Reading Assignment 2

Authors:
Daniele Montesi, Francesco Staccone

Paper title: **MLlib*: Fast Training of GLMs using Spark MLlib**

1 Motivation

Running machine learning workloads in an efficient and distributed way is becoming more and more relevant because of the increasing availability of large datasets used to carry out highly computational demanding tasks, such as training deep neural networks.
In this context, the paper takes into account the increasing popularity of Spark, a unified analytics engine for big data processing developed by Apache that many companies use to extract and transform their data, but not for machine learning workloads. In particular, a small percentage of them actually use MLlib, an official built-in module for machine learning, while the rest prefer other specialized systems such as TensorFlow or XGBoost, even if this implies significant data movement overhead for dataset migrations from Spark.
That is due to the general belief that Spark is slow when it comes to distributed machine learning, so that the paper aims to understand the reasons for MLlib slowness figuring out if it is actually caused by architectural barriers or by implementation issues, and propose a solution.

2 Contributions

Focusing on training generalized linear models (GLM) as a case study, the paper reveals that are actually implementation issues rather than fundamental barriers that prevent Spark from achieving superb performance. Specifically, the study identifies **two major bottlenecks** in the MLlib implementation of gradient descent (GD), one of the most popular optimization algorithms used for training GLMs. They are:

- **Pattern of Model Update**, that is not efficient. In MLlib there is a driver node responsible for updating the (global) model, whereas the worker nodes simply compute the derivatives and send them to the driver. This is inefficient because the global model shared by the workers can only be updated once per communication step between the workers and the driver.

- **Pattern of Communication**, that can be improved. In MLlib while the driver is updating the model, the workers have to wait until the update is finished and the updated model is transferred back. Apparently the driver becomes a bottleneck, especially for large models.

3 Solution

By carefully designing and consolidating two state-of-the-art techniques in MLlib, the authors implemented a new machine learning library on Spark called MLlib*. They address the issues mentioned in the previous section by leveraging on **model averaging**, a widely adopted technique in distributed machine learning systems. The basic idea behind it is that each worker updates its local view of the model and the driver simply takes the average of the local views received from individual workers as the updated global model. In this way the global model is actually updated many times per communication step and therefore the number of communication steps towards convergence is reduced. Moreover, model averaging can be performed in a distributed manner across the workers, so that the centralized MLlib implementation based on the driver can be completely removed, reducing latency. The basic idea

of this **distributed aggregation** is to partition the global model and let each executor own one partition, so that all the executors participate in the distributed maintenance of the global model simultaneously.

4 Strong Points

The paper is well-explained by the authors. Among the strengths, there are:

- **Communication improvements** between the driver and the nodes are widely shown in the **Solution** section of this review and represent the main strenght of the paper. Thanks to the **model averaging** and **distributed aggregation**, one communication step of MLlib* corresponds to *many* updates of the Global model, aiming for a faster convergence that is really evident on the benchmark comparison section of the paper;
- **Lower latency** between communication steps has been achieved by MLlib* after a two-phases procedure: *Reduce-Scatter* and *AllGather*. The procedure let MLlib* achieve a great improvement if compared with the *MLlib + model averaging* system. The comparisons are made visibly clear by the author thanks to the "Fig. 3B" and "Fig. 3C" showed in the paper;
- **Exhaustiveness of the background** lets the reader understand all the steps followed by the authors. Starting from the distributed MGD description, the well-explained pseudo-codes, and the figures, it is possible to easily understand the general distributed mechanism for training an algorithm. Secondly, the authors introduce the existing distributed machine learning systems and compare different approaches underlining the similarities and the differences, as well doing accurate benchmark comparisons. For instance, the authors found out that in *Petuum*, the *model summation* procedure was performing slowly and decided replace the model summation with the *model averaging* (which worked the best with MLlib* as well) calling the new system *Pentuum**. The authors decided also to fairly benchmark this new system.

5 Weak Points

Three drawbacks of the paper are here listed and discussed:

- **System may decrease performance when regularization is not null**. In particular, in the paper is shown that when the objective function is too determined, the gap between the performances of MLlib* and its predecessor become smaller. This happens with all the 4 datasets. For instance, the loss of performance for *avazu* is 94.5% of the timing for convergence, passing from 123x (obtained with L2 = 0) to 7x time (L2 = 0.1);
- **Poor description** of some results obtained benchmarking MLlib* over the other existing platforms. The performance of the MLlib* is pretty different on every dataset. I would have expected more details explaining why one dataset performs better than the others in the opinion of the authors. Moreover, the authors do not explain why in some datasets (kddb, url) MLlib does not converge.
- **MLlib* takes more iterations to converge** due to the larger overhead that it needs in comparison with MLlib. This overhead is given on the one hand because MLlib* needs to pass the entire dataset on every iteration (MLlib passes a smaller batch); on the other hand, because if model size is small, communication steps are also less and hence the benefits achieved by the **Allreduce** procedure are less effective.