SciKit Learn vs Dask vs Apache Spark Benchmarking on the EMINST Dataset

Filip Zevnik, Din Music, Carolina Fortuna, Gregor Cerar
Department of Communication Systems, Jozef Stefan Institute
Ljubljana, Slovenia
zevnikfilip@gmail.com

***Abstract*—As datasets for machine learning tasks can become very large, more consideration to memory and computing resource usage has to be given. As a result, several libraries for parallel processing that improve RAM utilization and speed up computations by parallelizing ML jobs have emerged. While SciKit Learn is the typical go to library for practitioners, Dask is a parallel computing library that can be used with SciKit and Apache Spark is an analytics engine for large-scale data processing that includes some machine learning techniques. In this paper, we benchmark the three solutions for developing ML pipelines with respect to data merging and loading and subsequently for training and predicting on the extended MNIST (eMNIST) dataset under Linux and Windows OS. Our results show that Linux is the better option for all of the benchmarks. For low amounts of data plain SciKit learn is the best option for machine learning, but for more samples, we would choose Apache Spark. On the other hand, when it comes to dataframe manipulation Dask beats a normal pandas import and merge.**

***Index Terms*—Apache Spark, Dask, machine learning, Pandas, import**

I. Introduction

As datasets for machine learning tasks can become very large, more consideration to memory and computing resource usage has to be given. As a result, several libraries for parallel processing that improve RAM utilization and speed up computations by parallelizing ML jobs have emerged. While SciKit Learn [1] is the typical go to library for practitioners, Dask [2] is a parallel computing library that can be used with SciKit to improve memory and CPU utilization. Dask improves memory utilization by not immediately loading all the data, but only pointing to it. Only part of the data is loaded on a per need basis. It also enables using all available cores on a system to train a model. Apache Spark is an analytics engine written in Java and Scala for processing large-scale data that incorporates some machine learning techniques and is tightly integrated with the Spark architecture.

While there are other libraries [3] that enable parallelization of ML, when it comes to distributed computing tools for tabular datasets, Spark and Dask are the most popular choices today. Even though Spark is an older, more stable solution, Dask is part of the vibrant Python ecosystem and both technologies excel at parallelization. While the two solutions have been already been benchmarked on big data pipelines [4] and on various image processing and learning scenarios [5]–[7]. The work in [7] is the closest to this one, however they focused on evaluating the tradeoffs in parellelizing feature extraction and clustering while this work focuses on evaluating data loading and merging and subsequent classification.

This work was funded by the Slovenian Research Agency ARRS under program P-0016.

In this paper, we benchmark the three solutions for developing ML pipelines with respect to data merging and loading and subsequently for training and predicting on the extended MNIST (eMNIST) dataset under Linux and Windows OS. Our results show that Linux is the better option for all of the benchmarks. For low amounts of data plain SciKit learn is the best option for machine learning, but for more samples, we would choose Apache Spark. On the other hand, when it comes to dataframe manipulation Spark is behind Dask, and Dask beats a normal pandas import and merge. The contribution of this paper is the benchmarking of three ML libraries across various data sizes and two operating systems on two parts of the ML model development pipeline.

The remainder of the paper is structured as follows. Section II discusses related work. Section III presents the methodology used in the benchmarking. Section IV evaluates the comparison. Finally, Section V presents our conclusions.

II. Related work

Chintapalli et al. (2016) [8] compared streaming platforms Flink, Storm and Spark. The paper focuses on real-world streaming scenarios using ads and ad campaigns. Each streming platform was used to build a pipeline that identifies relevant events, which were sources from Kafka. In addition, Redis was used for storing windowed count of relevant events per campaign. The test system contained 40 nodes, where each node contained 2 CPUs with 8 cores and 24GB of RAM. All nodes were interconnected using a gigabit ethernet connection. The experiment encompassed Kafka producing events at set rate with 30 minutes interval between each batch was fired. The results showed that both Flink and Storm were almost equal in terms of event latency, while Spark turned out to be the slowest of the three.

Dugré et al. (2019) [4] compared Dask and Spark on the neuroimaging big data pipelines. As neuroimaging requires a large amount of images to be processed, Spark and Dask were in the time of writing the best suited Big Data engines. The paper compares the technologies with three different pipelines. First is incrementation, second is histogram and the final

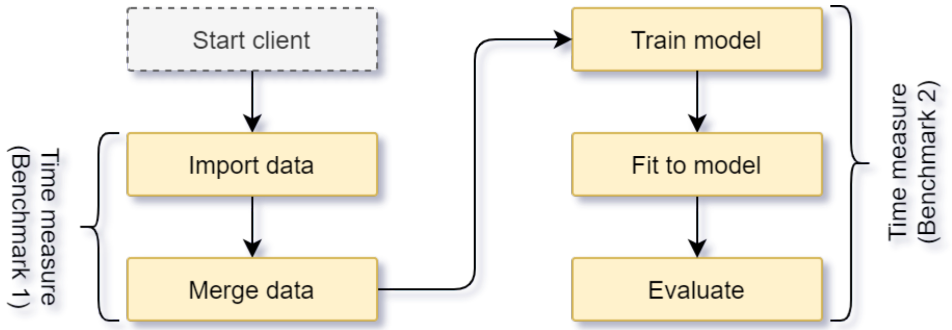

Fig. 1. Workflow of the Machine learning test example used for benchmarking.

one is a BIDS app example (a map-reduce style application). All comparisons were done on BigBrain and CoRR datasets, with sizes of 81GB and 39GB respectively. The authors have concluded that all platforms perform very similarly and that the incrementation of worker nodes is not always the optimal solution due to the transfer times and overall overhead. While all platforms yielded similar results, the Spark is claimed to be the fastest out of the three platforms.

Nguyen et al. (2019) [6] evaluated SciDB, Myria, Spark, Dask and TensorFlow to figure out which system is best suited for image processing. Similarly to [4], the authors compared the systems using different pipelines. For comparison, the authors used 2 datasets, both over 100GB in size. The comparison reveled that Dask and Spark are comparable in the performacnce as well as the ease of use.

Mehta et al. (2016) [5] presented the satellite data processing pipeline. The pipeline consists of two steps, a feature extraction step and a clustering step. The baseline pipeline used the Caffe deep learning library and SciKit. The improved pipeline used Keras along with Spark and Dask for multi-node computation. They found that while Spark was the fastest in terms of computational time required per task, Dask used almost half the memory compared to Spark due to recalculation of the intermediate values. SciKit Learn was not able to complete the task and was excluded from the final comparison. It was concluded that Spark is the best performer, while Dask is the easiest to use.

Cheng et al. (2019) [7] presented a comparison of the RADICAL-Pilot, Dask and Spark for image processing. All three systems were tested using watershed and a blob detector algorithms. Each test was split into two parts, a weak scaling algorithm where the amount of data to be processed was increased alongside the number of nodes, and a strong scaling algorithm where the amount of data stayed the same and the number of nodes increased. The evaluation showed that Dask outperformed Spark on weak scaling, while Spark excelled in the strong scaling part.

III. Methodology

To benchmark the three solutions, namely SciKit learn, Dask and Spark, we single out two parts of the end-to-end model development process depicted in Figure 1. We first time the data importing and merging process, referred to as Benchmark 1 in the figure, followed by model training and evaluation denoted by Benchmark 2. While the time required to train the model is usually the most important metric because it takes up most of the computation time, importing and merging the input data cannot be ignored. As described in Algorithm 1, for Benchmark 1, training data was imported and then merged. For SciKit Learn dataframes were used all along and no parallelization was used while for Dask and Spark parallelization was turned on.

Algorithm 1: Import and merge benchmarking process.

```
Enable parallelization
Require: data_a and data_b
Merge the DataFrames
Convert data to a pandas DataFrame
```

Algorithm 2: Train/fit and evaluate benchmarking process.

```
Enable parallelization
Import and setup data
train = [80% of the samples], test = [20% of the samples]
Define ML algorithm
Fit the data
Predict the samples
Evaluate - F1
```

As described in Algorithm 2, for Benchmark 2 in Figure 1, an example of machine learning with a decision tree classifier depicts the workflow of the machine learning test example. First, parellelization is enabled for Dask and Spark and immediately after that the data is imported and modified accordingly to fit the test scenario. Next, the decision tree classifier is trained using various training data size, dividing the data set into a training subset and a test subset. The training subset represents 80% of the original dataset and for the training subset the remaining data is used, representing 20% of the original dataset. Each task is run with 5 different sample sizes, ranging from 50k to 250k samples, with a step of

50k samples. Finally, the execution report with the calculation times of each task is generated.

To realize these benchmarks[1], we used the extended MNIST or EMNIST dataset[2]. The data set contains approximately 250k samples of handwritten digits, resulting in total size of 516MB. The size of all images is exactly the same, 28 by 28 pixels and each pixel has a value ranging from zero to 255. The dataset is represented in the CSV (Comma Separated Values) format with the first column being the label and the rest of the columns representing 784 pixels. For the benchmarks, different data set sizes, ranging from 50k to 250k samples with a 50k step were generated.

In addition, each data set size was tested on Dask and Spark with 1, 2 and 4 workers. Therefore, the programs used to test computation time on Windows and Linux operating systems have the same complexity. All tests were performed on equivalent Windows and Linux virtual machines running on the 6 CPU core machine with 10 GB of RAM.

IV. Results

In this section we provide the results of the benchmarks collected using the methodology described in Section III.

A. *Import and merge*

First, we present in Figure 2 the import and merge times for 100k samples on Linux without parallelization across the the three platforms. In the first bar, it can be seen that importing (i.e. loading the data into memory) takes most of the time with Pandas. Merging (i.e. concatenation) is relatively negligible while computation is not relevant in this case as after merging it already returns the desired data structure. The total import and merge time is slightly above 4s.

From the second bar, it can be seen that importing and merging is negligible with Dask as doesn't load anything into memory at these steps, rather it prepares only recipes that will be executed during the most time consuming compute phase. During compute, Dask turns a lazy collection into its in-memory equivalent, in our case, the Dask dataframe turns into a Pandas dataframe. Overall, it can be seen that on a single node, Dask is comparable to Pandas, with a total import and merge time slightly below 4s.

Finally, from the last bar, it can be seen that Spark import and merge are very fast and efficient, taking below 2s. However, when transforming the internal data structure of Spark into pandas (i.e. during the compute phase in this case) is very time consuming. We added this step so that the final outcome is consistent with the other two (i.e. Pandas data structure), however in the end-to-end ML pipeline the ML algorithm will be trained directly using Spark's internal data structure.

Figure 3 shows how the import and merge times fare as a function of worker nodes for Dask across Linux and Windows. As expected, a decreasing tendency of the import/merge times with the increase of the working nodes can be seen. When testing Spark on the import and merge benchmark, both Windows and Linux ran out of memory with two and four workers. Swap memory could be used to overcome this shortcoming, however, the resulting comparison would not be fair because the Dask benchmarks didn't need the swap memory.

[1]Scripts for the benchmarks, https://github.com/sensorlab/parMLBenchmarks

[2]EMINST dataset - https://www.kaggle.com/crawford/emnist (accessed: 30.07.2022)

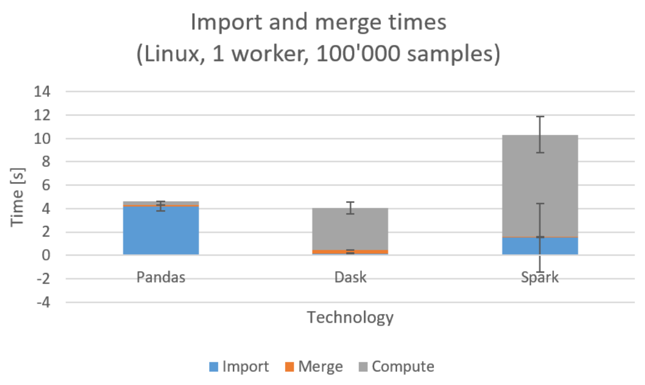

Fig. 2. Benchmark results of import and merge times at 100k samples: raw data to Pandas.

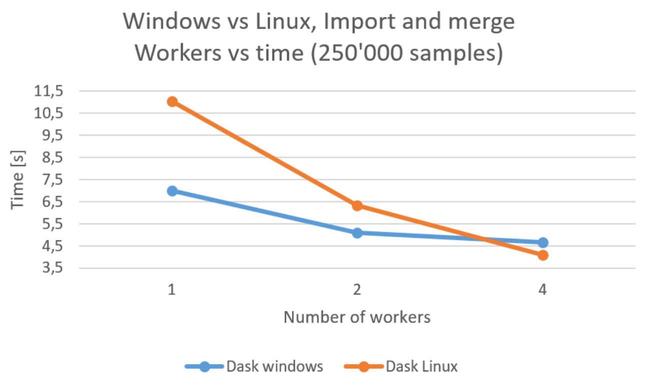

Fig. 3. Benchmark results two operating systems, Dask with import and merge on 250k samples.

B. *Machine learning*

Figure 4 shows the comparison of computation time between Dask, Spark, and SciKit on the Windows operating system for different dataset sizes. Each column in the figure represents the average computation time of 5 test runs. The results show that Dask and Spark are almost equivalent when the input dataset size is around 150k samples. Dask performs better on smaller datasets, while Spark's performance is best on larger datasets. Interestingly, SciKit outperforms both Dask and Spark on all dataset sizes, although it is not able to parallelize tasks. This is most likely because of the transfer times between nodes and the overall overhead of Dask and Spark. Since the datasets fit completely into the computer's memory, SciKit has no problems computing them, while Dask and Spark only cause unnecessary overhead. However, Dask and Spark are meant for large clusters with hundreds or even thousands of nodes, while SciKit is meant for computations on a single computer.

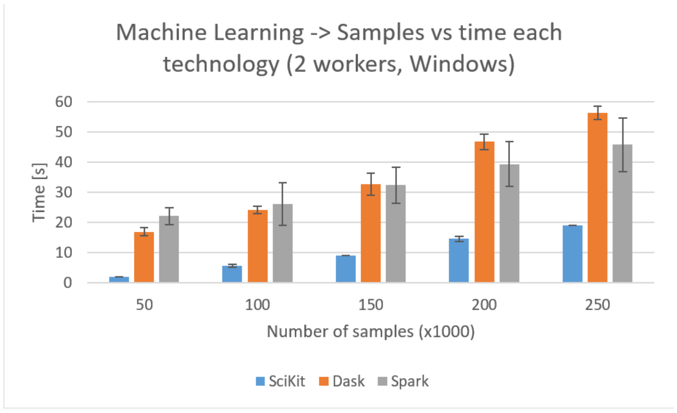

Fig. 4. Computational time for different dataset sizes on Windows operating system.

Figure 5 shows the results of the same experiment performed on the Linux operating system. Compared to the Figure 4, the results are very similar, with only difference that on Linux operating system Dask performs better then Spark even when input data set contains 150k samples.

Table I shows the F1 scores. An F1 score is the harmonic mean (alternative metric for the arithmetic mean) of precision and recall. The precision gives information on how many of the predicted samples that have been predicted as positive are correct. The recall gives information on how many of all positive samples the model managed to find.

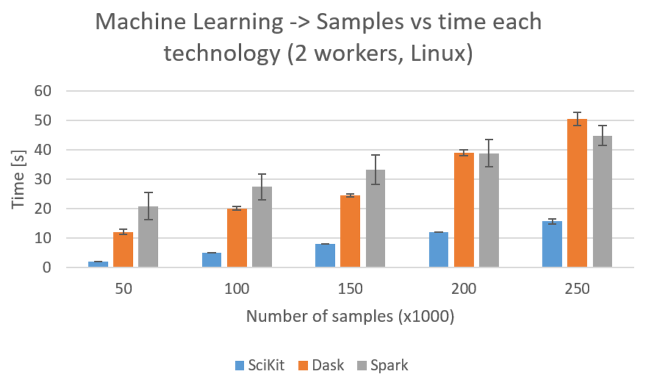

Fig. 5. Computational time for different dataset sizes on Linux operating system.

TABLE I
TABLE OF F1 SCORES FOR WINDOWS BENCHMARKS FOR VARIOUS SAMPLE SIZES (SIMILAR FOR LINUX).

	Number of samples (x1000)				
	50	100	150	200	250
Spark	0.71	0.73	0.73	0.71	0.71
Dask	0.71	0.72	0.73	0.71	0.70
Scikit	0.70	0.71	0.70	0.71	0.73

The machine learning benchmark measured the time to cast all columns into smaller data types. It seems that Dask has a dedicated function to cast all of the columns of a Dask dataframe at once whereas with the Spark function you have to cast each column one by one. The Dask casting was faster (0.06s) than Sparks (7.2s).

V. Conclusions

In this paper we benchmarked two parallel computing technologies, Dask and Apache Spark, against each other and against the single node SciKit Learn. The benchmarks were computed on the EMNIST dataset for various subsets from 50k to 250k samples on different operating systems and various degrees of parallelization. The results show a slight advantage on running the training pipeline on Linux rather than on Windows. Dask is seen as superior in dataframe manipulation while Apache Spark has a superior end-to-end processing performance on larger datasets with comparable final F1 scores.

Acknowledgments

This work was funded in part by the Slovenian Research Agency under the grant P2-0016.

References

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.
[2] M. Rocklin, "Dask: Parallel computation with blocked algorithms and task scheduling," in *Proceedings of the 14th python in science conference*, vol. 130, p. 136, Citeseer, 2015.
[3] S. Celis and D. R. Musicant, "Weka-parallel: machine learning in parallel," in *Carleton College, CS TR*, Citeseer, 2002.
[4] M. Dugré, V. Hayot-Sasson, and T. Glatard, "A performance comparison of dask and apache spark for data-intensive neuroimaging pipelines," in *2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*, pp. 40–49, 2019.
[5] P. Mehta, S. Dorkenwald, D. Zhao, T. Kaftan, A. Cheung, M. Balazinska, A. Rokem, A. Connolly, J. Vanderplas, and Y. AlSayyad, "Comparative evaluation of big-data systems on scientific image analytics workloads," vol. 10, p. 1226–1237, VLDB Endowment, aug 2017.
[6] M. H. Nguyen, J. Li, D. Crawl, J. Block, and I. Altintas, "Scaling deep learning-based analysis of high-resolution satellite imagery with distributed processing," in *2019 IEEE International Conference on Big Data (Big Data)*, pp. 5437–5443, 2019.
[7] M. T. S. J. William Cheng, Ioannis Paraskevakos, "Image processing using task parallel and data parallel frameworks," pp. 1–7, 2019.
[8] S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, and B. J. Peng, "Benchmarking streaming computation engines: Storm, flink and spark streaming," 2016.