Processing Big Data in Field of Marketing Models Using Apache Spark

Tomáš Janečko(✉), Ondřej Grunt, Jan Plucar, Markéta Štáková, and Ivan Zelinka

Department of Computer Science, VŠB - Technical University of Ostrava, 17. listopadu 15, 708 33 Ostrava - Poruba, Czech Republic
{tomas.janecko,ondrej.grunt,jan.plucar,marketa.stakova.st, ivan.zelinka}@vsb.cz

Abstract. This paper presents an application of Apache Spark cluster for processing of selected marketing data. Based on available realistic data, Azure cluster is reused. Due to a complexity of the infrastructure and running environment, we used cloud resources for deploying and executing target simulations. Outputs then represents analysis of the links between the structured data and performance measurements.

Keywords: Apache Spark · Big data · Data analysis · Cloud computing · Statistics · Telemetry

1 Introduction

Current technological development enables financial institutions to collect extreme amounts of data every day. This situation poses new demands and requirements to sort these bulks of information and extract valuable contexts. Emerging links help corporations and advertisers to understand sequences of past actions, to learn from them and to extrapolate for future actions [1].

To [2] fulfill the computational requirements of massive data analysis, an efficient framework is essential to design, implement and manage the required pipelines and algorithms. In this regard, Apache Spark has emerged as a unified engine for large-scale data analysis across a variety of workloads. Following its advanced programming model, Apache Spark has been adopted as a fast and scalable framework in both academia and industry. It has become the most active big data open source project and one of the most active projects in the Apache Software Foundation.

2 Overview of Apache Spark

Apache Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that

V.H. Duy et al. (eds.), *AETA 2017 - Recent Advances in Electrical Engineering and Related Sciences: Theory and Application*, Lecture Notes in Electrical Engineering 465, https://doi.org/10.1007/978-3-319-69814-4_5

supports general execution graphs. It also supports a rich set of higher-level tools including *Spark SQL* for SQL and structured data processing, *MLib* for machine learning, *GraphX* for graph processing, and *Spark Streaming* [3]. Composition of the layers in system are presented in Fig. 1.

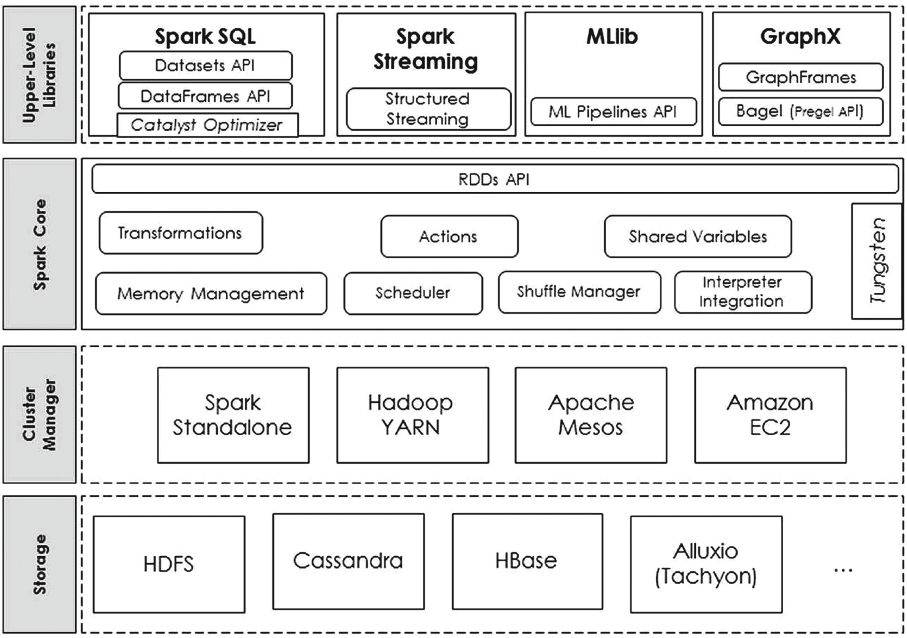


Fig. 1. Apache Spark overview

For the purposes of the experiment were used the following bindings and steps as follows:

1. Upper-Level Libraries
   - Spark SQL
   - Python programming language
2. Spark Core
   - RDDs API
   - Transactions
   - Actions
3. Cluster Manager
   - Hadoop YARN
4. Storage
   - HDFS

As the main query language and accessible interface was used Spark SQL that [4] provides DataFrames, which is a new data structure for structured (and semi-structured) data. DataFrames offers us the possibility of introducing SQL queries in the Spark programs. It provides SQL language support, with command-line interfaces and ODBC/JDBC controllers.

2.1 Cluster Manager

Spark uses master/worker architecture. There is a driver that talks to a single coordinator called master that manages workers in which executors run. The driver and the executors run in their own java processes. Physical machines are called hosts or nodes [5]. A detailed view of the cluster manager architecture is shown in Fig. 2.

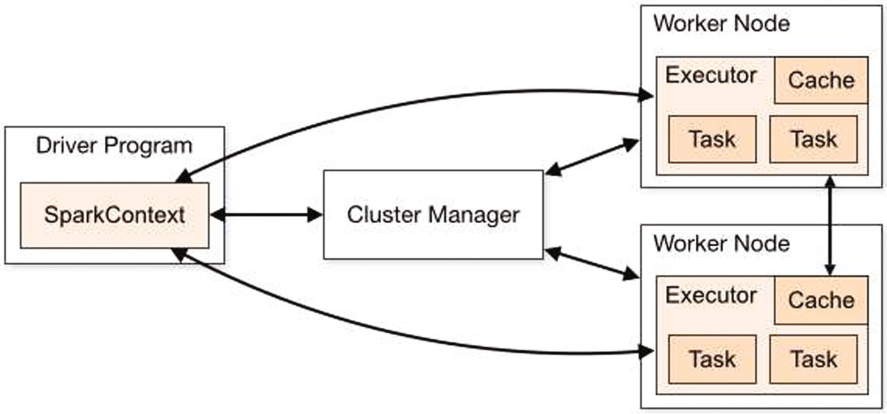


Fig. 2. Cluster overview

3 Experiment Design

Data used for deeper exploration [1] described customer relations with a single bank and consisted of multiple data columns each representing separate attribute related to individual customer ID (such as balance movement, number of ATM used, etc.). Values of these attributes were recorded monthly for each ID. However, most data columns still contained either missing or error values, preprocessing of the data was required.

Data were provided by PricewaterhouseCoopers Česká republika, s.r.o.

3.1 Preprocessing Data

Due to a complexity and large volume of obtained data, it was necessary to carry out the preprocessing phase. During this phase data was cleaned and normalized with the goal to reduce complexity of the data set.

Main operations of the preprocessing phase were as follows:

- selection of the most influential attributes
- selection of appropriate records
- handling of incomplete data columns

Originally, obtained data contained 3.3 billion atomic cells in total each representing a specific value. After all operations were performed size of the data set was reduced to 2.1 billion cells resulting into 36.65% reduction caused mainly due to the sparsity of some data columns.

3.2 Input Data Set

In our solution we have received several types of documents. The files had to be processed, categorized and passed through their semantic value. The original data structure was adopted in a form of comma-separated values. In the first series of assignments was necessary to select subsets that clearly interpreted the given set of data. Documents were separated into the next groups where each group contained the final set of files.

1. Overall group - Contains all received files
2. Live data group - Include files that doesn't represent dictionaries
3. Candidate group - Include not empty files that can be used for further analysis
4. Preprocessed group - Include files that have been subjected to the advanced analysis using Apache Spark

In the next Fig. 3, it is possible to see the partition of the total number of files per each group. From the collected data was observed that 57% of the files belongs to "Live data group" collection. This kind of data was obtained during the client usage of the system held by financial institution. From the other point of view 86% of the files contain values that can be reused for deeper analysis and from the measurement flows that 24% of the files requires the necessary preprocessing in the cloud computing infrastructure.

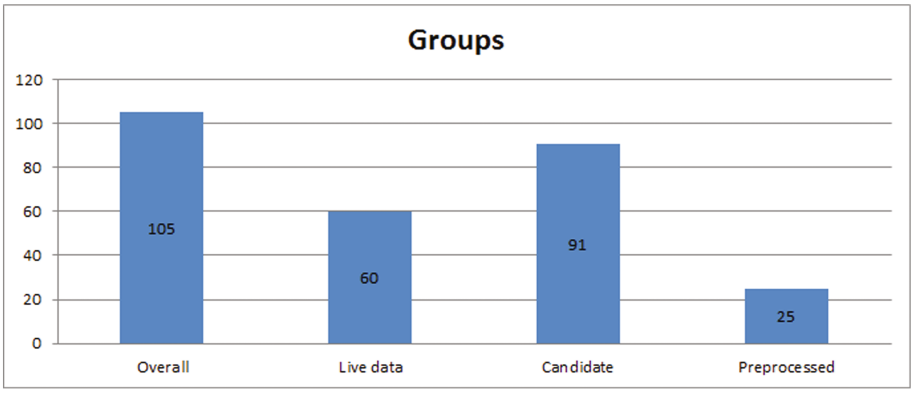


Fig. 3. Total count of files per group

3.3 Data Clustering

To obtain more detailed statistics the source files were clustered according to their meaning and content value. The resulted distribution was made to the following sections:

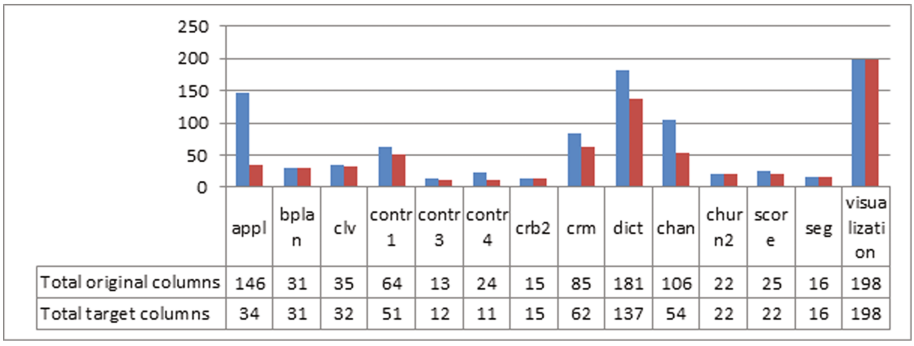
- Application - *appl*
- Business plan - *bplan*
- Customer lifetime value - *clv*
- Contract - *contr*
- Credit bureau - *crb2*
- Campaign - *crm*
- Dictionary - *dict*
- Channel - *chan*
- Churn - *churn2*
- Scoring - *score*
- Segment - *seg*
- Grouped visualizations - *visualization*

After the individual sections were identified we had to reduce the complexity of the data model by correctly selecting the appropriate features (e.g. customer attributes that are relevant to the amount of revenue generated by said customer) in next steps:

- selection of suitable attributes
- selection of appropriate records

In our example, attributes related to balance movement, offer success ratio and DTI (debt-to-income) were considered to be the most relevant ones.

The outputs of the selection operations are presented in Fig. 4. It shows the original total amount of attributes in the system before the preprocessing phase happened and their target total amount of attributes after the preprocessing phase was performed. The results are also clustered by sections.



	appl	bpla n	clv	contr 1	contr 3	contr 4	crb2	crm	dict	chan	chur n2	scor e	seg	visua lizati on
Total original columns	146	31	35	64	13	24	15	85	181	106	22	25	16	198
Total target columns	34	31	32	51	12	11	15	62	137	54	22	22	16	198

Fig. 4. Attributes per section

By examining we came to the conclusion that the largest drop in attributes was in the application section, because this one mostly contained data from previously discovered "Live data" group, which primarily includes records applied to the users and to their unambiguous identification. Additionally this loss of data is caused by the issues related to the data confidentiality, personal data protection and processing by third parties.

3.4 Cloud Computing

Preprocessing step for handling an incomplete data columns is introduced in this chapter. Partial algorithms and code snippets from the overall solution are mentioned later in this section. Brief survey of mandatory algorithm for determining ratio of "NULL" records per column was designed as followed.

The algorithm described below calculates the ratio:

```
from pyspark.sql.functions import col, sum

def count_null(c):
    """Use conversion between boolean and integer
    - False -> 0
    - True ->  1
   """
    return sum((col(c) == 'NULL').cast("integer")).alias(c)

exp = [(count_null(c) / totalCount).alias(c) for c in df.columns]
collectedRDD = df.agg(*exp).cache();
collectedRDD.show()
```

After the previous step was completed there was necessary to specify the ratio condition to match the sufficient features. For our purposes the variable was set to 80%. This variable describes surface for dropping features and is applied in next part of our algorithm.

```
columnNullRatioDict = collectedRDD.first().asDict()
columnNamesToFilter = []

for item in columnNullRatioDict.items():
    value = item[1]
    if value is None:
        value = 0

    if(value >= 0.8):
        columnNamesToFilter.append(item[0])

columnNamesToFilter
```

All of the above mentioned steps belongs to the transformations and the operation itself is executed on Apache Spark from the next code.

```
filteredDF = df.select([column for column in df.columns
if column not in columnNamesToFilter])
```

One of the main ideas and advantages of Apache Spark to cluster that big data is that we can smoothly consume larger volumes of workloads. As well we can reapply part of algorithms mentioned above to produce live outputs. These results are achieved through the use of in-memory computing instead of using file storages, as is usually the case in other systems.

3.5 Report Outputs

If we look at the results of preprocessing phase from the overall ratio, also here is the data cleansing a major factor. The ratio of the filtered attributes is described in Table 1. The results are compared with the standard environment and cloud environment.

- Filtered attributes - the ratio of all removed attributes across the system
- Filtered processing attributes - the ratio of all dropped attributes in files that have been preprocessed in Apache Spark

Table 1. Attributes filter ratio

Attributes	Filter ratio
Filtered attributes	28.21%
Filtered processing attributes	35.59%

The same steps that were applied to the attributes (vertical axis) were also performed within individual records (horizontal axis). This has resulted to the reduction of entities that would ultimately distort the calculated models. The ratio of filtered entities is described in Table 2.

- Filtered rows - the ratio of all deleted records across the system
- Filtered processing rows - the ratio of all deleted records in files that have been preprocessed in Apache Spark

Table 2. Rows filter ratio

Rows	Filter ratio
Filtered rows	15.90%
Filtered processing rows	20.90%

The resulting view of the entire dataset is presented in Table 3.

- Total original cells - total number of cells in the system
- Total cells - total number of cells in the system after the preprocessing phase
- Cell loss ratio - ratio of all filtered cells

Table 3. Total cells ratio

Cells	Cells ratio
Total original cells	3303352118
Total cells	2092834204
Cells loss ratio	36.65%

In summary, we have reached the data structure, which is reduced by about 1.2 billion cells, while a significant part is still formed by the dictionaries.

3.6 Performance

During all operations, the duration of individual operations and system performance was recorded. The technical specification of the infrastructure that was used is as follows:

- Apache Spark - version 2.1.0 (Linux)
- Master - 2 nodes
- Slave - 4 nodes

Master nodes are machines that are memory optimized due to the high demands on the synchronization of the entire process. These individual machines can be interpreted in Fig. 2. In the detailed description of each machine, the configurations were as follows:

- Mater node - 4 cores, 28 GB RAM, 200 GB SSD
- Slave node - 8 cores, 28 GB RAM, 400 GB SSD

The overall robustness of the architecture is presented in Table 4.

Table 4. Infrastructure resources overview

Total cores	Total memory (GB)	Total storage (TB)
40	168	2

The results of the measurements and the average values obtained are presented in Table 5. The table aggregates metrics related to attributes, records and also a conversion of performance to 1 million records. The measurements obtained are output from the Apache Spark platform.

Table 5. Performance metrics

Average processing time per column/file (s)	Average processing time per column/mil. rows (s)	Average rows processing time (s)
0.84	0.21	0.02

4 Results

To uncover the nature and patterns in the data, it was necessary to analyze the links between the structured files and to perform operations that work with the raw data structure on the basis of the discovered relations.

Upon closer measurement, it was found that the dataset contains a total of approximately 3.3 billion atomic cells representing a specific value of the attribute in the system. From the above findings, it has become clear that the *Big Data Ecosystem* was essential for further data processing. For our solution was selected *Apache Spark* technology stack.

Using Apache Spark helped us to get deeper insight into the patterns hidden behind the data. It was necessary to get the input values into a uniform form. Because as was expected, a lot of source files contained incomplete data or the data was in the wrong format.

The use of cloud technology accelerated the process and helped straighten inconsistencies in the data structures, that the input files contained. Thanks to use the Apache Spark environment, the whole process could be parallelized and applied over such a large-scale solution, which was not feasible from a performance point of view by using a local resources.

The task of preprocessing data was also necessary in terms of following investigations. This preprocessed data served as the basis for developing a client revenue computing program. When MDP (Markov Decision Processes) are used for modeling of selected marketing process to maximize return value of the customer [1].

5 Conclusion

In this paper, Apache Spark was used for data processing. From the above findings, we came to the conclusion that the main problem in the data was their confidentiality. This whole area is therefore based on a cloud data protection solution. One possible solution to this problem is to use homomorphic encryption. A fully homomorphic scheme can be used in many applications, and one of these can be, for example, the implementation of secure cloud computing [6].

This problem was resolved by Craig Gentry in his dissertation, describing the use of a fully homomorphic encryption scheme (FHE). Thanks to Gentry's technology, the analysis of encrypted information can produce as detailed results as if the original data were fully readable to all [7].

For further research, it's crucial to find solution how to use FHE in field of cloud computing. For better continuous development and improvement of the results, we also recommend ability to dynamically scale out running environment.

Acknowledgement. The following grants are acknowledged for the financial support provided for this research: Grant Agency of the Czech Republic - GACR P103/15/06700S, Grant of SGS No. SGS 2017/134, VSB-Technical University of Ostrava. The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science - LQ1602. Research presented in this article was conducted in collaboration with PricewaterhouseCoopers Česká republika, s.r.o.

References

1. Grunt, O., Plucar, J., Štáková, M., Janečko, T., Zelinka, I.: An approach to customer behavior modeling using Markov decision process. MENDEL - Soft Comput. J. **23**(1) (2017). ISSN 1803-3814
2. Salloum, S., Dautov, R., Chen, X., Peng, P.X., Huang, J.Z.: Big data analytics on Apache Spark (2016). ISSN 2364-4168. https://doi.org/10.1007/s41060-016-0027-9
3. Zaharia, M.: Apache Software Foundation, UC Berkeley AMPLab, Databricks (2017). http://spark.apache.org/docs/latest/. Accessed 3 May 2017
4. García-Gil, D., Ramírez-Gallego, S., García, S., Herrera, F.: A comparison on scalability for batch big data processing on Apache Spark and Apache Flink (2017). ISSN 2058-6345. https://doi.org/10.1186/s41044-016-0020-2
5. Laskowski, J.: Mastering Apache Spark 2 (2017). https://www.gitbook.com/book/jaceklaskowski/mastering-apache-spark/details. Accessed 20 June 2017
6. Pejlová, A.: Homomorphic encryption schemes. Charles University, Faculty of Mathematics and Physics (2013)
7. Gentry, C.: A fully homomorphic encryption scheme. In: Symposium on the Theory of Computing (STOC), pp. 169–178 (2009)
8. Microsoft Corporation: Azure Windows VM (2017). https://docs.microsoft.com/en-us/azure/virtual-machines/windows/sizes-general. Accessed 20 June 2017
9. Stehlé, D., Steinfeld, R.: Advances in Cryptology - ASIACRYPT 2010: Proceedings of 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, 5–9 December 2010. ISBN 978-3-642-17373-8
10. Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J., Ghodsi, A., Gonzalez, J., Shenker, S., Stoica, I.: Apache Spark: a unified engine for big data processing. Commun. ACM **59**(11), 56–65 (2016). https://doi.org/10.1145/2934664
11. Estrada, R., Ruiz, I.: Big Data SMACK A Guide to Apache Spark, Mesos, Akka, Cassandra, and Kafka (2016). ISBN 978-1-4842-2175-4