
Scalable Computing: Practice and Experience, ISSN 1895-1767, http://www.scpe.org
© 2021 SCPE. Volume 22, Issues 4, pp. 401–412, DOI 10.12694/scpe.v22i4.1945

PERFORMANCE-EFFICIENT RECOMMENDATION AND PREDICTION SERVICE FOR BIG DATA FRAMEWORKS FOCUSING ON DATA COMPRESSION AND IN-MEMORY DATA STORAGE INDICATORS

HRACHYA ASTSATRYAN* ARTHUR LALAYAN† ARAM KOCHARYAN‡ AND DANIEL HAGIMONT§

Abstract. The MapReduce framework manages Big Data sets by splitting the large datasets into a set of distributed blocks and processes them in parallel. Data compression and in-memory file systems are widely used methods in Big Data processing to reduce resource-intensive I/O operations and improve I/O rate correspondingly. The article presents a performance-efficient modular and configurable decision-making robust service relying on data compression and in-memory data storage indicators. The service consists of Recommendation and Prediction modules, predicts the execution time of a given job based on metrics, and recommends the best configuration parameters to improve Hadoop and Spark frameworks' performance. Several CPU and data-intensive applications and micro-benchmarks have been evaluated to improve the performance, including Log Analyzer, WordCount, and K-Means.

Key words: Hadoop; Spark; MapReduce; data compression; in-memory file system

AMS subject classifications. 68T09

1. Introduction. Big Data has become a critical research area due to the massive data generation [1]. Managing such data is a resource-intensive operation demanding parallel processing. The rapid development of Big Data frameworks addresses the distribution, communication, and processing of a vast number of data. For instance, Hadoop and Spark popular frameworks [2] may handle massive amounts of data relying on the MapReduce paradigm [3] to process and generate extensive data sets [4]. The data sets are stored across distributed clusters to run a distributed processing scheme in each cluster. The Hadoop Distributed File System (HDFS) [5] is a storage layer and a back-end file system for the MapReduce model, providing a scalable, fault-tolerant, and portable architecture for MapReduce jobs.

The MapReduce framework splits large data sets into a set of distributed blocks, executes Map tasks in parallel on these blocks, and finally reduces tasks for the aggregation of results to process the data. HDFS performance mainly depends on data access and data movement, where memory data access bandwidth is higher than the disk access bandwidth. Therefore, an influential organization of an intensive I/O application to disk is a challenge. Data compression is a critical approach to reduce I/O operations, whereas the in-memory file systems using HDFS lazy persist strategy [6] improves the I/O rate. From one side, data compression reduces the input data size, HDFS storage usage, and network traffic. Map and Reduce tasks are executed on different machines in a parallel fashion in the MapReduce programming model. The final result is achieved only after the completion of the Reduce tasks. The compression at various stages of MapReduce jobs or minimization of mapper output saves the disk space and minimizes spilling. From another side, HDFS Data nodes flush in-memory data to disk asynchronously to remove checksum computations and expensive disk I/O from the performance-sensitive I/O path. HDFS offers best-effort persistence guarantees for Lazy Persist Writes, where data are written in a faster off-heap memory located in a RAM than writing on a hard drive. Therefore, it may decrease the waiting time for the written data to the disk and improve I/O rate.

*Institute for Informatics and Automation Problems National Academy of Sciences of Armenia 1, Paruyr Sevak str. 0014 Yerevan, Armenia (hrach@sci.am).
†National Polytechnic University of Armenia 105, Teryan str. 0009 Yerevan, Armenia (arthurlalayan97@gmail.com).
‡Université Fédérale Toulouse Midi-Pyrénées Toulouse Cedex 7, 31000 Toulouse, France (ar.kocharyan@gmail.com).
§Université Fédérale Toulouse Midi-Pyrénées Toulouse Cedex 7, 31000 Toulouse, France (daniel.hagimont@irit.fr).

Hadoop and Spark frameworks support different data compression methods to reduce I/O and improve performance. The methods require configuring the frameworks to recognize compression codecs by specifying the codec implementation paths. Then to compress the input data and save it to HDFS. Before processing data, the framework decompresses the data if the input file is compressed in HDFS. Our recent studies show [7, 8, 9] that the average memory usage for selected scientific workflows is 13-17% for Hadoop and 20-40% for Spark jobs, which neglect the full utilization of the RAM of HDFS nodes. Therefore, the usage of RAM-free space may boost the performance of HDFS processing. Our primary approach is to benefit from combining RAM and disk space by setting up HDFS on top of the unused space by implementing virtual RAMDisks in all data nodes. That approach allows saving the input data in RAM before processing it. The article presents a codeless performance-efficient decision-making robust service relying on data compression techniques and in-memory file systems. The service, which consists of Recommendation and Prediction modules, has been evaluated for several MapReduce workflows.

The content of this paper is organized as follows. The related work is presented in Section 2. The Recommendation and Prediction service can be found in Section 3. The evaluation results are given in Section 4. Finally, the conclusion and directives for future research are drawn in Section 5.

2. Related work. The performance efficiency, memory caching techniques, and in-memory computing models are widely used to improve the performance of Hadoop and Spark frameworks. The Intra-Node Combiner approach is proposed by [10, 11] using in-node and in-mapper combiners that rely on memory caching technologies. This approach changes the MapReduce job mapping methods in the Map and Reduce phases using in-node and in-map design patterns to combine the data of a node in the Map phase and then send it to the Reduce phase. The in-memory Redis data cache improves job execution time by 23% over the standard approach. Notably, the in-map combiner reduces execution time by 25%, and the in-node combiner reduces execution time by 20%. Besides, the in-memory cache observed that the average Map completion time is reduced by 14.8%. The combined design pattern improves performance but depends on the codebase memory caching techniques like Redis and doesn't provide a codeless performance efficiency. Therefore, it is necessary to redesign the job in such an approach, read input from the cache, and develop mechanisms for managing the memory cache to get a fully customized environment.

The impact of the usage of high throughput, high bandwidth, and low latency networks over RDMA is studied by to reduce I/O and improve performance focusing on the InfiniBand network [12, 13]. The studies show that the RDMA-based approach enhances the performance of some MapReduce workflows (RandomWriter, PageRank, Sort, TeraSort, TestDFSIO, etc.) by 1.2-1.8 times. It was observed 42%-49% and 46%-50% latency reduction compared to default Hadoop RPC over 10GigE and IPoIB QDR (32 Gbps). Besides, a 10% performance improvement for CloudBurst application and 26% cache improvement of a put operation for HBase. The studies show that RDMA Hadoop reduces I/O to affect the performance optimizations using several compression techniques with an overhead to the uncompressed process. The essential advantage of this approach is the performance-boosting of using network interface cards, which may have high throughput limitations. The network card supports the shuffle phase to exchange intermediate outputs from the Map phase over the network.

Another perspective is a distributed multi-tier caching system on top of HDFS. The experiments on such a multi-tier caching system show that for a 10MB file, the new cache was 56% faster than Redis, and 26- and 60-times fasters for GET and SET operations [14]. Performance and power analysis models estimate the performance per watt for various cloud benchmarks with an error of less than 10% based on comparative studies of performance, the power consumption of multiple clusters with heterogeneous processor configurations (with and without cache memory). Their focus is on performance studies using heterogeneous processor architectures instead of developing or integrating new caching techniques.

RAMCloud storage system is suggested by [15] to integrate the available memory resources in an HDFS cluster to form a cloud storage system. The authors [16] suggest a memory-based Hadoop distributed file system to improve I/O rate because disks have poor performance. In addition to the new memory cache, an in-memory data replacement strategy is proposed to allocate memory space. Such an approach cannot consider compression techniques reducing I/O.

Instead of performance-boosting, the execution time of different scientific workflows in Hadoop or Spark

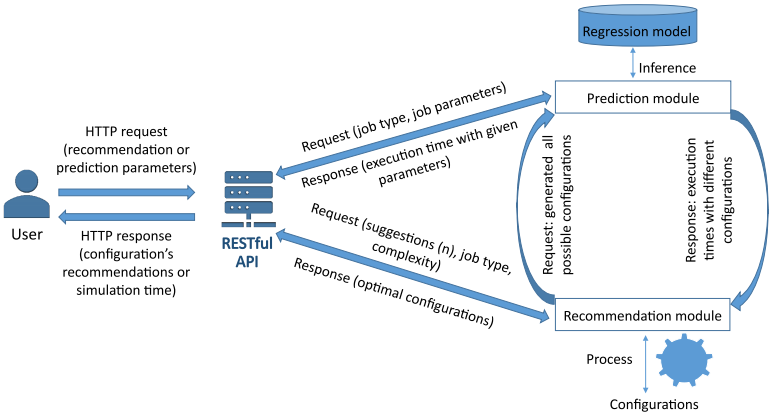

Fig. 3.1: The structure of the performance-efficient Recommendation and Prediction service

is studied by [17, 18]. The authors suggest a weighting system to estimate job execution times in Hadoop using the historical simulation data or machine learning regression models to predict the execution time of various machine learning jobs and SQL queries in Spark. Our studies focus on input data compression and saving in-memory approaches for both Hadoop and Spark frameworks. Besides the Prediction module, a Recommendation module is suggested to advise users to get optimal parameters to improve the performance. The article presents a codeless performance-efficient decision-making system mixing the approaches mentioned above to enhance our previous studies [8].

3. Recommendation and Prediction service for Big Data frameworks. The suggested Recommendation and Prediction service for Big Data frameworks (see fig. 3.1) consists of the following three modules[1]:

- RESTful (Representational State Transfer) - to provide an interface to the MapReduce service [19];
- Recommendation - to recommend a configuration parameters according to the user request;
- Prediction - to analyze and predict suitable configurations satisfying the user requests.

3.1. RESTful module. The RESTful module fully utilizes the architectural principles of REST API to develop a modular and configurable service delivering two different endpoints for the suggested Prediction and Recommendation modules. The Prediction module forecasts the simulation time considering various configuration parameters in regression models. In comparison, the Recommendation module depends on configuration files and the Prediction module to recommend optimal configuration parameters to improve performance. The service modules are abstracted from the system, enabling the implementation of a custom module (using static values, linear or non-linear ML models) and loading the module to the system. The user's requests to the REST API are addressed and transformed via HTTP protocol. The Post method executes the query and fetches the suitable configuration. The JSON (JavaScript Object Notation) lightweight, text-based, language-independent data interchange format empowered the request and response bodies based on simple key/value pairs. The input parameters of the service are the JSON attributes of the request body (see Table 3.1).

The specific metrics are identified per each application or micro-benchmark having its local dataset on the top of the general metrics. The replication and block sizes are specified as static metrics, considering the direct impacts on all the collected metrics. TestDFSIO has an additional option parameter (with values for reading and writing). At the same time, TeraSort consists of three different tasks (TeraGen, TeraSort, and TeraValidate), and K-Means has the number of clusters to find parameters. The input parameters are different

[1]https://github.com/ArmHPC/MapReduce-Optimization-Solutions

Table 3.1: Service input parameters

Attribute name	**Value**
Environment	Hadoop or Spark
Compression method	Uncompressed, gzip, bzip2, lzo, lz4, snappy, zstandard
Data nodes count	number
Complexity	Input data size
Additional	Application specific parameters

Table 3.2: List of metrics

Metric name	**Value**
Environment name	Hadoop or Spark
RAM over HDFS	True/False
Number of nodes	Number
Application	Name
Input Data Size	Number
Input Data Type	Method
Execution time	Value in seconds
CPU	%
Memory	%
I/O Read	Number
I/O Write	Number
Network received	Number
Network sent	Number

for Prediction and Recommendation modules since the Prediction module requires all the described parameters. Still, the Recommendation module requires only job type and complexity (other parameters are optional).

The following types of workflows have been studied:

- data/CPU intensive - WordCount, Log Analyzer;
- ML - K-Means;
- micro-benchmarks - TestDFSIO, TeraSort.

The WordCount is a simple data and CPU-intensive benchmark to count the number of occurrences of each word in a given input set. At the same time, the Log Analyzer is a generic log analyzer benchmark using MapReduce framework [20]. The K-Means focuses on the MapReduce implementation of standard K-means, as clustering algorithms require scalable solutions to manage large datasets [21]. TestDFSIO micro-benchmark is for stressing HDFS with the options of reading and writing, whereas the TeraSort is for generating, sorting, and validating data.

3.2. Prediction module. As the input metrics of each MapReduce application are dissimilar, the Prediction module relies on regression models to predict the job execution time. Different linear and polynomial regression models were trained using key features from the simulation dataset (see Table 3.2).

The regression model requires encoded inputs as there are qualitative variables such as environment or compression method. The qualitative data of the simulation dataset is encoded using the one-hot encoding schema, and the training process is performed on the encoded input data. The REST call encodes the parameters specified by the user based on identical encoders to pass the encoded data to the regression model. The encoders of each MapReduce application are unique as the models, and input parameters are different. As soon as the Prediction module receives the request, the module finds a regression model using the job type and returns the model output for a given input. All the trained models and encoders are accessible through the .pkl format.

The service abstraction easily extends the system for new MapReduce workloads. The trained regression

model, the encoder, and the configuration file are provided per each new application. The configuration file corresponds to the trained regression model and contains information about the trained regression model features [22]. As a result of the training series, the polynomial regression models with degree 3 are provided for the studied applications with the following loss function:

$$L = (\ln y - X\beta)^T(\ln y - X\beta) + \lambda\beta^T\beta \tag{3.1}$$

The β weights are determined during the training phase by minimizing the loss function using the gradient descent method. While X is the dataset consisting of different features and all polynomial combinations with degrees less than or equal to 3, y is execution times with given X features, λ is the regularization parameter, I is an identical matrix. The degree of polynomial and regularization hyperparameters are determined using the cross-validation technique. After the training, the Prediction is made by the following formula:

$$\hat{y} = e^{X\beta} \tag{3.2}$$

where $\hat{y}$ is the predicted execution time.

3.3. Recommendation module. The Recommendation module aims to provide an optimal recommendation according to the user request relying on the Prediction module. The module generates all possible configurations and sends the configurations to the Prediction module to get the execution times of the given possible configurations. Finally, the Recommendation module sorts the results according to the execution time and returns top-n configuration parameters, the first of which is the optimal configuration. Before using the Recommendation module, all configurations per application are uploaded to indicate a set of significant metrics, such as the input type, size, or environment. The configuration files are described in JSON format.

Moreover, the configuration must not be loaded every time, as the system caches every successfully loaded configuration. When the user first requests the Recommendation module, it packs a set of configurations using the application name and stores them in the cache. The metrics of the configuration files depend on the key features from the simulation datasets (see Table 2) and correspond to the features of the trained regression models.

The user may specify some fixed metrics. In that case, the Recommendation module will not generate possible values for that metric. For instance, if the user is interested in the Spark environment, the module will not consider the Hadoop environment. The maximum number of configurations is equal to:

$$N_{conf} = N_{env} \cdot N_{datatype} \cdot N_{nodes} \tag{3.3}$$

where N_{env} is the number of environments, $N_{datatype}$ is the number of input data types, and N_{nodes} is the number of nodes in the cluster.

In the current version of the module, the number of possible configurations is equal to 48 per application, as two environments (Hadoop and Spark), eighth input data types (uncompressed, compressed with 7 possible compression methods and using the RAM over HDFS approach), and three cluster configurations (4, 8, and 16 nodes) are used.

4. Evaluation. Various workloads as input data have been evaluated based on diverse data node configurations and compression methods using CPU, memory, I/O, network, and job execution time metrics. Computational and storage resources of the IaaS (Infrastructure as a Service) cloud service of the Armenian hybrid research computing platform have been used for the experiments [23, 24]. The hardware and software configurations of the experimental environment is described in Table 4.1.

With the RAM over HDFS approach, 2GB of RAM was allocated from each data node for HDFS. Different input data sizes, compression methods (gzip, bzip, lzo, snappy, lz4, and zstandart), RAM over HDFS approach, data nodes (up to 16), and additional application-specific parameters have been evaluated for selected MapReduce applications. The outputs of the experiments have been used to train the regression models. 20% of the collected data was selected as a test dataset and the remaining 80% as a training dataset. The k-Fold Cross-Validation was used to find the optimal regression model's hyperparameters for all types of applications [25].

Table 4.1: List of the software and hardware configurations

Configuration name	**Description**
Operating system	Ubuntu 18.04
Master nodes count	1
Data nodes count	16
RAM	4 GB per node
Hard disk	120 GB SATA per node
Hadoop version	3.2.1
Spark version	2.4.5
Java JDK version	1.8
HDFS block size	128 MB
HDFS block replication	2

Table 4.2: Prediction errors of the regression model

Workflow	**Train**		**Test**	
	R^2	**RMSE**	R^2	**RMSE**
Log Analyzer	0.99	11.5	0.96	25.3
WordCount	0.99	65.2	0.99	158.8
K-Means	0.97	71.4	0.96	87.4
TestDFSIO	0.96	13.6	0.92	22.6
TeraSort	0.98	48.4	0.94	58.2

Table 4.2 shows the experimental results of R^2 and RMSE per job, where R^2 score and root mean square error evaluate the regression models linked with the service. R^2 measures the percentage of variation, while the RMSE measures of the size of the error in the regression model. For each type of application, one polynomial regression model was trained, which at the input receives the necessary metrics described in Table 3.1. The output gives the execution time of the application with the given configuration parameters.

The best R^2 score on the train and test datasets has the regression model of the WordCount application. In contrast, the lowest RMSE on the training dataset has the regression model of the LogAnalyzer application and the test dataset TestDFSIO's regression model. The highest RMSE on the training dataset has K-means's model and on the test dataset WordCount's model. WordCount is expected to have a stable execution time as it is a CPU-intensive application, but the RMSE of the regression model is highest on the test dataset. The reason is that the execution times of WordCount in different configuration parameters are pretty diverse (e.g. the difference between the execution time of WordCount if the input data is compressed with gzip and is uncompressed is high compared with the other applications with the same scenario). This means that the WordCount application execution times with different configurations have higher variance than the other applications.

4.1. Applications. The experiments aim to evaluate the boosting of real applications by considering compression technique to reduce I/O and RAM over the HDFS approach to improve I/O rate compered to the default approach. The optimal compression method provides minimal execution time compared to the other compression methods. The Recommendation module offers the optimal performance considering compression or RAM over HDFS approaches. The evaluation results are given on the figures for 16 data node configurations.

4.1.1. LogAnalyzer. The real performance boosts of using compression methods, RAM over HDFS approach and the predicted performance boost of LogAnalyzer job are shown in Fig. 4.1.

The best performance boosts for 4GB, 8GB, and 16GB are correspondingly 2%, 7.5%, and 3% for Hadoop and 49%, 53% and 83% for Spark. Fig. 4.1 shows that the best performance boost for 4 GB and 16 GB inputs in the Hadoop environment gives compression techniques, while for 8 GB RAM over HDFS approach. Besides,

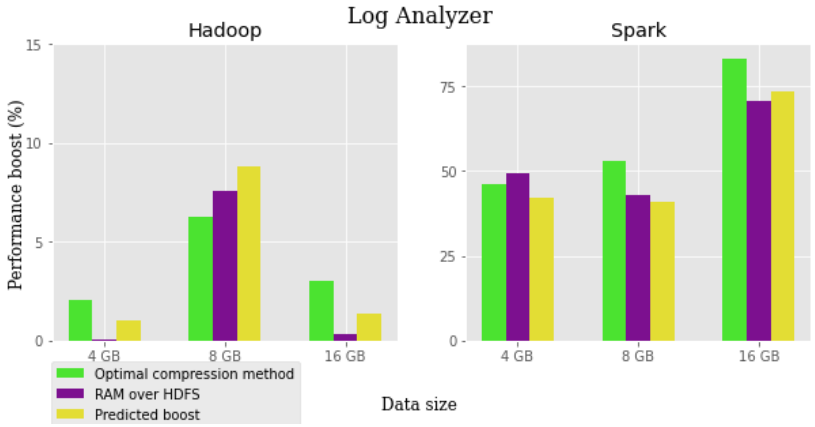

Fig. 4.1: Log analyzer performance boosts for 4GB, 8GB, and 16GB data on Hadoop and Spark

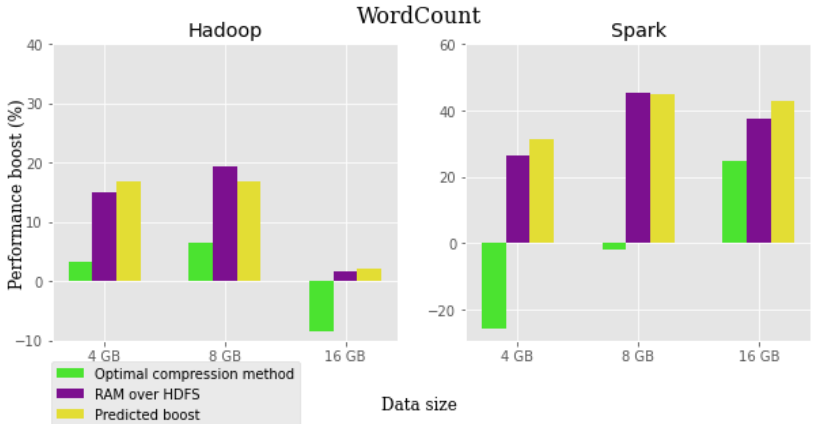

Fig. 4.2: Wordcount performance boosts for 4GB, 8GB, and 16GB data on Hadoop and Spark

the predicted performance boost compared with real ones has less than 3% of relative error. For the Spark environment, the scenario is a bit different, as the best actual performance boosts compression techniques for 8 GB and 16 GB inputs, while for 4 GB input data RAM over HDFS approach gives more performance boost. The predicted performance boost for Spark environment is also close to real ones and has less than 10%.

4.1.2. WordCount. Compared with LogAnalyzer, the real and predicted performance boosts of a CPU and data-intensive WordCount application are shown in Fig. 4.2.

The best performance boosts for 4GB, 8GB and 16GB are correspondingly 15%, 19% and 1.5% for Hadoop and 26%, 45.5% and 37.5% for Spark. As the figure shows, compression methods in the Hadoop environment give performance boost only in 4 GB and 8 GB of input data. When the input data size is 16 GB, compression methods add an average of 10% to the job execution time. RAM over HDFS approach for Hadoop environment compared with compression techniques shows more accurate performance boosts and in three sizes of input data gives performance boost. When the input data is 4 GB and 8GB, the performance boosts are big (15-20%), but for 16 GB of input data, the performance boost is approximately 2%. In this case, the predicted performance boost is also closed to real ones and has less than 3.5% relative error than real performance boosts. The Spark environment is quite different from the Hadoop, as Spark delivers fast performance, while Hadoop has an advantage of linear processing. For 4 GB and 8GB of input data, compression techniques don't give a

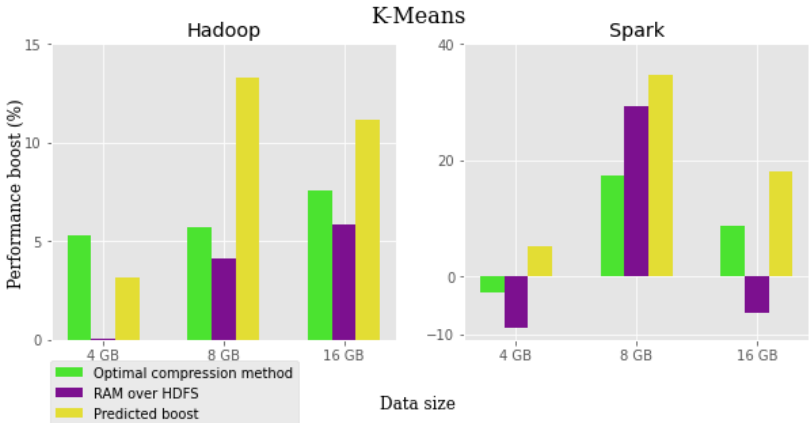

Fig. 4.3: K-Means performance boosts for 4GB, 8GB, and 16GB data on Hadoop and Spark

performance boost, while 16GB of input compression improves the performance. As in Hadoop, the RAM over HDFS approach gives more performance boost against compression techniques in a Spark environment. In this case, the predicted performance boost is also close to real ones and has less than 6% relative error compared with the best real performance boost.

4.1.3. K-Means. The following real application is K-Means, in which performance boosts are shown in Fig. 4.3.

The best performance boosts for 4GB, 8GB, and 16GB are correspondingly 5.3%, 5.7%, and 7.6% for Hadoop and Spark if the input data is 4 GB, there is not positive performance boost. In contrast, for other sizes, the best performance boost is correspondingly 29.3% and 8.7% for Spark. The figure shows that both approaches improve the performance in the Hadoop environment. Still, compression techniques for the K-Means job help reach a better performance boost than RAM over the HDFS approach. In the Spark environment, compression techniques help get a performance boost for 8GB and 16GB input data, while RAM over the HDFS approach improves performance only for 8GB input. Compared with real ones for Hadoop, the predicted performance boost has approximately less than 7.3% and for Spark less than 10%.

Considering 4GB, 8GB, and 16GB are respectively light, medium, and heavy workload, the average CPU and memory usages across all nodes for real-world applications is shown in a RAM over HDFS approach and using the optimal compression method in Table 4.3.

For all types of real applications, the performance boost of Spark is significantly better than Hadoop because Spark utilizes more CPU and RAM than Hadoop.

4.2. Micro benchmarks. Real application examples show that using compression techniques or the RAM over HDFS approach can achieve different performance gains for various applications. The scenario for micro-benchmarks compared with real applications is the opposite. On the one hand, using compression for micro-benchmarks is inconvenient because the generated TestDFSIO or TeraGen data has many repetitions, and compression methods may show fake and colossal performance boosts. On the other hand, some micro-benchmarks, such as the TeraSort, do not support compression. Therefore, for micro-benchmarks, the performances is compared without considering compression techniques.

4.2.1. TestDFSIO. The execution time of the TestDFSIO application for reading and writing options compared to RAM over the HDFS approach is shown in Fig. 4.4.

The figure shows that not always RAM over HDFS helps to improve performance. TestDFSIO with the read option is faster in the original case, while the write option is faster in the RAM over HDFS approach. Hence, the RAM is much faster than the hard disk; the simulation time boost is considerable for write-option, as it depends on faster RAM than the hard disk.

Table 4.3: Average CPU and memory usage across cluster nodes

Workflow				LogAnalyzer		WordCount		K-Means	
Framework				Hadoop	Spark	Hadoop	Spark	Hadoop	Spark
Complexity	Light	Original approach	CPU	6.47	24.54	5.78	47.94	7.16	39.89
			RAM	15.82	16.47	13.24	34.39	18.09	29.88
		Best compression	CPU	6.51	35.48	6.11	36.55	7.05	31.82
			RAM	16.51	18.77	16.57	40.47	18.14	26.69
		RAM over HDFS	CPU	6.87	55.49	6.87	55.49	6.97	40.23
			RAM	23.31	29.83	23.31	29.83	18.07	29.64
	Medium	Original approach	CPU	5.81	14.70	5.28	21.58	6.46	50.46
			RAM	14.40	17.16	15.92	37.91	16.54	30.49
		Best compression	CPU	5.87	53.42	5.57	41.21	6.32	43.68
			RAM	1.39	22.12	16.45	66.99	16.83	29.64
		RAM over HDFS	CPU	6.97	61.27	6.97	61.27	7.01	70.14
			RAM	29.44	29.49	29.44	29.49	18.58	31.83
	Heavy	Original approach	CPU	6.37	15.53	6.72	37.61	6.69	37.65
			RAM	13.99	23.56	17.36	42.69	18.92	44.37
		Best compression	CPU	6.39	71.19	6.29	51.53	6.68	44.39
			RAM	16.23	20.44	16.55	67.69	17.16	35.62
		RAM over HDFS	CPU	6.92	49.17	6.92	49.17	7.36	58.02
			RAM	41.14	47.45	41.14	47.45	23.37	43.71

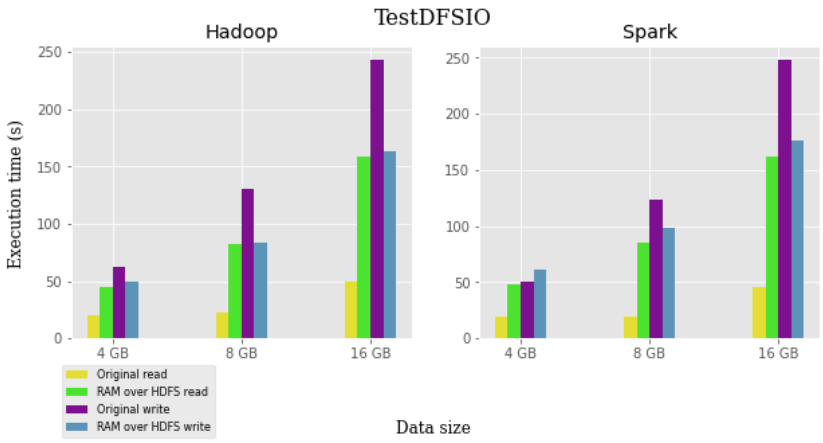

Fig. 4.4: Evaluation of TestDFSIO for read and write options compared to RAM over HDFS approach

In the Hadoop environment original approach for 4GB, 8GB, and 16GB data sizes for the read option is correspondingly 1.25, 2.5, and 2.3 times faster than RAM over HDFS approach, while the write option is correspondingly 1.21, 1.35, and 1.33 times faster than the original approach. In the Spark environment, the scenario is similar. The only difference is that the 4 GB data write option is also 1.2 times faster than the original approach. For 8GB and 16 GB input data for the read option original approach is 3.47 and 2.52 times faster than RAM over HDFS, while for the write option RAM over HDFS approach shows correspondingly 1.2 and 1.29 times fast processing time.

4.2.2. TeraSort. The results of the TeraSort micro-benchmark, which is similar to TestDFSIO, show that in most cases, the RAM over HDFS improves the performance (TestDFSIO with write option and TeraGen job of the TeraSort package). In contrast, the scenario is different in some cases (TestDFSIO read option and the remaining two jobs of the TeraSort package). The increase and decrease of performance mainly depend on

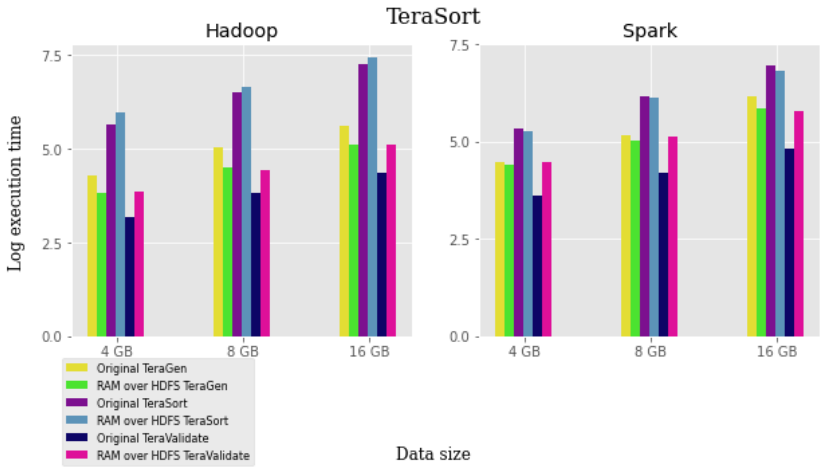

Fig. 4.5: Evaluation of TeraSort compared to RAM over HDFS approach

the overloading and underloading memory usage. Hence, the underloading memory amount can be effectively used, as RAM over HDFS, to boost the performance. The logarithmic scale evaluation of the TeraSort micro-benchmark for TeraGen, TeraSort, and TeraValidate jobs compared to RAM over the HDFS approach is shown in Fig. 4.5. The logarithmic scale improves the graph's appearance because the execution time difference between TeraSort micro-benchmark jobs (TeraGen, TeraSort, TeraValidate) is significant.

The figure shows that the Teragen job works faster in Hadoop and Spark environments with RAM over the HDFS approach. In contrast, the other two TeraSort and TeraGen jobs are faster in the original approach.

In the Hadoop environment for 4GB, 8GB, and 16 GB data TeraGen works 1.36, 1.32, and 1.4 times faster than the original approach. TeraGen and TeraSort are faster in RAM over the HDFS approach for the Spark environment, while TeraValidate is faster in the original one. The TeraGen for 4 GB, 8GB, and 16 GB data works correspondingly 1.07, 1.11, and 1.26 times, and TeraSort 1.04, 1.15, and 1.12 times faster than the original approach. At the same time, TeraValidate is faster, correspondingly 2.32, 2.56, and 2.59 times than RAM over the HDFS approach.

The experimental results show that for micro-benchmarks using RAM over the HDFS approach, not always provide a performance boost, while for real applications mostly there is a way to reach performance improvement using either compression techniques or RAM over the HDFS approach.

5. Conclusion. The article presented performance-efficient Recommendation and Prediction service for Hadoop and Spark Big Data frameworks based on data compression and in-memory file system. The suggested service consists of Prediction and Recommendation modules to predict an application's execution time with given metrics and provide optimal configurations considering the simulation time. The codeless service can be easily expanded for new scientific workflows by adding configuration files describing all the applications' features and train regression model and encoder.

The service has been evaluated using diverse configuration parameters, like the environment, several data nodes, or the compression method. Some applications and micro-benchmarks have been studied to improve the performance, such as Log Analyzer, WordCount, and K-Means. Each approach has a different effect on performance for different types of applications. The relative error of the predicted boost for Log Analyzer and WordCount in Hadoop is about 3–3.5% and in Spark is about 6-10%, while for K-Means, it is less than 7.3% and 10%, respectively.

It is planned to continue the experiments and studies to improve regression models' accuracy and find similar patterns in different applications considering the energy efficiency alongside the simulation time.

Acknowledgement. The European Commission through the NI4OS, EaPConnect Projects and the COST Action IC1305, 'Network for Sustainable Ultrascale Computing (NESUS)', and State Thematic Project entitled "Towards the Green e-Infrastructures" supported the work presented in this paper.

REFERENCES

[1] A. Mohamed, M. K. Najafabadi, Y. B. Wah, E. A. K. Zaman, R. Maskat, *The state of the art and taxonomy of big data analytics*, Artificial Intelligence, Vol. 53, 2020, No. 2, pp. 989–1037, doi:10.1007/s10462-019-09685-9.

[2] U. R. Pol, *Big data analysis: comparision of Hadoop MapReduce and apache spark*, International Journal of Engineering Science, Vol. 6389, 2016, doi: 10.4010/2016.1535.

[3] J. Dean, S. Ghemawat, *MapReduce: a flexible data processing tool*, Communications of the ACM, Vol. 53, 2010, No. 1, pp. 72–77, doi: 10.1145/1629175.1629198.

[4] H. Won, M.C. Nguyen, M.S. Gil, Y.S. Moon, K.Y. Whang, *Moving metadata from ad hoc files to database tables for robust, highly available, and scalable HDFS*, The Journal of Supercomputing, Vol 73, 2017, No. 6, pp. 2657–2681, doi: 10.1007/s11227-016-1949-7.

[5] K. Shvachko, H. Kuang, S. Radia, R. Chansler, *The hadoop distributed file system*, IEEE 26th symposium on mass storage systems and technologies (MSST), 2010, pp. 1–10, doi: 10.1109/MSST.2010.5496972.

[6] Al-Laham, M.— El Emary, I.M., *Comparative study between various algorithms of data compression techniques*, IJCSNS International Journal of Computer Science and Network Security, Vol. 7, 2007, No. 4, pp. 281–291.

[7] A. Kocharyan, B. Ekane, B. Teabe, G.S. Tran, H. Astsatryan, D. Hagimont, *A remote memory sharing system for virtualized computing infrastructures*, IEEE Transactions on Cloud Computing, 2020, doi: 10.1109/TCC.2020.3018089.

[8] H. Astsatryan, A. Kocharyan, D. Hagimont, A. Lalayan, *Performance Optimization System for Hadoop and Spark Frameworks*, Cybernetics and Information Technologies: Vol. 20, 2020, No. 6, pp. 5–17, doi: 10.2478/cait-2020-0056.

[9] A. Kocharyan, B. Teabe, V. Nitu, A. Tchana, D. Hagimont, H. Astsatryan, H. Kocharyan, *Intra-node Cooperative Memory Management System for Virtualized Environments*, IEEE Ivannikov Memorial Workshop, 2018, pp. 56–60, doi: 10.1109/IVMEM.2018.00018.

[10] D.C. Vinutha, G. Raju, *In-Memory Cache and Intra-Node Combiner Approaches for Optimizing Execution Time in High-Performance Computing*, SN Computer Science, Vol. 1, 2020, No. 2, pp. 1–7, doi: 10.1007/s42979-020-0089-6.

[11] Lee, Woo-Hyun— Jun, Hee-Gook— Kim, Hyoung-Joo, *Mapreduce performance enhancement using in-node combiners*, International Journal of Computer Science and Information Technology, Vol. 7, 2015, No. 5, pp. 1–17, doi: 10.5121/ijcsit.2015.7501.

[12] N.S. Islam, M.W. Rahman, R. Jose, H. Wang, H. Subramoni, C. Murthy, D.K. Panda, *High performance RDMA-based design of HDFS over InfiniBand*, Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, 2012, pp. 1–12, doi: 10.1109/SC.2012.65.

[13] L. Xiaoyi, S. I. Nusrat, Md. Wasi-Ur-Rahman, J. Jithin, S. Hari, W. Hao, K. P. Dhabaleswa, *High-performance design of Hadoop RPC with RDMA over InfiniBand*, 42nd international conference on parallel processing: 2013, pp. 641–650, doi: 10.1109/ICPP.2013.78.

[14] Z. Jing, W. Gongqing, X. Hu, W. Xindong, *A distributed cache for hadoop distributed file system in real-time cloud services*, ACM/IEEE 13th International Conference on Grid Computing: 2012, pp. 12–21, doi: 10.1109/Grid.2012.17.

[15] Y. Luo, S. Lou, J. Guan, S. Zhou, *A RAMCloud Storage System based on HDFS*, Architecture, implementation and evaluation: Journal of Systems and Software, Vol. 86, 2013, No. 3, pp. 744–750, doi: 10.1016/j.jss.2012.11.025.

[16] S. Aibo, M. Zhao, X. Yingying, L. Junzhou, *MHDFS: A memory-based hadoop framework for large data storage*, Scientific Programming: 2016, pp. 1–12, doi: 10.1155/2016/1808396.

[17] P. Narges, M. Ali, *Estimating runtime of a job in Hadoop MapReduce*, Journal of Big Data, Vol. 7, 2020, No. 44, pp. 1–18, doi: 10.21203/rs.2.20701/v1.

[18] M. Sara, E. Iman, A. I. Mohamed, *A Machine Learning Approach for Predicting Execution Time of Spark Jobs*, Alexandria Engineering Journal, Vol. 57, 2018, No. 4, pp. 3767–3778, doi: 10.1016/j.aej.2018.03.006.

[19] M. L. Alberto, S. Sergio, R. C. Antonio, *RESTest: Black-Box Constraint-Based Testing of RESTful Web APIs*, International Conference on Service-Oriented Computing: 2020, pp. 459-475, doi: 10.1007/978-3-030-65310-1_33.

[20] N. Sayalee, B. Tripti, *HMR log analyzer: Analyze web application logs over Hadoop MapReduce*, International Journal of UbiComp: Vol. 4, 2013, No. 3, pp. 41-51, doi: 10.5121/iju.2013.4304.

[21] K. Krishna, N. M. Murty, *Genetic K-means algorithm*, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics): Vol. 29, 1999, No. 3, pp. 433–439, doi: 10.1109/3477.764879.

[22] E. Ostertagová, *Modelling using polynomial regression*, Procedia Engineering, Vol. 48, 2012, pp. 500–506, doi: 10.1016/j.proeng.2012.09.545.

[23] H. Astsatryan, V. Sahakyan, Y. Shoukourian, J. Dongarra, P.H. Cros, M. Dayde, P. Oster, *Strengthening compute and data intensive capacities of Armenia*, 14th RoEduNet International Conference-Networking in Education and Research: 2015, No. 3, pp. 28–33, doi: 10.1109/RoEduNet.2015.7311823.

[24] Yu. Shoukourian, V. Sahakyan, H. Astsatryan, *E-Infrastructures in Armenia: Virtual research environments*, Ninth International Conference on Computer Science and Information Technologies Revised Selected Papers, 2013, pp. 1–7, doi: 10.1109/CSITechnol.2013.6710360.

[25] T. Fushiki, *Estimation of prediction error by using K-fold cross-validation*, Statistics and Computing, vol. 21, 2011, No. 2, pp. 137–146, doi: 10.1007/s11222-009-9153-8.

Edited by: Dana Petcu
Received: Nov 1, 2021
Accepted: Dec 1, 2021