A Performance Prediction Model for Spark Applications *

Muhammad Usama Javaid, Ahmed Amir Kanoun,Florian Demesmaeker, Amine Ghrab and Sabri Skhiri

EURA NOVA R&D
Mont-Saint-Guibert, Belgium
Email: firstname.lastname@euranova.eu

Abstract. Apache Spark is a popular open-source distributed processing framework that enables efficient processing of massive amounts of data. It has a large number of configuration parameters that are strongly related to performance. Spark performance, for a given application, can significantly vary because of input data type and size, design & implementation of algorithm, computational resources and parameter configuration. So, involvement of all these variables makes performance prediction very difficult.In this paper, we take into account all the variables and try to learn machine learning based performance prediction model. We ran extensive experiments on a selected set of Spark applications that cover the most common workloads to generate a representative dataset of execution time. In addition, we extracted application and data features to build a machine learning based performance model to predict Spark applications execution time. The experiments show that boosting algorithms achieved better results compared to the other algorithms.

1 INTRODUCTION

Nowadays it is common for companies to face big data challenges, as they deal with increasingly large and complex data. Yet, they often struggle to reduce the processing time of such amount of data. Spark is a popular distributed processing framework used in big data. However, it comes with too many parameters (above 200 parameters) that need to be tuned to get the best performance. Hence, it is not easy to manually choose the values of the parameters that will lead to the best performance. Also, not all the parameters are equally important and does not have the same impact on performance. So more often in practice, only a subset of spark parameters is tuned.
In this paper, we discuss the influence of a subset of Spark parameters on the commonly used applications in industry. We present our work in three main steps. First,

* The elaboration of this scientific paper was supported by the Ministry of Economy, Industry, Research, Innovation, IT, Employment and Education of the Region of Wallonia (Belgium), through the funding of the industrial research project Jericho (convention no. 7717).

a framework to run a set of experiments in Spark and to record aggregate measures of execution time. Second, we discuss the selection of spark parameter and extraction of spark job features to capture the job based effect on execution time. Third, a performance model to predict spark application execution time regarding a set of spark parameters, application features and input data size.
Therefore, through this paper, we aim to build a robust performance model using machine learning algorithms to predict the execution time of a given Spark application.To train a machine learning model, we generated an execution time dataset of various spark applications. Different spark applications were executed with different sets of spark platform parameters along with various sizes of input data and execution time was recorded. The most widely used spark jobs were selected for the generation of the dataset. As shown in [1], some of the existing works tried to predict Spark application performance using execution time as measure. However, the work is limited to a small subset of algorithms and did not explore the application native features.
To overcome this limitation, we built a performance prediction framework that, given a Spark application and the chosen configuration values, extracts the application features and combine them with the dataset to predict the execution time of the application. The contributions of our work can be summarized as follows:

- We built a representative training dataset by identifying the most common Spark application families and running extensive experiments with these families.
- We extracted applications features at different levels (application, stage and job) and input data features to built a performance model using machine learning algorithms to predict Spark applications execution time.
- We built different machine learning prediction models to predict execution time of a Spark application and discussed their accuracy.

The remaining part of this paper is structured as follows: section 2 reviews the state of the art in the domain of performance models for distributed processing frameworks. Section 3 presents overview of the framework and building blocks of the framework. Section 4 discusses the results and Section 5 concludes the paper.

2 RELATED WORK

In the recent times, big data frameworks have been very popular as data being produced is increasing rapidly. Many big data frameworks have emerged to process this ocean of data. Among these big data analytics platforms, Apache Spark[2], an open source framework with implicit data parallelism and fault tolerance, is the most popular framework for big data analytics. Due to its efficiency and in-memory computation, Apache Spark is preferred over other frameworks like Hadoop which is slower because of its disk based processing.

Performance prediction for big data analytics platforms is one of the emerging domains that received so much attention in the recent time. The performance of an application is affected by the settings parameters of the framework. There has been a significant number of researches on performance prediction for various big data

analytics frameworks, in this paper we are focusing on performance prediction for Apache Spark.

Most of the work in the state of the art focused on MapReduce computing framework or Hadoop platform. Starfish[3] uses a cost-based modeling and simulation to find the desired job configurations for MapReduce workloads. AROMA [4] proposes a two phase machine learning and optimization framework for automated job configuration and resource allocation for heterogeneous clouds. Hadoop's scheduler can degrade the performance in heterogeneous environments and introduced a new scheduler called Longest Approximate Time to End in [5]. In [6], the author analyzed the variant effect of resource consumption for map and reduce. In [7], an automated framework for configuration settings for Hadoop and in [8], the presented KMeans based framework recommends configuration for new Hadoop jobs based on similar past jobs which have performed well.

To the best of our knowledge, there is not a lot of work on the Apache Spark performance prediction. A simulation-driven prediction model has been proposed in [9], in order to predict a Spark job performance. In [10], the authors proposed a support vector regression (SVR) approach for an auto-tuning model and they shown its efficiency in terms of computation. In [11], the authors propose a Spark job performance analysis and prediction tool, by parsing the logs of small scale job and then simulating it for different configuration for performance prediction.

Recently, other machine learning based algorithms have shown significant performance. In [12], the authors selected, what they considered as, the 12 most important Spark parameters and shown a significant performance optimization by tuning them. In [13], the authors proposed an approach to reduce the dimensionality. First, a binary classification model is built to predict the execution time of an Apache Spark application under a given configuration. Then, they used second classification model to define the range of performance improvements.

In [14], the authors proposed a parameter tuning framework for Big data analytics with time constraint. In this work, a Spark application first needs to run at a smaller scale with different parameters - called autotune testbed - capturing the effect of parameters on execution time. Then, an autotune algorithm finds the best parameters configuration for an application within a time constraint. In [15], authors present a machine learning approach to predict execution time of a Spark job. For predicting the execution time, they extract the job level features from Directed Acyclic Graph (DAG) of Spark job and combine it with cluster features naming number of machines, number of cores and RAM size to train a machine learning model. Their work is limited to predicting execution time of SQL queries and Machine Learning workloads.

In this paper we propose to learn a Spark performance function. In our approach we do not need to run the application before estimating its performance. Our algorithm applies prior knowledge learned on a dataset that we tried to make as representative as possible of the reality. In this context, for a given Spark application, configuration, resource and input data we can predict the application performance.

3 OVERVIEW

In this section, we present the experimental framework for performance prediction of Spark platform. We explore machine learning algorithms and several aspects of the experimental framework such as feature selection, data collection methods and techniques, training, testing and evaluation. Finally, we evaluate these models w.r.t. accuracy of predicting the execution time (performance) and their sensitivity to various variables such as the Spark configuration parameters, training data size or even the application characteristics.

3.1 An experimental framework for performance prediction of Spark

A performance model for Spark is required in order to predict the execution time of a Spark job. Here is the Equation (1) proposed in [16] to denote the performance prediction.

$$perf = F(p, d, r, c) \tag{1}$$

Where *perf* denotes the performance of a Spark application p processing d as input data, running on r resources under the configuration c. As a result, F is the function we want to approximate.

In this paper, our work focuses on performance prediction of a Spark application. Here, the performance is the execution time of a Spark application. In order to learn such a performance model, we needed first to have a dataset representative of the typical Spark applications. Our intuition is that if we are able to create such a dataset, we could end up with an algorithm able to generalise what it learned to predict the performance of any new Spark applications. Therefore, based on our industrial experience in this field, we selected different types of Spark applications: (1) data science workload such as Kmeans, Binomial Logistic Regression, Decision Tree Classifier, Linear Regression,(2) data integration such as SQL Groupby, sorting data, (3) Graph processing such as PageRank, Single Source Shortest Path, Breadth First Search in graphs, (4) general workload such as wordcount. In the rest of the paper, we call these four types of workdloads the four *families*. We ran extensive experiment on these Spark applications and used the approach described in [14] to collect the data. We evaluated two approaches for modeling the performance function. First, we tried to train a dedicated algorithm for each of the four workload family. We had the intuition that a single model would be less accurate in predicting the performance for so many types of workloads. However, in order to valid this first assumption, we tried to train a single model for any kind of applications. We found out that the single model approach was actually way better. In order to create an extensive data set, we ran a significant number of experiments. For a given application p, we ran different input data d, on different resources r with different configurations c. All variables from equation 1 have been used in a systematic way to cover all the involved aspects of Spark application affecting the execution time. We divided the result data into four datasets w.r.t. the family it comes from. We evaluated the performance prediction of our algorithm as the ability to correctly predict the execution time.

3.2 Parameter selection

Table 1: Spark Parameters and Value Ranges

Parameter	Default	Search Space	Description
Spark.executor.cores	1	1-8	Number of cores assigned to each executor
Spark.executor.memory	1 GB	1 G-8 G	Size of the memory allocated to each executor
Spark.shuffle.compress	True	True/False	Whether to compress map output files
Spark.shuffle.spill.compress	True	True/False	Whether to compress data spilled during shuffles
Spark.io.compression.codec	lz4	lz4/lzf/snappy	The codec used to compress internal data such as RDD partitions, event log, broadcast variables and shuffle outputs.
Spark.reducer.maxSizeInFlight	24m	24m-72m	The max size of map output each reducer could fetch simultaneously

In this work, we have selected 6 Spark configuration parameters shown in the table 1, along with the size of the input data and features extracted from execution of Spark applications. The default columns contains the default values for Spark configuration parameters. The search space column defines the range of parameter values considered and explored. These six configuration parameters are selected from [1] [12][13][14].In these papers, the authors selected a small subset of spark parameters for parameter tuning. Out of the selected parameters, 4 parameters: spark.shuffle.compress, spark.spill.shuffle.compress, spark.io.compression and spark.reducer.maxSizeInFlight parameters are used in previously mentioned 4 papers. the 2 other parameters naming spark.executor.cores and spark.executor.memory are common in 3 papers and given their direct relevance, made them to make the list for parameters to be considered. The reason to limit to 6 parameters is to cover the effect of important parameters thoroughly. Nevertheless, parameter selection for spark platform not only a different domain related to Spark platform. This research domain can also be targeted for Spark platform optimization and could possibly lay the foundation for future work.

3.3 Dataset Generation

Ten different Spark workloads naming KMeans, Binomial Logistic Regression, Decision Tree Classifier, Linear Regression, WordCount, Sorting, PageRank, Single Source Shortest Path, Breadth First Search and Groupby were selected to build performance prediction model for Spark. In this paper, we call Spark execution data the result of a spark application run where we collect performance metrics. In order to collect such execution data for different workloads, we adopted an approach inspired by [14]. Experiments are run starting from minimum resources for Spark platform and small input datasets. As the parameter search space is large and could not be explored entirely, we divided it into bins based on domain knowledge and expert opinion. In addition, we also used Bayesian optimization to identify the potential sub-space where to run more experiments. Experiments were scaled up with respect to resources for the Spark and input dataset size. The strategy is to capture the complex relationship between spark parameters and input datasets size for different spark applications. Also, each experiment is run 3 times in order to capture the variance in performance. These experiments were run on Google Kubernetes Engine.

3.4 Application features selection

In order to build a robust prediction model, application native features were extracted and included in dataset. Based on our experimentation and state-of-the-art,taking into account the input dataset size only, is not enough. Thus, we aimed at leveraging significant information regarding how Spark deals with a given application and an input dataset. To do so, we rely on the execution graph of a Spark application. This graph is called the Directed Acyclic Graph (DAG). With in a DAG, the nodes represent the Spark internal state (RDD) while the edges represent the actions to transform an RDD to another. Spark application DAG is leveraged because it contains the application complexity per se.
Since we want to build a machine learning model to predict execution time, the DAG needed to be summarized for our model. In fact, in Spark core, Spark Driver is responsible for creating execution plans, also known as Jobs, for any Spark application. Once the plan is created, each job, represented by a DAG, is divided by the driver into a number of stages. These stages are then divided into smaller tasks before sending them to workers (executors) in order to be executed. Spark prepares the logical plan on the Spark Driver before the execution step, this plan defines the steps and operation to be performed for the execution of an application. Fortunately, Spark platform has its history Server's web UI from which the following features were extracted:

- Number of jobs
- Aggregate
- Collect
- Count
- Number of completed tasks
- Number of completed stages

Execution time as a performance measure Several measures can be taken into consideration for a batch parallel processing job like speedup, efficiency and redundancy. However, those performance measures can be related to a benchmark between one processing unit and multiple processing units or comparing sequential execution to parallel execution. In this work, we focus on recording execution time as a performance measure for the experiments we run.

3.5 Performance model

The purpose of this work as mentioned above is to predict the execution time of a given Spark application under specified spark configuration. In this section, we present the performance model used to make this prediction.
The dataset is composed of 3 kinds of features: Spark parameters, application features and input data size as a dataset feature in addition to applicatoin execution time as target. Each row of the data represents an experiment run on a cluster with given Spark configuration, a Spark workload and a dataset. Given the complex relationship and vast range of spark parameters adds to the complexity of the problem,

making it harder to predict the execution time with accurary.
Different machine learning regressor algorithms: Linear Regression, Boosting, Random Forest and Neural Networks were implemented to build a performance model.

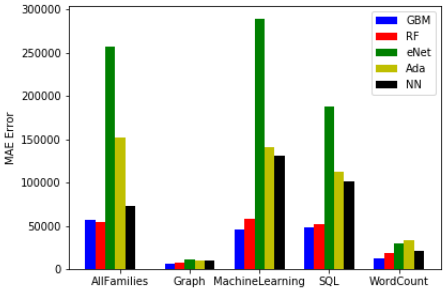

Fig. 1: Accuracy MAE application features ingested

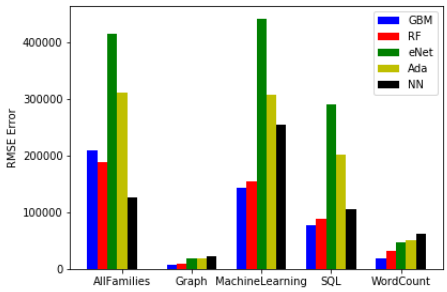

Fig. 2: Accuracy RMSE application features ingested

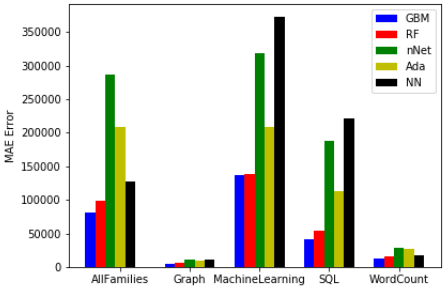

Fig. 3: Accuracy MAE application features not ingested

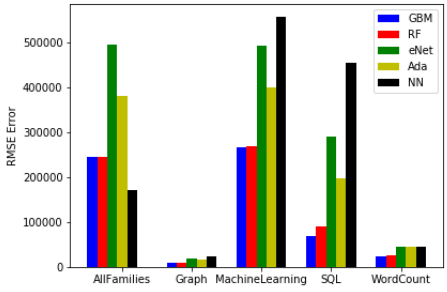

Fig. 4: Accuracy RMSE application features not ingested

4 Experimental Evaluation

4.1 Experimental Setup

In this work, each experiment is an execution of a Spark application with defined set of Spark parameters values, application features and dataset feature (size). We proceed in following steps: First, all selected spark applications were run with all the possible combination of selected spark parameters. Each experiment was run 3 times in a Kubernetes cluster. Spark application features are extracted for each

application. Finally, all the experiments results are appended to create a dataset for machine learning algorithms.
We used Google Kubernetes Engine with 5 nodes with three different resource configuration according to the size of the input dataset. The three different resource configurations are 1 core & 1 GB RAM per node, 4 cores & 4 GB RAM per node and 8 cores & 8 GB RAM per node.
Each application was run with an input datasets of different sizes. We classified the datasets regarding the size into 3 major classes: (1) small: for datasets less than 1GB, (2) medium: for datasets between 1GB and 10 GB, (3) big: for datasets bigger than 10GB.
Before starting the modeling part, it was important to check a potential dispersion between execution time values for one experiment since we run it 3 times. As a result, although we record a 6.73% as coefficient of variation's mean, we record a maximum value of 83.06% for the same metric. This large variation for some experiments can be caused by networking latency for example. We took this variation into account when building the performance model in order to check its impact.

4.2 Evaluation of performance model's Accuracy

The machine learning prediction model was built under two different settings. First model was built without taking into account the application features and the later was built including the application features.The reason for this approach was to check the contribution of application features in the prediction of execution time by the models. Figure 1, Figure 2 show both mean absolute error (MAE) and root mean squared error (RMSE) of the machine learning algorithms per family with application features ingested and Figure 3, Figure 4 show both mean absolute error (MAE) and root mean squared error (RMSE) of the machine learning algorithms per family without application features ingested.
On average, Gradient Boosting Machine (GBM) and Random Forest (RF) perform the best compared to other methods. The execution time's mean is 399025 ms, the mean absolute error of GBM is 46662 and for Random Forest it was 44621. The error is approximately 10% for all families together. All ML algorithms provide good results on Word Count and Graph family thanks to the instrinsic simplicity of the chosen workloads. However, it proved difficult to predict the execution time of the Machine Learning family because of different types and complexity of the workloads, as well as for the GroupBy in SQL.
The performance model was almost the same with and without the application, however for some families of workloads, slight improvement was recorded.

5 Conclusion

In this paper, we presented a novel method for predicting the performance of a Spark application using machine learning models. An experimental framework for generating relevant datasets for performance prediction was developed and several

machine learning algorithms were evaluated. Experimental results show that GBM and RF have a good accuracy and computational performance across diverse spark workloads.
In the future, further research could be conducted on parameter selection in order to add more suitable spark parameters. The other revelent aspects affecting the performance prediction, such as the resource competition between the jobs and fluctuation of network bandwidth could also be explored.

References

1. Zemin Chao, Shengfei Shi, Hong Gao, Jizhou Luo, and Hongzhi Wang. A gray-box performance model for apache spark. *Future Generation Computer Systems*, 89, 06 2018.
2. Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.
3. Herodotos Herodotou, Harold Lim, Gang Luo, Nedyalko Borisov, Liang Dong, Fatma Bilgen Cetin, and Shivnath Babu. Starfish: A self-tuning system for big data analytics. In *In CIDR*, pages 261–272, 2011.
4. Palden Lama and Xiaobo Zhou. Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud. In *Proceedings of the 9th International Conference on Autonomic Computing*, ICAC '12, pages 63–72, New York, NY, USA, 2012. ACM.
5. Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, pages 29–42, Berkeley, CA, USA, 2008. USENIX Association.
6. Karthik Kambatla, Abhinav Pathak, and Himabindu Pucha. Towards optimizing hadoop provisioning in the cloud. *Hot Topics in Cloud Computing*, 06 2009.
7. D. Wu and A. Gokhale. A self-tuning system based on application profiling and performance analysis for optimizing hadoop mapreduce cluster configuration. In *20th Annual International Conference on High Performance Computing*, pages 89–98, Dec 2013.
8. R. Zhang, M. Li, and D. Hildebrand. Finding the big data sweet spot: Towards automatically recommending configurations for hadoop clusters on docker containers. In *2015 IEEE International Conference on Cloud Engineering*, pages 365–368, March 2015.
9. Kewen Wang and Mohammad Maifi Hasan Khan. Performance prediction for apache spark platform. In *Proceedings of the 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conf on Embedded Software and Systems*, HPCC-CSS-ICESS '15, pages 166–173, Washington, DC, USA, 2015. IEEE Computer Society.
10. Nezih Yigitbasi, Theodore L. Willke, Guangdeng Liao, and Dick H. J. Epema. Towards machine learning-based auto-tuning of mapreduce. *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 11–20, 2013.

11. Rekha Singhal, Chetan Phalak, and Praveen Singh. Spark job performance analysis and prediction tool. pages 49–50, 04 2018.
12. Anastasios Gounaris and Jordi Torres. A methodology for spark parameter tuning. *Big Data Research*, 11, 05 2017.
13. G. Wang, J. Xu, and B. He. A novel method for tuning configuration parameters of spark based on machine learning. In *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 586–593, Dec 2016.
14. Liang Bao, Xin Liu, and Weizhao Chen. Learning-based automatic parameter tuning for big data analytics frameworks. *CoRR*, abs/1808.06008, 2018.
15. Sara Mustafa, Iman Elghandour, and Mohamed A. Ismail. A machine learning approach for predicting execution time of spark jobs. *Alexandria Engineering Journal*, 57(4):3767 – 3778, 2018.
16. Herodotos Herodotou and Shivnath Babu. Profiling, what-if analysis, and cost-based optimization of mapreduce programs.