EPJ Web of Conferences **245**, 08016 (2020) https://doi.org/10.1051/epjconf/202024508016
CHEP 2019

Delivering a machine learning course on HPC resources

Stefano Bagnasco[1], *Gabriele Gaetano* Fronzé[1], *Federica* Legger[1,*] *Stefano* Lusso[1], and *Sara* Vallero[1]

[1]Istituto Nazionale di Fisica Nucleare, via Pietro Giuria 1, 10125 Torino, Italy

Abstract. In recent years, proficiency in data science and machine learning (ML) became one of the most requested skills for jobs in both industry and academy. Machine learning algorithms typically require large sets of data to train the models and extensive usage of computing resources, both for training and inference. Especially for deep learning algorithms, training performances can be dramatically improved by exploiting Graphical Processing Units (GPUs). The needed skill set for a data scientist is therefore extremely broad, and ranges from knowledge of ML models to distributed programming on heterogeneous resources. While most of the available training resources focus on ML algorithms and tools such as TensorFlow, we designed a course for doctoral students where model training is tightly coupled with underlying technologies that can be used to dynamically provision resources. Throughout the course, students have access to a dedicated cluster of computing nodes on local premises. A set of libraries and helper functions is provided to execute a parallelized ML task by automatically deploying a Spark driver and several Spark execution nodes as Docker containers. Task scheduling is managed by an orchestration layer (Kubernetes). This solution automates the delivery of the software stack required by a typical ML workflow and enables scalability by allowing the execution of ML tasks, including training, over commodity (i.e. CPUs) or high-performance (i.e. GPUs) resources distributed over different hosts across a network. The adaptation of the same model on OCCAM, the HPC facility at the University of Turin, is currently under development.

1 Introduction

Data science is one of the fastest growing fields of information technology, with wide applications in key sectors such as research, industry, and public administration. The volume of data produced by business, science, humans and machines alike has been growing exponentially in the past decade, and it is expected to keep on following this trend in the near future. In the next decade research infrastructures such as the High Luminosity LHC (HL-LHC) or the Square Kilometer Array (SKA) are expected to produce a comparable amount of data with respect to the leading technology companies (Facebook, Google, Netflix, Apple, Amazon), with each experiment producing tenths of Exabytes of data. Even more data will be produced by the growing number of interconnected devices (the so called Internet of Things, IoT). Skills to handle, manage and process large amounts of data are quickly

*e-mail: federica.legger@cern.ch

becoming paramount in the formation of young physicists, to prepare them for the upcoming challenges in both the scientific and industry domains.

Machine Learning (ML) techniques are becoming increasingly popular and have proven to be effective in extracting information from large datasets. ML is based on a set of algorithms that are able to learn patterns and relationships directly from the data without relying on rule-based programming. Deep Learning (DL) is a subset of ML algorithms designed to learn complex functions mapping the input to the output, by exploiting multiple layers to progressively extract higher level features from the raw input. The training and optimisation of ML and DL models may require substantial computing power, often exceeding that of a single machine. On the other hand, substantial computing resources may nowadays be available through university clusters, HPC centers, or opportunistic resources such as public or commercial clouds, and it is desirable that students learn how to leverage such opportunities. In addition, GPUs are particularly performant on matrix operations such as those executed during the training phase of DL models, but using them efficiently might not be so straightforward.

There is an abundance of educational resources and academic training dedicated to ML and DL models, however notions of distributed computing and paralleling processing are often not sufficiently acquired in standard courses. At the University of Turin we introduced for the academic year 2018-2019 a new course for doctoral students, with the title "Big Data Science and Machine Learning", to bridge the gap between the development and optimisation of ML models, and the exploitation of a distributed computing infrastructure. In the following sections we will discuss the course goals and program, the hands-on sessions, the computing infrastructure used throughout the course, and future developments.

2 The course

The course "Big Data Science and Machine Learning" aims at giving doctoral students a brief introduction to modern data science (including some of the most popular and recent developments in ML and DL methods), and then focus on the technical implementation of various ML algorithms. We discuss key aspects such as parallelisation of the ML model training, and the deployment of ML applications on distributed resources and different architectures (CPUs and GPUs). For the academic year 2018-2019 we had a total of ten hours of lessons, where about half of the time was spent in hands-on sessions to allow students to become familiar with several ML libraries. In addition students received a brief introduction to a variety of topics relevant to modern scientific computing, such as collaborative analysis tools and reproducibility, continuous integration and the *devops* approach, virtualisation and orchestration technologies. Various computing infrastructures (opportunistic, HPC or HTC - High Throughput Computing), together with the fundamental principles of distributed file systems and frameworks for distributed computing are discussed.

For the academic year 2019-2020 we will increase the amount of hours to sixteen, allowing us to give in addition a basic introduction to the current computer architecture, with a focus on parallel computing paradigms aimed at the exploitation of the full potential of parallel architectures. An overview of the fundamental OpenMP [1] and MPI [2] coding patterns will be added to the hands-on sessions. The program of the course will be the following:

- Introduction to big data science;
- The big data pipeline: state-of-the-art tools and technologies;
- ML and DL methods: supervised and unsupervised training, neural network models;

- Introduction to computer architecture and parallel computing patterns;
- Initiation to OpenMP and MPI;
- Parallelisation of ML algorithms on distributed resources;
- Beyond the CPU: ML applications on distributed architectures and GPUs.

Since the novelty of this course is the focus on hands-on sessions and on efficient programming on distributed architecture, we will give more details on the hands-on content and on the computing infrastructure used by the students.

2.1 The hands-on sessions

Widely used open source tools and libraries were introduced during the hands-on sessions, to enhance the possibility that the students will profit from what they learned also outside of the scientific environment. The sessions were based on Jupyter [3] notebooks written in Python, which is a de-facto standard in data science. The input dataset used throughout the hands-on session is an open dataset publicly available at the UCI ML repository [4]. The dataset [5] contains ten millions of Monte Carlo generated events produced in two benchmark processes, referred to as signal and background, in proton-proton ($p - p$) collisions at the Large Hadron Collider (LHC). About half of the events are labelled as signal, and the other half as background. Signal events were generated by simulating a process where a heavy charged theoretical Higgs boson is produced in $p - p$ collisions together with a W boson, and decays through a Standard Model Higgs boson (h_0) and another W boson. The h_0 decays to two b-quarks, one W decays leptonically to either an electron or a muon in association with a neutrino, and the other W decays hadronically into two quarks. Background events were generated by simulating the production of $t\bar{t}$ quark pairs in $p - p$ collisions. Their decay products are the same as those of the signal process: two W bosons and two b quarks. As an example, the invariant mass distribution of the two reconstructed b quarks for signal and background events, as obtained by the students during the first hands-on session, is shown in Fig. 1.

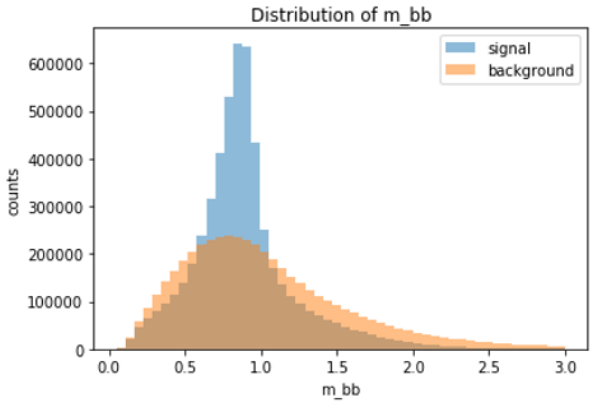

Figure 1. The invariant mass distribution (denoted as m_{bb}) of the two reconstructed b quarks for signal (blue) and background (orange) events.

The students applied various ML models for supervised classification (gradient boosting trees, multilayer perceptron classifier, sequential neural networks) to distinguish signal

from background events using several ML frameworks (scikit-learn [6], Spark MLLib [7], Keras [8], Intel BigDL [9]). They learned how to measure the performances of the various models (using standard metrics such as accuracy, ROC - Receiver Operating Curve, AUC - Area Under the Curve), how to perform model parameter training, how to avoid overfitting and underfitting, and how to tune the model hyper-parameters.

The students used Spark [10] to read the input dataset from HDFS [11] and to distribute the training of the ML models and the hyperparameter optimisation on a shared computing cluster on local premises. Spark is a general-purpose distributed data processing engine that provides high level APIs in several languages (Scala, Python, Java, R, and SQL) and a distributed data object, the dataframe, which is at the core of Spark distributed processing. Spark applications are handled by a driver process, which converts the user code into a set of multiple tasks, and delegates them to the executor processes that are in charge of executing the tasks on separate nodes in the cluster. The parallelisation of the user code is therefore handled by Spark, and is transparent for the user.

3 The computing infrastructure

Throughout the course, students have access to a dedicated cluster of computing nodes. The cluster comprises five physical servers and ten virtual machines hosted at the INFN Torino Cloud, for a total of 216 cores, 1.9 TB of RAM and 2.3 TB of disk, linked by a 1 Gbps Ethernet connection. Task scheduling is managed by an orchestration layer (Kubernetes [12]), leveraging Docker [13] containers to define and isolate the runtime environment for several virtual clusters (one for each student). The virtual clusters are accessed through a web interface based on Jupyter Hub [14]. When a user authenticates on the Hub through GitHub OAuth [15], a notebook server is created as a containerized application. A set of libraries and helper functions is provided to execute a parallelized ML task by automatically deploying a Spark driver and several Spark execution nodes as Docker containers. This solution automates the delivery of the software stack required by a typical ML workflow and enables scalability by allowing the execution of ML tasks, including training, over commodity resources distributed across a network.

A custom Kubernetes *operator* [16] was developed to grant a minimum number of executors to each tenant, whilst not enforcing static quotas. Instantiating multiple executors per tenant allows for the transparent execution of parallel tasks through Spark.

The operator controls two types of Custom Resource Definitions (CRDs):

- The "Farm" CRD is a collection of *pods* matching a given selector (defined by a namespace and a label). Pods are the smallest deployable units of computing that can be created and managed in Kubernetes. The selector identifies the Spark executor pods belonging to a given user. A Farm resource is created for each user and it keeps track of the number of Spark executors created for that user by the Spark driver. The custom Kubernetes operator implements the scale-down functionality for the Farm resource: whenever a scale-down trigger is sent to a given Farm, the number of pods belonging to that Farm is reduced, while granting a configurable minimum number of pods to be kept alive (quota).
- The "FarmManager" CRD keeps track of the status of all Farms in the cluster, therefore a single instance of this resource is deployed in the Kubernetes cluster. The FarmManager sends a scale-down trigger to those Farms found over quota when other Farms are requesting resources and are below their own quota. The number of deleted pods per Farm is proportional to the number of pods over quota for that Farm. This simple but effective algorithm will be refined in the future.

The Spark driver of each user tries to scale up the number of executor pods in order to match the user request. It is therefore able to occupy all available resources in the cluster. The FarmManager un-deploys exceeding executors only when needed to grant the configured minimum number of executors to other tenants.

3.1 Scaling tests

We performed scaling tests to optimise two parameters of the Spark cluster, namely the number of cores per executor and the total number of cores, which is given by the number of executors multiplied by the number of cores per executor. The tests were reviewed during the course, and the students were asked to perform them at a smaller scale.

To optimise the number of cores per executor, we trained two ML models (Gradient Boosting trees - GBT, and Multilayer Perceptron Classifier - MCP) using Spark MLLib on a single machine with 28 physical cores (56 with hyper threading) and 260 GB RAM with a reduced dataset containing one million events. We set the number of executors equal to one and varied the number of cores per executor (with hyper threading enabled). We measured the strong scaling efficiency, defined as

$$\epsilon = \frac{t(1)}{N \cdot t(N)}$$

where $t(1)$ is the training time with one core, $t(N)$ is the training time with N cores, and N is the number of cores. As can be seen in Fig. 2a, the performances of the GBT model training do not scale very well when increasing the number of cores. This is to be expected, since the implementation of parallel algorithms for building decision trees is a notoriously complex task [17]. The strong scaling efficiency of the MPC model training is greater than 80% up to five cores, and then linearly decreases when increasing the number of cores further. We therefore set the number of cores per executor equal to five when using MLLib.

To optimise the total number of cores per Spark application, we used four identical machines with 28 cores and 260 GB RAM. We compared the performances of training two ML models: the MPC model from Spark MLLib that we used in the previous scaling tests, and a sequential neural network (NN) trained with Intel BigDL. Note that the BigDL library requires the number of cores per Spark executor to be set to one, whereas for the MPC model we set the number of cores per executor to five, as a result of the previous test. The scaling performances of the two models have a similar behaviour, but in general the strong scaling efficiency of the NN is higher than that of the MPC (see Fig. 2b). We set for both models the optimal number of cores equal to 25, in order to have a strong scaling efficiency greater than 50%, and guarantee a minimum number of cores to all students.

3.1.1 Details of the benchmark ML models

The hyper-parameters of the benchmark ML models used in the scaling tests are:

- MLLib GBT: maximum number of iterations equal to 50, maximum tree depth equal to 10;
- MLLib MPCb: one input layer of size equal to 28, two intermediate layers of size equal to 30;
- BigDL NN: trained for 50 epochs, one hidden layer, 100 units per layer, with a batch size equal to 3200.

The training performances obtained on 25 cores and one million events can be found in Table 1.

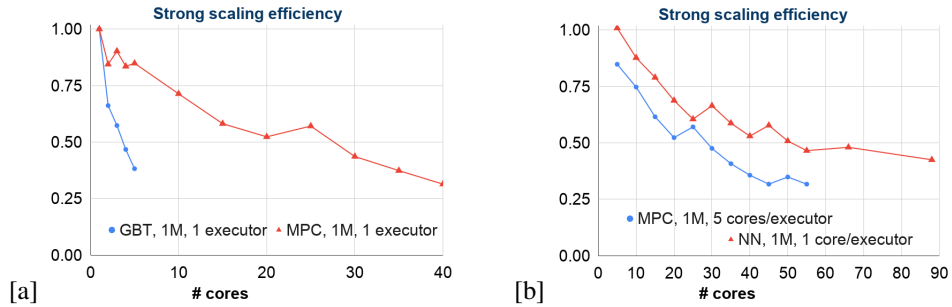

Figure 2. [a]: Strong scaling efficiency measured on a single machine, obtained by measuring the training time of two MLLib ML models (GBT, blue circles, and MPC, red triangles) as a function of the number of cores. The number of Spark executors is set to one. [b]: Strong scaling efficiency measured on a homogeneous cluster, obtained by measuring the training time of two ML models from MLLib (MPC, blue circles) and BigDL (NN, red triangles) as a function of the number of cores. The number of cores per Spark executor is set to five for the MLLib model, and to one for the BigDL model.

Table 1. Training time in minutes and AUC in percentage for the three ML models used in the scaling tests.

Model	Training time [m]	AUC [%]
GBT MLLib	1	81%
MPC MLLib	1	74%
BigDL NN	14	81%

4 Future developments: HPC and OCCAM

The *Open Computing Cluster for Advanced data Manipulation* (OCCAM) [18] of the Torino University and INFN section is a unique HPC facility managed in a rather unconventional manner. It relies on container-based cloud-like technologies, where computing applications are run on virtual clusters deployed on top of the physical infrastructure. This is achieved with Docker and Apache Mesos as virtualization and orchestration layers respectively.

This particular management strategy allows OCCAM the flexibility to accommodate different computing models, which currently are: batch execution, multi-step pipelines, interactive workstations. The transition to a Kubernetes orchestrator is ongoing, which will allow, together with other improvements, the seamless inclusion of an additional computing model: the ML framework described in Section 3. In this way, the execution of computing-intensive ML workflows will benefit from the availability of high-performance computational resources, including GPUs. In the 2018/2019 edition of the course the transition to Kubernetes was not yet achieved for the OCCAM facility, but it will for future replicas of the course.

5 Conclusion

Since last year, a new course on ML methods has been activated at the University of Turin for doctoral students. During the course, the students learn the basics of ML and DL, with particular emphasis on the implementation of the algorithms on distributed computing resources. A dedicated cluster is setup on local premises, and the students learn how to distribute data processing tasks such as model parameter training on several nodes using

Spark. Task scheduling on the computing cluster is managed by Kubernetes leveraging Docker containers. Work to port the same model on OCCAM, the HPC facility at the University of Turin, is currently on-going. This will allow the students to access also OCCAM resources, such as fast CPUs or GPUs, for the next edition of the course.

6 Acknowledgements

This work has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement LHCBIGDATA No. 799062.

References

[1] OpenMP, *https://www.openmp.org/*
[2] MPI, *http://www.mpi-forum.org*
[3] Jupyter, *https://jupyter.org*
[4] The UCI repository, *https://archive.ics.uci.edu/ml/index.php*
[5] Baldi, P., P. Sadowski, and D. Whiteson, Nature Communications **5** (2014)
[6] The SciKit library, *https://scikit-learn.org/stable/*
[7] MLLib, *https://spark.apache.org/mllib/*
[8] Keras, *https://keras.io/*
[9] bigDL, *https://bigdl-project.github.io/*
[10] Apache Spark, *http://spark.apache.org*
[11] Apache Hadoop, *http://hadoop.apache.org*
[12] Kubernetes, *https://kubernetes.io/*
[13] Docker, *https://www.docker.com/*
[14] JupyterHub *https://github.com/jupyterhub/jupyterhub*
[15] OAuth, *https://oauth.net/*
[16] Kubernetes Farm Operator, *https://github.com/svallero/farmcontroller*
[17] Amado N., Gama J., Silva F., Lecture Notes in Computer Science, **2258** (2001)
[18] M. Aldinucci, S. Bagnasco, S. Lusso, P. Pasteris, S. Rabellino, and S. Vallero, Journal of Physics: Conference Series, 898 **(8):082039** (2017)