Analysing Data with Spark

Henrique Figueiredo Conte

January 2023

Contents

1 Introduction

This project is about analysing Google's large dataset using Apache Spark, learning how to solve complex problems in large scale data analysis. To do that, we will conduct a series of analysis in Spark using Python, experimenting different Spark functionalities and plotting results with Matplotlib when possible.

The Google dataset is a set of machines, jobs and tasks. The machines run jobs that are composed by tasks, they have CPU/memory allocations and events informing if the machine was added/removed. The jobs and tasks also have events informing their execution, such as task eviction or task killed, and they also have their requested resources usage. More details about the dataset can be found in the official Google cluster-usage traces documentation.

The next section will cover both the analysis executed for each problem and the results obtained from them. All the analysis were made with 5 parts of each table (when available) in order to get a more accurate conclusion.

The code for this project and its report can also be found on Github.

2 Analysis conducted in Spark

2.1 What is the distribution of the machines according to their CPU capacity?

In a first iteration, the approach was to use the *machine_events* data table and get a distribution of the values of the *CPU Capacity* column. To get this distribution, I first filtered empty values in this column and then used the **.histogram** method from Pyspark.

The histogram will be divided in 5 sections. Since CPU capacity is a normalised value from 0 to 1, the first section will group the machines that have a CPU capacity from 0 to 0.2, the second will group from 0.2 to 0.4, and the same logic applies to the rest of the sections.

After executing this with the first part of the *machine_events* table (*part_0000_of_00500*), we got the following results:

Figure 1: Exercise 1 results with duplicated machines

```
Machines analysed: 37748
CPU capacity distribution:
 ([DISTRIBUTION], [AMOUNT_OF_VALUES]
 ([0.0, 0.2, 0.4, 0.6, 0.8, 1.0], [106, 10698, 21736, 2983, 2225])
```

However, I noticed that since this table considered machine events instead of

the machines themselves, we would count some machines multiple times, since a machine could be added/removed multiple times during the analysis period. Then, I implemented a new version removing duplicated rows based on the **machineID**, considering only the first occurrence of the machine.

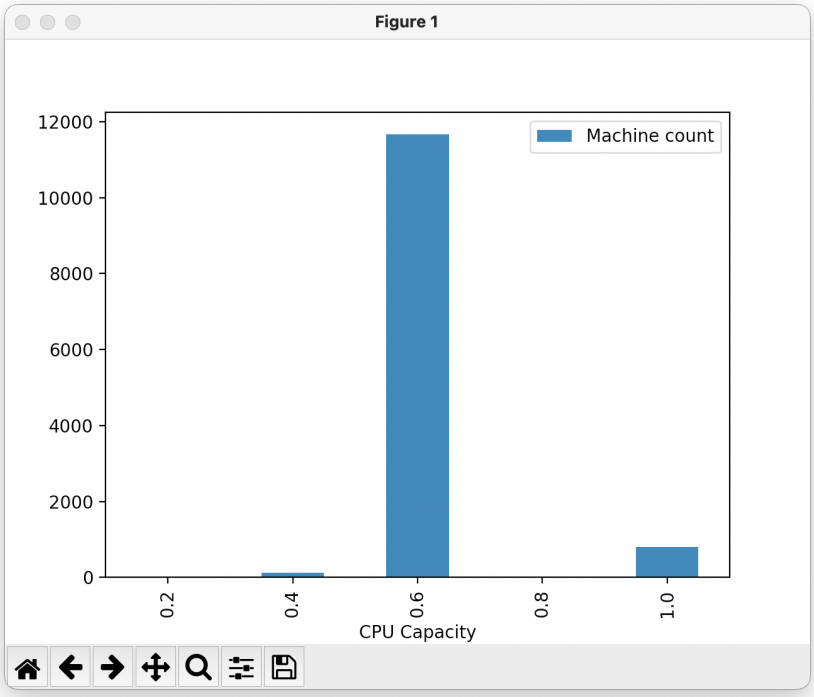
We can see the results below:

Figure 2: Exercise 1 results without duplicated machines

```
Machines analysed: 12583
CPU capacity distribution:
 ([DISTRIBUTION], [AMOUNT_OF_VALUES]
 ([0.0, 0.2, 0.4, 0.6, 0.8, 1.0], [0, 126, 11659, 0, 798])
```

As we can see, the amount of analysed machines reduced to one third of the original value. On the other hand, we can see that some sections, like 0.0-0.2 didn't have any machines, meaning that Google changes the machines CPU capacity according to their needs during execution.

Figure 3: Machines capacity histogram

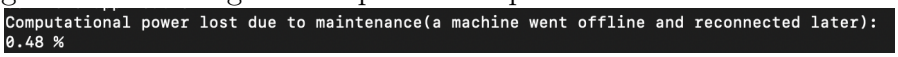


2.2 What is the percentage of computational power lost due to maintenance (a machine went offline and reconnected later)?

To calculate the percentage of computational power lost due to maintenance, I used the *machine_events* table, and grouped the events by machine ID and ordered by their timestamp. The machines could be connected and disconnected multiple times, and this made the problem quite hard.

The strategy to calculate the percentage of computational power lost was to sum the amount of time each machine spent offline and compare it with the time span of the dataset. Then, the next step is to get the average percentage offline percentage of all machines, resulting in the desired result.

Figure 4: Percentage of computational power lost due to maintenance



2.3 What is the distribution of the number of jobs/tasks per scheduling class?

To get the distribution of jobs and tasks per scheduling class, I used the *job_events* and *task_events* tables and grouped its elements by the scheduling class column. Then, it was only necessary to count the number of elements per group by using the method **count()**. Let's see the results for the distribution of jobs and tasks per scheduling class:

Figure 5: Jobs/tasks distributions per scheduling class

```
Jobs distribution by scheduling class
+---------------+-----+
|SchedulingClass|count|
+---------------+-----+
|              0| 7738|
|              1| 8450|
|              2| 7237|
|              3| 2012|
+---------------+-----+

Tasks distribution by scheduling class
+---------------+------+
|SchedulingClass| count|
+---------------+------+
|              0|565620|
|              1|110724|
|              2|183251|
|              3| 59924|
+---------------+------+
```

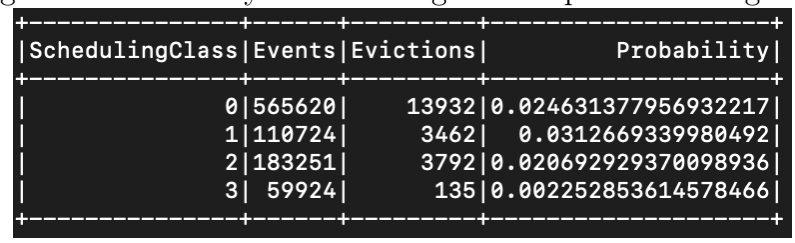
We can conclude that a job and its tasks can have different scheduling classes. For example, a job with scheduling class 2 can have tasks ranging with all scheduling classes.

2.4 Do tasks with a low scheduling class have a higher probability of being evicted?

To analyse the relation between scheduling class and probability of being evicted, I grouped the events by scheduling class, counted the total amount of events and the amount of events that were evicted.

Then, to calculate the probability of being evicted, I created a new column with the amount of evicted events per scheduling class and another column with the probability of being evicted.

Figure 6: Probability of task being evicted per scheduling class

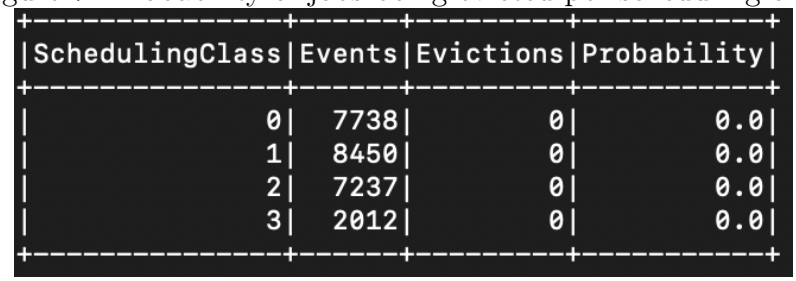


SchedulingClass	Events	Evictions	Probability
0	565620	13932	0.024631377956932217
1	110724	3462	0.0312669339980492
2	183251	3792	0.020692929370098936
3	59924	135	0.002252853614578466

As we can see, tasks with high priority (scheduling class 3) have the lowest eviction probability, 10 times lower than the second lowest probability.

Besides, I also checked if jobs followed the same pattern, however jobs seem unlikely to be evicted.

Figure 7: Probability of jobs being evicted per scheduling class



SchedulingClass	Events	Evictions	Probability
0	7738	0	0.0
1	8450	0	0.0
2	7237	0	0.0
3	2012	0	0.0

2.5 In general, do tasks from the same job run on the same machine?

To solve this problem, I ran a series of experiments to understand the data we are working with.

The first thing I did to analyse if the tasks of the same job run on the same machine was to take a look in a small sample of the data. By using the *task_events* table, I grouped it by jobID and the amount of distinct task indexes and distinct machineID's per jobID.

Figure 8: Relation between jobs and machine execution sample

JobID	count(TaskIndex)	count(MachineID)
1005190908	10	10
1005246041	5	5
1005831088	1	1
1005831090	51	52
1015033007	1	1
105360316	1	1
106664144	2	2
109341524	2	2
109344935	2	2
1097525707	1	1
1106173310	50	52
1128637731	1	1
1161381522	1	1
1170482497	1	1
1184713860	10	10
1202636597	5	6
1205597814	1	1
1218322450	1	2
1218322587	1	1
1221166900	1	1

only showing top 20 rows

The 20 rows result will gives us an idea on what to expect. As we can see, the jobs that have multiple tasks are also executed in multiple machines. On the first row, for example, the jobID 1005190908 has 10 tasks, and those tasks were executed in 10 different machines.

Let's expand our analysis to all the data. The approach to understand if the rest of the jobs are also executed in multiple machines will be to get the percentage of jobs that have more than one task AND are executed in more than one machine.

Therefore, I got three important numbers that allow us to analyse the data:

1. Total amount of distinct jobs.
2. Total amount of distinct jobs that have more than one task.
3. Total amount of distinct jobs that have more than one task and that run in more than one machine.

Finally, we know the amount of jobs that have multiple tasks and run in multiple machines:

Figure 9: Tasks running on the same/different machines

```
29.05% of the jobs run in multiple machines.
Among jobs with multiple tasks, 99.58% of them run in multiple machines.
```

2.6 Are the tasks that request the more resources the one that consume the more resources?

To analyse the correlation between resources request/usage, we have to use both *task_usage* and *task_events* tables. In this analysis, we will compare the requested **CPU cores** from *task_events* with **MeanCPUUsage** and **MaxCPUUsage** from *task_usage*, and the requested **RAM** from *task_events* with **AssignedMemUsage** and **MaxMemUsage**.

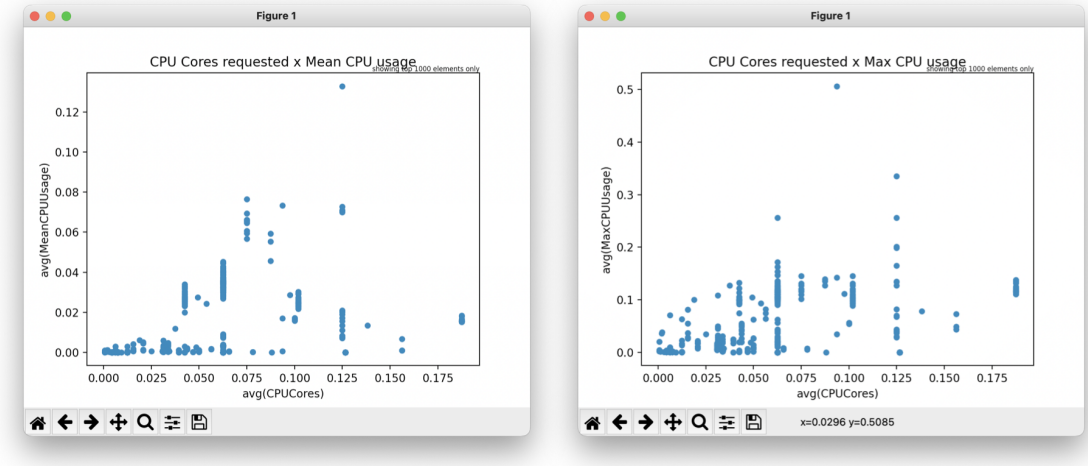
To get all this information together, we will create a dataframe with the CPU cores and RAM requested per task. Since a task might be evicted and started again, we will get the average CPU cores and RAM requested by the tasks.

Then, we will create a second dataframe with the MeanCPUUsage, MaxCPU-Usage, AssignedMemUsage and MaxMemUsage consumed per task.

Finally, we will merge those two datasets by joining by jobID and task index. This will associate each task with their respective resources request and consumption.

Now, to check if there is a correlation between those parameters, the easiest way is to create visual graphs of those comparisons using the Spark plotting framework, that uses Pandas and Matplotlib to build the graphs.

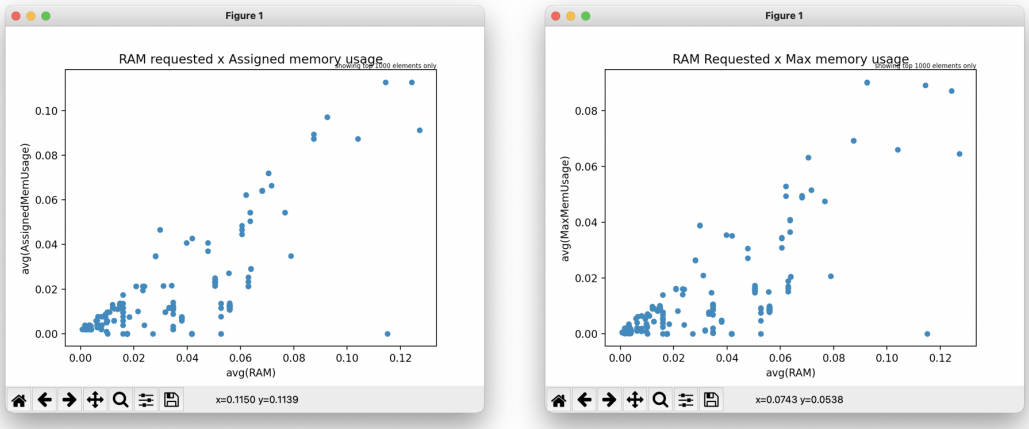
Figure 10: CPU cores request x CPU mean/max consumption



On the CPU usage, the consumption increases in the beginning and stabilises

in the middle. Even among the tasks that requested the most CPU cores, the consumption doesn't really increase. This probably happens when users request a higher CPU capacity to ensure that the machines will handle the traffic even in higher peaks.

Figure 11: RAM request x Memory mean/max consumption



On the other hand, the correltion between RAM request and memory usage is clearly visible. The bigger the RAM request, the higher is the consumption.

2.7 Can we observe correlations between peaks of high resource consumption on some ma- chines and task eviction events?

The last mandatory question (and the hardest one), used the *task_usage* and *task_events* tables.

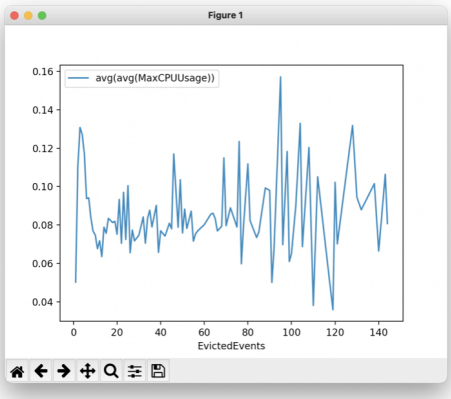
The strategy was to join the tasks from task_events with the respective machines and timestamp window from task_usage rows. This gives us a new dataframe with all the events that occurred in each timestamp window + machine.

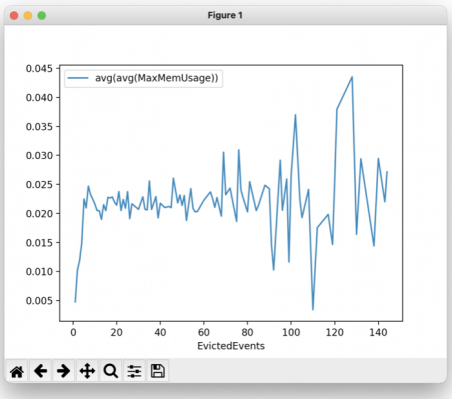
Then, we group them by machineID, start and end timestamp, counting the amount of evicted events per timestamp window + machine and getting the average Max CPU Usage and Max Mem Usage of those events. This will give me a dataframe with the timestamps, amount of evicted events per machine and their resources usage.

Finally, we create a new dataframe grouping by the amount of evicted events, getting the average of resource usage per amount of evicted events. For example,

the machines that evicted 54 events in a given timestamp window had an average of 0.13 MaxCPUUsage.

Figure 12: CPU and memory relation with task eviction





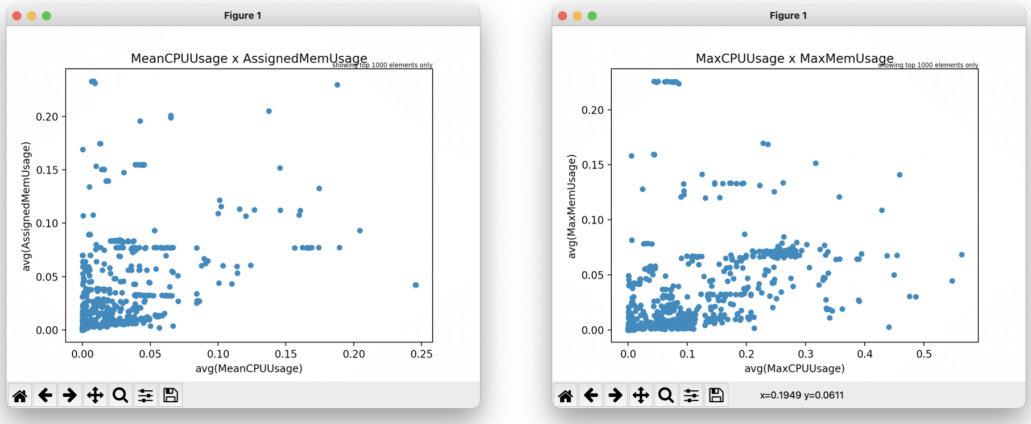
According to my results, there's no direct relation between CPU and memory consumption with task eviction. The only possible relation we can see is that the higher resource consumption values are concentrated on higher eviction count groups. Maybe my code is incorrect, but I checked it twice with the teacher and we couldn't figure it out if the code was the problem or if there's not a relation between resource consumption and task eviction.

2.8 Do tasks that have a higher CPU usage also have a higher memory usage?

To check if there is a correlation between CPU and memory usage, we will use the *task_usage* table. Then, we need to group by **jobID** and **taskIndex**, getting the average **MeanCPUUsage**, **AssignedMemUsage**, **MaxCPUUsage** and **MaxMemUsage**. Next, we will compare the MeanCPUUsage with AssignedMemUsage, and also MaxCPUUsage with MaxMemUsage.

Finally, one way to see if there is a correlation between those parameters is by plotting a graph. Since we have a huge amount of CPU usage entries, we will use a scatter plot to better see the data.

Figure 13: CPU and memory usage relation



In addition, we can check the correlation value between columns by using the method *stat.corr* from Pyspark. This method will use the Pearson Correlation Coefficient to calculate the correlation.

Figure 14: Pearson Correlation Coefficient between CPU and memory usage

```
Correlation between mean CPU and Mean memory usage: 0.41388996280316326
Correlation between max CPU and max memory usage: 0.257013673793216
```

As a result, we can conclude that the memory usage tends to increase as the CPU usage increases up until a certain point. What we can mostly notice is that most machines have a low CPU and memory usage, and as the CPU usage grows we start using more memory, until it gets stable in medium usage of memory.

2.9 How many submitted tasks actually finish without being evicted, failing, being killed or lost?

This question will allow us to see how often tasks are evicted or fail completing. This will help us to realise how important it is to implement fault tolerant systems, since tasks in large scale systems will often face tasks interruptions.

To implement this analysis, we will filter the tasks that were evicted, killed, lost or failed at some point and see their proportion compared to the total amount of tasks.

Figure 15: Tasks that were interrupted at some point

```
Amount of tasks: 304279
Amount of tasks that were either evicted, killed, lost or failed: 47499

15.61% of the tasks were evicted, killed, lost or failed at some point.
```

As we can see, out of 304279 tasks, 47499 were interrupted, representing 15.61% of the total amount of tasks.

3 Extended work

3.1 Comparison of different solutions

When I started implementing the project, I was trying to implement the solutions using RDD. After a couple of hours struggling to get results, I managed to implement the question 1 with RDD, as we can see on the commit history.

However, while implementing the other exercises, I realised how hard it is to manipulate data with RDD, since we have to use indexes to access data and it has less methods when compared to dataframes. Therefore, I gave up on using RDD's and did the rest of the project with dataframes, and I don't regret the decision.

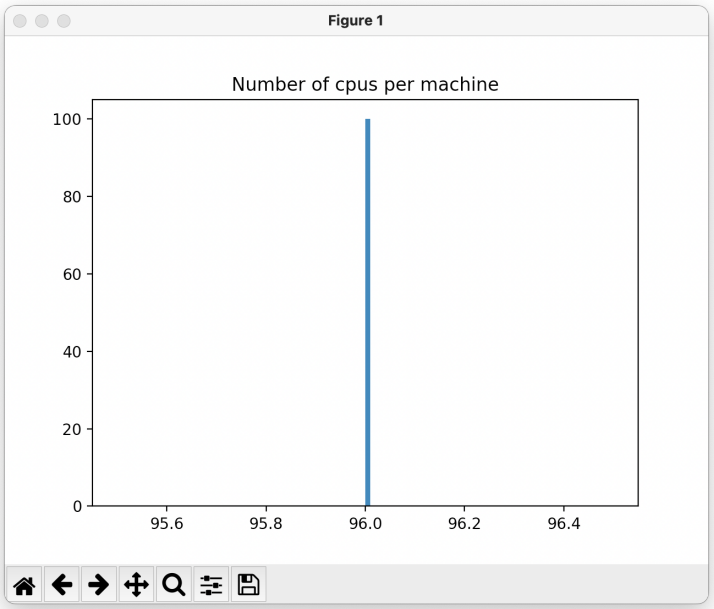
In my opinion, dataframes are easier to manipulate, since they resemble SQL. In addition, they have more built-in methods that help analysing the data, so we don't have to implement everything from scratch.

3.2 Analysing Alibaba's cluster data

In order to understand more about the Google's cluster data, we also extended the analysis to see how Alibaba's machines compare to Google's. The Alibaba dataset is available here, and I chose the data from 2018. Even though the default download link isn't working, I tried downloading from the Chinese link and it worked!

Based on the analysis we executed with Google's machines, I tried to analyse similar parameters with Alibaba's, but the results weren't satisfying. The first analysis I ran was to see the CPU distribution among the machines. This data can be found on the Machine Meta table, and the complete schema is available here.

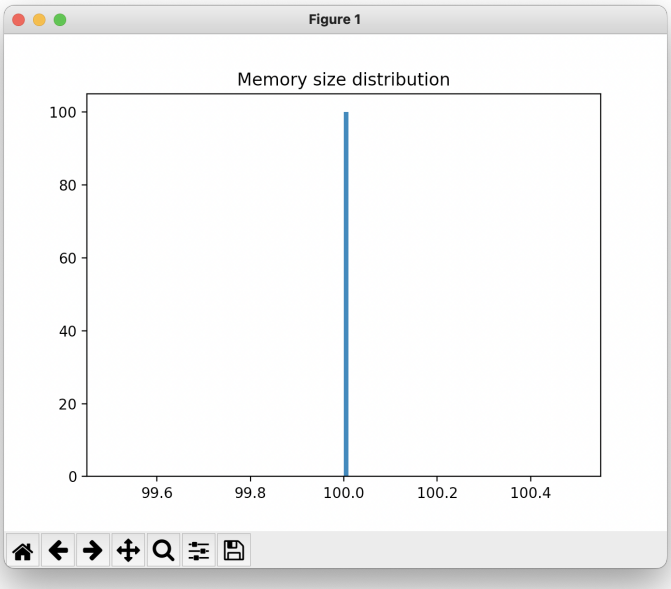
Figure 16: CPU distribution on Alibaba machines



As we can see, all the machines from Alibaba have the value 96 as CPU. According to the schema, this column is the "number of cpu on a machine", which doesn't make it clear what 96 actually is. Besides, it is disappointing that all machines have the same amount of CPU.

In sequence, I tried to analyse their normalised memory size:

Figure 17: Memory size distribution on Alibaba machines



Since the value is normalised, we can't exactly know how much memory they have. In addition, what is actually disappointing on this data is the fact that all machines have the same memory as well, making the analysis less interesting.

Finally, I decided to check how the other tables from the dataset looked like, and they also had a lot of repetition in their data. Therefore, I stopped the Alibaba's dataset analysis because the data didn't seem interesting to analyse.

4 Conclusion

The project was harder than I expected it to be, with long and sometimes tough questions. However, that improved my knowledge in Spark, which might be useful in the future. Besides, the Google dataset is interesting, especially on the analysis related to task eviction. I didn't know that task eviction was that common, and it was nice to see its relation with the scheduling classes.