each Map and Reduce task to allocate at most 1.5 GB and 3 GB of memory, respectively. These limits were adequate for the two supervised classifiers of our test, namely, Naive Bayes and Linear Regression, and allowed them to complete their task without memory shortage problems. However, the memory requirements of k-Means clustering were significantly higher. For this reason, we used *Setup B*, according to which the aforementioned limits were doubled; each Map and Reduce task was permitted to consume at most 3 GB and 6 GB respectively.

These two setups combined with the aforementioned storage units and the selected HDFS block sizes create additional execution plans. For instance, Setup A-SSD-128 refers to a cluster with the following attributes: i) 1.5 GB maximum allowed memory for the Map tasks, ii) 3 GB maximum allowed memory for the Reduce tasks, iii) HDFS block size equal to 128 MB, and iv) workstations equipped with the Samsung SSDs. Table 3 summarizes all the possible execution plans.

Regarding other important settings, the value of the `slowstart` parameter in the sixth row of Table 2 was set to prevent early shuffling and dictates that Reducers will be launched only after all Map tasks have been completed[b]. Furthermore,

[b]The Reduce phase includes three stages: shuffle, sort, and reduce; the former may be executed in

Table 3. Cluster setups

Setup	Map Memory	Reduce Memory	Block Size	Disk
A-128-OPT	1.5 GB	3 GB	128 MB	Optane SSD
A-128-SSD	1.5 GB	3 GB	128 MB	Samsung SSD
A-128-HDD	1.5 GB	3 GB	128 MB	WD Blue HDD
A-512-OPT	1.5 GB	3 GB	512 MB	Optane SSD
A-512-SSD	1.5 GB	3 GB	512 MB	Samsung SSD
A-512-HDD	1.5 GB	3 GB	512 MB	WD Blue HDD
B-128-OPT	3 GB	6 GB	128 MB	Optane SSD
B-128-SSD	3 GB	6 GB	128 MB	Samsung SSD
B-128-HDD	3 GB	6 GB	128 MB	WD Blue HDD
B-512-OPT	3 GB	6 GB	512 MB	Optane SSD
B-512-SSD	3 GB	6 GB	512 MB	Samsung SSD
B-512-HDD	3 GB	6 GB	512 MB	WD Blue HDD

we prevented speculative execution, since any backup Map or Reduce tasks would distort our comparative results[c]. In all cases, the replication factor of the framework was equal to 1 (`dfs.replication=1`), that is, the replication was essentially disabled.

Given the mandatory presence of one AM in the cluster, the aforementioned configuration will allow the parallel execution of $(4N-1)$ Map and $(2N-1)$ Reduce tasks for Naive Bayes and Linear Regression, where N represents the total number of nodes in the cluster. On the other hand, during the execution of k-Means the maximum number of the simultaneous Map and Reduce tasks was set to $(2N-1)$ and $(N-1)$ respectively. For instance, in case we choose to deploy Naive Bayes across a cluster which consists of 5 nodes ($N=5$), YARN will allow the concurrent creation of 19 Map or 9 Reduce tasks plus the AM across the nodes of the cluster.

It is clear that between these two scenarios, Setup A favors parallelization. Consequently, we anticipate that a job will be executed faster under this scenario.

4.2. *Methodology*

In the following experiments we evaluate the performance benefits which derive from the usage of SSDs on MapReduce clusters, compared to traditional magnetic hard drives. In contrast to other similar works, we examine this enhancement in combination with other characteristics such as the degree of parallelization (namely, the number of processing cores which can be launched simultaneously), the applied memory limits, and the selected HDFS block size.

To achieve this goal, each task was launched on clusters of 5 different sizes, namely, by employing different number of nodes each time (1–5 nodes). The ex-

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 schedule the data shuffling phase to also start after the Map phase.

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

periments were repeated for all three storage units of Table 1, under two different settings for the HDFS block size parameter, i.e., 128 MB and 512 MB. The job execution times were retrieved from the corresponding log files; in total, a set of 30 measurements were recorded for each examined algorithm.

Before the execution of each experiment, Hadoop and YARN were stopped and the new cluster parameters were applied (that is, the slave nodes, the employed storage unit, the memory limits for the Map/Reduce tasks, and the HDFS 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.

5. Experiments

This Section includes the presentation of the execution times of various parallel data engineering and machine learning methods on the aforementioned cluster. More specifically, the performance of several dataset preprocessing algorithms is described Subsection 5.1. In the sequel, the running times of the k-Means, Naive Bayes, and Linear Regression algorithms are presented in Subsections 5.2, 5.3, and 5.4 respectively.

5.1. *Dataset Preprocessing Methods*

When a dataset is going to be used in combination with a machine learning algorithm, it usually has to be converted to a form which is suitable for parallel processing. For instance, the typical statistical classification models operate on datasets which are expressed by sets of feature vectors, that is, expressions which combine the features of an observation with their respective values. The process of converting a dataset from its original form to a set of such feature vectors is broadly known in the literature as *vectorization*.

In this Subsection we present performance measurements for various parallel data preprocessing methods which were applied to the latest publicly available dump of the English version of Wikipedia[d]. This collection consists of a single XML file which accommodates roughly 5.8 million articles and occupies approximately 71.3 GB in uncompressed form. Notice that this dataset is different than the one utilized in;[17] the size of corresponding uncompressed XML file of that dataset was about 64 GB, or 11.5% smaller than the one we used here. Therefore, although a qualitative

[d]The Wikipedia dump was downloaded from https://dumps.wikimedia.org/enwiki/latest/ and its date was 20/03/2019.

comparison of the results is possible, the absolute execution times between these two experiments cannot be compared directly.

The goal of this procedure is to prepare the Wikipedia dataset with the aim of using it on two kinds of problems:

- *Clustering:* this process requires the grouping of similar articles into an arbitrary number of 10 clusters by using the k-Means algorithm.
- *Classification:* in this case the objective is the retrieval and classification of the 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.

The dataset preprocessing task was achieved by utilizing Apache Mahout[e], a popular linear algebra distributed framework. Mahout implements a wide variety of parallel machine learning and data engineering algorithms for MapReduce and Spark. More specifically, we executed the `seqwiki` and `seq2sparse` programs of this library; the first one splits the aforementioned XML file into a set of *Sequence files*, whereas the second performs a series of transformations to convert the Sequence files into a set of normalized vectors. In the following presentation we describe the details of each method and we compare their running times on the cluster with the aforementioned characteristics.

5.1.1. *Sequence Files Creation*

The single huge XML file which accommodates the Wikipedia articles cannot be processed in parallel by multiple nodes. In contrast, it is required that we split it into a series of smaller files and distribute these files to the nodes of the cluster through HDFS. The `seqwiki` program of Mahout achieves this goal by parsing each document within the XML file. Then, the text of the extracted articles are recorded to a Sequence file in a <Text,Text> format. The Sequence files are then written to HDFS which transparently transfers them to the nodes for further processing.

The execution times of this procedure are depicted in the two diagrams of Fig. 3. The left and right diagrams concern variable-sized clusters with Setup A and B respectively. Notice that the 58 GB of the Optane SSDs were not adequate to accommodate the 71.3 GB of the Wikipedia dump. Therefore, we were not able to run any algorithm on single-node clusters with this type of storage medium. This in turn means that the curves which represent the Optane SSDs begin from the clusters which contain 2 machines.

According to the previous discussion, we present measurements for two settings of the HDFS block size for each storage type, namely 128 MB and 512 MB. The results indicate that the selected block size indeed affects performance; each storage unit with a block size of 512 MB was consistently faster than its corresponding unit

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

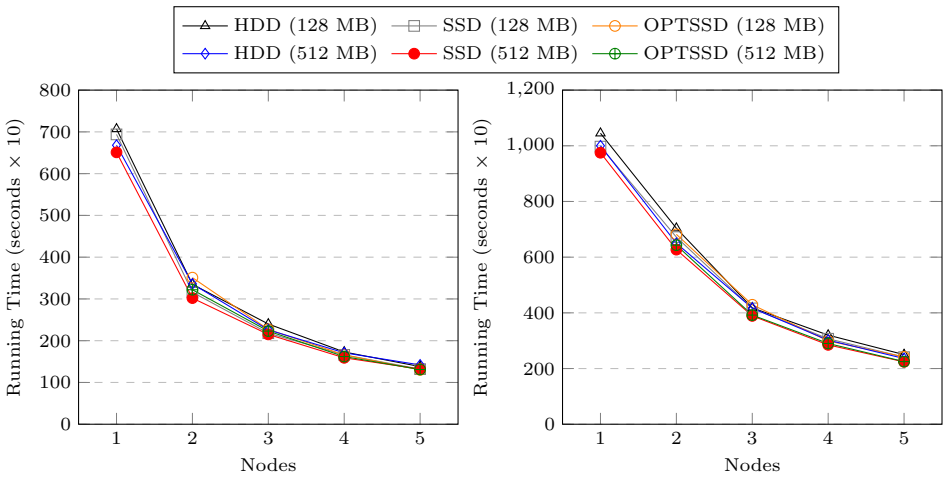

Fig. 3. Running times for the process of Sequence Files creation from the Wikipedia XML dump. The left diagram concerns a variable-sized cluster with Setup A, whereas the right one a cluster with Setup B. Each curve represents a different storage unit with a different HDFS block size.

with a block size of 128 MB for all cluster sizes and for both setups. However, the performance gap gradually decays as the degree of parallelization increases.

More specifically, for a cluster which consists of only two machines and it is configured according to the Setup A scenario, A-512-SSD outperformed A-128-SSD by about 4.5%, whereas A-512-OPT was faster than A-128-OPT by 9%. However, in larger clusters (i.e. with 5 nodes), this gap decreases to under 1% in both cases. Similar conclusions derive for clusters with Setup B; B-512-SSD and B-512-OPT improved the execution times by over 7% and 6% compared to B-128-SSD and B-128-OPT respectively. Nonetheless, these gains gradually decay as the cluster grows, and ultimately become infinitesimal for clusters comprised of 5 nodes.

For equally sized blocks (either 128 MB, or 512 MB), SSDs (both Samsung and Optane) 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.

As expected, the execution times are considerably lower for Setup A (left diagram of Fig. 3). Since the Map and Reduce tasks are programmed to consume less memory in this scenario, YARN is able to launch more containers, which in turn leads to improved performance. The gains range from about 30% for small degrees of parallelization, to over 70% for larger cluster sizes.

5.1.2. *Vectorization & tf-idf Vectors Pruning*

As stated earlier, vectorization is the process of converting a dataset into a set of feature vectors. Under this form, the records are expressed by vectors whose compo-

nents are pairs of: i) an integer identifier (featureID) which represents a feature of the record, and ii) a numerical value which denotes the value of the feature for this particular record. This process is particularly important in the area of data mining, since the majority of the supervised learning algorithms (including the regression and classification methods) accept such vectorized data on their input.

The relevant literature contains a considerable number of state-of-the-art vectorization approaches. One of the most popular among them is *tf-idf*, a method which originates from the research area of Information Retrieval. More specifically, tf-idf vectorization is broadly utilized to transform textual datasets (such as the Wikipedia dump of our case) into feature vectors. Mahout provides a special program, called `seq2sparse`, which vectorizes a dataset according to tf-idf by sequentially performing the following steps:

- *Tokenization:* This procedure parses the Sequence files of the previous phase and extracts the unique words and/or the n-grams out of each document. In the sequel, the extracted words and n-grams are stored back to HDFS accompanied by their in-document frequency values.
- *WordCount:* This task has been used extensively as an introductory example to MapReduce; it merely counts and stores the number of the occurrences of each word in the entire document collection.
- *Vectorization:* In this stage, the main operations are performed for converting the Wikipedia articles into feature vectors. More specifically, during the Map phase each word and/or n-gram in the collection is initially assigned a unique integer identifier. Then, a sequence of (document,feature ID) pairs is created and passed to the Reducers. The pairs are subsequently grouped by key and finally, they are merged together to output the desired (document, list of feature IDs) vectors.
- *Dimensionality Reduction:* The vectors of the previous procedure may consist of many millions of features, of which some may be of small value for the effectiveness of a supervised learning algorithm. More importantly, such long vectors often require huge amounts of memory to be stored and processed. For this reason, a dimensionality reduction (or vector pruning) method must be applied to the dataset.

  In the tf-idf scheme, this process is divided into two phases; during the first phase, the tf-idf value of all features is computed (based on the aforementioned in-document and collection-wide frequencies). In the second phase, a percentage of the features with the lowest tf-idf values are removed from the vectors. In our experiments, we set this percentage to 30%.

Now let us discuss the performance measurements of these vectorization steps. The results are illustrated in Figures 4 and 5. More specifically, the first one depicts the execution times of tokenization (top), WordCount (middle) and vectorization (bottom), on various cluster sizes with the three experimental storage devices. The

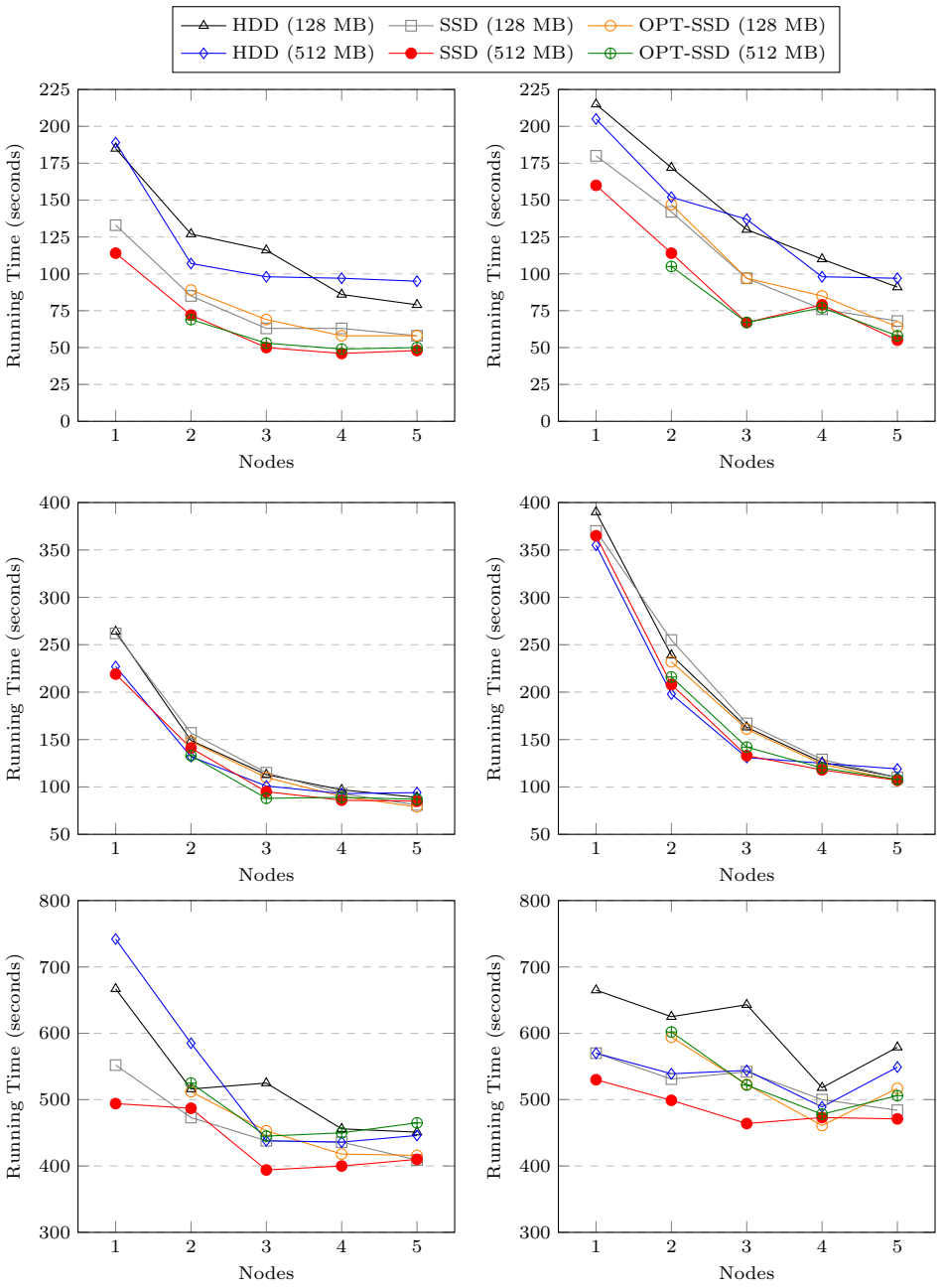

Fig. 4. Performance comparison of various dataset preprocessing methods on a 5-node cluster configured with Setup A (left column) and Setup (right column): i) document tokenization (top), ii) WordCount (middle), iii) dataset vectorization (bottom). Each curve represents a different storage unit with a different HDFS block size.

left handed diagrams concern a cluster configured according to Setup A, whereas the righted handed ones represent clusters configured according to Setup B. On the other hand, Fig. 5 presents the running times of tf-idf scores calculation (top), vectors pruning (middle), and the total times of all 5 dataset vectorization algorithms together (bottom).

The two top diagrams of Fig. 4 depict 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 Samsung and Optane SSDs had a similar performance and both of them were faster than A-128-HDD and A-512-HDD by roughly 37% and 64% respectively. In smaller clusters the benefits are similar, whereas for larger block sizes SSDs perform even better; the A-512-SSD scheme outperformed A-128-HDD and A-512-HDD by approximately 65% and 98% respectively.

Moreover, notice that for Setup A, there is essentially no performance benefit on clusters with more than 3 nodes. However, in Setup B the execution times continue to improve even after this point. This is justified by the fact that in Setup A, each node runs more concurrent tasks. For instance, there are at most 11 Mappers in a 3-node cluster with Setup A (recall that in Setup A, YARN can deploy $(4N - 1)$ concurrent Map tasks). On the other hand, in the case of the Setup B configuration, an approximate number of Mappers can only be deployed on 5-node clusters (i.e., 9). This explains why the execution times between a 3-node cluster with Setup A are almost identical to a 5-node cluster with Setup B.

Regarding the WordCount task (middle diagram of Fig. 4), the results indicate that the usage of more than 4 nodes practically leads to no further acceleration; in both setups the execution times converge after this point. Moreover, the execution is considerably faster in clusters with Setup A. Indicatively, the A-128-OPT, A-128-SSD, and A-128-HDD scenarios on a 5-node cluster are approximately 35%, 36%, and 24% more efficient than B-128-OPT, B-128-SSD, and B-128-HDD respectively.

In the same algorithm, the larger HDFS block sizes exhibit better performance for small degrees of parallelization (1–2 nodes), however, in 5-node clusters the 128 MB setting leads to slightly improved efficiency of 3.5% for all storage units. 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 general, in both setups the Optane SSDs performs equally to the Samsung SSDs on 5-node clusters; in smaller clusters and for both setups, the Optane SSDs are slightly faster by a margin which ranges between 3% and 10%.

The next method, text vectorization, is an example algorithm which does not benefit significantly from parallelization. The two bottom diagrams of Fig. 4 show that performance is similar on clusters of 3–5 nodes with only a few exceptions for HDDs. However, the selected block size and the type of storage media result in different efficiency measurements as the size of the cluster increases. Therefore, the Samsung SSD with HDFS block size equal to 512 MB was consistently faster

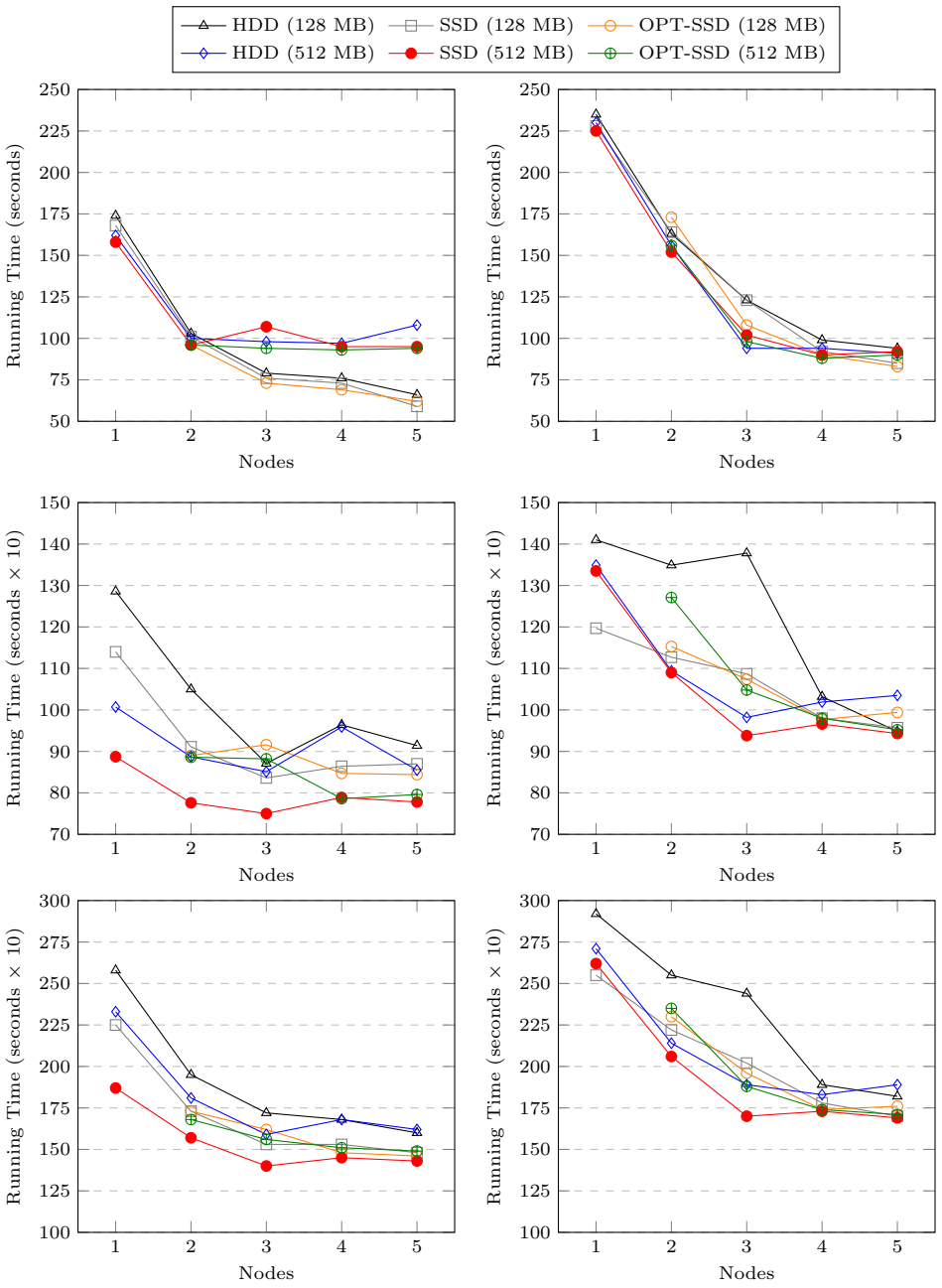

Fig. 5. Performance comparison of various dataset preprocessing methods on a 5-node cluster configured with Setup A (left column) and Setup (right column): iv) tf-idf scores calculation (top), v) tf-idf vectors pruning (middle), and vi) total times of the 5 vectorization procedures (bottom). Each curve represents a different storage unit with a different HDFS block size.

than the other storage units independently of the block size and the cluster setup. More specifically, on a 5-node cluster A-512-SSD was faster than A-128-HDD, A-512-HDD, A-128-OPT and A-512-OPT by 10%, 8.7%, 1.5%, and 13.5% respectively. A-128-SSD performed almost equally well, whereas similar improvements were observed on clusters under Setup B.

We continue our presentation with the methods depicted in Fig. 5. The two top diagrams illustrate the execution times of the process which calculates the tf-idf scores of words of the Wikipedia articles. The analysis of these two diagrams leads to three significant observations: The first one is that the execution times do not improve significantly on clusters which comprise of 3 or more nodes. This means that the degree of parallelization is not an important factor for the parallel computation of the tf-idf scores. The second observation is that the times in the left diagram are considerably lower than those in the right diagram. Consequently, the most important factor which affects performance is the amount of memory that is provided to the Map and Reduce tasks. And finally, it seems that the selected HDFS block size is more important than the type of the storage medium itself. The left diagram reveals that by keeping the HDFS block size equal to the default setting (i.e. 128 MB), we achieve execution times which are almost half compared to those that we get under the 512 MB setting.

Similarly to the vectors creation process, the performance of the tf-idf vectors pruning algorithm does not improve significantly as the cluster size increases. In fact, in some cases (such as A-512-SSD, A-512-HDD, B-512-SSD, etc), the execution times get larger if we include more than three nodes in the cluster. Both SSD devices were faster than HDDs, with A-512-SSD being the fastest scenario among all. For a 5-node cluster configured with Setup A, A-512-SSD outperformed A-128-HDD and A-512-HDD by 24% and 10% respectively, whereas the Optane SSD was only 2% slower. Regarding the same cluster under Setup B, the performance gap was smaller, however the SSDs were still faster.

The two last diagrams illustrate the accumulated execution times of the five previous tasks of Figures 4 and 5. In other words, they provide the complete picture of the dataset preprocessing procedure. The conclusions are:

- The size of the cluster is important only for small numbers of nodes. Therefore, the performance does not improve, or improves slightly for clusters which consist of 3 or more nodes.
- Setup A is faster than Setup B. Hence, it is important to provide the correct amount of memory to the Map and Reduce tasks, because a correct configuration will allow the deployment of additional processing containers. This will increase the degree of parallelization with the same number of nodes.
- Samsung SSDs outperformed the other two types of storage, however Optane SSDs also perform well.
- The HDFS block size is important for small cluster sizes and its effect decays as the number of nodes increases.

Finally, among the experiments which were presented in this Subsection, 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 reports such results.

5.2. *k-Means Clustering*

The vectorization process of the previous Subsection has transformed the dataset into a set of normalized feature vectors. Each Wikipedia article is now represented by a vector which has been pruned by using the tf-idf scoring technique; according to it, only the 30% of the highest scoring components have been preserved in the vector, whereas the rest 70% have been removed. We shall now use this vectorized dataset to examine the performance of k-Means on our experimental cluster with the three aforementioned storage units.

k-Means is an unsupervised clustering algorithm, that is, its goal is to create clusters of similar entities. It initially accepts a number k which denotes the number of the clusters to be created, and then it randomly chooses k points $\mu_1, \mu_2, \ldots, \mu_k$ in the vector space that will serve as the initial centers of the clusters (which are called centroids). In the sequel, it assigns each feature vector x_i into the closest cluster c_i by computing the value of a distance function such as the Euclidean distance:

$$c_i = \arg\min(x_i - \mu_j)^2 \tag{1}$$

After all data points have been assigned to a cluster, the algorithm adjusts the centroids of each cluster by averaging the values of the current data points of the cluster. This procedure is repeated until either a maximum number of iterations have been performed, or convergence (i.e. the centroids do not change any more) has been achieved. It is proved that the algorithm converges after a finite number of iterations.

Several tweaks concerning the distance measure, the initial selection of the centroids and the computation of the new average centers have been explored, as well as the estimation of the number of clusters k. Yet the main principle always remains the same. In our parallel execution, we set $k = 20$, and the maximum number of iterations to be performed equal to 10. The distance measure which was selected is the well-known Euclidean distance.

Fig. 6 depicts the running times of the iterative centroids computation (left diagram) and the clustering of the vectorized articles after the centroids have been stabilized (right diagram). Recall that with the selected parameters (10 iterations, $k = 20$), the memory limits dictated by Setup A do not suffice for the execution k-Means. Consequently, the presented results concern Setup B only.

The left diagram of Fig 6 reveals a remarkable outcome; the selected HDFS block size plays a huge role in the overall performance of the iterative calculation of the centroids of the clusters, independently of the employed storage device. More

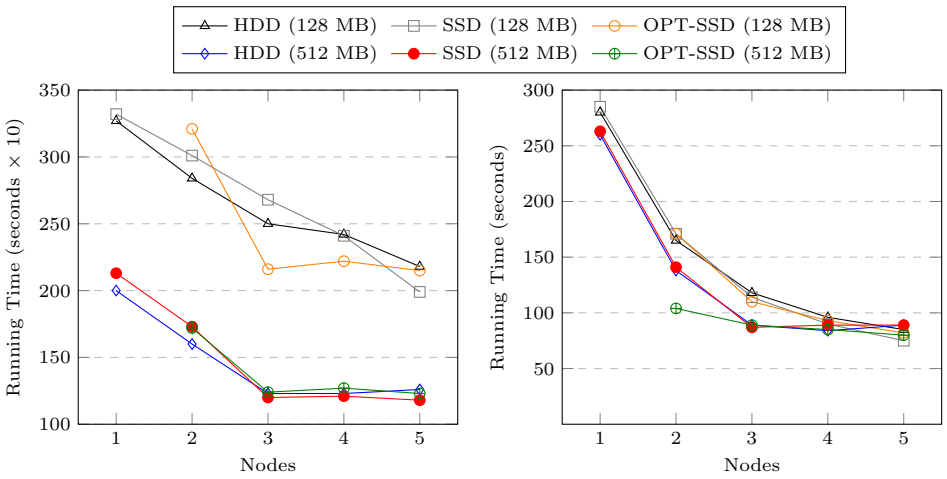

Fig. 6. k-Means performance on a 5-node cluster with the vectorized Wikipedia dataset with $k = 20$: i) computation of the centroids (10 iterations, left), and ii) clustering (right). Each curve represents a different storage unit with a different HDFS block size.

specifically, the 512 MB setting in some cases doubles the execution speed. Especially in clusters which consist of 3 and 4 nodes, all storage devices operate two times faster compared to the 128 MB setting for the HDFS block size. In larger clusters with 5 nodes, B-512-HDD, B-512-SSD, and B-512-OPT outperformed B-128-HDD, B-128-SSD, and B-128-OPT by 73%, 69%, and 75% respectively.

Samsung SSD was the fastest storage device in this case (5-node cluster). B-512-SSD was faster than B-512-OPT and B-512-HDD by rougly 8% and 9.5% respectively, and the results were similar for HDFS block size equal to 128 MB. However, for small sizes of the cluster, it was the traditional Hard Drive (B-512-HDD) which achieved the best performance. Similarly to some of the previous experiments, the increase of the nodes of the cluster beyond 3, practically leads to no, or infinitesimal gains in the speed of execution.

The right diagram of Fig. 6 illustrates the running times of the clustering process after determination of the clusters and the convergence of the previous algorithm. The number of nodes is the most important factor in this experiment. The 512 MB setting for the HDFS block size delivers the best results for clusters comprised of 1–4 nodes, however, for larger clusters the value of 128 MB is the winning setting. Samsung SSDs is again the best performing storage unit for block size equal to 128 MB. More specifically, in the 5-node cluster B-128-SSD outperformed B-128-HDD and B-128-OPT by approximately 13.3% and 3.6% and respectively. On the other hand, Optane SSDs dominate for the 512 MB setting; hence, for the same cluster size B-512-OPT was faster than B-512-HDD and B-512-SSD by a margin which exceeded 11%. A final remarkable observation in these experiment was the poor scaling of k-Means clustering when the HDFS block size is set equal to 512 MB; this is particularly true for the Optane SSD.

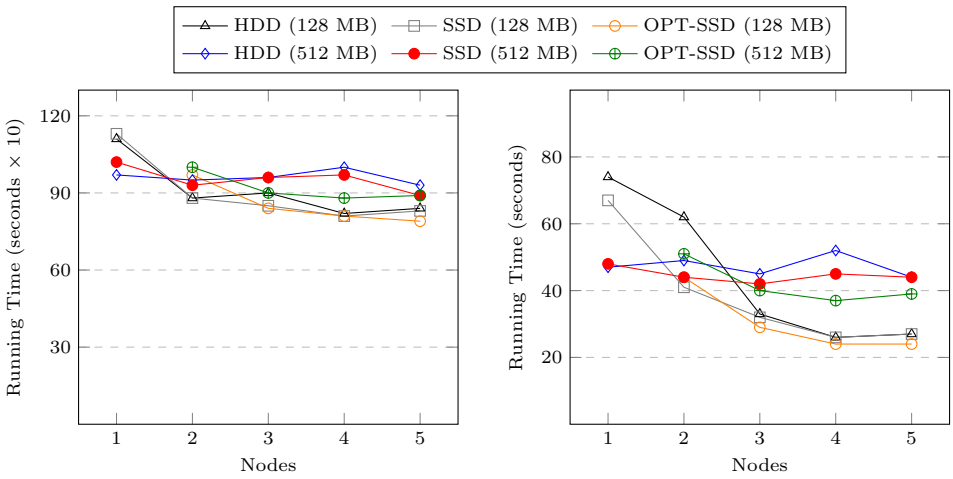

Fig. 7. Naive Bayes performance on a 5-node cluster with the vectorized Wikipedia dataset: i) Model training (left), and ii) testing (right). Each curve represents a different storage unit with a different HDFS block size.

5.3. *Naive Bayes Classifier*

Naive Bayes is one of the most popular supervised learning classification algorithms. Its popularity is mainly due to its simplicity and its capability on handling large-scale datasets. It derives from the Bayes theorem, according to which the likelihood of a feature x given a class y is given by the following formula:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} \tag{2}$$

where $P(y)$ and $P(x)$ represent the prior probabilities of a class y and a feature x respectively. The algorithm assumes independence among the predictors (i.e. features). During the training phase, the classifier constructs a special frequency table according to the observations of the training set. More specifically, this table stores the frequency of each (feature x, class y) pair and it is utilized to compute the three aforementioned probabilities. Then, it is utilized during the test phase to classify an unlabeled observation.

The left diagram of Fig. 7 depicts the running times of the Naive Bayes model training process. The input of this task is the vectorized data which was generated by the preprocessing methods of Section 5.1. Additionally, the cluster was configured according to Setup A, that is, the memory limits for the Map and Reduce tasks were set equal to 1.5 GB and 3 GB respectively.

It is apparent that the size of the cluster plays a very small role in the efficiency of the model's training phase. Consequently, the running times do not improve significantly for clusters which consist of more than 3 nodes. Moreover, the default HDFS block size seems to achieve better scaling compared to the 512 MB setting.

Hence, in single-node clusters the 512 MB setting exhibits lower running times, whereas the situation reverses as more nodes enter the cluster. The Optane SSD was the most efficient storage medium under the default setting for HDFS block size, although the performance gap was small. Indicatively, in a 5-node cluster A-128-OPT outperformed A-128-HDD, A-128-SSD and A-512-OPT by 6.3%, 5%, and 12.6% respectively. Notice that on smaller clusters, although the 128 MB setting still yields the lowest running times, the differences in performance become even smaller.

Regarding test phase (right part of Fig. 7), the execution times are primarily affected by the number of nodes in the cluster. Initially (from 2 to 3 nodes) the performance gains are linear, whereas on larger clusters they become sublinear. Similarly to the training phase, the most reasonable choice is to leave the HDFS block size unchanged (i.e. equal to the default value of 128 MB). In some cases (i.e. clusters with 4 and 5 nodes), this selection leads a performance improvement of approximately 62-65% for the two SSDs, and up to 100% for HDD.

The Optane SSD achieved the lowest running times for both block sizes. More specifically, A-128-OPT outperformed both A-128-HDD and A-128-SSD by 12.5% (the WD HDD and the Samsung SSD performed about the same on clusters with 3–5 nodes). Similarly, A-512-OPT outperformed both A-512-HDD and A-512-SSD by a margin of 12.8%.

5.4. *Linear Regression*

Linear Regression is a popular statistical method for supervised classification. Given a sample x with features $x_1, x_2, ..., x_n$, the algorithm assigns a label y to x by plotting a regression line which is defined by the following equation:

$$y = \theta_0 + \sum_{i=1}^{n} \theta_i x_i \tag{3}$$

where $\theta_0, \theta_1, ..., \theta_n$ are the features coefficients which we need to learn. In the current literature we encounter multiple approaches for their computation. The commonest technique is based on the iterative least squares method, which is used to minimize the overall squared test error:

$$\min_{\theta} \sum_{i=1}^{n} (y - y')^2 \tag{4}$$

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

In this case we decided to employ a dataset with different characteristics than the Wikipedia dump which has been used in all previous experiments. In particular, we desired a large-scale repository of observations with low dimensionality. A dataset

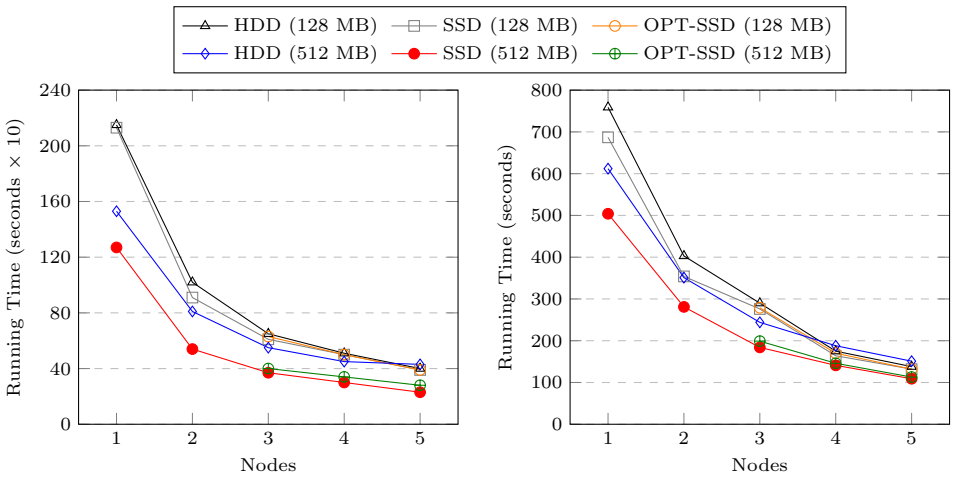

Fig. 8. Linear Regression performance on a 5-node cluster with the vectorized HIGGS dataset: i) Model training (left), and ii) testing (right). Each curve represents a different storage unit with a different HDFS block size.

which satisfied some of these requirements was the HIGGS dataset[f], a collection of about 11 million Monte Carlo simulations of high-energy particle collisions. Each sample within HIGGS consists of a binary class label, accompanied by a vector of 28 features. These features represent several kinematic and other properties of the particles which participate in the collisions.

One requirement which was not satisfied by the HIGGS dataset was its relatively small size. More specifically, we considered that the 8 GB of the dataset were inadequate to conduct a reliable analysis of large-scale data processing. For this reason, we created a new collection by generating an arbitrary number of copies (between 10 and 20) of the records of the HIGGS dataset. This process resulted in a 127 GB synthetic dataset comprised of more than 236 million samples. From this dataset we utilized the 80% of the records to train the classification model, whereas the rest 20% was left for testing.

In this experiment we did not employ the Mahout library; instead, we used our own Java implementation of Linear Regression. Fig. 8 illustrates the execution times of the algorithm for the training (left diagram) and test phases (right diagram). Regarding the former, the measured times concern the accumulated durations of 10 iterations for the calculation of the θ values by employing the Batch Gradient Descent algorithm.

The SSDs outperformed the HDDs by a substantial margin for block sizes equal to 512 MB and for all cluster sizes. In particular, in a 5-node cluster A-512-SSD completed the model training task after 231 seconds, whereas A-512-OPT and A-512-HDD were 22% and 87% slower respectively. Regarding smaller cluster sizes, the

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

performance gap becomes is also large; for instance, in a 3-node cluster the Samsung SSD and the Optane SSD outperformed HDD by 49% and 37% respectively. In contrast, in case the HDFS block size is set equal to 128 MB, the three storage devices perform almost equally.

Regarding the test phase, the performance measurements were very similar to those of the training phase. In short, the devices perform considerably better be setting the HDFS block size equal to 512 MB. In this case, both SSD devices outperformed HDD by 30–35%. The 6 curves in right diagram of Fig. 8 indicate that all storage media exhibit a satisfactory scaling; hence the classification times improve significantly as the size of the cluster increases.

6. Discussion & Conclusions

In this paper we conducted an extensive research of the effects of the modern storage devices in the efficiency of multiple parallel machine learning and data engineering algorithms on MapReduce clusters. More specifically, we installed a laboratory cluster comprised of 5 nodes and we applied various configurations with the aim of measuring the performance of these algorithms with three storage units: i) a standard spinning disk drive with magnetic plates, ii) a Samsung SSD, and iii) an Optane SSD, a modern device which implements the latest 3D XPoint technology.

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) the amount of memory provided to the Map and Reduce tasks (which partially affects the degree of parallelization since lower memory limits allow the creation of additional YARN containers), iii) the HDFS block size, iv) different datasets, and v) different types of MapReduce jobs including CPU, network, and disk intensive tasks. These tasks included: i) data partitioning and distribution procedures, ii) a series of data preprocessing methods, such as vectorization and tf-idf vectors pruning, iii) one clustering algorithm (k-Means), and iv) two supervised classifiers, that is, Naive Bayes and Linear Regression.

In the vast majority of cases, the two solid state drives outperformed the hard drive. However, in a significant number of tasks the performance gains were small. The results indicate that the magnitude of the benefits depends on the nature of the algorithm itself. Moreover, the cluster configuration played a significant role in the efficiency of the algorithms. The most important observation is that in different algorithms, there is a different set of parameters which affect performance. For instance, in some cases it is the size of the cluster which is the most significant factor, whereas in other cases it is the storage medium or the HDFS block size.

The diversity of the experimental results indicates that there is no golden configuration; this renders the modeling of the influence of these parameters an extremely difficult problem. To clarify the situation, we created Table 4, which attempts to capture the essence of our work. In this Table we summarize the results of the experimental analysis of the 13 examined tasks. The tasks have been grouped into 4

Table 4. Factors which affect the performance of various machine learning algorithms

Algorithm	**Type**	**Important Factors**	**Winning Scenario**	
			$N = 3$	$N = 5$
1. Sequence files creation	Data partitioning & distribution	i) Cluster size, ii) Map/Reduce RAM, iii) HDFS block size	A-512-SSD	A-128-SSD A-512-SSD
2. Document tokenization	Data preprocessing	i) Storage medium, ii) Map/Reduce RAM, iii) HDFS block size	A-512-SSD	A-128-SSD
3. Word count	Data preprocessing	i) Map/Reduce RAM, ii) Cluster size, iii) Storage medium	A-512-OPT	A-128-OPT
4. Text vectorization	Data preprocessing	i) Storage medium, ii) HDFS block size	A-512-SSD	A-128-SSD
5. tf-idf scores computation	Data preprocessing	i) Map/Reduce RAM, ii) HDFS block size	A-512-OPT	A-128-SSD
6. tf-idf vectors pruning	Data preprocessing	i) Storage medium, ii) Map/Reduce RAM, iii) Cluster size	A-512-SSD	A-128-SSD
7. total preprocessing procedure (tasks 2–6)	Data preprocessing	i) Map/Reduce RAM, ii) Storage medium, iii) Cluster size iv) HDFS block size	A-512-SSD	A-128-SSD
8. k-means - iterative centroids computation	Clustering	i) HDFS block size, ii) Storage medium, iii) Cluster size	B-512-SSD	B-512-SSD
9. k-means - clustering	Clustering	i) Cluster size, ii) Storage medium	B-512-SSD	B-128-SSD
10. Naive Bayes - training phase	Classification	i) Storage medium, ii) Cluster size, iii) HDFS block size	A-128-OPT	A-128-OPT
11. Naive Bayes - testing phase	Classification	i) Cluster size, ii) Storage medium, iii) HDFS block size	A-128-OPT	A-128-OPT
12. Linear Regression - training phase	Classification	i) Cluster size, ii) Storage medium, iii) HDFS block size	A-512-SSD	A-512-SSD
13. Linear Regression - testing phase	Classification	i) Cluster size, ii) Storage medium, iii) HDFS block size	A-512-SSD	A-512-SSD

categories according to their nature: one is about data partitioning and distribution, 5+1 concern data preprocessing procedures, the next two cover the clustering problem, wheres the last 4 is about supervised classification. Furthermore, in the first two columns of Table 4 we report the examined algorithm and its type. The third column contains lists with the parameters which most affected performance. These lists are ranked; their first element shows the most important parameter, followed by the less significant ones. The two last columns indicate the winning (i.e. best performing) scenario on a cluster comprised of 3 and 5 nodes respectively. Recall

that these scenarios are described in Table 3.

Now let us use Table 4 to derive some conclusions. The two last columns reveal that in all tasks, SSDs were the best performing storage devices. In a 3-node cluster, the Samsung SSD outperformed the other two devices in 9 out of 13 tasks, whereas in the other 4, the Optane SSD won the test. Similarly, in the larger 5-node cluster the number of tasks in which the Samsung and the Optane SSD exhibited the best performance were 10 and 3 respectively. This is mainly due to the higher read and write rate of the former.

An important question is *how large is the performance gap between SSDs and HDDs?* The answer to this question can be given by the third column of Table 4 which tells us that in 3 out of 13 tasks (Sequence files creation, WordCount, tf-idf scores computation) the storage medium does not affect performance. This means that although SSDs were faster than HDDs by an infinitesimal margin. In contrast, in 4 tasks (tokenization, text vectorization, vectors pruning and Naive Bayes training phase), the storage medium had a large impact in the execution times. In the other tasks the storage medium had a moderate impact in performance. Consequently, the benefits of utilizing SSDs on MapReduce clusters are diverse and depend on the nature of the executed task.

Another conclusion is that all configuration parameters were the most important factors which affected performance in at least one case. Even the HDFS block size had a huge impact in the iterative process which calculated the centroids of the clusters in k-Means. From the last two columns we can also conclude that in case a cluster consists of a small number of nodes, then we may increase performance by setting the HDFS block size equal to 512 MB. However, these gains are either nullified or reversed as the size of the cluster increases (i.e. more machines are appended to the cluster). In these occasions, the default setting of 128 MB yields the best results, as indicated by the last column of Table 4.

Finally, not all algorithms benefit from parallelization. There are cases where the addition of more machines to the cluster may hurt efficiency, even a little. Examples of such tasks are document tokenization, text vectorization and k-Means clustering.

References

1. M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker and I. Stoica, Spark: Cluster Computing with Working Sets, in *In Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing*2010, pp. 1–7.
2. P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi and K. Tzoumas, Apache Flink: Stream and Batch Processing in a Single Engine, *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering* **38** (2015) 28–38.
3. 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.
4. 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.

5. 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 Data, *Genome Research* **20**(9) (2010) 1297–1303.
6. 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.
7. L. Akritidis and P. Bozanis, Computing Scientometrics in Large-Scale Academic Search Engines with MapReduce, in *Proceedings of the 13th International Conference on Web Information System Engineering, Lecture Notes in Computer Science (vol. 7651)*2012, pp. 609–623.
8. 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.
9. K. Zhang and X.-W. Chen, Large-Scale Deep Belief Nets with MapReduce, *IEEE Access* **2** (2014) 395–403.
10. W. Chen, Z. Wang and J. Zhou, Large-scale L-BFGS using MapReduce, in *Advances in Neural Information Processing Systems*2014, pp. 1332–1340.
11. 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.
12. 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.
13. 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.
14. 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* **64**(3) (2013) 1055–1074.
15. 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.
16. E. Gal and S. Toledo, Algorithms and data structures for flash memories, *ACM Computing Surveys (CSUR)* **37**(2) (2005) 138–163.
17. L. Akritidis, A. Fevgas, P. Tsompanopoulou and P. Bozanis, Effective Product Categorization with Importance Scores and Morphological Analysis of the Titles, in *Proceedings of the 30th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*2018, pp. 213–220.
18. 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* **6**(1) (2013) 201–210.
19. 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.
20. 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 Com-*

*puting, Networking, Storage and Analysis (SC)*2012.
21. 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.
22. 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.
23. 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.
24. 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.
25. 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* **2**(5) (2014).
26. 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.
27. S. Moon, J. Lee, X. Sun and Y.-s. Kee, Optimizing the Hadoop MapReduce Framework with High-Performance Storage Devices, *The Journal of Supercomputing* **71**(9) (2015) 3525–3548.
28. S. Ahn and S. Park, An Analytical Approach to Evaluation of SSD Effects under MapReduce Workloads, *Journal of Semiconductor Technology and Science* **15**(5) (2015) 511–518.
29. 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.
30. 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* **11**(2) (2014) 1–26.
31. J. Dean and S. Ghemawat, MapReduce: Simplified Data Processing on Large Clusters, *Communications of the ACM* **51**(1) (2008) 107–113.
32. R. Micheloni, *3D Flash memories* (Springer, 2016).
33. F. T. Hady, A. Foong, B. Veal and D. Williams, Platform Storage Performance With 3D XPoint Technology, *Proceedings of the IEEE* **105**(9) (2017) 1822–1833.
34. I. Koltsidas and V. Hsu, IBM Storage and NVM express Revolution, tech. rep. (2017).