Characterizing Distributed Machine Learning and Deep Learning Workloads

Yasmine Djebrouni, Isabelly Rocha, Sara Bouchenak, Lydia y Chen, Pascal Felber, Vania Marangozova-Martin, Valerio Schiavoni

ComPAS'2021 : Parallélisme/ Architecture/ Système / Temps Réel
MILC - Lyon, France, 5-9 juillet 2021

Characterizing Distributed Machine Learning and Deep Learning Workloads

Yasmine Djebrouni[1], Isabelly Rocha[2], Sara Bouchenak[3], Lydia Y. Chen[4], Pascal Felber[2], Vania Marangozova-Martin[1], and Valerio Schiavoni[2]

[1]University of Grenoble Alpes – LIG, France
[2]University of Neuchatel, Switzerland
[3]INSA Lyon – LIRIS, France
[4]TU Delft, The Netherlands

Abstract
Nowadays, machine learning (ML) is widely used in many application domains to analyze datasets and build decision making systems. With the rapid growth of data, ML users switched to distributed machine learning (DML) platforms for faster executions and large-scale training datasets. However, DML platforms introduce complex execution environments that are overwhelming for uninitiated users. To provide guidance for the tuning of DML platforms and achieve good performance, it is crucial to characterize DML workloads. In this work, we focus on popular DML and distributed deep learning (DDL) workloads leveraging Apache Spark. We characterize the impact of several platform parameters related to distributed executions such as parallelization, data shuffle and scheduling on performance. Based on our analysis, we derive key takeaways on DML/DDL workload patterns, as well as unexpected behavior of workloads based on ensemble learning methods.

Keywords : Distributed Machine Learning, Distributed Deep Learning, Workload Characterization

1. Introduction

Machine learning (ML) and deep learning (DL) models are widely used in many application domains to analyze real-life datasets and help in the decision making process. Search engines [21], recommendation systems [4] and medical science for disease diagnoses [17] are some of the many examples of ML applications. As for DL models, they bring an important breakthrough for vision and natural language applications [11]. In order to guarantee a high prediction quality and make ML/DL a viable solution, a massive amount of training data is necessary. However, the size of data makes the training phase impractical on centralized platforms. This has lead to the emergence of distributed machine learning (DML) and distributed deep learning (DDL) where distributed systems are used to leverage advantages such as increased task parallelization or total amount of storage disks bandwidth. To simplify the development of DML/DDL solutions, several DML/DDL libraries have been proposed including MLlib [22], BigDL [9] and TensorFlow [3]. As DML/DDL's performance is highly dependent on their workloads, workload characterization is a major issue [7]. Workload characterization captures the execution characteristics of workloads and identifies the factors for performance degradation or optimization. In the context of DML/DDL methods, workload characterization

can provide a better understanding of resource utilization and guide performance optimization at both the hardware and the software levels. Moreover, it may provide insights into the actual effectiveness of configuration strategies of the underlying distributed computing platform, especially for data scientists and ML/DL users who are not familiar with distributed settings.
In this paper, we present our work on DML/DDL workload characterization based on real-world collected traces [13], resulting from extensive experiments conducted on two popular libraries, MLlib [22] and BigDL [9], running on a Spark cluster [29]. We present an analysis of the statistical distributions of the DML/DDL workloads, and discuss the common patterns and characteristics that we observe in these workloads. We derive key takeaways, among which the main aspects that characterize DML/DDL workloads, and unexpected behavior of workloads involving ensemble learning methods under configuration tuning.
The rest of the paper is organized as follows. In §2 we present the necessary background on DML/DDL systems and workloads, as well as the main aspects considered when tuning those systems. Section §3 presents an overview of the traces we rely on in our analysis. Section §4 presents a detailed characterization of DML/DDL workloads. We survey related work in §5, and conclude in §6 with key takeways derived from our detailed characterization.

2. Background

DML and DDL Architecture. Figure 1 depicts the infrastructure we use to perform the DML and DDL experiments. This architecture represents the typical deployment scenario, either on premises or outsourced to third-party cloud deployments. The computing architecture consists of a physical cluster, a distributed computing platform (*i.e.*, Spark [5]), and a distributed data storage (*i.e.*, the Hadoop distributed file system HDFS [28]). The Spark platform is composed of a set of nodes, called *workers*. During training, an interactive process iteratively executes Spark *jobs* which define computation steps. Each of these jobs is scheduled and processed on a dedicated *executor* which is a process on a Spark worker. Jobs are composed of a sequence of *stages* running several parallel and independent *tasks* on data partitions.

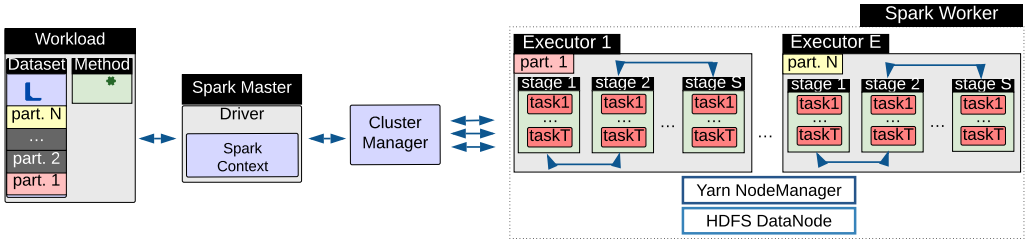


Figure 1: Infrastructure of a typical distributed machine learning or deep learning environment

DML and DDL Workloads. A DML/DDL workload is a tuple consisting of a *ML or DL method* and a *dataset*. There are several types of ML methods, such as classification (*e.g.*, naive Bayes [26], decision trees [25]), regression (*e.g.*, linear models [24]) or clustering (*e.g.*, K-means [20]). When it comes to DL methods, they mostly consist of different types of artificial neural networks which layers and overall structure best fit the domain of interest (*e.g.*, CNNs [27], LSTMs [15], GRU [34]). There exist several libraries that provide open source implementations for DL and ML methods on distributed paltforms, for example, MLlib [22] and BigDL [9] are respectively the ML and DL libraries running on top of Spark. Regarding datasets, there exist multiple real-world datasets that are publicly available and widely used in ML and DL research [2, 12, 10, 6]. Their structure and content vary and depend on the type of learning we are interested in. For

example, News20 is a collection of 20,000 messages [2], collected from 20 different newsgroups, and DrivFace is a popular dataset of 606 images of car drivers [12].

Configuration of DML and DDL. There are many configuration parameters which can be tuned in distributed platforms. For instance, Spark [5] and Hadoop [30] have over 200 and 50 configurable platform parameters, respectively. These parameters usually fall in one of the following types: *(i)* parameters related to memory management, *(ii)* parameters related to data management and compression, *(iii)* parameters related to job scheduling and *(iv)* parameters related to parallel execution.

3. DML and DDL Workload Traces and Setup

We use DML and DDL traces. The DML traces have been obtained from the execution of 26 DML workloads on Spark under different configurations, as described in [13]. We consider 9 popular ML methods from MLlib [22], the Spark ML library, including methods such as K-means, decision trees and linear models. The methods have been applied on three datasets, namely DDF, DGS and DHG, that differ in terms of features and records characteristics. DDF corresponds to the DrivFace dataset [12] that contains a small number of records (606 images) with a big number of features (6,400 pixels). DGS is the Drift dataset [31] with several thousands of records, each corresponding to 128 chemical sensors measurements (features). Finally, DHG references the Higgs dataset [6] which is a collection of 11 millions records of a small number of kinematic measures (features). The DDL traces come from the executions of 9 DDL workloads. We have used 4 popular DDL methods from BigDL [9], the Spark DL library, including CNN [27], GRU [34], LENET [18] and LSTM [14]. The used datasets are Mnist [10], Fashion Mnist [33] and News20 [2]. They respectively contain 70,000 images of written digits, 70,000 images of fashion articles and 20,000 newsgroup messages. The traces contain measures collected from system level (CPU, Memory, Energy, etc.), Spark level (task duration, data size per task, etc.) and application level (training time, inference time and accuracy). In the following, we investigate how the application-level metrics are impacted by configuration tuning. More details about the considered datasets and methods, the collected metrics and the considered Spark parameters can be found in the Appendix. All traces have been collected on a Spark cluster of 4 identical nodes with a quad-socket Intel E3-1275 CPU processor, 8 cores per CPU, 64 GiB. The machines run Spark 2.4.0 as the distributed learning platform, MLlib 2.4.0 as the ML library and BigDL 2.4.0 as the DL library. Each workload execution was replicated 3 times.

4. Characterization of DML and DDL Workloads

In the following, we investigate the impact of 13 Spark configuration parameters (cf Appendix Table 6) on the training time, the inference throughput (*i.e.*, inference responses per time unit) and the accuracy of our workloads. The parameters of interest cover the main aspects of distributed execution. Regarding parallelization, `EXEC_NUM` and `EXEC_COR` control respectively the number of executors running concurrently on the cluster, and the number of cores assigned to each executor. Regarding memory management, `EXEC_MEM` helps setting the memory used by each executor. Other parameters include `SHF_COMPR` which determines whether the data resulting from shuffle[1] should be compressed, and `LOC_WAIT` which defines how long Spark should wait before giving up on a task and rescheduling it. More parameters related to serialization and data compression are also considered .

[1]Shuffle is an operation that involves redistributing data across multiple partitions. It occurs whenever data transfers are needed between stages.

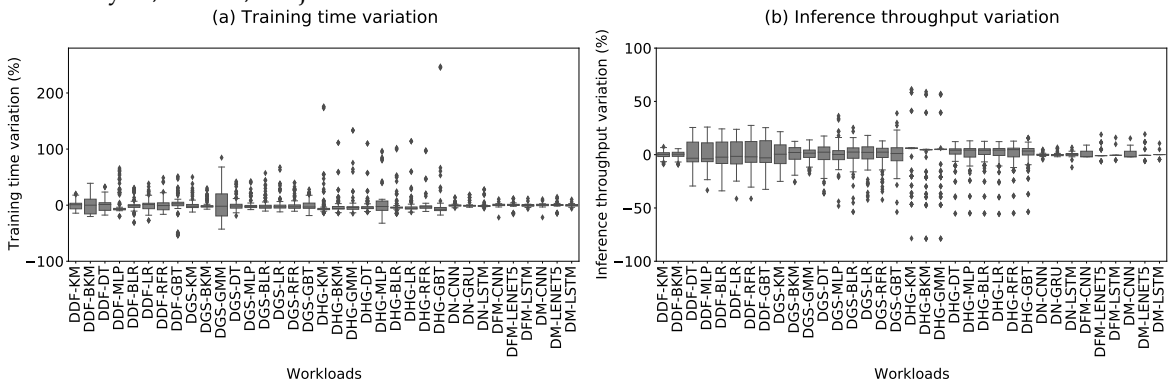


Figure 2: Performance variability per DML/DDL workload

Global Impact of Platform Configurations. We start by investigating whether varying a single platform parameter impacts the performance of the underlying system. We execute several runs for each workload where we individually vary each platform parameter. The variations of training time and inference throughput are shown in Figure 2. We use a box-and-whiskers representation to conveniently show the data distribution through their quartiles, as well as outliers. On the horizontal axis we indicate the different workloads[2], while on the vertical axis we plot the variations in training time and inference throughputs of all the runs grouped by workload. The variation of a run is computed as the difference between the run's time and the mean execution time of the workload, divided by the mean time. We can observe important training time variations spanning between -54% and +247%. For two workloads, namely `DDF-GBT` and `DGS-GMM`, configuration tuning result in significant training speedups (respectively 54% and 43%). Regarding inference throughput, we observe that its variation ranges between -78% and +61%. In most scenarios there are configurations resulting in lower throughputs. These observations support our assumption that both training and inference are strongly affected by the platform parameters' configurations.

Individual Parameter Impact. To clarify the impact of each configuration parameter on training time and inference throughput, we use a heatmap plot as shown in Figure 3. For each of the considered parameters (vertical axis) and workloads (horizontal axis), we use a three-color scheme to represent the parameters impact, from highest (over 20%, dark grey) to negligible (less than 10%, white). The impacts of a parameter is equal to the relative variation obtained in performance, i.e., the difference between the lowest and highest performance, when different values of that parameter are considered while the other parameters have fixed values. What we can directly observe from this heatmap is that the two parameters (number of executors `EXEC_NUM` and number of cores per executor `EXEC_COR`) that configure the parallelization of the execution, highly affect the large majority of the workloads.

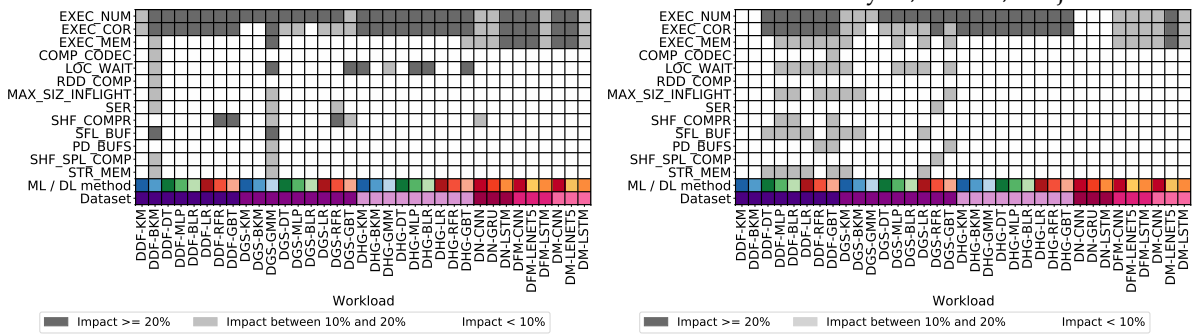
To further explore the impact of each parameter on both the training time and inference throughput, we calculate the coefficient of variation of these two measures inside each workload when testing several values for each parameter (cf Appendix Table 6). The coefficient of variation is defined as the ratio of the standard deviation σ to the mean μ, *i.e.*, $c_v = \frac{\sigma}{\mu}$. We highlight in the following our key observations.

Observation 1 : *Most of DML and DDL workloads significantly benefit from parallelization.*

This observation comes from figures 4a and 4b which present respectively the coefficients of variation of training time and inference throughput for each workload when varying `EXEC_NUM` and `EXEC_COR`. The results report variations beyond 40%.

Observation 2 : *Training times of large datasets workloads are highly impacted by task re-scheduling.*

[2]Workload names are tuples composed of the dataset name followed by the learning method name (cf. Table 1,2)



(a) Impact on training time (b) Impact on inference throughput

Figure 3: Platform parameters impact on performance

As shown in Figure 5a, when working with large datasets, tasks may take arbitrarily long time to complete. This is due to task rescheduling or preemption variations, which are impacted by `LOC_WAIT` parameter.

Observation 3 : *Training of workloads using ensemble learning methods on datasets with a large number of features are affected by shuffle data size reductions.*

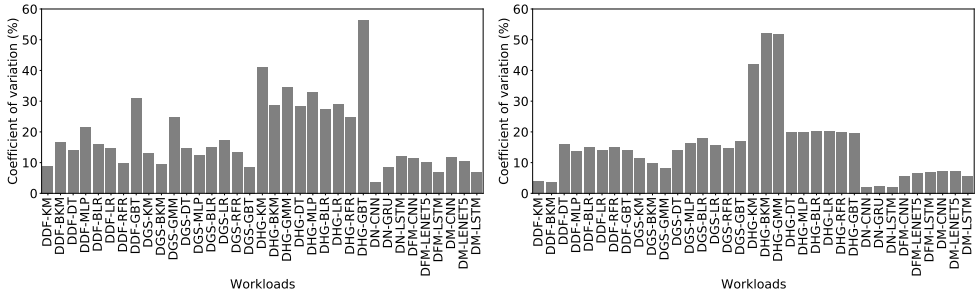
We observe that, for ensemble learning methods used on datasets with many features (`DDF-RFR`, `DDF-GBT`, `DGS-RFR` and `DGS-GBT`, Figure 5b), the training phase triggers frequent data transfer (shuffle) operations. One can use the `SHF_COMPR` parameter to reduce the amount of data during shuffle and reduce the training time.

Observation 4 : *Inference throughput of workloads involving datasets with a relatively high number of features is impacted by memory, scheduling and shuffle operations.*

The number of features presented by the dataset is highly correlated with the resulting inference throughput. Figure 3b shows that there are several parameters interwined that are impacting the inference throughput of workloads involving datasets with high number of features (`DDF` and `DGS`). The concerned parameters are `EXEC_MEM`, `MAX_SIZ_INFLIGHT`, `STR_MEM`, `LOC_WAIT` and `SFL_BUF`. Figure 5c illustrates the impact of one of these parameters (`EXEC_MEM`) on `DDF-BLR`, `DDF-GBT`, `DGS-KM` and `DGS-LR`.

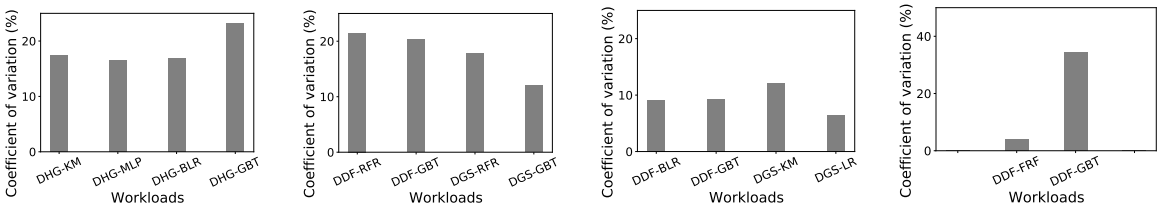
Observation 5 : *Model accuracy of workloads involving ensemble learning methods on datasets with a relatively large number of features is surprisingly impacted by parallel computing configurations.*

We conclude the analysis by studying the effect of configuration tuning on the R2 score of our regression workloads. R2 is a metric that evaluates the precision of regression methods. Figure 5d presents the coefficient of variation of R2 for `DDF-RFR` and `DDF-GBT`. The Figure shows an unexpected behavior of (`GBT` and `RFR`) that are ensemble learning methods, when executed on the dataset that has the highest number of features (`DDF`). R2 for these two workloads surprisingly vary when `EXEC_NUM` and `EXEC_COR` are tuned. In fact, both `RFR` and `GBT` are based on multiple decision trees. Before splitting a node of a decision tree, and in order to determine



(a) Training time (b) Inference throughput

Figure 4: Coefficient of performance variation when varying `EXEC_NUM` and `EXEC_COR`



(a) Training time: LOC_WAIT (b) Training time: SHF_COMPR (c) Infer. throughput: EXEC_MEM (d) R2: EXEC_NUM and EXEC_COR

Figure 5: Coefficient of variation for different performance measures

the best split segment for features, a typical approach is to sample each partition of the dataset, attempting to reduce data transmission operations. On the other hand, when dealing with many features, splitting usually affects the accuracy of the model, and data partitioning comes with a cost. Consequently, for such datasets and learning methods, the impact of parallelization on model quality should be carefully analyzed, and platform configuration and the model hyperparameters should be jointly configured.

5. Related Work

While there are numerous studies on the usage and optimisation of Spark, few studies tackle the performance of ML workloads. SparkBench [19] is a benchmarking suite in which three ML methods are considered. It characterizes the workloads in terms of data access patterns, job execution time and system resources consumption and briefly explores the variation of one Spark parameter. In [23], the authors compare ML performances between Spark and Hadoop [1] considering resource consumption and variations of dataset sizes and Spark cluster size. However, the work focuses only on the KNN [8] method working on a single dataset. Some characterization works describe their traces and share them with the community. The authors of [32] trace the execution of DL workloads using TensorFlow. They use metrics on job scheduling, resource allocation, DL computation features and data I/O. However, they focus on DL methods, while we analyze both ML and DL methods. The authors of [16] have published two-month traces about resource consumption, job scheduling and workload results. They focus on GPU cluster resource usage and scheduling issues of DL workloads.

6. Key Takeaways

From our DML/DDL workload characterization, we derive the following key takeaways:
(1) **Most of DML and DDL workloads significantly benefit from parallelization.** For DML/DDL service operators, we report how platform parameters related to parallel computing, should be systematically considered to efficiently handle model training requests.
(2) **Shuffle and scheduling related parameters are important in the tuning process.** Shuffle operations usually involve data transfer and memory access. Reducing shuffle data size helps improving the execution times. Tasks scheduling parameters also show an important impact on big datasets where tasks are relatively longer due to the big amounts of processed data.
(3) **Model accuracy of workloads involving ensemble learning methods on datasets with relatively large numbers of features is impacted by parallel computing tuning.** This is counter-intuitive, since model accuracy is usually impacted by model hyperparameters, and not by the underlying distributed computing platform parameters. Thus, improved training times of such workloads thanks to parallel computing can come at the cost of lower model accuracy. This trade-off should be carefully analyzed by data scientists and DML/DDL service operators.

As future work, we intend to explore the impact of the joint configuration of platform parameters and hyper-parameters on DML/DDL workloads execution time and model quality.

Bibliography

1. Apache Hadoop. – `https://hadoop.apache.org/`. Accessed: 2021-04-09.
2. News 20 dataset. – `http://qwone.com/~jason/20Newsgroups`. Accessed: 2021-04-09.
3. 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.
4. Aher (S. B.) et Lobo (L.). – Combination of machine learning algorithms for recommendation of courses in e-learning system based on historical data. *Knowledge-Based Systems*, vol. 51, 2013, pp. 1–14.
5. Apache Spark. – Spark configuration, 2021.
6. 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.
7. Calzarossa (M. C.), Massari (L.) et Tessera (D.). – Workload characterization: A survey revisited. *ACM Computing Surveys (CSUR)*, vol. 48, n3, 2016, pp. 1–43.
8. Cover (T.) et Hart (P.). – Nearest neighbor pattern classification. *IEEE transactions on information theory*, vol. 13, n1, 1967, pp. 21–27.
9. Dai (J. J.), Wang (Y.), Qiu (X.), Ding (D.), Zhang (Y.), Wang (Y.), Jia (X.), Zhang (C. L.), Wan (Y.), Li (Z.) et al. – Bigdl: A distributed deep learning framework for big data. – In *Proceedings of the ACM Symposium on Cloud Computing*, pp. 50–60, 2019.
10. Deng (L.). – The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, vol. 29, n6, 2012, pp. 141–142.
11. Deng (L.) et Yu (D.). – Deep learning: methods and applications. *Foundations and trends in signal processing*, vol. 7, n3–4, 2014, pp. 197–387.
12. 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.
13. Djebrouni (Y.), Bouchenak (S.) et Benabdeslem (K.). – Collecting and characterizing distributed machine learning workloads. – In *Conférence d'informatique en Parallélisme, Architecture et Système*, 2020.
14. Greff (K.), Srivastava (R. K.), Koutník (J.), Steunebrink (B. R.) et Schmidhuber (J.). – Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, vol. 28, n 10, 2016, pp. 2222–2232.
15. Hochreiter (S.) et Schmidhuber (J.). – Long short-term memory. *Neural computation*, vol. 9, n 8, 1997, pp. 1735–1780.
16. Jeon (M.), Venkataraman (S.), Phanishayee (A.), Qian (u.), Xiao (W.) et Yang (F.). – Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. – In *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference*, *USENIX ATC '19*, USENIX ATC '19, p. 947–960, USA, 2019. USENIX Association.
17. Kourou (K.), Exarchos (T. P.), Exarchos (K. P.), Karamouzis (M. V.) et Fotiadis (D. I.). – Machine learning applications in cancer prognosis and prediction. *Computational and structural biotechnology journal*, vol. 13, 2015, pp. 8–17.
18. LeCun (Y.) et al. – Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, vol. 20, n5, 2015, p. 14.
19. 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*, pp. 1–8, 2015.

20. Likas (A.), Vlassis (N.) et Verbeek (J. J.). – The global k-means clustering algorithm. *Pattern recognition*, vol. 36, n2, 2003, pp. 451–461.
21. McCallumzy (A.), Nigamy (K.), Renniey (J.) et Seymorey (K.). – Building domain-specific search engines with machine learning techniques. – In *Proceedings of the AAAI Spring Symposium on Intelligent Agents in Cyberspace. Citeseer*, pp. 28–39. Citeseer, 1999.
22. Meng (X.), Bradley (J.), Yavuz (B.), Sparks (E.), Venkataraman (S.), Liu (D.), Freeman (J.), Tsai (D.), Amde (M.), Owen (S.) et al. – Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, vol. 17, n1, 2016, pp. 1235–1241.
23. Mostafaeipour (A.), Jahangard (A.), Ahmadi (M.) et Arockia Dhanraj (J.). – Investigating the performance of Hadoop and Spark platforms on machine learning algorithms. *The Journal of Supercomputing*, vol. 77, 02 2021.
24. Nelder (J. A.) et Wedderburn (R. W.). – Generalized linear models. *Journal of the Royal Statistical Society: Series A (General)*, vol. 135, n3, 1972, pp. 370–384.
25. Quinlan (J. R.). – Induction of decision trees. *Machine learning*, vol. 1, n1, 1986, pp. 81–106.
26. Rish (I.) et al. – An empirical study of the naive bayes classifier. – In *IJCAI 2001 workshop on empirical methods in artificial intelligence*volume 3, pp. 41–46, 2001.
27. Sainath (T. N.), Mohamed (A.-r.), Kingsbury (B.) et Ramabhadran (B.). – Deep convolutional neural networks for lvcsr. – In *2013 IEEE international conference on acoustics, speech and signal processing*, pp. 8614–8618. IEEE, 2013.
28. Shvachko (K.), Kuang (H.), Radia (S.) et Chansler (R.). – The hadoop distributed file system. – In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pp. 1–10. Ieee, 2010.
29. Spark (A.). – Apache spark. *Retrieved January*, vol. 17, 2018, p. 2018.
30. The Apache Software Foundation. – Hadoop commands guide, 2021.
31. 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.
32. Wang (M.), Meng (C.), Long (G.), Wu (C.), Yang (J.), Lin (W.) et Jia (Y.). – Characterizing deep learning training workloads on alibaba-pai. – In *IEEE International Symposium on Workload Characterization, IISWC 2019, Orlando, FL, USA, November 3-5, 2019*, pp. 189–202. IEEE, 2019.
33. Xiao (H.), Rasul (K.) et Vollgraf (R.). – Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.
34. Zhang (Z.), Robinson (D.) et Tepper (J.). – Detecting hate speech on twitter using a convolution-gru based deep neural network. – In *European semantic web conference*, pp. 745–760. Springer, 2018.

7. Appendix

Dataset	Description	#Records	#Features	Size
DDF (Driveface)	Images sequences of subjects while driving in real scenarios [12].	606	6 400	19.9 MB
DGS (Drift)	Measurements from 16 chemical sensors utilized in a discrimination task of 6 gases at various levels of concentrations [31].	13 910	128	40.3 MB
DHG (Higgs)	A collection of kinematic measures to detect signal processes which produce Higgs bosons [6].	11 000 000	28	7.5 GB
DN (News20)	Messages collected from 20 different newsgroups [2].	20 000	2	15 MB
DFM (fashion MNIST)	28x28 grayscale images of fashion articles, associated with labels from 10 classes [33].	70 000	784	31 MB
DM (MNIST)	Handwritten digit images for ML research [10].	70 000	784	12 MB

Table 1: Datasets

Library	Category	ML / DL Method
MLlib	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)
BigDL	Classification	CNN (Convolutional Neural Network)
		GRU (Gated Recurrent Unit)
		LENET5 (Convolutional Neural Network)
		LSTM (Long Short-Term Memory)

Table 2: Learning Methods

Application-level metrics	**Description**
Training accuracy	Percentage of predicted values that match the actual values in the training dataset.
Inference accuracy	Percentage of predicted values that match the actual values in the testing dataset.
Inference throughput	Number of records inferred by unit of time.
Silouhette	Evaluation metric for clustering methods. It measures how similar an object is to its own cluster.
R2 Score	TProportion of the variance in the dependent variable that is predictable from the independent variable(s).
MAE (Mean Absolute Error)	Average of the differences between original values and predicted values.
MSE (Mean Square Error)	Average of the square of the difference between the original values and the predicted values.
RMSE (Root Mean Square Error)	The square root of MSE.
F1 Score	Harmonic mean between precision and recall.

Table 3: Application-level Metrics

Platform-level metrics	**Description**
Task duration	Total elapsed time.
Task deserialization time	Elapsed time spent to deserialize this task.
Garbage collection time	Total JVM garbage collection time.
Result Serialization Time	Elapsed time spent serializing the task result.
Shuffle wait time	Time that tasks spent blocked waiting for shuffle data to be read from remote machines.
Shuffle read size	Total shuffle bytes read, includes both data read locally and data read from remote executors.
Shuffle write size	Total shuffle bytes written.

Table 4: Platform-level Metrics

Infrastructure-level metrics	**Description**
CPU usage	Percentage of used CPU.
Memory usage	Percentage of memory utilization
Network traffic	Amount of bytes read from and written to the network.
Disk usage	Amount of bytes read from and written to disk.
Energy consumption	Trapezoidal integral of power measurements collected per second.

Table 5: Infrastructure-level Metrics – From /proc/stat, /proc/meminfo, etc.

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

Table 6: Considered Spark Parameters