Collecting and Characterizing Distributed Machine Learning Workloads

Yasmine Djebrouni, Sara Bouchenak, Khalid Benabdeslem

Compas'2020 : Parallélisme / Architecture/ Système/ Temps Réel
MILC - Lyon, France, du 30 juin au 3 juillet 2020

Collecting and Characterizing Distributed Machine Learning Workloads

Yasmine Djebrouni[1,2], Sara Bouchenak[1], and Khalid Benabdeslem[1]

[1]INSA Lyon – LIRIS, France
[2]ESI (Ecole nationale Supérieure d'Informatique) – Algiers, Algeria

Abstract
Machine learning is a key for transforming data into actionable knowledge. The rapid increase in the amount of analyzed data forced the switch to distributed ML platforms. However, the complexity of such platforms is overwhelming for uninitiated users, who may not understand the trade-offs and the challenges of parameterizing such systems to achieve good performance. In order to better analyze and understand ML workloads running on ML distributed platforms, we conducted extensive experiments with various ML methods and real-world datasets, and collected the execution traces of these distributed ML workloads, that represent a total of 12 GB of traces and tens of millions of data records. We then provide a statistical analysis of the collected traces, and illustrate through a use case how different ML workloads' are characterized and their needs identified.

Keywords : Distributed Machine Learning, ML Workload Characterization, Trace Collection

1. Introduction

Nowadays, Machine learning (ML) is widely used in many application domains. Originally, ML workloads were designed to be executed on a single machine using libraries such as Weka, R or Scikit-learn [11, 18], mainly because the processed datasets could fit on a single machine and the computing capacity was sufficient. However, this situation has changed with the rapid growth of data sensing and collection technologies in recent years [10]. This leads to distributed ML platforms such as Spark MLlib [14], or Tensorflow [1]. Despite the benefits of distributed ML platforms, among which faster executions and the larger scale datasets, they also introduce a much more complex execution environment. On a single machine, users had a better control of the underlying infrastructure and a better understanding of their ML workload behavior, which is no longer the case in a cluster. Here, users fail to grasp the implications of distributed executions and do not necessarily understand the behavior of their distributed ML workloads. Thus, it is crucial to deeply analyze and characterize ML workloads running on distributed platforms. In this paper, we precisely address this question.
The paper makes the following contributions: (i) We conduct extensive experiments with various distributed ML workloads, made of 9 ML methods and 7 real-world datasets, running on Spark/ MLlib. (ii) We collect execution traces of these distributed ML workloads consisting of a total of 12 GB of traces and tens of millions of data records; these traces will be made publicly available to other researchers and practitioners. (iii) We provide a deep statistical analysis of the

collected ML workload traces, and illustrate through a use case how different ML workloads' are characterized and their needs identified.
The remainder of the paper is organized as follows. Section 2 presents the necessary background. Section 3 describes the methodology that we followed to conduct extensive experiments and collect distributed ML workloads traces, and section 4 analyzes the actual collected traces. Section 5 presents some related works. Finally, Section 6 draws our conclusions.

2. Brief Overview of Spark and MLlib

Apache Spark is a general-purpose, in-memory, fault-tolerant cluster computing framework [20]. It is designed to run different kinds of distributed analytics or Big Data jobs. The main abstraction in Spark is Resilient Distributed Datasets, or RDDs for short, which are basically an immutable distributed collection of data structures partitioned across many worker nodes. A machine learning workload (ML algorithm and dataset) on Spark is split into jobs. Each job in Spark consists of stages, with between these stages that usually represent shuffle boundaries. A stage is composed of a set of tasks running in parallel, each on a subset of the input data.
Spark MLlib is a machine learning library that runs on top of Spark [14]. It uses either RDDs or dataframes, which provide extra layers for usability, functionality and performance. ML Lib provides a wide range of out-of-the-box machine learning algorithms for regression, classification and clustering. ML Lib also provides the required tools and methods for evaluating the model's performance.

3. Methodology for Collecting Distributed ML Workload Traces

We deployed Spark on a cluster of machines, and conducted extensive experiments with MLlib's distributed ML algorithms and various real-world datasets. In the following, we describe how we collected traces of these distributed executions, at three layers: the ML application layer, the distributed platform layer, and the infrastructure layer.

3.1. ML Application Layer

At this layer, we consider 9 widely used distributed ML algorithms provided by MLlib [14], for regression, clustering and classification as described in Table 1. We also consider 7 real-world datasets [6, 7, 9, 17, 19, 5, 15, 16], with different properties in terms of number of data records (from hundreds to millions), number of data features (from few to thousands) and data size, as shown in Table 2. We call a *ML workload* the execution of a given distributed ML algorithm with a given dataset. The objective of this study is to cover representative ML workloads that exibit different behaviors and, thus, to collect heterogeneous execution traces of distributed ML workloads.
For each ML algorithm and each dataset, we run the training phase and the prediction phase. Each dataset is split into two subsets, 80% of the overall dataset is used for the training, and 20% of the overall dataset is used for conducting prediction requests. Each dataset is used with each ML algorithm, except for GMM which is weak against high dimensionality and could not be run properly with D_{DF} and D_{DR}. At this layer, we measure for each ML workload several applicative metrics, a total of 15 different metrics, among which the accuracy of the learned ML model for that workload, the execution time of the training phase of that ML workload, the prediction throughput for that ML workload (in #prediction requests per unit of time), etc.

Category	Algorithm
Clustering	KM (K-Means)
	BKM (Bisecting K-Means)
	GMM (Gaussian Mixture Model)
Classification	DT (Decision Tree)
	MLP (Multilayer Perceptron)
	BLR (Binomial Logistic Regression)
Regression	LR (Linear Regression)
	RFR (Random Forest Regressor)
	GBT (Gradient-Boosted Tree)

Table 1: ML algorithms

Dataset	#Records	#Features	Size	Description
D_{DF}	606	6 400	19.9 MB	Driving Face Images: images sequences of subjects while driving in real scenarios [7].
D_{GS}	13 910	129	40.3 MB	Gas Sensors: measurements from 16 chemical sensors utilized in a discrimination task of 6 gases at various levels of concentrations [19].
D_{SS}	94 514	13	10.2 MB	Smartphones' and smartwatches' Wi-Fi and geo-magnetic sensor data [6].
D_{DR}	215 063	6	111.6 MB	Drug Review: patient reviews on specific drugs [9].
D_{HG}	11 000 000	28	7.5 GB	Higgs: data to distinguish between a signal process which produces Higgs bosons and a background process which does not [4].
D_{AT}	1 087 140	47	275.3 MB	IOT Attacks: real modern normal activities and synthetic contemporary attack behaviours [15].
D_{GC}	2 012 242	6	322.8 MB	Google Cluster Jobs: traces from Google cluster management software and systems [16].

Table 2: ML datasets

3.2. Distributed Platform Layer

We use Apache Spark as a distributed platform, combined with Hadoop distributed storage system. Spark has several configuration parameters, among which 12 most important parameters [3]. Table 3 describes this parameters such as the number of cores assigned to a task executor in Spark (EXEC_COR, i.e., executor.cores), the amount of memory for a tsak executor (EXEC_MEM, i.e., executor.memory). Spark configuration parameters have default values, that can be modified by the operator of the Spark distributed platform to better suit a given workload. In order to capture various configurations behaviors in our study, we consider several values of Spark configuration parameters, in addition to their default values.

In our study, for each ML workload training phase and each ML workload prediction phase at the application layer, we first run these phases with the default values of Spark configuration parameters. In addition, for each considered configuration parameter, we vary the values of that parameter (while keeping the other parameters with their default values), and run the training phase and prediction phase of the ML workload. These values are described in Table 3. Their combinations result in a total of 61 configuration combinations. At this layer, for each ML workload run with each combination of Spark configuration values, we measure for the training phase and the prediction phase several platform metrics (a total of 35 metrics), such as Spark task execution times, data serialization times, garbage collection times, the number of blocks written to memory, the result size, etc.

3.3. Distributed Infrastructure Layer

At this layer, and for each node involved in the distributed platform, and for each distributed ML workload running with a set of Spark configuration values, we measure low level system metrics. This is done for both the training phase and the prediction phase of the ML workload. These system metrics include CPU usage, memory usage, disk access, network bandwidth consumption.

4. Experimental Results

4.1. Experimental Setup

Our experiments were conducted on Grid'5000's Taurus cluster. Workloads related to D_{DF}, D_{GS}, D_{SS}, D_{DR} or D_{HG} datasets were executed on 5 nodes, and workloads related to D_{AT} and

Name	Values (default value in brackets)	Description
EXEC_COR (executor.cores)	1, 2, 3, 5, (4), 6, 7, 8, 9, 10	Number of cores per executor.
EXEC_MEM (executor.memory)	2 GB, (5 GB), 10 GB, 15 GB, 20 GB, 25 GB, 30 GB, 50 GB, 70 GB, 100 GB	Amount of memory per executor.
LOC_WAIT (locality.wait)	1ms, (3s), 10ms, 50ms, 100ms, 500ms, 1s, 2s, 5s, 10s	How long to wait to launch a data-local task before giving up and launching it on a less-local node.
SER (serializer)	(Java), Kryo	Data serialization mechanism.
COMP_CODEC (io.compression.codec)	(snappy), lz4	Codec to compress internal data such as RDD partitions, shuffle outputs, etc.
MAX_SIZ_INF (reducer.maxSizeInFlight)	6 MB, 12 MB, 24 MB, (48 MB), 72 MB, 96 MB, 128 MB, 192 MB, 256 MB, 512 MB	Maximum size of map outputs to fetch simultaneously from reduce tasks.
PD_BUFS (shuffle.io.preferDirectBufs)	(true), false	Wether to use off-heap buffers to reduce garbage collection during shuffle and cache block transfer.
SHF_SPL_COMP (shuffle.spill.compress)	(true), false	Wether to compress data spilled during shuffles.
SHF_COMPR (shuffle.compress)	true, (false)	Wether to compress map output files.
RDD_COMP (rdd.compress)	true, (false)	Wether to compress serialized RDD partitions.
SFL_BUF (shuffle.file.buffer)	4 KB, 8 KB, 16 KB, (32 KB), 48 KB, 64 KB, 128 KB, 192 KB, 256 KB, 512 KB	Size of in-memory buffer of shuffle file output stream.
STR_MEM (storage.memoryFraction)	5%, 10%, 20%, 30%, 40%, 50%, (60%), 70%, 80%, 90%	Fraction of Java heap to use for Spark's memory cache.

Table 3: Spark configuration parameters

D_{GC} datasets run on 10 nodes because they are more compute intensive. All the nodes have the following hardware configuration: 2 x Intel Xeon E5-2630 CPU, 8 cores/CPU, 128 GB RAM, 600 GB HDD, 2 X 10 Gbps Ethernet. We used Spark MLlib 2.4.0, Hadoop / HDFS 2.7.7, and LACAN to deploy the distributed ML experiments [8]. Each run was repeated three times.

4.2. Overview of Collected Traces

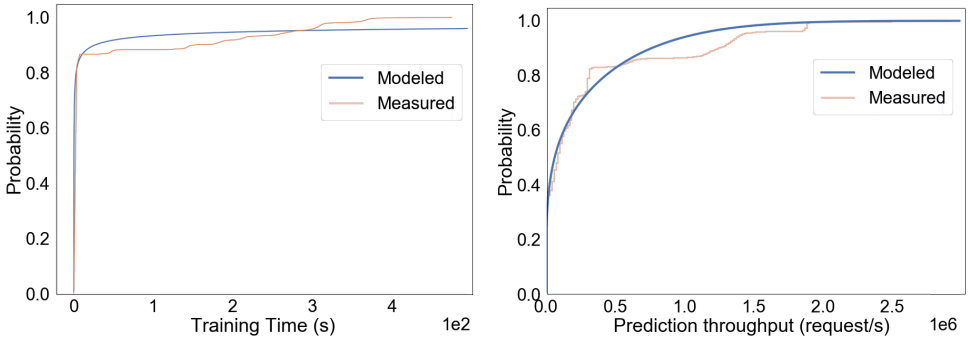
As described in Section 3, we collected traces at three layers, namely the ML application layer, the distributed platform layer, and the infrastructure layer. This results in a total of 12 GB of data, consisting of three sets of data: ML application-level traces that consist of more than 11 thousand records, platform-level traces that consist of more than 37 million records, and infrastructure-level traces which contain more than 8 million records. These traces are briefly described in Table 4. In the following and due to space limitation, we will present a subset of the statistical analysis and the behavioral analysis of ML application-level traces.

Trace type	#Records	#Features	Size	Description
Application traces	11 036	15	1.4 GB	Application metrics such as ML model accuracy, ML training times, prediction times, etc.
Platform traces	37 066 879	32	9 GB	Spark platform metrics such as tasks execution times, tasks deserialisation and serialisation times, tasks results sizes, etc..
Infrastructure traces	8 361 243	6	1.6 GB	System metrics such as CPU usage, memory usage, etc.
Total	**45 439 158**	–	**12 GB**	

Table 4: Description of collected traces

4.3. Statistical Analysis of Collected Traces

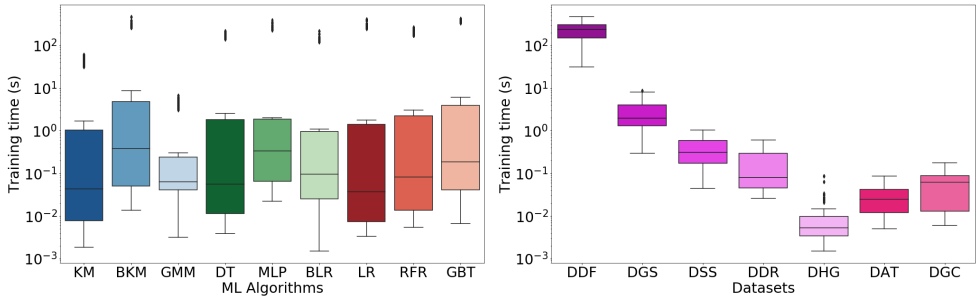
Based on all ML application-level traces that we collected (c.f., Table 4), we conducted a statistical analysis on the different measured metrics. In Figure 1b, we present the CDF of all measured ML training times of all considered distributed ML workloads and all considered Spark platform configurations. Here, since the application datasets used in the ML workloads have different sizes (c.f., Table 2), the training times were normalized for datasets of 1 000 records. We fitted a total of 88 statistical distribution functions (.e.g, normal, chi, gamma, etc.), and chose the one that minimizes the Kolmogorov-Smirnov distance. We found that the distribution function that captures the best the normalized training times is the F-distribution with parameters $d_1 = 1.28$ and $d_2 = 0.63$. Similarly, we present in Figure 1b the CDF of all measured ML prediction throughputs of all considered distributed ML workloads and all considered Spark platform configurations. We observe that ML prediction throughputs follow a chi distribution with parameter $k = 0.25$.



(a) CDF of training times (b) CDF of prediction throughput

Figure 1: Overall statistical analysis

In order to get more insight into the statistical analysis of the collected traces, we present in Figure 2 the dispersion of normalized ML training times, on the one hand, per ML algorithm (c.f., Figure 2a) and, on the other hand, per dataset (c.f., Figure 2b). Note the logarithmic scale here. From Figure 2a, we observe that the training time spread is high for each algorithm compared to the spread per dataset. It means that for a given ML algorithm, training times have a wide range, they are probably impacted by the other considered variables such as the dataset and the platform configuration. Figure 2b presents the results per dataset, we can see that there is much more heterogeneity between the datasets than between ML algorithms. In other words, ML training times are more influenced by the characteristics of the dataset (e.g., #features) than by the ML algorithm.



(a) Training times per ML algorithm (b) Training times per dataset

Figure 2: Statistical analysis of training times for different ML algorithms and datasets

4.4. Use Case: One Size Does Not Fit All

In the following, we consider the following question: What is the actual impact of the different Spark platform configuration parameters on the performance of distributed ML workloads? In other words, which Spark parameters have a *high impact* on the performance of a distributed ML workload? Which ones have a *medium impact*? And which ones have a *low impact*? In the following, we answer this question for the measured ML prediction throughput in the collected traces. Figure 3 presents a matrix where the lowest row (in pink) represents the different datasets, respectively D_{DF} (for 8 successive columns), D_{GS} (for 9 successive columns), D_{SS}, D_{DR}, D_{HG}, D_{AT}, D_{GC} (c.f., Table 2). The second lowest row in the matrix represents the different ML algorithms, respectively KM, BKM and GMM (in blue), DT, MLP and BLR (in green), LR, RFR and GBT (in red). The combination of these two lowest rows represents the different ML workloads. The highest rows of the matrix represent, each, a Spark configuration

parameter. A cell in the highest rows of the matrix may have one of the three colors: (i) orange if the corresponding configuration parameter has a high impact on performance (here, ML prediction throughput), (ii) dark yellow if the corresponding parameter has a medium impact on performance, and (iii) light yellow if the parameter has a low impact on performance. For instance, the first cell in the first row shows that EXEC_COR Spark parameter has a high impact on the ML workload that consists of KM ML algorithm running on the D_{DF} dataset. Here, we consider that a parameter has: a high impact if it improves performance by at least 20%, a medium impact if it improves performance by at least 10% but lower than 20%, and a low impact otherwise. We can see that the ML workloads involving the D_{HG} dataset are highly impacted by one Spark configuration parameter that is EXEC_MEM, And the ML workloads involving the D_{GC} dataset highly impacted by up two parameters, namely EXEC_COR and EXEC_MEM. The other workloads are highly impacted by more parameters. Such an analysis can guide the operator of the distributed ML platform to better choose the parameters to configure depending on the workloads.

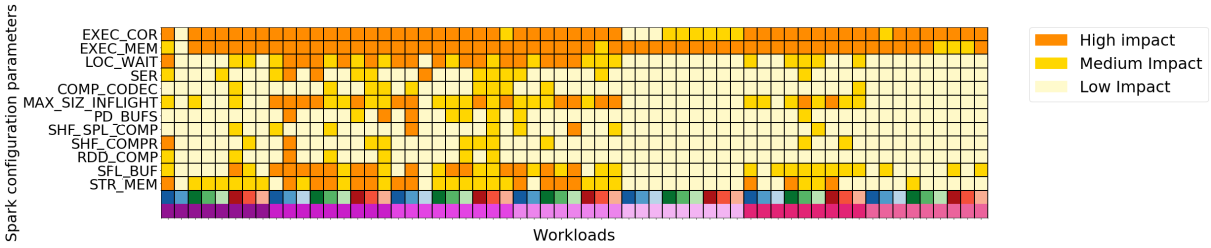


Figure 3: Impact of configurations on prediction throughput of different ML workloads

5. Related Work

Some tools were proposed for benchmarking ML worloads running on a single node [2, 13]. Here, we are interested in distributed ML workloads running on several nodes. Tools for ML workload profiling on Spark and Tensorflow have been proposed. Spark-Bench is a tool for benchmarking and simulating Spark jobs in a distributed environment [12]. Spark-Bench allows to run ML workloads on Spark, and collects high-level metrics such as CPU and memory usage, disk and network bandwidths. TensorBoard is a tool for inspecting and understanding TensorFlow runs and graphs [1]. TensorBoard provides several graphs to help users understand the behavior of the workloads during runs. Although these tools provide measured metrics, none of them provides a mean to analyze and interpret the produced results. Finally, in a previous work, we proposed a software tool to automate the deployment of distributed ML workloads on a cluster of machines [8]. In the present study, we use that tool to conduct extensive experiments with various distributed ML workloads. From the collected ML workload execution traces, we propose a deep statistical analysis and behavioral analysis.

6. Conclusion

In this paper, we first presented extensive execution traces that we collected from various distributed ML workloads running on Spark/ ML lib. These traces consist of tens of millions of data records, for a total of 12 GB, and , that will be made publicly available to other researchers and practitioners. In this paper, we also provided a statistical analysis of the collected ML workload traces, and illustrated through a use case how different ML workloads are characterized and their needs identified. Future work includes collection of execution traces of distributed deep neural networks.

Bibliographie

1. Abadi (M.), Barham (P.), Chen (J.), Chen (Z.), Davis (A.), Dean (J.), Devin (M.), Ghemawat (S.), Irving (G.), Isard (M.), Kudlur (M.), Levenberg (J.), Monga (R.), Moore (S.), Murray (D. G.), Steiner (B.), Tucker (P.), Vasudevan (V.), Warden (P.), Wicke (M.), Yu (Y.) et Zheng (X.). – Tensorflow: A system for large-scale machine learning. – In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, *Osdi'16*, Osdi'16, pp. 265–283, Berkeley, CA, USA, 2016. USENIX Association.
2. Adolf (R.), Rama (S.), Reagen (B.), Wei (G.) et Brooks (D. M.). – Fathom: Reference workloads for modern deep learning methods. *CoRR*, vol. abs/1608.06581, 2016.
3. Apache Spark. – Spark configuration, 2019.
4. Baldi (P.), Sadowski (P.) et Whiteson (D.). – Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, vol. 5, nC, juillet 2014.
5. Baldi (P.), Sadowski (P.) et Whiteson (D. O.). – Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, vol. 5, 2014, p. 4308.
6. Barsocchi (P.), Crivello (A.), La Rosa (D.) et Palumbo (F.). – A multisource and multivariate dataset for indoor localization methods based on wlan and geo-magnetic field fingerprinting. – In *2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pp. 1–8, octobre 2016.
7. Diaz-Chito (K.), Hernández-Sabaté (A.) et López (A. M.). – A reduced feature set for driver head pose estimation. *Appl. Soft Comput.*, vol. 45, nC, août 2016, pp. 98–107.
8. Gaci (Y.), Moll (A.), Teabe (B.) et Bouchenak (S.). – Lacan: Characterizing distributed machine learning workloads. – In *Conférence d'informatique en Parallélisme, Architecture et Système*, juillet 2019.
9. GraBer (F.), Kallumadi (S.), Malberg (H.) et Zaunseder (S.). – Aspect-based sentiment analysis of drug reviews applying cross-domain and cross-data learning. – In *Proceedings of the 2018 International Conference on Digital Health*, *Dh '18*, Dh '18, pp. 121–125, New York, NY, USA, 2018. Acm.
10. Kraska (T.), Talwalkar (A.) et Duchi (J.). – Mlbase: A distributed machine-learning system. – In *In CIDR*, 2013.
11. Lantz (B.). – *Machine Learning with R*. – Packt Publishing, 2013.
12. Li (M.), Tan (J.), Wang (Y.), Zhang (L.) et Salapura (V.). – Sparkbench: A comprehensive benchmarking suite for in memory data analytic platform spark. – In *Proceedings of the 12th ACM International Conference on Computing Frontiers*, *Cf '15*, Cf '15, pp. 53:1–53:8, New York, NY, USA, 2015. Acm.
13. Mazuecos Pérez (M. D.), Seiler (N. G.), Bederián (C. S.), Wolovick (N.) et Vega (A. J.). – Power efficiency analysis of a deep learning workload on an ibm "minsky" platform. – In Meneses (E.), Castro (H.), Barrios Hernández (C. J.) et Ramos-Pollan (R.) (édité par), *High Performance Computing*, pp. 255–262, Cham, 2019. Springer International Publishing.
14. Meng (X.), Bradley (J.), Yavuz (B.), Sparks (E.), Venkataraman (S.), Liu (D.), Freeman (J.), Tsai (D.), Amde (M.), Owen (S.), Xin (D.), Xin (R.), Franklin (M. J.), Zadeh (R.), Zaharia (M.) et Talwalkar (A.). – Mllib: Machine learning in apache spark. *J. Mach. Learn. Res.*, vol. 17, n 1, janvier 2016, pp. 1235–1241.
15. Moustafa (N.) et Slay (J.). – Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). – In *2015 Military Communications and Information Systems Conference (MilCIS)*, pp. 1–6, 2015.
16. Reiss (C.), Wilkes (J.) et Hellerstein (J. L.). – *Google cluster-usage traces: format + schema*. – Technical report, Mountain View, CA, USA, Google Inc., novembre 2011.

17. Rodriguez-Lujan (I.), Fonollosa (J.), Vergara (A.), Homer (M.) et Huerta (R.). – On the calibration of sensor arrays for pattern recognition using the minimal number of experiments. *Chemometrics and Intelligent Laboratory Systems*, vol. 130, 2014, pp. 123–134.
18. Thornton (C.), Hutter (F.), Hoos (H. H.) et Leyton-Brown (K.). – Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. – In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *Kdd '13*, Kdd '13, pp. 847–855, New York, NY, USA, 2013. Acm.
19. Vergara (A.), Vembu (S.), Ayhan (T.), Ryan (M. A.), Homer (M. L.) et Huerta (R.). – Chemical gas sensor drift compensation using classifier ensembles. *Sensors and Actuators B: Chemical*, vol. 166-167, mai 2012, pp. 320–329.
20. Zaharia (M.), Chowdhury (M.), Franklin (M. J.), Shenker (S.) et Stoica (I.). – Spark: Cluster computing with working sets. – In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, *HotCloud'10*, HotCloud'10, pp. 10–10, Berkeley, CA, USA, 2010.