RESEARCH PROJECT APPLICATION FORM

Performance Optimization System for Hadoop and Spark Frameworks

Promoting academia-industry alliances for R&D through collaborative and open innovation platform

Project reference number: 598719-EPP-1-2018-1-MK-EPPKA2-CBHE-JP

Project duration: 15 November 2018 – 14 November 2021

EU funding instrument: European Neighbourhood Instrument (Erasmus+: KA2 CBHE)

Erasmus+ (CBHE) grant amount: 531,165.00 €

Partner countries: Armenia, Bosnia and Herzegovina, North Macedonian, Austria, Germany, Finland

Target groups: University management and students, companies, research institutions, intermediaries.

Grant holder: Ss. Cyril and Methodius University in Skopje, North Macedonia

Coordinator: Prof. Elena Dumova-Jovanoska, Ss. Cyril and Methodius University in Skopje

This project has been funded with support from the European Commission. This document reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Research project application form information	
PROJECT NAME	**Performance Optimization System for Hadoop and Spark Frameworks**
RESEARCH AREA	**Big Data**
NARROW RESEARCH AREA	**Energy consumption and performance optimization**
TYPE OF RESEARCH	☐ fundamental ☒ scientific ☒developing ☐ applicative ☐ educative ☐ cooperative
PROJECT LEADER	**Hrachya Astsatryan**
LEAD ORGANIZATION	**Institute for Informatics and Automation Problems of the National Academy of Sciences of the Republic Armenia**
LEAD ORGANIZATION ADDRESS	**1, Paruyr Sevak str.,**
R&D UNIT	**NPUA/IIAP Cooperative R&D Unit, Open-Source Software Lab**
PROJECT DURATION	**from: 01/05/2021 to: 30/04/2022**
PROJECT COST (€)	**8000**
DATE AND PLACE OF APPLICATION FORM SUBMISSION	**20 April, 2021** **Yerevan, Armenia**

PROJECT LEADER

R&D UNIT LEGAL REPRESENTATIVE

Contents

Project summary

The optimization of large-scale data sets depends on the technologies and methods used. The MapReduce model, implemented on Apache Hadoop or Spark, allows splitting large data sets into a set of blocks distributed on several machines. Data compression reduces data size and transfer time between disks and memory but requires additional processing. Therefore, finding an optimal tradeoff is a challenge, as a high compression factor may underload Input/Output but overload the processor. The project aims to present a system enabling the selection of the compression tools and tuning the compression factor to reach the best performance in Apache Hadoop and Spark infrastructures based on simulation analyzes.

Keywords: Hadoop, Spark, data compression, CPU/IO tradeoff, performance optimization, energy consumption.

1. State of the art

Big Data processing [1] is a resource-intensive operation that uses specific hardware and software. Due to the intense Input/Output (I/O) nature of the processing, the hardware architecture is different from the traditional high-performance computing (HPC) clusters or supercomputers, particularly, local disks are required for all data nodes. Moreover, the data processing application stack is also significantly different from traditional approaches. For instance, the data volume is substantially larger than in other operations, and the data sets are poorly structured, and various data types are available.

The traditional relational database management systems, like SQL queries, are incapable of tackling semi-structured or unstructured Big Data processing. Thus, the MapReduce model has been introduced, a critical technology for processing and generating extensive data sets. Its implementations, such as Apache Hadoop [2] or Spark [3], split large data sets into a set of distributed blocks, execute map tasks in parallel on these blocks, and finally reduce tasks for the aggregation of results. Data compression techniques are used to overcome data storage and network bandwidth limitations to process a massive volume of data. In Big Data infrastructures, it decreases the size of data chunks to minimize the time delay forced by the I/O operation and save space on local disks. Therefore, it is a challenge to find an optimal tradeoff, as high compression factor may underload I/O but overload CPU, while a weak compression factor may underload CPU but overload I/O. The ideal configuration is when both I/O and CPU are used entirely. CPU (respectively I/O) should not be waiting for I/O (respectively CPU) to reach the best performance.

The project aims to present a system enabling the selection of the compression tools and tuning the compression factor to reach the best performance in Hadoop and Spark infrastructures based on simulation analysis.

2. Recent results from own research

Mainly, the Hadoop and Spark cluster consisting of a master and 16 slave nodes is used for the experiments with five distinct configurations: 1+4, 1+8, and 1+16. Each node in the cluster runs the Openstack middleware with one virtual machine per node using Ubuntu server 18.04 operating system, 3 GB of memory, and a 120 GB SATA shared hard disk. The Hadoop version 3.2.1, Spark version 2.4.5, Java JDK version 1.8, and HDFS block size 128MB are used. The replication factor is set at 2 (default value is 3) to facilitate the decommissioning of data nodes. The total number of experiments per Apache Hadoop and Spark environment is 240. 4GB, 8GB, and 16GB data workload are carried out for all experiments. Data compression reduces the storage usage. The analyzes of compressed and raw-files compression ratios are illustrated in Figure 1.

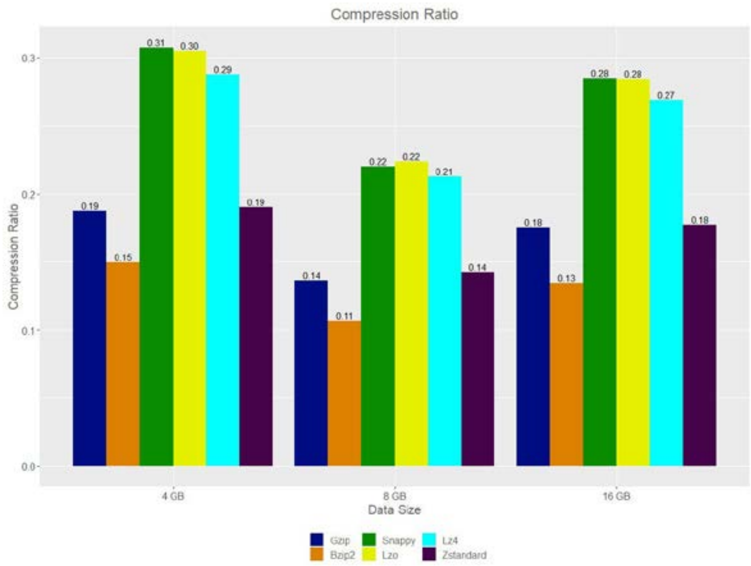

Fig. 1. Compression ratio for 4GB, 8GBB, and 16GB data workloads

The figure shows the best compression ratio with a 13-17% of the average value for gzip, zstandard, and bzip2 algorithms. The compression ratio difference between gzip and bzip2 is about 4%. According to the benchmarks, the execution time of gzip is about seven times faster than the bzip2 compression. The lzo, lz4, and snappy algorithms have 26-27% low compression ratios with about seven times faster execution time compere to the gzip compression. The remarkable outcome from this experiment is that with Spark with the lz4 compression format, and with 8GB and 16 GB data seta, it was able to obtain a 47% improvement at the cost of a 15-25% and 18-28% memory usage for uncompressed input data; 20-70% CPU and 18-20% memory usage for splittable compressed data; and 8-10% CPU and 14-28% memory usage for non-splittable compressed data. The LogAnalyzer's execution time for Hadoop is optimized up to 4.4% with the lz4 compression format regardless of the input data size (see fig. 2).

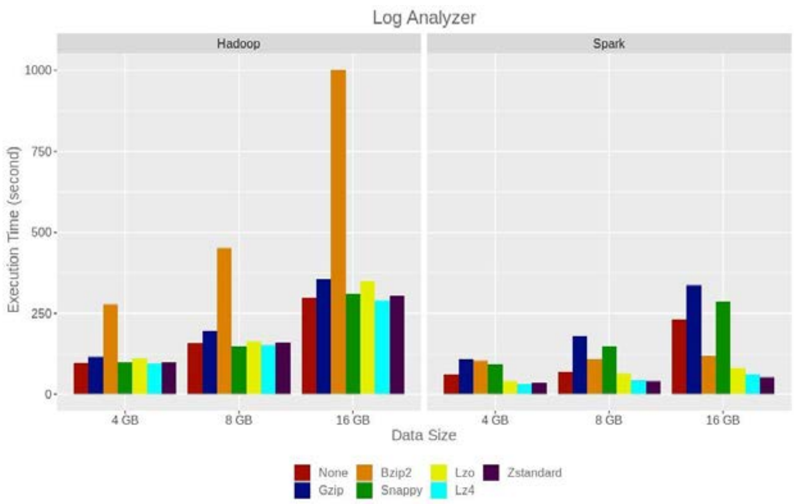

Fig. 2. The LogAnalyzer experiment performs for 4GB, 8GB, and 16GB data on 16 nodes Hadoop and Spark configuration

The standard deviation for Hadoop is up to 2% when eight-node and four-node are implemented, and 9% for eight-node and 27% for four-node configurations for Spark. The average CPU usage of all nodes on the Hadoop cluster is 6–6.5%, while the memory usage is 12-16.5%. On Spark for uncompressed input data, the average CPU usage is 15-25% and 18-28% memory, for splittable compressed data 20-70% CPU and 18-20% memory, for non-splittable compressed data 8-10% CPU and 14-28% memory. On Hadoop, average resource usage is almost the same.
The picture is different if the WordCount massive simulation application is studied instead of the LogAnalyzer (see fig. 3). The experiments show that in the case of using 16 GB input data, the compression codec slightly improves the execution time for the Hadoop framework and significantly improves the execution time framework. lz4 and lzo codecs show the best performance for both cases. Within the Hadoop lz4 has a bit higher performance than lzo and on Spark the opposite (lzo shows lower execution time). The Hadoop execution time for 8 and 16 nodes configuration is almost the same, but on four-node, the average execution time increases by 1.4%. On the Spark 8 node cluster, the average execution time increases by 17% and on the four-node cluster by 51%. On the Hadoop cluster, the average CPU usage is 5.3-6.7% and memory 12-17.3%. On the Spark cluster with uncompressed input data, CPU usage is 20-47% and memory 30-42%. For input data compressed with non-splittable codec, the average CPU usage is 6-7%, memory 20-30%, and for data compressed with splittable codec CPU -20-50%, memory 30-70%. As LogAnalyzer for WordCount job on the Hadoop environment, the average resource usage is almost the same. The best performance for Wordcount job shows lzo codec, which is 8% faster than uncompressed data but uses 12% more CPU and 23% more memory on average.

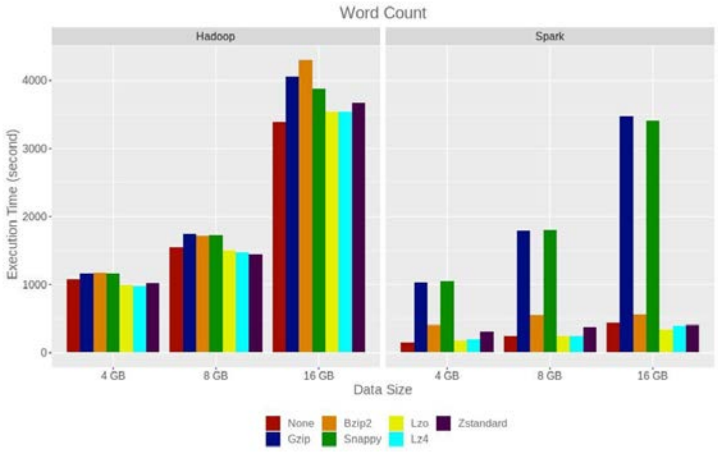

Fig. 3. WorCount experiment performance for 4GB, 8GB, and 16GB data on 16 nodes Hadoop and Spark configuration

The experiments show that the splittable codecs improve the execution time of LogAnalyzer and WordCount applications, besides the Bzip2 slow compression algorithm for the Hadoop cluster. Gzip and Snappy non-splittable codecs decrease the storage size and increase execution time. The splittable compression codecs have a substantial impact on the Spark environment. The compression codecs were not used for TereGen and TestDFSIO benchmarks, as an algorithm artificially generates the data. Figure 4 shows the TestDFSIO benchmark's execution time on Hadoop and Spark environments with 16 node configurations.

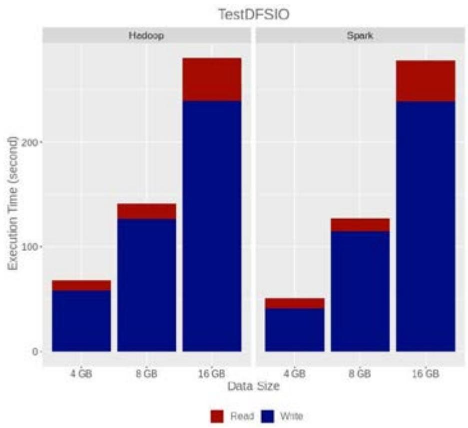

Fig. 4. TestDFSIO benchmark execution time for 4GB, 8GB, and 16GB data on 16 node Hadoop/Spark

On eight-node configuration cluster benchmarks with write option work in the approximate same time. The deviation for Hadoop is 2%, and Spark is 1%. For reading option execution time increases by 82% are on Hadoop and 31% on Spark. On four-node configuration, write works 4% slower on Hadoop and 18% on Spark, for read option works three times slower on both environments. Figure 6 shows TeraGen, TeraSort, TeraValidate benchmark's execution time on Hadoop and Spark environments with 16 node configurations. TeraGen and TeraValidate work faster on Hadoop and TeraSort on Spark. On average, the simulation time of benchmarks is 12% smaller for Spark compared to Hadoop. On eight-node Hadoop and Spark clusters, the results of TeraGen and TeraSort are almost the same with only a 2% difference, but for TeraValidate, the benchmark execution time increases by 20% on Hadoop and 50% on Spark. On four nodes, Hadoop cluster TeraGen on average is faster by 13%, Terasort 3%, and Teravalidate is slower by 43% compared with 16 node configurations. On four nodes, Spark cluster TeraGen is faster by 7%, TeraSort is slower by 4%, and Teravalidate by 72%. In both environments, the average CPU usage is 5-7%, memory 12-14% on Hadoop, and 15-16% on Spark.

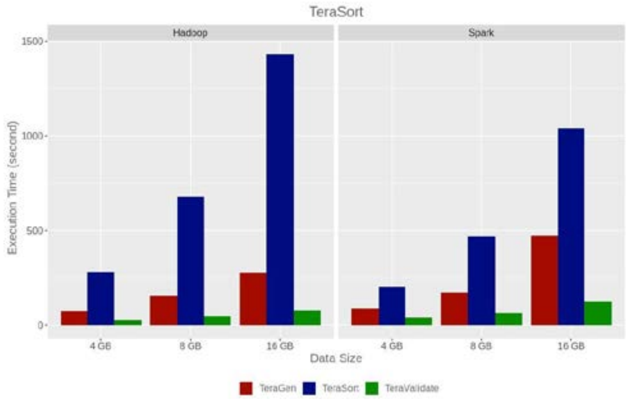

Fig. 6. TeraSort benchmark execution time for 4GB, 8GB, and 16GB data on 16 nodes Hadoop/Spark.

In the k-means clustering application, the 1GB, 2GB, 4GB input data sizes are used for the experiments. According to Figure 6, gzip, snappy, and zstandart codecs show almost the same performance as if the input is uncompressed.

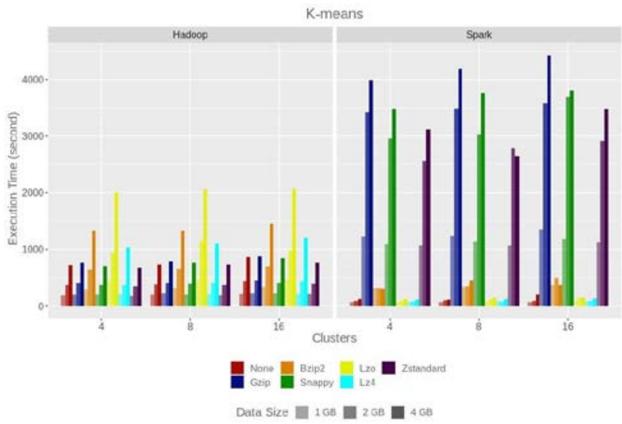

Fig. 6. K-means benchmark execution time for 1GB, 2GB, and 4GB data on 16 nodes Hadoop/Spark.

The scenarios are entirely different in the Spark cluster case, as the splittable codecs besides bzip2 show better performance than if data is uncompressed. The best performance is reached using the lz4 codec by having about 93% of the compression ratio. Instead of Hadoop (deviation is 1%), the execution time on average increases by 30% and 93% on four-node and eight-node configuration of Spark. On Hadoop, the best performance shows zstandard codec 6.4% faster than for uncompressed data. On Hadoop k-means cluster, the average resource usage is almost the same compere to the LogAnalyzer and WordCount. The average CPU usage is 6-7%, while memory usage is 16-18%. The worst performance on spark show gzip, zstandard and snappy codec, which use, on average, 6-8% CPU and 30-48 % memory. If k-means input data is uncompressed, the average CPU usage is 37-50%, while the memory is 30-44%. In the case of the other codecs, the average CPU usage is 11-56%, while the memory is 26-44%. The best performance on Spark cluster show lz4, which is, on average, 8.8% faster than for not compressed input data, but uses on average 3% more CPU and 1% less memory.

The statistical analyzes of the memory and processor usages are presented in Table 1 to present the characteristics of the data and to study the dispersion. In the case of TestDFSIO and TeraSort that is very reliable, while the LogAnalyzer, WordCount, and K-means, there is a significant variance between the data and the statistical average.

Table 1. SD and Means analyzes five workloads.

No	Job	Framework	CPU usage		Memory usage	
			Mean (%)	SD	Mean (%)	SD
1	LogAnalyzer	Hadoop	6.29	0.17	15.23	1.18
		Spark	31.21	20.36	21.97	3.66
2	WordCount	Hadoop	6.01	0.35	15.57	1.43
		Spark	28.93	15.54	42.11	16.81
3	TestDFSIO	Hadoop	4.74	0.49	17.76	0.15
		Spark	4.74	0.80	14.85	0.21
4	TeraSort	Hadoop	6.15	0.68	12.96	1.08
		Spark	6.21	0.59	15.31	0.44
5	K-means	Hadoop	6.67	0.33	17.21	0.72
		Spark	22.58	16.21	34.09	5.70

References

1. Chen, J., Y. Chen, X. Du, C. Li, J. Lu, S. Zhao, X. Zhou. Big data challenge: a data management perspective. - Frontiers of Computer Science, Vol. 7, 2013, No 2, pp. 157-164.
2. Lublinsky, B., K. T. Smith, A. Yakubovich. Professional Hadoop Solutions, p. 504, John Wiley & Sons, Indiana, USA, 2013.
3. Zaharia, M., R.S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi. Apache spark: a unified engine for big data processing. - Communications of the ACM, Vol. 59, 2016, No 11, pp. 56-65.
4. Cheng, D., X. Zhou, P. Lama, Ji Mike, C. Jiang. Energy efficiency aware task assignment with dvfs in heterogeneous hadoop clusters. - IEEE transactions on parallel and distributed systems, Vol. 29, 2017, No 1, pp. 70-82.
5. Nitu, V., A. Kocharyan, H. Yaya, A. Tchana, D. Hagimont, H. Astsatryan. Working Set Size Estimation Techniques in Virtualized Environments: One Size Does not Fit All - ACM Meas. Anal. Comput. Syst., Vol. 2, 2018, pp. 1-21.
6. Kothuri, P., D. Garcia, J. Hermans. Developing and Optimizing Applications in Hadoop. - J PY - Journal of Physics: Conference Series, Vol. 898, 2017, No 5.
7. Dean, J., S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. - Communications of the ACM, Vol. 51, 2008, No 1, pp. 107-113.
8. Won, H., 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.
9. Uthayakumar, J., T. Vengattaraman, P. Dhavachelvan. A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications. - Journal of King Saud University - Computer and Information Sciences, 2018.
10. Liu, L.Y., J. F. Wang, R. J. Wang, J. Y Lee. Design and hardware architectures for dynamic Huffman coding – IEEE Proceedings-Computers and Digital Techniques, Vol. 142, 1995, No 6, pp.411-418.
11. Fenwick, P. M. The Burrows-Wheeler Transform for Block Sorting Text Compression: Principles and Improvements. - The Computer Journal, Vol. 39, 1996, No 9, pp. 731–740.
12. Fang, J., J. Chen, Z. Al-Ars, P. Hofstee, J. Hidders. Work-in-Progress: A High-Bandwidth Snappy Decompressor in Reconfigurable Logic. – In: IEEE International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS), Turin, Italy, 30 Sept.-5 Oct. 2018, pp. 1-2.
13. Liu, W., F. Mei, C. Wang, M. O'Neill, E. E. Swartzlander, Data Compression Device Based on Modified LZ4 Algorithm. - IEEE Transactions on Consumer Electronics, Vol. 64, 2018, No 1, pp. 110–117.
14. Rattanaopas, K., S. Kaewkeeree. Improving Hadoop MapReduce performance with data compression: A study using wordcount job. - In IEEE 14th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2017, pp. 564-567.
15. Haider, A., X. Yang, N. Liu, X. H. Sun, S. He. Ic-data: Improving compressed data processing in hadoop. - In IEEE 22nd International Conference on High Performance Computing (HiPC), 2015, pp. 356-365.

16. Chen, Y., A. Ganapathi, R. H. Katz. To compress or not to compress-compute vs. io tradeoffs for mapreduce energy efficiency. - In Proceedings of the First ACM SIGCOMM workshop on Green networking, 2010, pp. 23-28.
17. Lang, W., J. M. Patel. Energy management for MapReduce clusters. – In Proceedings of the VLDB Endowment, 3(1-2), 2010, pp.129–139.
18. Li, W., H. Yang, Z. Luan, D. Qian. Energy prediction for MapReduce workloads. - In IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, 2011, pp. 443-448.
19. Wirtz, T., R. Ge. Improving mapreduce energy efficiency for computation intensive workloads. - In IEEE International Green Computing Conference and Workshops, 2011, pp. 1-8.
20. Leverich, J., C. Kozyrakis. On the energy (in)efficiency of Hadoop clusters. - ACM SIGOPS Operating Systems Review, Vol. 44, 2010, No 1, pp. 61-65.
21. Tiwari, N., S. Sarkar, U. Bellur, M. Indrawan. An Empirical Study of Hadoop’s Energy Efficiency on a HPC Cluster. - Procedia Computer Science, Vol. 29, 2014, pp. 62–72.
22. Tatineni, M., J. Greenberg, R. Wagner, E. Hocks, C. Irving. Hadoop deployment and performance on Gordon data intensive supercomputer. - In Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery, 2013, pp. 1-3.
23. Narkhede, S., T. Baraskar. HMR Log Analyzer: Analyze Web Application Logs over Hadoop MapReduce. - International Journal of UbiComp (IJU), Vol. 4, 2013, No 3, pp. 41-51.
24. Krishna, K., M. Narasimha Murty. Genetic K-means algorithm. - IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), Vol. 29, No 3, 1999, pp. 433-439.
25. Zhao, W, Ma. H, He, Q. Parallel K-means clustering based on MapReduce. - In: CloudCom 2009, LNCS 5931. Berlin: Springer; 2009. pp. 674–9.
26. Astsatryan, H., Sahakyan, V., Shoukourian, Yu., Cros, P. H., Dayde, M., Dongarra, J., Oster, P. Strengthening compute and data intensive capacities of Armenia. - IEEE Proceedings of 14th RoEduNet International Conference - Networking in Education and Research (NER’2015), Craiova, Romania; September 2015, pp. 28–33.
27. Astsatryan, H., Narsisian, W., Kocharyan, A., da Costa, G., Hankel, A., Oleksiak, A. Energy Optimization Methodology for e-Infrastructure Providers. - Willey Concurrency and Computation: Practice and Experience, Vol. 29, 2017, No. 10, doi: 10.1002/cpe.4073.
28. Nitu, V., Kocharyan, A., Yaya, H., Tchana, A., Hagimont, D., Astsatryan, H. Working set size estimation techniques in virtualized environments: One size does not fit all. - Proceedings of the ACM on Measurement and Analysis of Computing Systems, 2(1), 2018, pp. 1-22.

3. Goals of the research project

The resource optimization addresses the growing needs of Big data processing and analysis. The traditional methods and tools are mainly dedicated to CPU resource optimization, but the memory and I/O consume a significant portion of Big data processing resources. Many scientific studies have been dedicated to the memory optimizations in hardware [4], kernel memory [5], and middleware [6] layers. The project aims to optimize the resources in the application level using several compression algorithms within the Apache Hadoop and Spark frameworks, aiming to reduce the size of the files to be processed (to be loaded into memory, and written back to the disk). This approach increases the CPU load of the system overall, but as already mentioned, the CPU is not the most consumed resource in such systems, and it often stays underutilized. In the meantime, the splittable compressing algorithms split and merge back the data while using the MapReduce development model. The suggested system is based on Apache Hadoop and Apache Spark general-purpose Big Data computing frameworks. If the Apache Hadoop is a model for reading and writing data processing based on disk, the Apache Spark performs in-memory calculations with the resilient distributed data sets. Apache Hadoop is an open-source Java-based distributed computing framework built for applications implemented using MapReduce parallel data processing paradigm [7] and Hadoop Distributed File System (HDFS) [8].

As a distributed file system, HDFS provides a reliable, scalable, and fault-tolerant distributed data storage. The data is stored as blocks for handling the hardware failures. The replication factor shows the number of copies of a block in HDFS. MapReduce has become a critical distributed processing model for large-scale data-intensive applications like data filtering, feature extraction, or web indexing. The Map and Reduce functions are the key components of the MapReduce programming model. The Map function processes a key/value pair for generating a set of intermediate key/value pairs, while the Reduce function aims to merge all intermediate values associated with the same intermediate key. When the Map tasks are completed, the intermediate output is shuffled and sorted. The shuffle step is the only communication step between data nodes in MapReduce, during which nodes begin to swap the intermediate outputs from the map tasks. After shuffling and sorting, the reduce phase calls the user-defined reduce task and stores the output on HDFS.

The data compression algorithms are used in the suggested system to reduce the data movement cost by increasing the computation time. MapReduce supports the implementations of several compression and decompression algorithms called a codec. Data compression methods are classified according to data quality, codec schemas, data, and application types [9]. The codec allows us to compress and decompress data using splittable and non-splittable compression algorithms. The splittable compression algorithm splits the file into the compressed and uncompressed data blocks with the fixed size of the HDFS file's block size setting, where each of them can be decompressed separately of the others. The Hadoop also supports a non-splittable algorithm with a serial decompression, which usually requires longer decompression time. Therefore, the tradeoff of data compression algorithms depends on various factors, such as degree of compression, data quality (with or without loss), compression algorithm type, or data type. The degree of data size or I/O reduction depends on the compression ratio, which equals compressed data divided into the uncompressed data size. The compression ratio relies on the data and the compression algorithm. A lower ratio means less memory and I/O usages.

The data compression in Hadoop and Spark frameworks increases the storage space and improves performance to compute the job. The compression can be implemented for input data, intermediate Map output data, and Reduce output data stages. Intermediate compression of the map output reduces network usage during the Mapreduce shuffle step. All nodes begin to communicate with each other and collect the map output as the phase reduces input. If the input or intermediate output of the map phase is compressed, the framework chooses a decompression algorithm before processing according to the file extension (see table 2).

Table 2. A summary of compression formats available in Hadoop

No	Compression format	File extension	Splittable
1	gzip	.gz	No
2	bzip2	.bz2	Yes
3	snappy	.snappy	Yes (container file formats)
4	Lzo	.lzo	Yes (indexing algorithm)
5	lz4	.4mc	Yes (4MC library)
6	zstandart	.4mz	Yes (4MC library)

The data is stored securely, as all selected compression codecs are lossless. The gzip and deflate codecs use the deflate algorithm as a combination of lz77 and Huffman Coding [10]. The lz77 compression algorithm replaces duplicate bit positions regarding their previous positions. The difference between gzip and deflate is the Huffman encoding phase. The splittable compression bzip2 codec uses the Burrows-Wheeler (block-sorting) text compression and Huffman coding [11] algorithms. Bzip2 compresses data blocks independently and can compress data blocks in parallel. As a fast data compression and decompression library, snappy uses the ideas from lz77 [12]. Snappy blocks are non-splittable, but the files in the snappy blocks are splittable. The lzo (Lempel-Ziv-Oberhumer) compression algorithm is a variation of the lz77 compression algorithm. The algorithm is divided into the find the match, write the unmatched literal data, determine the length of the match, and write the match tokens parts. The next compression algorithm is the lz4, where compressed data files consist of LZ4 sequences that contain a token, literal length, offset, and match length [13]. Zstandart is an lz77-based algorithm developed by Facebook to support dictionaries, a massive search box, and an entropy coding step using finite-state entropy and Huffman coding.

4. Outcomes

In this project, a system enabling to find an optimal tradeoff to reach optimal performance in Apache Hadoop and Spark frameworks is presented. 4GB, 8GB, and 16GB data workloads for diverse applications, including TestDFSIO, TeraSort, WordCount, LogAnalyzer, and K-means, have been evaluated in Hadoop and Spark environments. The evaluation results are used by the suggested system to choose an optimal configuration environment. The compressed data processing analyzes show that the lz4 codec reaches Hadoop's best performance regardless of the input data size. Meanwhile, Spark achieves the best performance with Iz4 only for 4GB input data, and zstandard codec for 8GB and 16 GB cases.

It is planned to study the energy-efficient data transfers of Apache Hadoop and Spark using RDMA-capable networks like InfiniBand based on the developed methodology [27] and techniques [28].

5. Research plan and outcomes from each research stage

Short description of the work package		
Work package No.	**1**	State-of-the-art
Work package leader		Aram Kocharyan
Researchers		Hrachya Astsatryan Aram Kocharyan Kristina Khudaverdyan
Duration		01/05/2021 - 30/11/2021

Targets	– To analyse the current tendences and the state-of-the-art

Work package description	A prominent data processing engine for data centers is Hadoop MapReduce enabling users to avoid the costs of maintaining physical infrastructures. Many studies focus on MapReduce jobs to boost the performance and minimize the energy consumption in data centers by orders of magnitude. The authors [14-16] have studied the effect of data compression to improve the performance and energy efficiency for MapReduce small workloads only on four nodes clusters. Several methods and algorithms have been constructed to determine compression approaches to reduce data loading time and increase concurrency. It dynamically changes the file block size based on the compression ratio. Two dynamically selectable algorithms (tentative selection and predictive decision) have been studied to achieve an optimal I/O performance with a periodical compression algorithm features profiling and real-time system resource status monitoring. The authors focus on old versions of Hadoop (based on slots) supporting limited compression algorithms. Several studies aim to evaluate the influence of various configuration parameters on energy efficiency in the Hadoop framework. In [17], different energy models have been developed to predict MapReduce jobs' energy consumption. The job execution time and energy consumption have been minimized simultaneously by adjusting the data replication coefficient and data block size parameters. In [18], the authors stressed each part of MapReduce (map, shuffle, and reduce) and energy-related components (CPU, IO, and network) of machines. It is recommended to configure various parameters, such as data replication coefficient, file block size, number nodes, or type of nodes. A linear regression model has been designed to predict the energy consumption of MapReduce workloads. The experimental results indicate that significant energy savings can be achieved from accurate resource allocation and intelligent dynamic voltage and frequency scaling scheduling for computation-intensive applications [19]. The paper [20] presents our early work on modifying Hadoop to allow the scale-down of operational clusters. In [21], strategies are proposed for adjusting the degree of parallelism, network bandwidth, and power management functions in the HPC cluster for energy-efficient execution of map-reduce jobs. They also noted that increasing concurrency usually means energy efficiency or speed-up. The presented papers mainly explore either data compression or the influence of various configuration parameters on energy efficiency in the Hadoop/Spark frameworks to boost the Hadoop MapReduce job performance. This paper aims to present a system that selects optimal compression tools and tunes the

	compression factor to reach the best performance. The latest versions of Apache Hadoop and Spark's compression codecs were used to evaluate the benchmarks, tools, and applications.

WP Deliverables	**Planned delivery date**
Report	01/12/2021

Short description of the work package		
Work package No.	**2**	To develop a decision-making service
Work package leader		Hrachya Astsatryan
Researchers		Hrachya Astsatryan Aram Kocharyan Kristina Khudaverdyan
Duration		01/12/2021 - 30/04/2022

Targets	– To develop a decision-making service

Work package description	The suggested system allows to study the tradeoff, with compression between saving CPU and saving I/O, to evaluate the efficiency of Big data applications using Hadoop and Spark frameworks based on compression tools and tuning the compression factor. The performance optimization methodology allows users to explore and optimize Big data applications. A decision-making service sends the application type and the complexity to the service trading module through the REST API to select an optimal configuration. Several MapReduce types benchmarks, tools, and applications have been studied and implemented in the simulation module. As a distributed I/O benchmark tool, the TestDFSIO benchmark is used to stress test HDFS and determine cluster I/O speeds [22]. TestDFSIO is also essential to identify bottlenecks in networks and stress the hardware, OS, and Spark/Hadoop configuration on cluster nodes. TestDFSIO performs parallel reading and writing bulk data using separate Map tasks (or Spark jobs). The statistics are collected in the Reduce task to get a summary of HDFS throughput and average I/O.

WP Deliverables	**Planned delivery date**
Method	01/5/2022

Short description of the work package		
Work package No.	**3**	To validate the suggested method
Work package leader		Hrachya Astsatryan
Researchers		Hrachya Astsatryan Aram Kocharyan Kristina Khudaverdyan

Duration	01/12/2021 - 30/04/2022

Targets	– To validate the suggested method

Work package description	There are many MapReduce applications used to test both layers of HDFS and MapReduce. The terasort package is used to check the HDFS and MapReduce layers, consisting of TeraGen designed to generate data, Terasort to sort data, and TeraValidate to verify data sorting. TeraGen is designed to generate a large amount of data, which is the input to TeraSort. The size of the generated data and the output are the input arguments. Terasort sorts the data generated by TeraGen. TeraValidate checks the sorted TeraSort output. The input and output paths are the TeraSort and TeraValidate benchmarks arguments. The WordCount and LogAnalyzer are studied, as MapReduce applications [23]. The WordCount workload reads text files and counts how often words are found. The LogAnalyzer workload reads log file as an input, detects lines that match the entered regular expression, and outputs a report that informs if the keyword is present or not and if present how many times. The clustering data analysis technique divides the entire data into groups according to a similarity measure. K-means clustering is one of the simplest, powerful, and popular unsupervised machine learning algorithms in Data Science [24]. Parallel K-means MapReduce application has been used, allowing to manage large datasets finding distances between objects [25]. 1, 2, and 4 centroids have been identified for the experiments to allocate every data point to the nearest cluster. The input data is compressed using the compression algorithms. Three types of input data, seven compression algorithms and five workloads (TestDFSIO, TeraSort, WordCount, LogAnalyzer, K-means), are evaluated in Hadoop and Spark environments metrics to study environment and compression algorithms for different workloads.

WP Deliverables	**Planned delivery date**
Report	01/5/2022

Short description of the work package		
Work package No.	**4**	**COORDINATION, FINANCIAL MANAGEMENT, DISSEMINATION**
Work package leader		Hrachya Astsatryan
Researchers		Hrachya Astsatryan Areg Mickaelian Kristina Khudaverdyan
Duration		01/05/2021 - 30/04/2022

Targets	– Coordination – Financial management – Dissemination

Work package description	The aim of WP4 is to ensure the consistency of the overall resources used and the work performed, control the progress of the work, organise the production of meetings, resolve any project internal conflicts, and act as an interface for reporting on project progress. In order to successfully manage the project, a stable management structure is being proposed, with a clear set of roles and responsibilities of all actors; as well as clear set of procedures for information flow and other key management aspects.

WP Deliverables	**Planned delivery date**
Report	01/5/2022

6. Beneficiaries of the outcomes, impact, dissemination and sustainability

The major dissemination and exploitation aims are raising awareness on the Big Data performance tradeoff services offered and the benefits of Big Data management frameworks and platforms; promoting the widespread use of developed services and creating incentives to attract new services and users; performing science communication activities with a focus on the Resource management optimization research communities, to increase visibility within scientific, academic and technical circles; supporting sustainability and visibility of the research results even after the projects lifetime, particularly through the creation and the establishment of the SaaS astro solutions; and participating to and shaping scientific outreach activities on research issues.

7. Technical prerequisites for project realization

We use a hybrid research computing platform combing HPC with Grid and Cloud Computing based on ArmCluster HPC cluster, resource sharing ArmGrid Grid, and on-demand service provisioning federated cloud infrastructures. The infrastructure is operated by IIAP. The Armenian e-infrastructure is used for the studies and experiments, a complex national IT infrastructure consisting of networking, data, and distributed computing infrastructures [26].

8. Participants in the project

LEAD RESEARCHER – PROJECT RESPONSIBLE PERSON	
First and last name	Hrachya Astsatryan
Academic/Professional rank	Dr., HDR
Affiliation	Institute for Informatics and Automation Problems of the National Academy of Sciences of the Republic Armenia

Lead Researcher CV

Dr. Hrachya Astsatryan (m) is the head of the “Scientific Computing” Centre at IIAP. H. Astsatryan is a country delegate of the European Open Science Cloud, board member of Internet Society Armenian Chapter, member of H2020-ICT Committee, NCP of European Research Infrastructures, and a member of the Institute of Electrical and Electronics Engineers (IEEE). He holds a doctoral fellowship at the KFKI Research Institute for Particle and Nuclear Physics, Budapest, Hungary (2005-2006) and a postdoctoral fellowship at the Institute de Recherche en Informatique de Tulouse, Toulouse, France (2006-2007). In 2005, the President of the Republic of Armenia awarded him for the best work in Technical Sciences and Information Technologies. In 2020, he received an HDR (Habilitation a Diriger les Recherches) degree from the doctoral school of the Institut National Polytechnique de Toulouse, France. During his career, H. Astsatryan participated in many international conferences and workshops in Europe and the USA and many international Projects and Grants. He is an author of more than 80 papers in international scientific journals and conferences. He has more than twenty years of experience in High performance, cloud and scientific computing, Big Data, and data analytics.

Participation in international research projects

1. Reforming ARMDOCT - Doctoral Education in Armenia in line with Needs of Academia, Industry and Current EU Practices, EC Erasmus+ KA2 Nr. 609850-EPP-1-2019-1-AM-EPPKA2-CBHE-SP, 2020-2023, (role: participant).
2. Machine Learning to tackle weather and air pollution using DAtasets of satellite imagery and digiTAl models, Philip Morris Armenia, 2019-2020, (role: coordinator).
3. NI4OS-Europe: National Initiatives for Open Science in Europe, EC Horizon 2020 Nr. 857645, 2019 – 2021, (role: participant).
4. PROfiling the atmospheric Boundary layer at European scale (PROBE), EC Cost action (European Cooperation in Science and Technology) CA18235, 2019 – 2021, (role: participant).
5. Understanding and modeling compound climate and weather events (DAMOCLES), EC Cost action (European Cooperation in Science and Technology) CA17109, 2018 – 2022, (role: participant).
6. ADC4SD: Supporting Armenia in Building the Armenian Data Cube, Swiss State Secretariat for Education, Research and Innovation, 2018 – 2020, (role: participant).
7. Promoting Academia-Industry Alliances for R&D Through Collaborative and Open Innovation Platform (ALL4R&D), EC Erasmus+ KA2 Nr. 598719-EPP-1-2018-1-MK-EPPKA2-CBHE-JP, 2018 – 2021, (role: participant).

Lead researcher signature:

RESEARCHER	
First and last name	Aram Kocharyan
Academic/Professional rank	Dr.
Affiliation	Institute for Informatics and Automation Problems of the National Academy of Sciences of the Republic Armenia

Lead Researcher CV

Dr. Aram Knyazyan (m) is a member of the “Scientific Computing” Centre at IIAP. He got a joint PhD degree both in Armenia and in France. The research interests of A. Kocharyan include resource management, energy consumption minimisation, and cloud computing. His professional interests include service tradeoff and OS resource management and provisioning.

Participation in international research projects

1. NI4OS-Europe: National Initiatives for Open Science in Europe, EC Horizon 2020 Nr. 857645, 2019 – 2021, (role: participant).

Lead researcher signature:

RESEARCHER – PhD student	
First and last name	Kristina Khudaverdyan
Academic/Professional rank	PhD student
Affiliation	Assistant Professor at National Polytechnic University of Armenia

Researchers’ CV

Mrs. Kristina Khudaverdyan is Assistant Professor and part time PhD student at National Polytechnic University of Armenia.

Participation in international research projects

1. Change in Classroom: Promoting Innovative Teaching & Learning to Enhance Student Learning Experience in Eastern Partnership Countries (PRINTeL), Erasmus+ KA Nr. 585760-EPP-1- 2017-1-AMEPPKA2-CBHE-JP, 2017-2020, (role: participant)
2. Promoting Academia-Industry Alliances for R&D Through Collaborative and Open Innovation Platform (ALL4R&D), Erasmus+ KA2 Nr. 598719-EPP-1-2018-1-MK-EPPKA2-CBHE-JP, 2018-2021, (role: participant)
3. Transforming Architectural and Civil Engineering Education towards a Sustainable Model (TACEESM), Erasmus+ KA Nr. 2618883-EPP-1-2020-1-IT-EPPKA2-CBHE-JP, 2020-2023, (role: participant)

Researcher signature:

9. Project budget and cost

Costs	Amount (Euro)	Comment
Personnel costs	7200	1PM
Travel	600	Participation and presentation in one conference
Equipment	0	Using the facilities operated by NPUA/IIAP Cooperative R&D Unit, Open-Source Software Lab
Dissemination costs	200	Booklets, brochures
Total	**8000**	