AMSD-2019 IOP Publishing
Journal of Physics: Conference Series **1441** (2020) 012049 doi:10.1088/1742-6596/1441/1/012049

Research of efficiency of technologies for data processing in tasks of big data analysis

S E Teryoshkin[1] **and I N Yakovina**[2]

[1]ITMO University, Saint Petersburg, Russia
[2]Novosibirsk State Technical University, Novosibirsk, Russia

Abstract. The paper compares the performance of technologies for big data processing applied for anomaly detection in a database of a billing system. Experiments have been conducted to evaluate the performance for processing big data of two technologies such as Hadoop MapReduce and Apache Spark. The datasets and configuration of the applications used in the experiments are described. The experiment results are presented as a description of the relation between calculation time, data size and imbalance in datasets. Based on the obtained results, a decision making system for choosing a technology for processing data depending on the characteristics of the datasets is proposed.

1. Introduction

In the modern world, big data technologies have been spreading across the various fields and the number of tasks may be solved by using these technologies increases. The examples of big data tasks analysis are:

- Customer analytics
- Fraud prevention
- Social Media analysis
- Preventive support
- Campaign optimization
- Artificial Intelligence
- Detection of anomalies, etc.

This is not a full list of cases where big data can be applied. Each task requires a particular approach to achieve better performance. This approach depends on task features such as data size, network configuration, data properties, etc. Since we plan to consider implementation of segmented cluster computation in future research (i.e. a cluster is divided into segments and one segment stand for calculations based on a one technique), it is important to estimate the technique and their application to a particular data, conditions, etc. Thus, cluster resources can be employed more efficient. In this regard, we should develop a decision-making system for choosing the technologies based on the criteria.

2. Overview of methods and tools for big data processing

Today there are many tools in the field of big data analysis, both open source and proprietary. The most famous of them is the Apache Hadoop ecosystem, which includes many frameworks, libraries and other solutions. Some companies have their own infrastructure for working with big data such as AWS (Amazon), GCP (Google), Azure (Microsoft), YT (Yandex). In this paper, we consider only open source tools. Hadoop MapReduce [1, 2] (MR) and Apache Spark [3] are the most popular of them.

2.1. Hadoop MapReduce

The main advantages of MR-technology are high reliability and throughput, low resource utilization. The disadvantages of MR are the impossibility of processing streaming data, the complexity of the iterative operations [4].

MR involves data processing using key-value pairs. Data is read from the Hadoop distributed file system (HDFS) and the nodes are assigned for processing blocks of input files. The stage called «Map» is intended for preliminary processing and allocation of keys that used to group data. Then data is stored in Hard Drive Disks (HDD) and the «Shuffle» stage is perform to transfer data between nodes according to the keys (data with the identical keys is delivered to the same node). The data grouped by the key is processed at the final «Reduce» stage. The result of the MR-job is saved in HDFS.

Figure 1 shows the schematic diagram of the MR application with signed main stages.

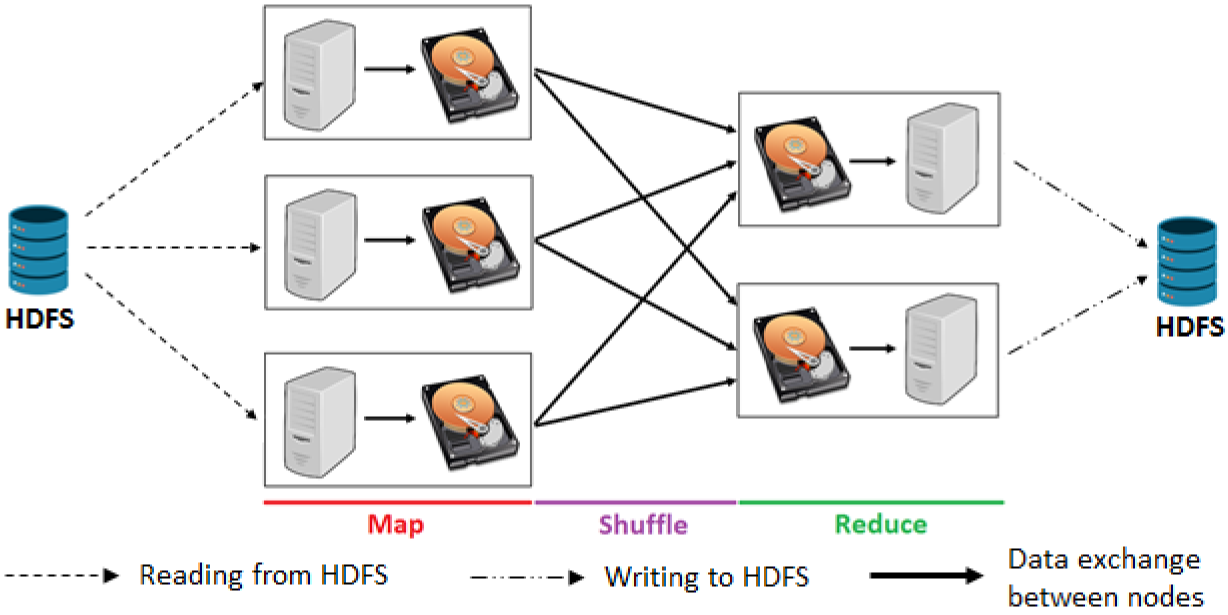

Figure 1. Schematic diagram of the MR application.

2.2.Apache Spark

The Spark advantages are high processing speed due to in-memory processing, good applicability for iterative algorithms, it has a set of different libraries (for working with graphs, machine learning, etc.), streaming processing. The disadvantages are high consumption of resources, increased requirements for network connection, lower reliability compared to MR, a strong performance decrease due to low RAM.

The schematic diagram of the Spark application is presented in figure 2. The important difference between Spark and MR is that nodes are not separated for the «Map» and «Reduce» stages. All nodes of Spark application perform the operations of each of the stages. Also, unlike in Hadoop MR, this technology stores intermediate data in RAM of nodes, thereby iterative tasks can be performed 10-100 times faster [5, 6]. If there is a shortage of RAM, some data can be written to the node's HDD and read from it when data is needed again. During the execution of Spark application, some operations can cause data exchanging between nodes, similar to the «Shuffle» stage in MR. At the end of the job, results can be stored in HDFS, transferred to the master node or used for further calculations.

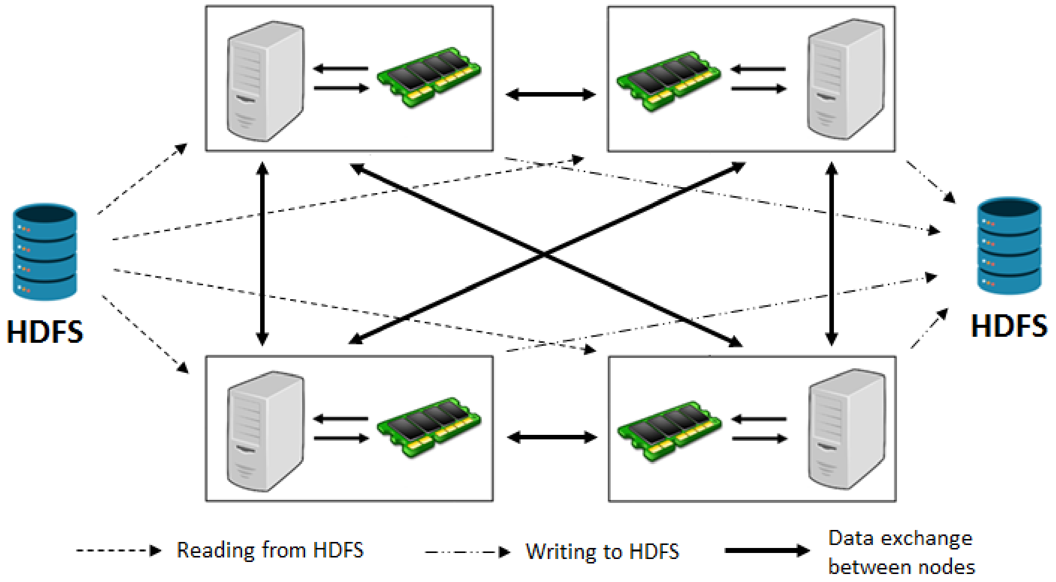

Figure 2. Schematic diagram of the Spark application.

Since these two technologies are quite common, many researches of comparing their performance [5, 6, 7] have been conducted. However, in most cases, the performance has been compared in iterative algorithms (e.g. graph algorithms, PageRank, K-Means, etc.), so the results cannot be applied to the current task. In some papers, the resource limit (RAM, CPU) for the execution of the application has been not specified.

Two features have been found during analysis of the database structure, existing process of delivering new data to database and development algorithms for detecting anomalies:

1. Data processing for anomaly detection have batch type
2. Algorithms are not iterative.

Based on these results, it has been decided to conduct experiments to identify the technology that provides with the minimum execution time under the existing conditions of anomaly detection in the billing system (BS) database (DB).

3. Task formalization

The technologies are chosen depending on their performance. The performance can be expressed as time to complete the task.

We formalize choosing the technology for non-iterative batch processing task as follows:

$$T = f(V, CoI, \overrightarrow{Conf}, Tech) \tag{1}$$

Explaining of these parameters:

- T – estimated time of execution duration.
- f – decision system.
- V – dataset size.
- CoI – the coefficient of dataset imbalance (CoI).

In machine learning, the term “imbalanced data” [8] is applied. The term stands for the data amount of one class exceeding the data amount of others classes. In “key-value” processing the similar term means than the value number of one key is significantly greater than the value number of other keys. Imbalanced datasets are known to cause increasing in execution time compared to balanced datasets [9].

In this task, this coefficient shows how many times the value number of the key with the maximum value amount is greater than the key with the median number of values. CoI can be expressed as follows:

$$I = \frac{H}{M} \tag{2}$$

In (2) H – the maximum number of values among the keys. M – the median number of values among the keys.

- $\overrightarrow{Conf}$ - configuration of computer network. It can be presented as a vector:

$$\overrightarrow{Conf} = (C, R, N) \tag{3}$$

In (3) C – the number of CPU cores (per node); R – the number of RAM, Gb (per node); N – the number of nodes.

- $Tech$ – technology of processing.

We consider the influence of the features on the performance for anomaly detection in DB of BS.

4. Experiment #1

The influence of the non-synthetic dataset size on performance has been explored. MR and Spark applications have the same algorithms for searching anomalies of two types in datasets. MR application has been developed using Java 8 and the libraries of Hadoop 3.1.0. Spark application has been developed by using Python 3.5 and the library pyspark.

For the experiment, 8 non-synthetic datasets varied in size have been chosen. Each dataset is represented by a set of database records and stored in 500 pieces of Sequence Files (SF) [10, 11] with block compression [11]. One record has a size equal to about 265 bytes. The properties of datasets are presented in table 1.

Table 1. Datasets properties.

№ of dataset	Count of records (millions)	SF size (Gb)	Raw size (Gb)
1	77.5	4	19.6
2	97.7	5	24.8
3	108.3	6	28.1
4	124.5	7	32.4
5	167.6	10	44.3
6	331.3	15	83.7
7	496	20	123.3
8	1361.9	44.4	320.1

These datasets contain information about services connected to the subscribers, connection parameters, service parameters and other technical information.

Maximum available resources for application execution are 420 Gb RAM and 128 CPU cores.

As the experiments have been conducted applying the productive cluster that already performs business tasks, the time with minimum cluster load has been chosen to carry out the experiments (load during experiments is less than 20%). Each application has been executed 5 times for each dataset. The result time is average of these 5 executions for the given dataset.

Based on the resource limit several Spark applications have been configured to compare their performance. Table 2 shows the configurations of the MR and Spark applications.

Table 2. Configurations of applications.

	Hadoop MR	Spark (1)	Spark (2)	Spark (3)
RAM per node (Gb)	4	4	6	12
CPU cores per node	1	1	2	4
Node count	105	105	64	32
Total RAM used (Gb)	420	420	384	384
Total CPU cores used	105	105	128	128

According to the experiment results, the dependence of execution time on data size (expressed in the record number) has been obtained. The dependence graph is shown in figure 3.

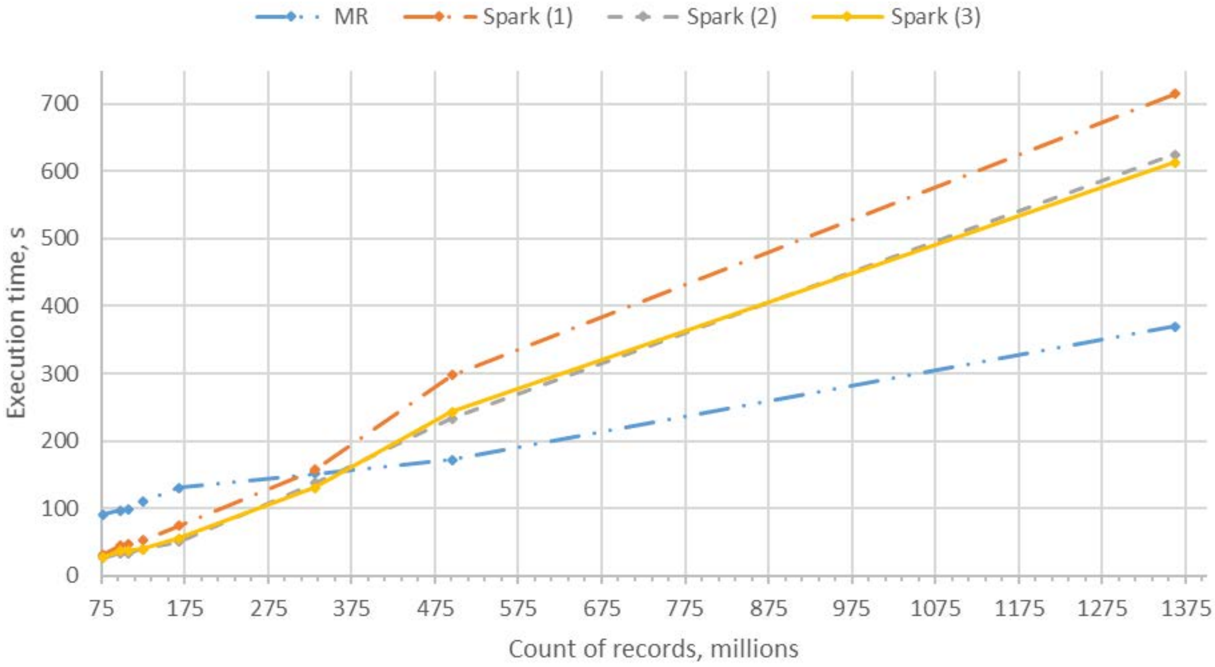

Figure 3. Execution time.

In addition, statistics on the shuffle data size (in Spark applications, values are the same) has been obtained. It is presented in table 3.

Table 3. Shuffle data size (Gb).

№ of dataset	1	2	3	4	5	6	7	8
MR	1.2	1.5	1.9	2	3	4.4	5.9	15.3
Spark	3.5	4.6	5.8	6.8	9.4	12.3	24.2	74.3

5. Analysis of the results obtained in experiment #1

According to the experiment results, we can conclude that the execution time of Spark and MR applications increases as the amount of data is increasing too. Shown in figure 3, the performance of the Spark applications with the second and third configurations is approximately identical, while the first configuration has a slightly worse result. We may assume that the number of CPU cores has a greater affect than RAM size for this task. Compared to MR, Spark applications work much faster with small datasets (less than 330 million records), but on for bigger datasets Spark shows a significant increase in processing time. For dataset №6 their performance is similar.

Possible reasons for this decrease in the Spark performance:

- A large amount of data arrives at the shuffle stage and as a result, this data is transmitted over the network between the nodes. E.g., for a dataset №7, the amount of shuffle data is 24.2 GB, while MR has only 5.9 GB. The differences are presented in the table 3. This can be caused by MR compressing data more effectively with compression codes than Spark. Using the Kryo serializer [12] for Spark applications has not improved the result.
- Using UDF (user defined functions). Recent research [13, 14] has revealed a significant decrease in the Spark performance when using Python UDF (compared to Scala UDF).
- Not full RAM of the node is allocated for data storage [15]. When there is a shortage of RAM, data is written to the HDD and read again to RAM when it is necessary. It causes a decrease in performance. This can be explained by a fact that the fewer number of nodes with the larger amount of RAM per node performs the task faster.
- Data imbalance. Some keys from datasets are known to have the large number of values. The influence of this fact to execution time is explored and described in the next experiment.

6. Experiment #2

In this experiment, the influence of imbalance data on the execution time of applications is studied. In order to eliminate influence of different configurations on execution time and to take into account resource limits, both MR and Spark applications are configured identically: 1 CPU core per node, 4 Gb RAM per node, 105 nodes.

In non-synthetic datasets, CoI can have any value, that is why we have decided to create synthetic datasets with CoI values: 1 (perfect balance), 100, 10.000, 40.000. The median number of the values is constant for all datasets.

In addition, we have changed the number of records: 175 millions (Spark has better performance than MR according to the results of the previous experiment), 330 millions (performance of MR and Spark are the same), 750 millions (MR is more efficient).

Data is stored as 500 pieces SF with Block compression (identical to non-synthetic datasets).

Thus, 12 synthetic datasets have been generated. Table 4 contains properties of these datasets (note that datasets with the same count of records and different CoI have the same SF size and real size).

Table 4. Properties of synthetic datasets.

№ of dataset	Count of records (millions)	SF size (Gb)	Raw size (Gb)
1	175	7.1	43.4
2	330	10.9	75.7
3	750	23.5	174.3

As in the previous experiment, each application has been executed 5 times and the result time is calculated as the average value of these execution durations.

The experiment results are presented in figure 5

Additionally, statistics (Δ MR and Δ Spark) on the growth rate (in percentage) of execution time with increasing CoI has been collected. It shows the amount of percentages by which execution time of imbalanced dataset is increased in compare with perfectly balanced dataset. Δ can be calculated as follows:

$$\Delta = \left(\frac{T_N}{T_1} - 1\right) \cdot 100\% \qquad (4)$$

In (4) T_1 – time of execution with CoI = 1, T_N – time of execution with CoI = N.

The statistics is presented in table 5.

Table 5. Statistics of Δ MR and Δ Spark.

CoI	Δ (%)					
	Dataset №1		Dataset №2		Dataset №3	
	MR	Spark	MR	Spark	MR	Spark
100	1.31	6.8	1.4	6.72	1.36	4.22
10000	5.25	16.35	4.89	14.16	5.8	9.1
40000	8.6	44.37	8.8	44.96	11.96	19.25

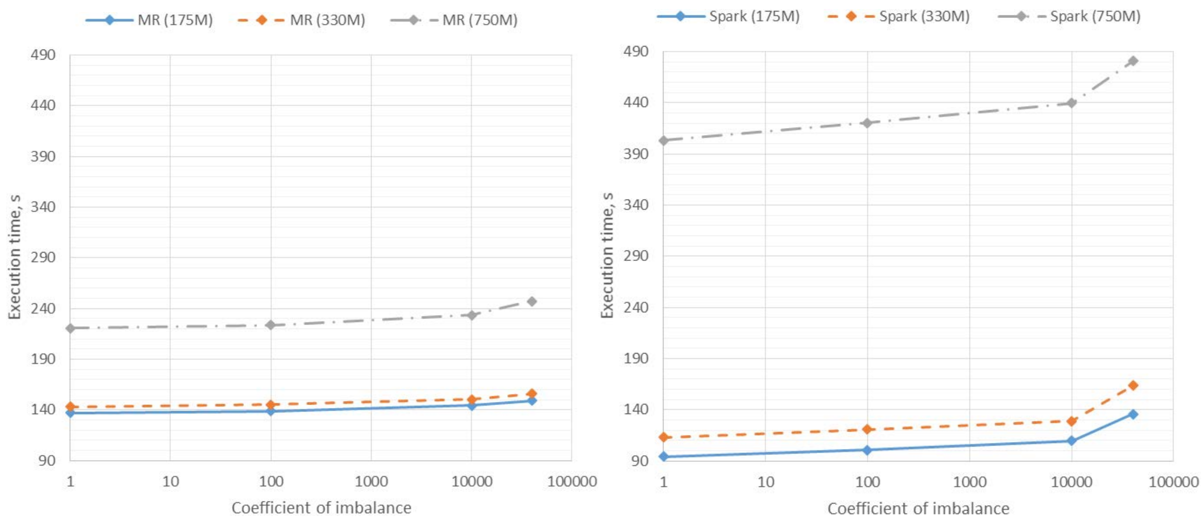

Figure 4. Influence of CoI on execution time

7. Analysis of the results obtained in experiment #2

We can observe that the execution time grows as CoI is increasing. However, MR is more tolerant to imbalanced data since Δ for MR is much smaller than Δ of Spark. E.g., with CoI = 40.000 execution time of MR grows only by 8.6% in dataset №1 but execution time of Spark has been increased by more than 44%. Imbalance does not has a great affect on small datasets. The performance of Spark is better than MR. However, if dataset is medium-sized (e.g., 330 million records) imbalanced data may significantly decrease the Spark performance. Even if datasets are perfectly balanced, the performance of Spark is smaller than MR, when processing large datasets (more than 330 million records).

8. Conclusion

Based on the conducted experiments, we can make following conclusions:

1. The most important feature for choosing technology for batch processing is data size. CoI is important for medium-sized datasets or if datasets are extremely imbalanced.
2. MR is much more tolerant to imbalanced data than Spark.
3. Spark application inflates the amount of shuffle data in compare to MR. For this reason, Spark more actively uses network resources.

We can classify dataset sizes into 3 categories: small (size is less than ~70 Gb), medium (70-100 Gb), large (more than 100 Gb). This classification is valid for configurations of computer networks identical to the configuration used during the experiments. According to the results obtained and analyzed during the experiments, we can develop a simple decision-making system. It is presented in table 6.

Table 6. Decision making system for configuration used in experiments
(1 CPU core, 4 Gb RAM, 105 nodes)

CoI	Estimated execution time, s					
	MR			Spark		
	Dataset size			Dataset size		
	Small	Medium	Large	Small	Medium	Large
1 –100	123 – 146	125 – 154	211 – 233	88 – 104	112 – 124	392 – 429
100 – 10000	128 – 153	141 – 158	216 – 242	96 – 116	118 – 134	411 – 454
10000 – 40000	135 – 157	146 – 161	226 – 254	102 – 147	126 – 168	432 – 505
>40000	>136	>151	>239	>122	>158	>467

By comparing of estimated execution time, we can select a technology with better performance.

Since the database of BS has more than 22 billion records (very large dataset) and data is imbalanced, we have decided to use Hadoop MapReduce to solve a task of detecting anomalies.

References

[1] Jeffrey Dean, Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA (2004), pp 137–150

[2] ‘Apache Hadoop – MapReduce Tutorial.’ [Online] Available: http://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. [Accessed: 30- May- 2019]

[3] ‘Apache Spark’ [Online]. Available: https://spark.apache.org/ [Accessed: 30- May- 2019)

[4] Anurag Sarkar, Abir Ghosh, Asoke Nath. MapReduce: A Comprehensive Study on Applications, Scope and Challenges. International Journal of Advance Research in Computer Science and Management 3(7), August 2015, pp 256–272

[5] Jeffrey Dean, Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA (2004), pp. 137-150.

[6] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. SOSP 2013. November 2013.

[7] Peter P. Nghiem. Best Trade-Off Point Method for Efficient Resource Provisioning in Spark. MDPI Algorithms 11(12), November 2018.

[8] Pasapitch Chujai, Kittipong Chomboon, Kedkarn Chaiyakhan, Kittisak Kerdprasop, Nittaya Kerdprasop. A Cluster Based Classification of Imbalanced Data with Overlapping Regions Between Classes. Proceedings of the International MultiConference of Engineers and Computer Scientists 2017 Vol I, IMECS 2017, March 15 - 17, 2017, Hong Kong, pp 353–358

[9] J. Wang, D. Crawl, I. Altintas, K. Tzoumas, V. Markl, "Comparison of distributed data-parallelization patterns for big data analysis: A bioinformatics case study", Proc. 4th Int. Workshop Data Intensive Comput. Clouds, 2013.

[10] 'SequenceFile – Hadoop Wiki' [Online]. Available: https://wiki.apache.org/hadoop/SequenceFile [Accessed: 30- May- .2019]
[11] 'Sequence File in Hadoop' [Online]. Available: https://netjs.blogspot.com/2018/06/sequence-file-in-hadoop.html [Accessed: 30- May- 2019]
[12] 'Esoteric Software. Kryo' [Online]. Available: https://github.com/EsotericSoftware/kryo [Accessed: 30- May- 2019]
[13] D. Bugayuchenko. 'Python vs. Scala for Apache Spark — expected benchmark with unexpected result' [Online]. Available: https://habr.com/ru/company/odnoklassniki/blog/443324/ [Accessed: 30- May- 2019]
[14] M.Merjan. 'Apache Spark: Scala vs. Java v. Python vs. R vs. SQL' [Online]. Available: https://mindfulmachines.io/blog/2018/6/apache-spark-scala-vs-java-v-python-vs-r-vs-sql26 [Accessed: 27.04.2019]
[15] 'Configuration – Spark 2.2.0 Documentation' [Online]. Available: https://spark.apache.org/docs/2.2.0/configuration.html#memory-management [Accessed: 30- May- 2019]