dislib: Large Scale High Performance Machine Learning in Python

Javier Álvarez Cid-Fuentes, Rosa M. Badia
Barcelona Supercomputing Center, Barcelona, Spain
E-mail: {javier.alvarez, rosa.m.badia}@bsc.es

Keywords—machine learning, python, distributed computing, high performance computing

I. Extended Abstract

In recent years, machine learning has proven to be an extremely useful tool for extracting knowledge from data. This can be leveraged in numerous research areas, such as genomics, earth sciences, and astrophysics, to gain valuable insight. At the same time, Python has become one of the most popular programming languages among researchers due to its high productivity and rich ecosystem. Unfortunately, existing machine learning libraries for Python do not scale to large data sets, are hard to use by non-experts, and are difficult to set up in high performance computing clusters. These limitations have prevented scientists from exploiting the full potential of machine learning in their research. In this work, we present dislib [1], a distributed machine learning library on top of PyCOMPSs programming model [2] that addresses the issues of other similar existing libraries.

A. Overview

As said before, dislib is built on top of PyCOMPSs programming model. PyCOMPSs is a task-based programming model that makes the development of parallel and distributed Python applications easier. PyCOMPSs consists of two main parts: programming model and runtime. The programming model provides a series of simple annotations that developers can use to define potential parallelism in their applications. The runtime analyzes these annotations at execution time, and distributes the computation automatically among the available resources. In essence, dislib is a collection of PyCOMPSs applications that exposes two main interfaces to developers: a distributed data structure called distributed array (ds-array), and an estimator-based API. A ds-array is a 2-dimensional matrix divided in blocks that are stored accross different computers. Ds-arrays offer a similar API to NumPy [3], which is one of the most popular numerical libraries for Python. The difference between NumPy arrays and ds-arrays is that ds-arrays are internally parallelized and use distributed memory. This means that ds-arrays can store and process much larger data than NumPy arrays. Implementing a NumPy-like API makes ds-arrays easy to use and intuitive to developers already familiar with NumPy. It also allows to parallelize NumPy codes by just replacing NumPy arrays with ds-arrays.

Distributed arrays serve as a building block for machine learning algorithms included in dislib. These algorithms are exposed through an estimator-based API inspired by scikit-learn [4], a widely used machine learning library for Python. The typical dislib application consists of the following steps:

1) Load data into a ds-array
2) Instantiate an estimator object with parameters
3) Fit the estimator with the loaded data
4) Retrieve information from the estimator or make predictions on new data

Each machine learning model in dislib is enclosed in a class implementing the estimator interface, where an estimator is anything that learns from data that implements two main methods: `fit` and `predict`. Using the estimator abstraction provides a unified API across different algorithms, which reduces the complexity of dislib, and facilitates the implementation of high level applications and utilities that can use different algorithms in an interchangeable manner. Distributed arrays and dislib estimators allow non-expert developers to easily create distributed machine learning applications using regular sequential code. Figure 1 shows an example application that runs the K-means clustering algorithm [5].

```
from dislib.cluster import KMeans
from dislib.data import load_txt_file

x = load_txt_file("data.csv", block_size=...)
kmeans = KMeans(n_clusters=10)
kmeans.fit(x)
print(kmeans.centers)
```

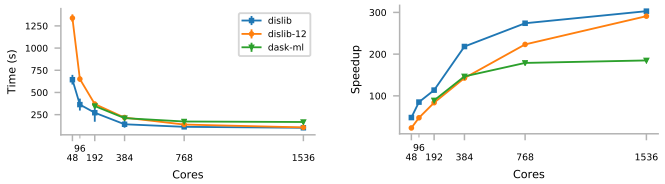
Figure 1: dislib code.

B. Performance evaluation

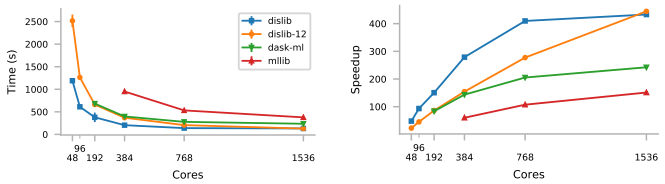
We compare dislib's performance to two similar libraries: MLlib and Dask-ML. We use these libraries because they share many characteristics with dislib, including an interface based on estimators, automatic data management and distribution, and support for HPC clusters. We use K-means as a benchmark, which is a popular unsupervised learning algorithm, and we run our experiments on the MareNostrum 4 supercomputer[1].

Figure 2 shows execution times and speedup of the K-means algorithm in MLlib, Dask-ML, and dislib. We experiment with two different granularities: 50 features and 50

[1] https://www.bsc.es/marenostrum



(a) 1 billion samples with low granularity (50 features and 50 clusters).



(b) 500 million samples with high granularity (100 features and 500 clusters).

Figure 2: Execution times and speedup of K-means.

clusters (low granularity), and 100 features and 500 clusters (high granularity). We run the algorithm for 5 iterations with randomly generated data divided in 1,536 partitions in all cases. Missing points in the plots mean that the execution failed due to memory issues. In the case of dislib, we run two sets of experiments: using 48 and 12 processes per node (labeled in Figure 2 as *dislib* and *dislib-12* respectively). This is to get a better comparison against Dask-ML, which can only use 12 processes per node due to memory limitations.

We see that dislib outperforms MLlib and Dask-ML both in terms of execution time and data size. More precisely, dislib can be up to 4 times faster than MLlib and Dask-ML. When running with 192 and 384 cores with 500 million samples, Dask-ML achieves similar performance to dislib with 12 processes per node. This means that some of the difference in performance between dislib and Dask-ML is due to dislib being able to utilize all the available cores. However, when running with 768 and 1,536 cores, dislib with 12 processes per node outperforms Dask-ML. This means that dislib scales better in terms of the number of nodes.

Our evaluation shows that dislib greatly outperforms MLlib and Dask-ML, both in execution time and ability to process large data sets. Moreover, running dislib using HPC queue systems like Slurm is completely automatic, whereas MLlib and Dask-ML require writing custom deployment scripts, and fiddling with configuration parameters to obtain good performance. Moreover, Dask-ML provides limited code portability as the IP of the Dask scheduler needs to be defined in the source code of the application. In terms of memory management, our experiments show that both Dask-ML and MLlib often raise out-of-memory errors when processing large data sets, while dislib is much more robust. Dask-ML can handle larger data sets than MLlib, but cannot use all the available computational resources. Unlike dislib, both MLlib and Dask-ML required manual configuration of the number of workers per node, and the amount of CPU and memory per worker. This makes MLlib and Dask-ML less accessible to non-experts, as finding the optimal configuration for a particular cluster is a difficult task that requires an extensive trial and error process. Moreover, this means that the performance of MLlib and Dask-ML is limited by the ability of the user to tune the configuration parameters.

II. Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement H2020-MSCA-COFUND-2016-754433. The research leading to these results has also received funding from the collaboration between Fujitsu and BSC (Script Language Platform).

References

[1] J. Álvarez Cid-Fuentes, S. Solà, P. Álvarez, A. Castro-Ginard, and R. M. Badia, "dislib: Large Scale High Performance Machine Learning in Python," in *Proceedings of the 15th International Conference on eScience*, 2019, pp. 96–105.

[2] F. Lordan, E. Tejedor, J. Ejarque, and et al., "ServiceSs: an interoperable programming framework for the Cloud," *Journal of Grid Computing*, vol. 12, no. 1, pp. 67–91, 2014.

[3] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy Array: A Structure for Efficient Numerical Computation," *Computing in Science & Engineering*, vol. 13, no. 2, p. 22, 2011.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Édouard Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[5] J. A. Hartigan and M. A. Wong, "Algorithm AS 136: A K-Means Clustering Algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

Javier Álvarez Cid-Fuentes is a researcher at the Workflows and Distributed Computing group of the Barcelona Supercomputing Center. His research interests include large scale distributed machine learning and parallel programming models for distributed infrastructures. Álvarez Cid-Fuentes received a Ph.D. in computer science from the University of Adelaide in 2018.