# Characterizing Distributed Machine Learning Workloads - Technical Report

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

[1] University of Grenoble Alps, France, first.last@univ-grenoble-alpes.fr
[2] University of Neuchatel, Switzerland, first.last@unine.ch
[3] INSA Lyon, France, first.last@insa-lyon.fr

## 1 Introduction

This document presents a technical report of the characterization work done in the paper entitled *Characterizing Distributed Machine Learning Workloads*. The rest of the document is organized as follows. In §2 we detail the key elements of our workload characterization methodology, along with the different metrics collected for our study. In §3 we provide statistical characteristics of the collected workloads traces and suggest ways to use them. We detail the single-level tuning results of our DML workloads in §4.

## 2 Trace Collection Methodology

In the following we describe the experimental setup and the DML workloads used to collect our traces. We also detail our exploration methodology of the configuration parameters space to understand the impact of different parameters on the workloads' performance.

### 2.1 Experimental setup

We use Spark 2.4.0 as distributed computing platform and HDFS 2.7.7 as distributed file system. The used DML library is MLlib (v2.4.0) [19] or BigDL (v2.4.0) [8]. We conduct our experiments on two clusters as described below.
**Cluster 1.** Cluster 1 is a 4-nodes cluster equipped with a quad-socket Intel E3-1275 CPU processor, 8 cores per CPU, 64 GiB of RAM, 480 GB SSD drives, on a switched 1 Gbps Ethernet LAN, running Ubuntu Linux 16.04.1 LTS.
**Cluster 2.** To consider more hardware diversity and run bigger workloads, we also deploy a 24-nodes cluster, dual-socket 8 core Intel Xeon E5-2630 CPU, 128 GB RAM, 600 GB HDD, 2× 10 Gbps Ethernet, running Debian GNU/Linux 9.7.

There are several reasons behind our choice of using CPU-only clusters. First, CPU clusters do not have the memory constraints of GPU clusters and can handle larger models [15, 24]. Second, recent works show that CPUs are still in the

**Table 1.** Learning datasets

| CC | Dataset | Description | #Records | #Features | Size |
|---|---|---|---|---|---|
| | DDF (Drive-face) | Images sequences of subjects while driving. | 606 | 6,400 | 19.9 MB |
| | DGS (Drift) | Measurements from 16 chemical sensors utilized in a discrimination task of 6 gases. | 13,910 | 129 | 40.3 MB |
| | DHG (Higgs) | A collection of kinematic measures to detect signal processes which produce Higgs bosons. | 11,000,000 | 28 | 7.5 GB |
| | DN (News20) | Messages collected from 20 different newsgroups. | 118,845 | 2 | 68.7 MB |
| | DFM (Fashion MNIST) | Images of fashion articles, associated with labels from 10 classes. | 700,000 | 784 | 54.9 MB |
| | DM (MNIST) | Handwritten digit images for ML research. | 70,000 | 784 | 52.4 MB |

competition and may even yield better performance than GPUs for deep learning [6]. Finally, we are interested in the relative impact of different configuration strategies and not in the absolute performance that may be obtained. If GPUs may accelerate training time supporting high parallelisation [11], we focus in our work on how different parallelisation settings, together with other platform and hyper- parameters, impact global DML performance.

### 2.2 DML Workloads

**ML Datasets.** We consider six commonly used and publicly available datasets [1, 25, 9, 10, 23, 4] shown in Table 1 (the CC columns in Tables 1 and 2 indicate the color codes used later in the graphs). We have chosen datasets to differ in terms of content type (*e.g.*, text, images), number of records, number of features and total size.

In our experiments, we use random split to define the training and inference sets. 80% of each dataset are used for model training and the remaining 20% are dedicated to inference.

**ML Methods.** We test 13 state-of-art ML methods commonly used by data scientists including 9 MLlib's methods and 4 BigDL's methods (see Table 2). They MLlib methods implement clustering, classification or regression and are based on different learning methods such as gradient descent, decision trees and neural networks. The deep neural networks have the following architectures.include a CNN consisting of 9 layers, a GRU with 7 layers, a LENET5 (a specific CNN for the MNIST dataset) with 5 layers and a LSTM with 7 layers.

In the rest of the paper, the names of the workloads are composed of the name of the dataset followed by the name of the learning method. For example, for the `DDF` dataset and the clustering methods we have the `DDF-KM`, `DDF-BKM` and `DDF-GMM` workloads.

### 2.3 Parameter Settings

In our experiments, we deploy each workload (*i.e.*, dataset and ML method) on the corresponding DML environment (*i.e.*, MLlib or BigDL on a Spark cluster).

**Table 2.** Learning methods

| Library | Category | ML Method | CC |
|---|---|---|---|
| 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) | |

For each deployed workload, we consider default values of hyper-parameters, default Spark platform parameters, variations for hyper-parameters and variations for different of Spark parameters. Each experiment is replicated three times, which is enough due to low variations in data, as shown by the confidence intervals in our plots.

**Hyper-parameters.** Based on previous works [12, 22, 21], we choose hyper-parameters that have a high impact on performance in terms of execution time and accuracy. These include the maximum depth of tree-based algorithms, the maximum number of iterations to reach model convergence and the learning rate and batch size of deep neural networks. More details can be found in Appendix 2.3.

**Platform Parameters.** We leverage existing studies [18, 3] that evaluate which Spark parameters affect performance most. These include scheduling, data transfers, data storage and representation, parallelisation and memory management. A detailed description of the used configuration parameters and the used experimental values are given in Appendix 2.3.

Most Spark parameters are left to their default values, except for executor memory; we changed the value from the default 1 GiB to 5 GiB to avoid out-of-memory issues. We used the same configuration values of Spark platform parameters for both MLlib and BigDL, except for executor configurations for BigDL. The configurations respect the constraints according to which the defined batch size has to be divisible by the total number of cores (*i.e.*, number of executors and the nomber of cores per executor are defined accordingly).

# 3 Traces Overview and Analysis

In this section, we first suggest possible uses of the traces collected during our experiments. Then, we present their statistical features.

**Table 3.** Spark platform parameters

| Configuration aspect | Spark platform parameters | Description | Values |
|---|---|---|---|
| Parallel computing | EXEC_NUM (executor.instances) | Number of executors | MLlib: 1, 2, 3, 4, 5, 6, 7, 8, 12, 48, 72, 96; BigDL: 1, 2, 4 |
| | EXEC_COR (executor.cores) | Number of cores per executor | MLlib: 1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 14, 16; 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., 50 GB., 70 GB., 100 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 data 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 compress serialized RDD partitions | true, false |
| | SHF_SPL_COMP (shuffle.spill.compress) | Must compress data spilled during shuffles | true, false |
| Scheduling | LOC_WAIT (locality.wait) | How long to wait to launch a data-local task on a less-local node | 10 ms, 100 ms, 500 ms, 1 s, 3 s, 10 s |
| 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 |

### 3.1 Possible Usage of Our Traces

**Synthetic Trace Generation.** Using our traces' statistical profiles, one can generate synthetic DML traces. Indeed, leveraging trace features such as numbers of tasks, duration and distribution, one can simulate and thus reason about DML systems without the need for real experimentations. We refer to [7, 16, 14] for tools and techniques for synthetic trace generation from real traces.

**Workload Modeling and Simulation.** Our traces can help modeling DML workloads. Models that capture workload patterns can be used to predict performance. They can also help the optimization of existing DML platforms and even the design of new platforms embedding the workloads' specificities. The analysis and the modeling of workloads has already been considered in different application domains including data centers, the cloud and the web [13, 17, 20].

### 3.2 Statistical Analysis

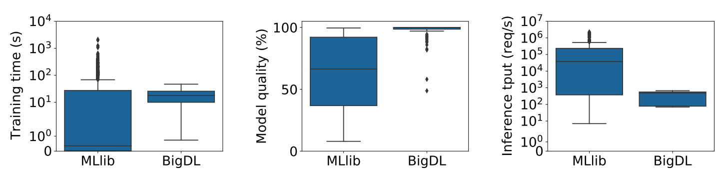
We have harvested metrics at the application-, platform- and the infrastructure levels. Our traces have been collected on both clusters 1 and 2 (see §2.1). They consist of 16.2 GiB of data with more than 80 millions records (see Table 5). The traces and their detailed description are available on a public archive for the research community [5].

**Table 4.** Hyper-parameters

| Hyper-parameter | Description | Values | MLlib / BigDL method |
|---|---|---|---|
| maxIter | Maximum number of iterations to achieve convergence. | 5, 10, 15, 20, 50, 100 | BLR, MLP, BKM, KM, GMM, GBT, LR, CNN, GRU, LSTM, LENET |
| maxBins | Number of classes for the discretization of continuous variables | 4, 16, 32, 48 | DT, GBT, RFR |
| maxDepth | Maximum depth of decision trees before convergence | 5, 10, 15, 20 | DT, RFR, GBT |
| tol | Convergence threshold for stopping the algorithm | 0.000001, 0.0001, 0.01, 0.1 | KM, GMM, MLP, BLR, LR |
| numTrees | The number of decision trees built | 10, 20, 50, 100, 150 | RFR |
| stepSize | Model learning rate | 0.003, 0.03, 0.3 | GBT, MLP, CNN, GRU, LSTM, LENET |
| blockSize | Batch size | 32, 128, 256, 512, 1024 | MLP, CNN, GRU, LSTM, LENET |

**Table 5.** Collected DML workload traces

| Trace description | #Records | #Features | Size |
|---|---|---|---|
| Infrastructure-level traces | 43,346,092 | 11 | 5.3 GiB |
| Platform-level traces | 41,481,857 | 42 | 10.8 GiB |
| Application-level traces | 12,934 | 18 | 9.6 MiB |
| **Total** | **84,840,883** | **Up to 42** | **16.2 GiB** |




(a) Training time (b) Model quality (c) Inference tput.

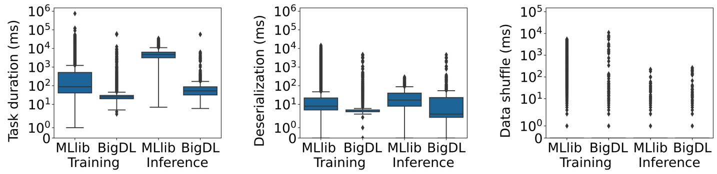
**Fig. 1.** Distribution of application-level metrics

We employ box-and-whiskers plots to report the statistical distribution of our metrics. The values are grouped by the ML library used in the experiments (MLlib or BigDL) and by the learning phase (training or inference).

**Application-level Traces.** Figure 1 reports the statistical distribution for three of the collected application-level metrics, *i.e.*, training time, accuracy and inference throughput.

Figure 1(a) shows normalized training times for a training set of 1,000 records. In our experiments, MLlib exhibits high variation while BigDL is more stable: 50% of the MLlib cases have very short training times ($\leq 0.4$ s) and 25% have training times between 25 s and 30 min. BigDL training times span between 4 s and 47 s. The greater dataset heterogeneity in our MLlib workloads explains these behaviours if compared to BigDL.

Figure 1(b) gives the models quality metrics: accuracy for classification models, $R^2$ for regression models and silhouette for clustering models. The median model quality for MLlib workloads is 66.4%, and up to 99.5%. For BigDL, it is between 98% and 100% for 75% of the workloads.




(a) Task duration (b) Deserialization time (c) Data shuffle

**Fig. 2.** Distribution of platform-level metrics

Finally, Figure 1(c) shows that the median inference throughput of our BigDL workloads (476 reqs/s) is less than that of MLlib workloads (around 37,747 requests/s). Infact, classical ML inference is cheaper and more time-efficient than DL-based inference, where the cost is proportional to the network complexity.

**Platform-level Traces.** Figure 2 represents the distribution of 3 Spark metrics: *(a)* task duration *(b)* task serialization, used upon loading tasks by executors, and *(c)* data shuffling which corresponds to data transfers. Results include both the training and the inference phases.

As shown in Figure 2(a), short tasks (in the 1–100 ms range) are very common with BigDL with up to 79% of the tasks. Less common for MLlib, they are up to 50% for training, and even less for inference where 75% of tasks last at least 3,000 seconds. Longer tasks are more frequent in the inference phase for both MLlib and BigDL. Inference tasks that last more than 100 ms represent 21% for BigDL and up to 98% for MLlib. In training, such tasks represent only 3% for BigDL and 47% for MLlib.

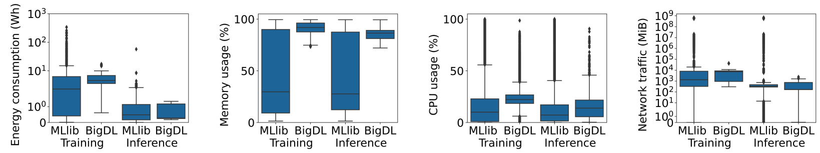
Task deserialization (Figure 2(b)) is very fast (*i.e.*, $\leq$ 10 ms) in 75% of BigDL training tasks, and in 25% to 50% for the other tasks. It is longer (10 ms-10,000 ms) in 75% of MLlib inference tasks but at maximum 50% of the other tasks.

Data shuffling (Figure 2(c)) is negligible for 75% of all tasks. However, it reaches up to 10,000 ms for training tasks and up to 100 ms for inference tasks, *i.e.*, equal to the duration of the whole task (Figure 2(a)).

**Infrastructure-level Traces.** Figure 3 reports the infrastructure measurements of energy consumption, network traffic, CPU and memory usage. Figure 3(a) indicates that up to 25% of MLlib workloads consume very little energy *i.e.*, $\leq$ than 0.4 Wh. BigDL inference executions consume at least 0.2 Wh and BigDL training at least 0.6 Wh. However, BigDL workloads consume at most 17 Wh whereas the consumption reaches up to 340 Wh for MLlib. This is directly related to the longer MLlib executions.

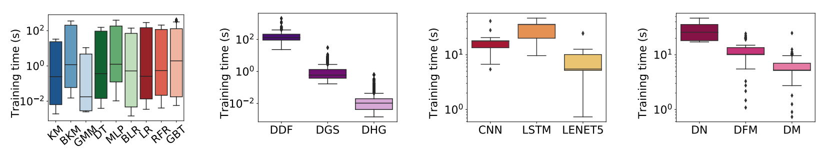
Regarding memory usage, Figure 3(b) shows all our BigDL workloads to be memory-intensive, with at least 70% of memory usage. Also, our workloads are memory-bound and not CPU-bound as CPU usage does not exceed 30% in 75% of all measurements (Figure 3(c)).

Finally, Figure 3(d) shows collected measurements of network traffic. We observe that inference for both MLlib and BigDL workloads involves the lowest network traffic: 75% of inference executions consume at most 800 MiB. In contrast, in 50% of MLlib and BigDL training network traffic exceeds 1 GiB. This is due to the per-iteration exchanges needed to build the model.




(a) Energy consumption (b) Memory usage (c) CPU usage (d) Network traffic

**Fig. 3.** Distribution of infrastructure-level metrics (CPU usage, memory usage, network usage and energy consumption)




(a) MLlib methods (b) MLlib datasets (c) BigDL methods (d) BigDL datasets

**Fig. 4.** Training time variability within same DML method (normalized) vs. within same dataset

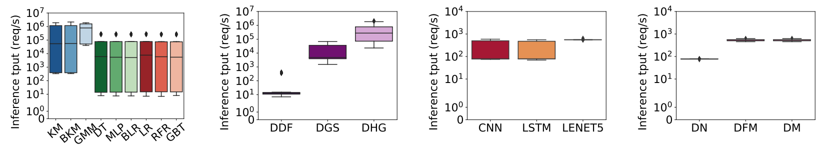
## 4 Characterizing DML Workloads

To characterize the sensitivity of DML workloads to different configuration parameters and strategies, we study the effect of varying only platform parameters (§4.1) and of varying only hyper-parameters (§4.2).

These results concern both Cluster 1 and Cluster 2 (see §2.1).

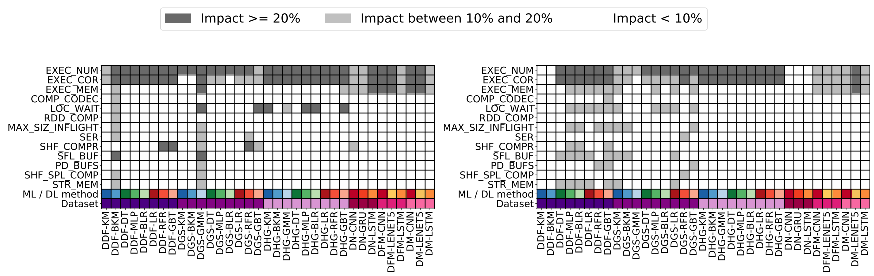
### 4.1 Tuning Platform Parameters

We characterize the impact of platform parameters on training time and inference throughput by varying each parameter individually, for each DML workload. We measure the relative variations obtained in performance by each platform




(a) MLlib methods (b) MLlib datasets (c) BigDL methods (d) BigDL datasets

**Fig. 5.** Inference throughput variability within same DML method vs. within same dataset




(a) Training time (b) Inference throughput

**Fig. 6.** Impact of platform parameter on performance (dark-grey for high: $> 20\%$, light-grey for medium: $10\% \leq 20\%$, white for low: $< 10\%$). The two bottom rows of each figure refer to the color codes specified in Table 1 and Table 2, respectively.

parameter. We consider parameters with high impact on performance if their corresponding variation $\geq 20\%$. Parameters which incur variations in the 10%-20% range have medium impact. Finally, if the variations are $\leq 10\%$, we consider those low impact. In the heat map representations of Figures 6(a) and 6(b), we use a 3-color schem (shades of grey), from dark grey (high) to white (low). The considered platform parameters (vertical axis) are introduced in §**??** (additional details in §2.3-Table 3).

For the majority of the workloads, the number of Spark executors (`EXEC_NUM`) and the number of cores per executor (`EXEC_COR`) play key roles, as they impact parallelization and performance, particularly while training.

***Observation 1 :*** *DML workloads' performance is significantly impacted by parallelization.*

Figure 6(a) shows that for ensemble learning methods (*e.g.*, random forests and gradient-boosted trees) on datasets with many features (from 100 and beyond) the training phase triggers frequent shuffle operations (`SHF_COMPR` parameter). Examples include `DDF-RFR`, `DDF-GBT`, `DGS-RFR` and `DGS-GBT`, with the the highest impacts of `SHF_COMPR` (impact $\geq$19%) among all workloads. Tuning Spark's shuffle compression parameter, one reduces the amount of data during shuffle, causing less network traffic and therefore faster training time.

***Observation 2 :*** *Training ensemble learning methods on datasets with large number of features significantly benefits from shuffle data size reductions.*

Our large datasets highlight that `LOC_WAIT` (*i.e.*, the timeout after which a data-local task is launched on a distant node) can significantly affect the training time. Indeed, waiting for an available nearby node for the job would prevent huge data transfers and shuffles that consume time and bandwidth, and thus would significantly impact efficiency. Thus, jobs that deal with large amounts of data benefit from increasing the `LOC_WAIT` parameter. We observe this phenomenon in particular in Figure 6(a), with the Higgs dataset ((DHG) and four methods for clustering and classification (`DHG-KM`, `DHG-MLP`, `DHG-BLR` and `DHG-GBT`).

***Observation 3 :*** *Training with large datasets is significantly impacted by task re-scheduling.*

### 4.2 Tuning Hyper-parameters

We investigate the impact of hyper-parameters on our models' quality, namely accuracy for classification tasks, $R^2$ coefficient for regression tasks and silhouette score for clustering tasks. We vary individually several hyper-parameters, such as number of iterations, total classes for discretization of continuous variables, the depth of decision tres and others (see the full list in §2.3-Table 4). For each hyper-parameter, we define its impact by the relative performance variations obtained while varying it.

In Figure 7 we distinguish: *(i)* high-impact hyper-parameters for variations beyond 5%, *(ii)* medium-impact for variations in the 5%-1% range, and *(iii)* low-impact parameters, for variations $\leq$ 1%. Note that empty (*i.e.*, white) cells indicate that the corresponding hyper-parameters have low impact or are irrelevant for the target learning methods. We observe that the *maxIter* hyper-parameter, *i.e.*, the parameter giving the number of iterations for a given learning method, has high impact on the quality of several methods like `BKM` (Bisecting K-means used in the `DDF-BKM`, `DGS-BKM` and `DHG-BKM` workloads) and `MLP` (Multilayer Perception, with `DGS-MLP` and `DHG-MLP` workloads). Further, *maxDepth* and *maxBins* are also impactful hyper-parameters for decision-tree (`DT`) methods.

For BigDL workloads, the number of epochs lightly affect the methods accuracy. Instead, the *stepSize* and *batchSize* hyper-parameters affect several BigDL workloads. Finally, we observe that for many MLlib and BigDL methods (*e.g.*, `KM`, `BKM`, `GMM`, `RFR`, `GBT`, `DT`, `LENET5`), the learning method is always impacted by the same hyper-parameters, and this holds for any given dataset.

We can thus confirm the general intuition stated below.

***Observation 4 :*** *For many DML workloads, the impact of hyper-parameters on model quality depends more on the type of the learning method than on the nature of data.*

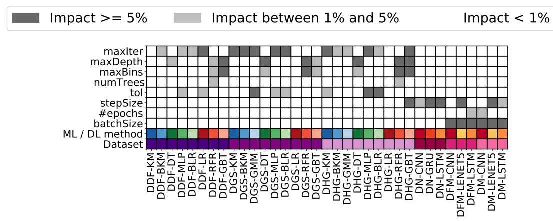



**Fig. 7.** Impact of hyper-parameter on performance (dark-grey for high: $> 5\%$, light-grey for medium: $1\% \leq 5\%$, white for low: $< 1\%$). The two bottom rows of each figure refer to the color codes specified in Table 1 and Table 2, respectively.

## 5 Conclusion

We presented our DML workload characterization conducted on diverse and heterogeneous workloads from 13 widely used learning methods, with 6 real- world datasets. We collected extensive execution traces, for a total of 16.2 GiB and over 80 million records. We presented a detailed analysis of the statistical distributions of the collected DML workload traces, showing their main characteristics, and we derived ten interesting observations to characterize DML workloads. We publicly release our collected traces [2] to help researchers and practitioners in future DML studies, such as building realistic modeling and simulation tools of DML workloads, or building tools for synthetic DML trace generation.

## References

1. News 20 dataset. `http://qwone.com/~jason/20Newsgroups`, accessed: 2021-02-04
2. DML Workload Characterization Git Repository (Feb 2023), `https://github.com/DMLCharacterization/DMLCharacterization`
3. Apache Spark: Spark configuration. `https://spark.apache.org/docs/2.4.3/configuration.html` (2021)
4. Baldi, P., Sadowski, P., Whiteson, D.: Searching for exotic particles in high-energy physics with deep learning. Nature Communications **5**(C) (Jul 2014)
5. Blind, D.: "anonymous dataset middleware'22 submission". `https://doi.org/10.5281/zenodo.5004927` (2022). https://doi.org/10.5281/zenodo.5004927
6. Chen, B., Medini, T., Farwell, J., Gobriel, S., Tai, T.C., Shrivastava, A.: SLIDE : In defense of smart algorithms over hardware acceleration for large-scale deep learning systems. In: Dhillon, I.S., Papailiopoulos, D.S., Sze, V. (eds.) Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020. mlsys.org (2020), `https://proceedings.mlsys.org/book/306.pdf`
7. Chen, J., Clapp, R.M.: Astro: Auto-generation of synthetic traces using scaling pattern recognition for mpi workloads. IEEE Transactions on Parallel and Distributed Systems **28**(8), 2159–2171 (2017). https://doi.org/10.1109/TPDS.2017.2649518
8. 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)

9. Deng, L.: The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. IEEE Signal Processing Magazine **29**(6), 141–142 (2012). https://doi.org/10.1109/MSP.2012.2211477
10. Diaz-Chito, K., Hernández-Sabaté, A., López, A.M.: A reduced feature set for driver head pose estimation. Appl. Soft Comput. **45**(C), 98–107 (Aug 2016). https://doi.org/10.1016/j.asoc.2016.04.027, `http://dx.doi.org/10.1016/j.asoc.2016.04.027`
11. Elshawi, R., Wahab, A., Barnawi, A., Sakr, S.: DLBench: a comprehensive experimental evaluation of deep learning frameworks. Cluster Computing (Feb 2021). https://doi.org/10.1007/s10586-021-03240-4, `https://doi.org/10.1007/s10586-021-03240-4`
12. Fränti, P., Sieranoja, S.: How much can k-means be improved by using better initialization and repeats? Pattern Recognition **93**, 95–112 (2019)
13. Gabriel, P., Mello, R.: Modelling distributed computing workloads to support the study of scheduling decisions. International Journal of Computational Science and Engineering **11**, 155–166 (01 2015). https://doi.org/10.1504/IJCSE.2015.071879
14. Galindo, H.E.S., Guedes, E.A.C., Maciel, P.R.M., Silva, B., Galdino, S.M.L.: Wgcap: A synthetic trace generation tool for capacity planning of virtual server environments. In: 2010 IEEE International Conference on Systems, Man and Cybernetics. pp. 2094–2101 (2010). https://doi.org/10.1109/ICSMC.2010.5641705
15. Kalamkar, D., Georganas, E., Srinivasan, S., Chen, J., Shiryaev, M., Heinecke, A.: Optimizing deep learning recommender systems training on cpu cluster architectures. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. SC '20, IEEE Press (2020)
16. Koltuk, F., Schmidt, E.G.: A novel method for the synthetic generation of non-i.i.d workloads for cloud data centers. In: 2020 IEEE Symposium on Computers and Communications (ISCC). pp. 1–6 (2020). https://doi.org/10.1109/ISCC50000.2020.9219577
17. Magalhães, D., Calheiros, R., Buyya, R., Gomes, D.: Workload modeling for resource usage analysis and simulation in cloud computing. Comput. Electr. Eng. **47**, 69–81 (2015)
18. Marcu, O., Costan, A., Antoniu, G., PÃĺrez-HernÃąndez, M.S.: Spark Versus Flink: Understanding Performance in Big Data Analytics Frameworks. In: 2016 IEEE International Conference on Cluster Computing (CLUSTER). pp. 433–442 (2016). https://doi.org/10.1109/CLUSTER.2016.22
19. 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 **17**(1), 1235–1241 (2016)
20. Piga, L., Bergamaschi, R., Klein, F., Azevedo, R., Rigo, S.: Empirical web server power modeling and characterization. In: 2011 IEEE International Symposium on Workload Characterization (IISWC). pp. 75–75 (2011). https://doi.org/10.1109/IISWC.2011.6114200
21. Probst, P., Wright, M.N., Boulesteix, A.L.: Hyperparameters and tuning strategies for random forest. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **9**(3), e1301 (2019)
22. Reyes, A.K., Caicedo, J.C., Camargo, J.E.: Fine-tuning deep convolutional networks for plant recognition. CLEF (Working Notes) **1391**, 467–475 (2015)
23. Vergara, A., Vembu, S., Ayhan, T., Ryan, M.A., Homer, M.L., Huerta, R.: Chemical gas sensor drift compensation using classifier ensembles. Sensors and Actuators B: Chemical **166-167**, 320–329 (May 2012).

https://doi.org/10.1016/j.snb.2012.01.074, https://doi.org/10.1016\%2Fj.snb.2012.01.074
24. Wang, Y.E., Wei, G.Y., Brooks, D.: Benchmarking tpu, gpu, and cpu platforms for deep learning (2019)
25. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747 (2017)