CS 597 - Special Problems Report

Alan Collet - A20453846
Summer 2020

Google Traces Analyses and Predictions

Alan Collet
Computer Science Student
Illinois Institute of technology
acollet@hawk.iit.edu

Zhiling Lan
Computer Science Professor
Illinois Institute of technology
lan@iit.edu

I. Introduction

This project represents different analyses and prediction models about various data from the Borg scheduler from Google. Google published in 2011 and lately in 2019, traces from one of their scheduler (Borg).
Borg deployment comprises a logically centralized cluster scheduler master, and a large number of machines (or nodes), each of which runs a local management agent. Each such deployment is called a cell and is operated as a single management unit.
Traces consist of time-stamped data from the master and the individual machines.

First steps of this project were to get familiar with these new data.
Some analyses are done to characterize the data from the scheduler. These analyses are adapted from the official Borg scheduler traces analyses [1].
A second task was to build prediction models to solve different problems that can occur in the system.
The two mains models were to predict the future workload of the system and to detect tasks than can fails. Tools used in order to achieve this objective was:

- Python as programming language with pandas libraries to manage datasets and Sklearn and Tensorflow/Keras to build the prediction models of the project
- Google BigQuery and Google Collaboratory were used together. It allows to manage the data without downloading it (as it represents Terabytes of data).

Using Google BigQuery have however some limitation, as with the free version, used data could not be infinite. So each analysis and each model has to be adapted to this limitation.
Every models done in this project are done only on one cluster managed by the scheduler to limit the size of data.

II. Trace Analysis

A. Global Analysis

The start of the project required to do some global analysis on the data from the Borg scheduler [4]. For this, it was necessary to first analyze traces from the scheduler using different perspectives. Experience done are the ones described in the Borg paper [1] adapted to the project.

The first analysis determines the distribution for the CPU and Memory utilization for any machine. It was necessary to gather data from 100 different machines during a specific period of time. One day was choose to find a compromise between the data size limitation and a representative-enough sample (day/night difference).

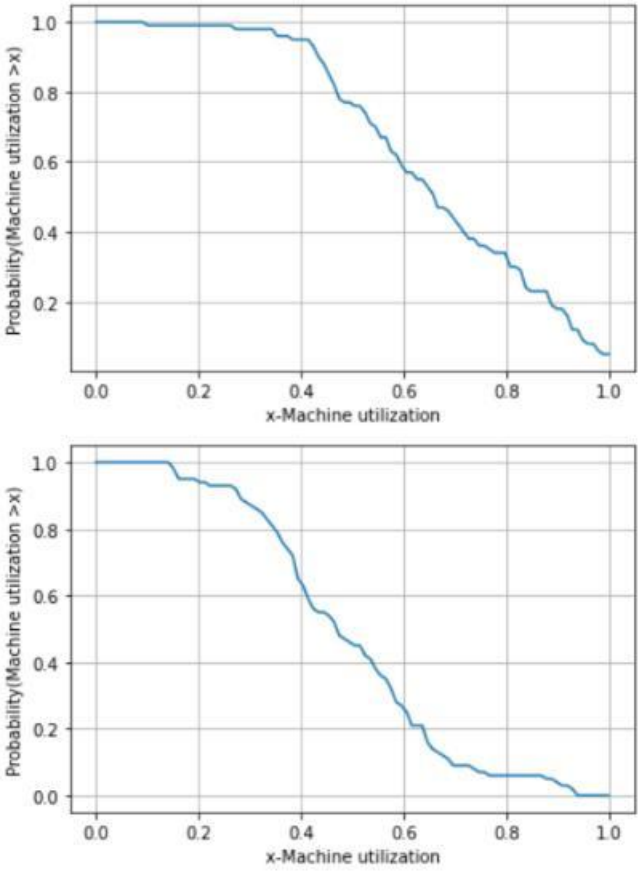

Fig. 1. Machine utilization (CPU on top and Memory on bottom)

Figure 1 describes this analysis. We can see that the scheduler divide the total workload within the cluster to the different machines (Every machine works almost all the time at 40% of CPU and memory usage whereas really few ones work at 90% or more of utilization). Moreover, we can see that the Memory and CPUs consumption is similar.

A second experiment is to estimate the incoming job number into the cluster. Data for this experience is given by gathering incoming task during one full day rather than on the global data set for the same reasons as the first experiment.

Figure 2 graph demonstrates that the number of incoming

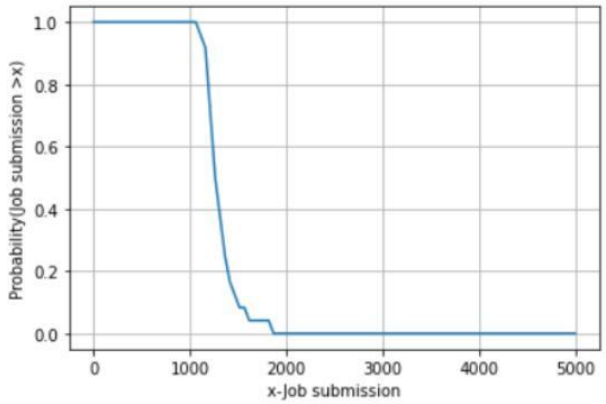

Fig. 2. Job submission rate for 1 hour

jobs is always between 1000 and 2000 no matter the hour of the day (closer to 1000 during the night and 2000 during the day).

However, these jobs are composed has different characteristics. According to [1] and [3], we can differ 4 different priority on these jobs which are:

1) $\mathbf{Free-tier}$: jobs running at these lowest priorities incur no internal charges, and have no Service Level Objectives (SLOs).
2) $\mathbf{Mid-tier}$: jobs in this category offer SLOs weaker than those offered to production tier workloads, as well as lower internal charges.
3) $\mathbf{Production-tier}$: jobs in this category require high availability (e.g., user-facing service jobs, or daemon jobs providing storage and networking primitives); internally charged for at "full price". Borg will evict lower-tier jobs in order to ensure production tier jobs receive their expected level of service.
4) $\mathbf{Best-effort\ Batch\ (beb)\ tier}$: jobs running at these priorities are managed by the batch scheduler and incur low internal charges; they have no associated SLOs.

It is necessary then to identify the task number for each job priority. Figure 3 represents this repartition of the task number for one job according to their priority.

We can see that for every priority most of the jobs are composed with only 1 task (90%) and less than 1% is composed of more than 100 tasks. Moreover, best-effort and mid priority tasks seems to have a bigger number of tasks. Prodcution-tier curve is not really representative as less than 1 task over 1000 corresponds to this priority and then really few was used for this analysis.

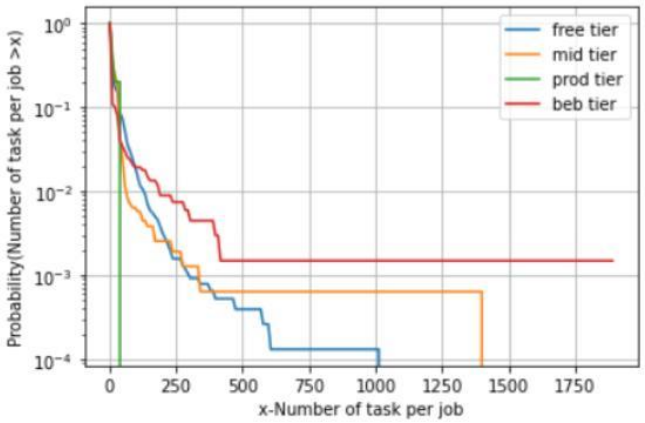

Fig. 3. Machine utilization (CPU on top and Memory on bottom)

B. Workload

The first main part of the project is to predict the workload of the system. An important characteristic of these jobs is their consumption (in terms of memory and CPU). We compute these results using NCU (Normalized CPU Unit) and NMU (Normalized Memory Unit). For this, it is necessary to add all the records of the task consumption during its lifetime using NCU-hours. A 20 NCU-hour job might have consumed 20 machines for an hour, or one machine for 20 hours, or used 20% of 100 machines for an hour, etc. Similarly, for memory usage, we total the Normalized Memory Unit-hours (NMU-hours) consumed by a job.

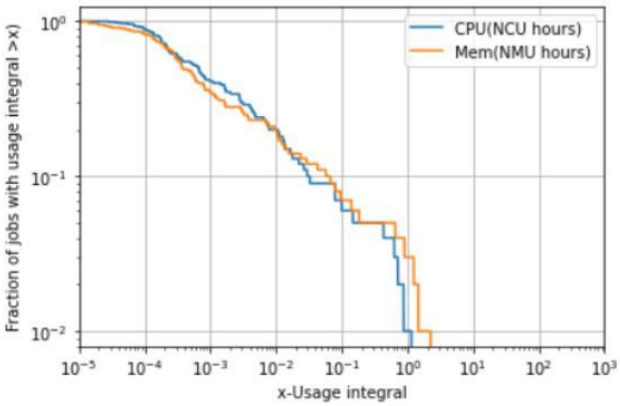

Fig. 4. Ressource consumption per task

In Figure 4, the consumption repartition of the task for a full day is represented. 90% of these tasks required less than 0.1 NCU/NMU during their lifetime really few needs more than 1 NCU/NMU (less than 1%). We also can see that NMU consumption and NCU consumption are correlated.

As we know that there is multiple class priority, another analysis is to identify this consumption given a specific priority. However, there are too few samples for Production tier and Best-effort tier to be relevant so it is done only

free-tier and mid-tier priority.

Figure 5 represents this split between priorities for CPU

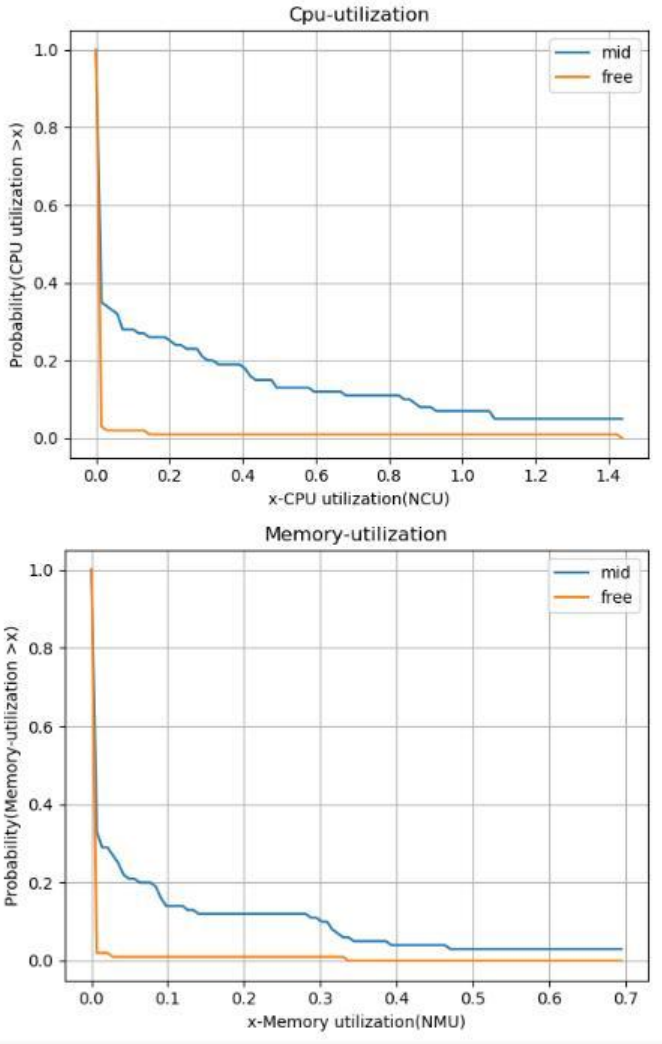

Fig. 5. Ressource consumption according to priority

and memory utilization. Mid-priority tasks have significantly more NCU and NMU consumption, even if only 40% of mid-priority tasks have a significant consumption. On average the consumption (NCU and NMU) is more than 10 times superior in mid-tier priority compared to free-tier priority tasks. Free-tier priority tasks almost never requires a consequent amount of CPUs/Memory utilization. Once more, we can see that the repartition for the memory and CPUs utilization are correlated.

C. Failures Analysis

The other main part of the project is to predict failures in the system using the different data we have from the scheduler. As well as Workload, the first step is to analyze traces from the scheduler.

The first analysis is to determine the distribution of the number of failed tasks within the cluster. For this, data from a full day is used as the best compromise between data size and representative-enough sample.

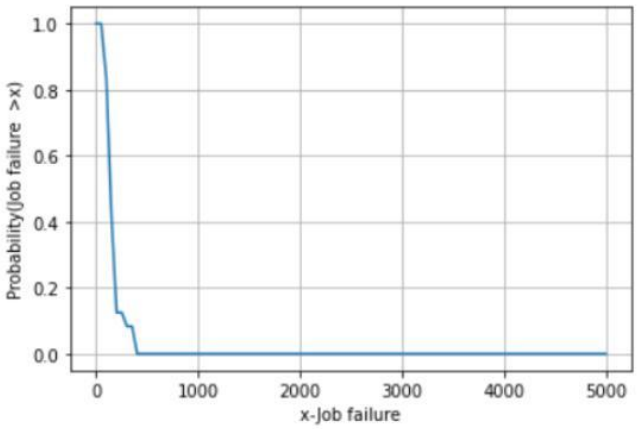

Fig. 6. Task Failure distribution (average on one hour)

Figure 6 illustrates this distribution. Only few jobs fails, 8.24% of the total tasks fails. However it is impossible to differentiate system failure and other type of failure (program failure). The failure number is stable during the day as the curve slope is abrupt.

III. Workload Prediction

The first problem developed is workload prediction. The goal of this section is to compare different methods used on the data in order to predict the workload of the system at any time. Different models for different utilization related to workload prediction are used. First, models used will be presented and in the second part, results obtained will be described for each problem.

A. Models Used

The first class of model tried on the data is statistical models (ARMA and ARIMA) to predict the workload of a task in the future. Both are moving average models. ARIMA adds an exponential factor to give more importance to the last samples. The main advantage of this algorithm is that it is really simple and then easily implemented and easy to understand as the target values are represented with the average (or weighted average) on a specific number of sample:

$$\hat{X}_t = \epsilon_t + \sum_{i=1}^{p} \alpha_i X_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}$$

with p and q the order of the model (the number of samples to consider), ϕ_i and θ_i, the coefficients of the model, and ϵ_i, error terms.

The main disadvantage of this model is that it is really slow

to compute. To use this model, the data was reduced 600 samples (400 for training and 200 for testing, corresponding to 30 hours of the workload) or the time of the training part was too long (It tooks 10 minutes using Google Collaboratory setup to train with 400 elements).
Another technique was used to predict the future workload on a task. Bunch of regressors (Lasso, ElasticNet, DecisionTree, ...) models was tried in order to reduce an error using previous samples as features. These previous samples was chosen when the partial autocorrelation of the model with this time shift was higher than a given threshold. The partial autocorrelation correspond to the correlation between a signal and the same signal with a shifted time. It is given by:

$$R_X(\tau, k) = E[X(t)X^*(t + k - \tau)]$$

with k the number of shifted samples.

Results with this technique were however mediocre. The best regressor model was the ElasticNet, trying to minimize the following loss function:

$$L_{ENet}(\hat{\beta}) = \frac{1}{2N}\sum_{i=1}^{n}(y_i - X_i\hat{\beta})^2 + \lambda(\alpha\sum_{i=1}^{m}\hat{\beta}_j^2 + (1-\alpha)\sum_{i=1}^{m}|\hat{\beta}_j|)$$

where α and λ are coefficients for the importance of L_1 and L_2 errors in the model and $\hat{\beta}$, coefficients of the model.
Another predictor is made to identify the next task priority given the previous ones. This time the problem is a classification problem. Only 3 classes are presented as targets (Mid, Free, and Best-effort priority) as production priority tasks are almost not present in the data (less than 1 over 1000). First, classical machine learning models for classification were used such as Logistic Regression and Support vector machine was tried, however, it returned poor results and then, ensemble learning methods were used.

The objective of ensemble learning methods is not to build the most accurate model but to build a lot of small models trained on the dataset or on parts of the dataset and put them together to build a really strong global model. It exists a lot of different ensemble learning methods but the most knowns are Random Forest, Bagging, Stacking and boosting methods.

The best model is given by a Gradient Boosting classifier. A Gradient Boosting Classifier builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage (3 as there are 3 target classes) regression trees are fit on the negative gradient multinomial deviance loss function given by

$$-\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\log h_\theta(x^{(i)}) + (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))\right]$$

where h_θ are the coefficients of the model

A final model is made to estimate the time took for a task given its priority. Data used for this model was the duration of different tasks (maximum possible with the free version of Google BigQuery). Different regressors was tested. First, classical machine learning regressor such as Perceptron or Linear Regression. Then the best result was given by a Random Forest Regressor (a bunch of regression decision trees over different parts of the dataset to reduce overfitting and maximize accuracy). Only mid-tier and free-tier priority was tested as the other priority tasks often do not finish (such as monitoring process). The loss function for each decision tree in the Random Forest is the same is given by the L_2 loss:

$$E = \frac{1}{2}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2$$

Then the final decision of the Random Forest is given by making an average on every results of the model. (The classes with the more represented will be the final label of the data). It allows to have a better accuracy than a standard machine learning method as it uses a lot of smaller models but reduce overfitting and bias as the final result is the mean of every little models (Decision Trees in this case).

B. Results

This part presents the results of models describes in the previous part (3.1)
Results for the statistical models (ARMA and ARIMA) are given with Figure 7 and Figure 8.

With these graphs, we can identify that the ARIMA method tends to better fit the curve than ARMA. However, as curves are really spiky, results are not so accurate (we can see more than 20% difference at each sample on average).

For the second model, to identify the next job priority, results are given by the following confusion matrix (class0: Free priority, class1: Mid priority, class2: Best-effort priority). The confusion matrix give a good description of the model showing the full output data description (True positive, True negative, False positive and False Negative)

With only the previous task priority, the estimation for mid-priority class is really good. For best-effort and free priority, it is more confusing. These results are probably caused by the fact that two following tasks have probably the same priority and mid-priority are generally bigger than the 2 other ones.

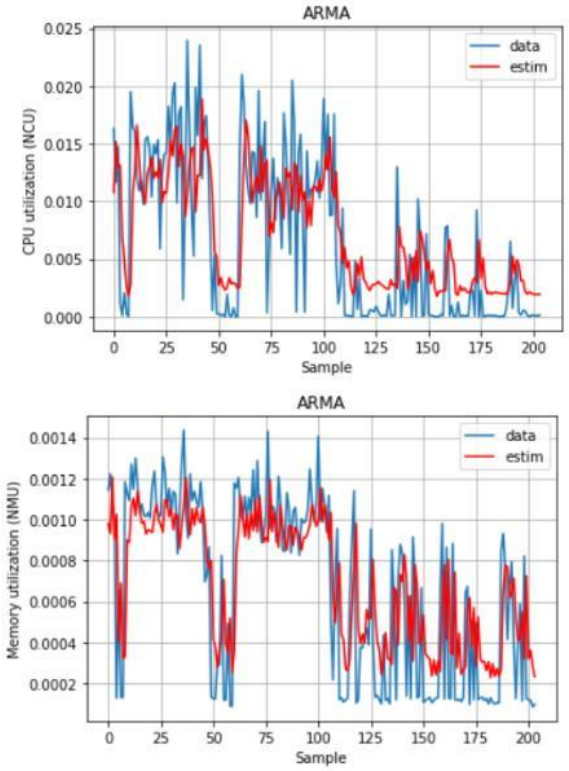

Fig. 7. Result graph with ARMA method

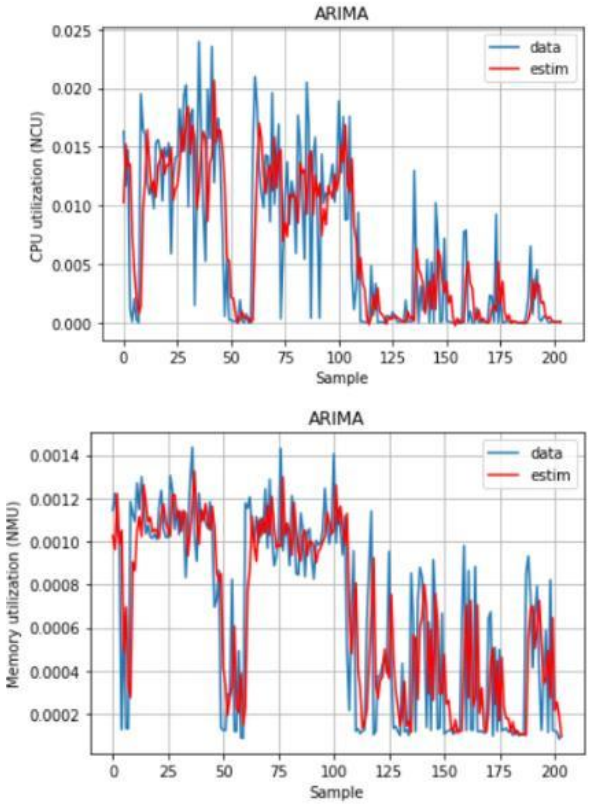

Fig. 8. Result graph with ARIMA method

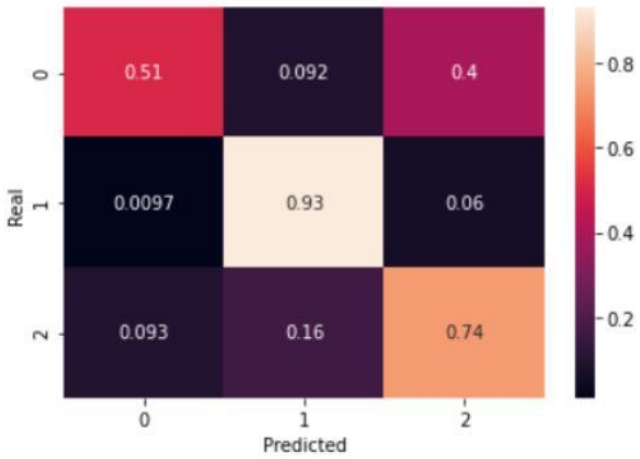

Fig. 9. Confusion matrix for the next priority class prediction

For the last problem (time estimation given its priority), the results of the Random Forest Regressor are characterized by their R^2 coefficient. It represents the quality of the estimator.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y}_i)^2}$$

with $\overline{y}_i$ the mean of the data.
Result for this problem are:
$R^2 = 0.003$ for free-tier priority, which corresponds to a mean estimator (just slightly better)
$R^2 = 0.203$ for mid-tier priority, which is a better estimator than a mean estimator without being exceptional. These bad results are caused by the fact that only metadata of a task are given (priority, requested CPU/Memory, ...) and no real data about the program (number of read/write , syscall, ...) are given which could be better descriptors of the task.

IV. Failure Detection

The other main part of the project is to predict the failure of the system. Using every available data about tasks, it was possible to build models to predict if a failure will happen during the execution of a task. For this, the maximum possible data for the free version of Google BigQuery is used. It represents the success or failure of thousands of tasks and with all the data gave by the scheduler (priority, required consumption, the machine(s) used, ...) and using the failure/success characteristic as target.

A. Models Used

The first method implemented was simplistic machine learning algorithms (Logistic Regression/Multi-Layer Perceptron). The advantages of these techniques are that there are easily understandable and implemented. Both trying to

optimize the L_2 error:

$$E = \frac{1}{2}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2$$

Multi-Layer Perceptron corresponds to a multi-layer of multiple units of Logistic Regression. It is the simples neural network possible as it corresponds to multiples units of Logistic Regression distributed among multiple layers.

These techniques returned bad results and then other models more involved are used using Ensemble Learning. 4 different algorithms were tried (Random Forest Classifier, Boosting technique, Stacking technique, and Bagging technique). The 2 classifiers with interesting results are the Random Forest Classifier and the model using Boosting technique. Both are already described in the II.A part. Boosting technique is exactly the same technique, and the Random Forest is used this time as a classifier rather than a regressor (This time the loss function to optimize is not anymore the L_2 loss but the entropy loss:

$$L = -\sum_{i=1}^{m}(y^{(i)}log(\hat{y}^{(i)}))$$

An important characteristic of these jobs is their consumption (in terms of memory and CPU). We compute these results using NCU (Normalized CPU Unit) and NMU (Normalized Memory Unit). For this, it is necessary to add all the records of the task consumption during its lifetime using NCU-hours. A 20 NCU-hour job might have consumed 20 machines for an hour, or one machine for 20 hours, or used 20% of 100 machines for an hour, etc. Similarly, for memory usage, we total the Normalized Memory Unit-hours (NMU-hours) consumed by a job.

B. Results

The 2 best techniques saved as valid are the Random Forest classifier and the classifier using Boosting techniques. 2 different analyses to identify the quality of models are used. First, the confusion matrix, that represents the full output of the model and then the ROC (receiver operating characteristic) curve with its AUC (area under curve). It represents the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The AUC of the ROC curve usually gives a really good estimator of the global quality of the model.

Results given by simplistic prediction models are given in the Figure 10

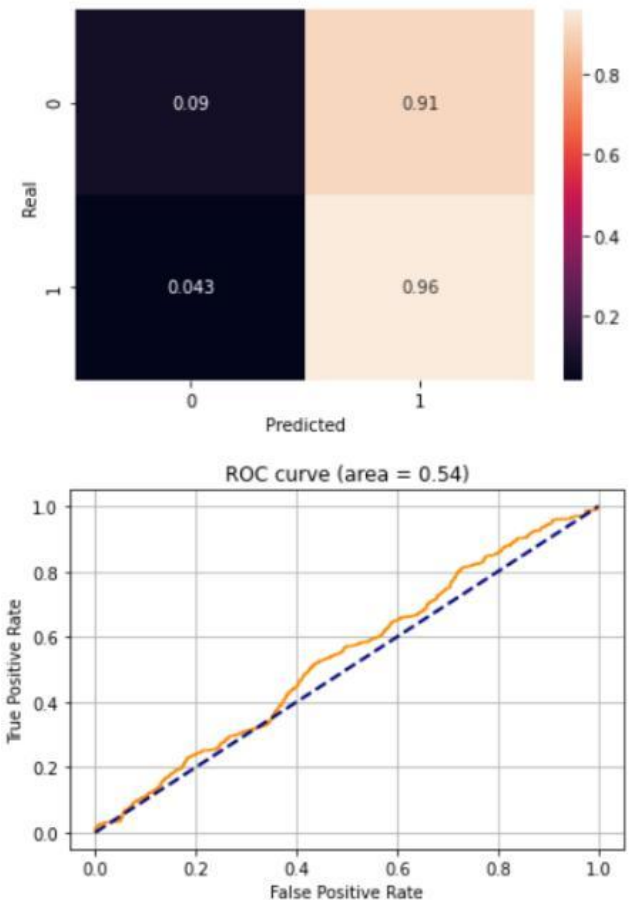

Fig. 10. Results using Logistic Regression Model

Results with simple machine learning methods are very poor. In fact, the model is just slightly better than a random classifier (defined by the blue line in the ROC curve graph. So the idea was to use ensemble learning to use a lot of small predictors that can be accurate when associated.

Results given by the Random Forest classifier and Boosting are presented in Figure 11 and Figure 12. For both models, in the confusion matrix, class0 represents a failure and class1 a success.

We could say that the classifier using boosting technique is the best as the area under curve is superior to the one of the Random Forest classifier. However, the task is to detect a failure and the classifier that have the best failure detection rate is the Random Forest. It represents only 42% of true failure detection but using only external program data, it is already a significant result. However 1.6% of the successful tasks will also be treated as a failure, but it is better to identify 42% of failure and miss treat 1.6% of good requests rather than predict no failure at all.
These results could be way better with data about tasks itself rather than only system data.

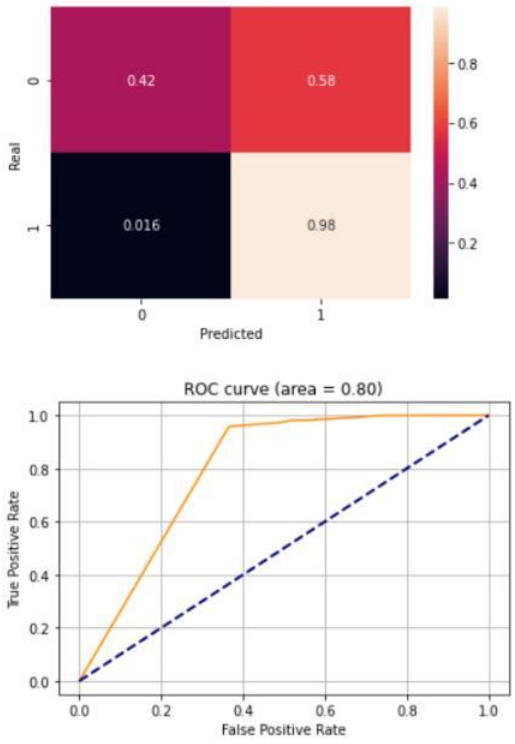

Fig. 11. Results using Random Forest Classifier

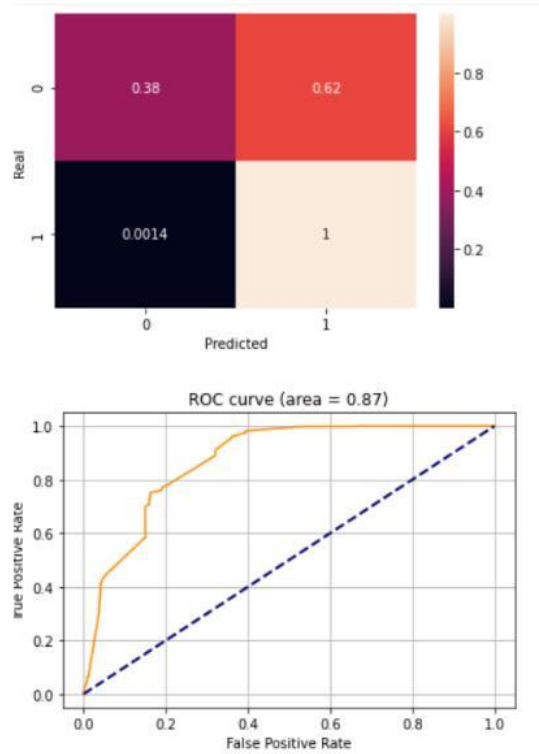

Fig. 12. Results using boosting technique

V. Related Studies

Different studies about the Borg scheduler was made before. The more important is Borg: The Next Generation published in May 2019 [1]. It was the main article for this project and is an updated version of the 2015 version [3]. It is a paper with a lot of different analyses about the full data from the Borg Scheduler. Some of them are adapted in this project (in the Trace Analysis part) when possible. Some information is given first about the Borg scheduler.
In both of these documents, analyses are done on the full data of the scheduler on every possible cluster (there is a total of 8 clusters).
Every prediction models are thought about these analyses.

Data used to compute analysis in the two previous papers and on this project is explained in [2]. Every different database is described with all the data inside them.

The other related studies are from the google community about the data from the Borg Scheduler. A lot of student and professional are working on this data and everyone exchanges about the data, models used to solve different problems and to help each other.

VI. Conclusion

The study of this data gave the opportunity to illustrate a lot of different analysis of the data. Analysis such as Job submission number, machine utilization (CPUs and Memory utilization), and the number of tasks per job gives a good interpretation of how the Borg scheduler works.
Analyses around the resource consumption and the number of failures asked the question if these data could be predicted with different models.

Where models used gave good results for predicting the next task priority, other models for the other tasks gave mitigate results. These results could be improved without limitations for this project:

- The amount of data was the biggest restriction, as the free version of Google BigQuery does not allows to use all the required data for analyses presented in [1] and [3]. Analyses required then to be adapted to the data restriction. Some could not be done at all due to this problem. The data limitation also decrease models efficiency as it reduces the size of training sets.
- At the end of the project, every timestamp in databases were not supported anymore by the Google Collaboratory API, the data had then to be downloaded directly from the BigQuery platform to perform models in a local machine (slower).
- The fact that only data from the system were provided and no about task and jobs themselves caused a lot of problem in prediction models. In fact for workload prediction, the

number of call of any type of function could be way more useful than the machine ID where the task run on.

These problems were however a great opportunity in a way as it was necessary to think in another way to explore what can be done.

REFERENCES

[1] M.Tirmazi A.Barker M.Deng Md E.Haque Z.G.Qin S.Hand M.Harchol-Balter J.Wilkes. EuroSys'20
Borg: The Next Generation May 2019.

[2] J.Wikes,
Google Cluster-usage traces v3 April 2020
https://drive.google.com/file/d/10r6cnJ5cJ89fPWCgj7j4LtLBqYN9RiI9/view

[3] A. Verma L. Pedrosa M.Korupolu D.Oppenheimer. Eurosys
Large-scale cluster management at Google with Borg 2015

[4] J.Wikes
Google compute cluster trace data, 2020
https://github.com/google/cluster-data/blob/master/ClusterData2019.md

[5] Google Traces chat community
https://groups.google.com/forum/!forum/googleclusterdata-discuss

[6] GitHub link for coding part
https://github.com/acollet001/GoogleTraceAnalysis