<u>Analyzing Data with Spark – Project</u>

M2 – MOSIG (Large Scale Data Management)

Done by,

Jayashri Govindan

BEN YOUSSEF Farah

(12/01/2024)

1. Introduction:

The *"Analysing data with Spark"* project provided us with a **practical experience in learning about Spark functionalities** and how large amount of data can be processed efficiently and with minimal latency. The dataset *ClusterData 2011 traces* provided us with a clear **picture of how big data is presented** in a real-time large-scale distributed environment and how a cluster management system allocates work to the machines.

In the analysis presented, spark functions such as *groupBy, agg, collect_list, intersect, udf* (user defined function), and others were used to perform **data aggregations, filtering, transformations, and other desired functionalities** with fewer lines of code. We also used **Spark.sql** in some of our analysis to explore different functionalities in Spark.

We wanted to learn more about other similar datasets and have a **better grasp of the diverse structures and challenges** that would arise while **processing and analysing large-scale distributed data**. The approach to data analysis with Spark from a code perspective is given in the source code file and our understanding, analysis, and the challenges we encountered are given in this report.

2. Dataset Information:

A) **ClusterData 2011 traces:**
https://github.com/google/cluster-data/blob/master/ClusterData2011_2.md
For the analysis, we used the first datasets in each category, as follows:
- machine_events: "data/machine_events/part-00000-of-00001.csv"
- job_events_path: "data/job_events/part-00000-of-00500.csv"
- task_events_path: "data/task_events/part-00000-of-00500.csv"
- task_usage_path: "data/task_usage/part-00000-of-00500.csv"

B) **Alibaba 2017cluster traces:**
https://github.com/alibaba/clusterdata/blob/master/cluster-trace-v2017/trace_201708.md
- Machine events: “server_event.csv”
- Machine utilization: “server_usage.csv”
- Task table: “batch_task.csv”
- Instance table: “batch_instance.csv”
- service instance event: ”container_event.csv”
- service instance usage: “container_usage.csv”

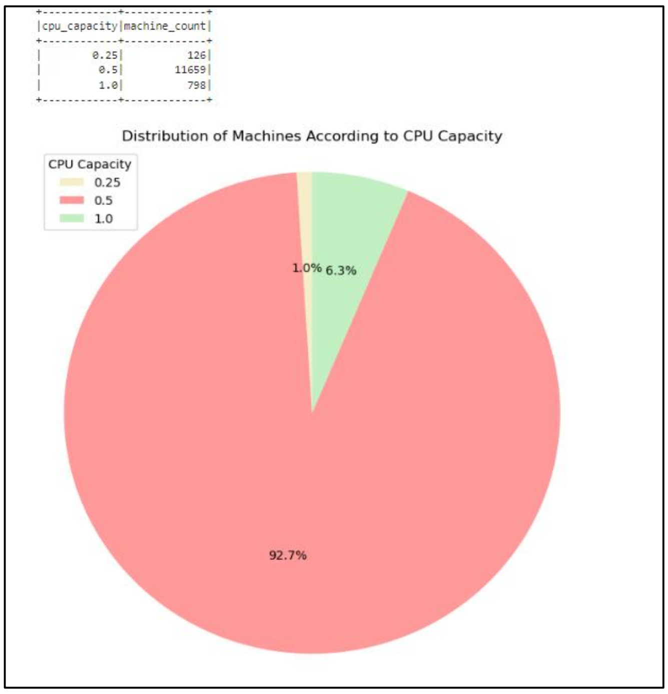
3. Analyses Conducted:

The questions given for this project for which analysis was conducted:

1) **What is the distribution of the machines according to their CPU capacity?**

- The majority of machines (11,659) **have 0.5 CPU capacity, 92.7% a dominant category**. There is considerable variety, with machines having *0.25* and *1.0* CPU capabilities. Some machines (*126*) have a **lesser capacity** (*0.25*), which could be used for **simpler processing requirements**. Machines (*798*) with **greater capacities** (*1.0*) may be able to **handle more demanding tasks**.

- We could understand that this diversity of CPU capacity enables jobs to be completed simultaneously on numerous machines (**parallel processing**), but we must be smart on how we split and distribute tasks to make the most of each machine's capabilities (**workload distribution**).



```
+------------+-------------+
|cpu_capacity|machine_count|
+------------+-------------+
|        0.25|          126|
|         0.5|        11659|
|         1.0|          798|
+------------+-------------+
```

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

Percentage of computational loss due to maintenance: **24.22%**

In general, we understood that **computational loss** caused by maintenance might have **an impact on s
ystem performance.** Using our dataset, we may observe a 24.22% percentage of computational loss c
aused by maintenance. The observed loss has a significant influence on the system and also highlights
the **need of addressing maintenance-related difficulties** in the system in order to improve overall effi
ciency. Additionally, to better understand why they occurred, we could identify specific patterns and c
auses of maintenance-related computation loses.

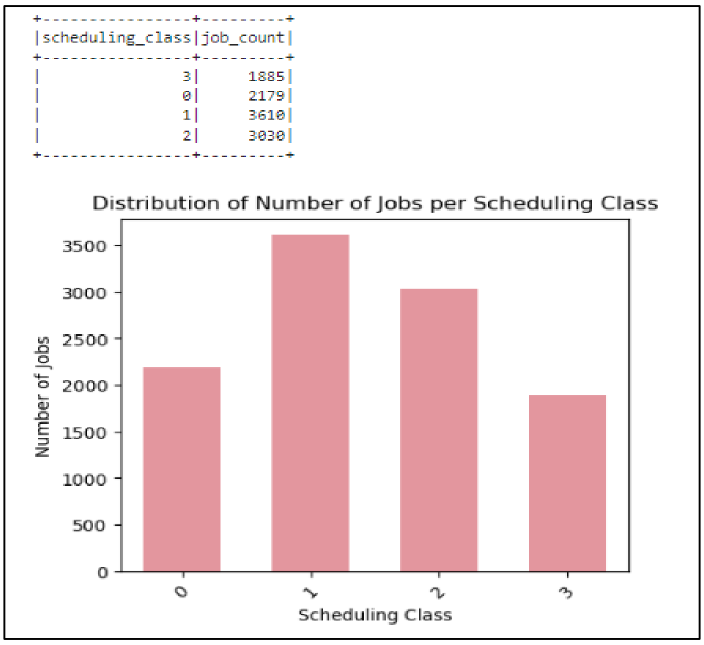
3) What is the distribution of the number of jobs/tasks per scheduling class?

Based on the provided Google documentation, we interpreted that each scheduling class (0 to 3) indicates a distinct level of **latency sensitivity**, with **higher number (3) indicating more time-critical operations**. The job and task counts help us understand the distribution and diversity of workload characteristics within each scheduling class.

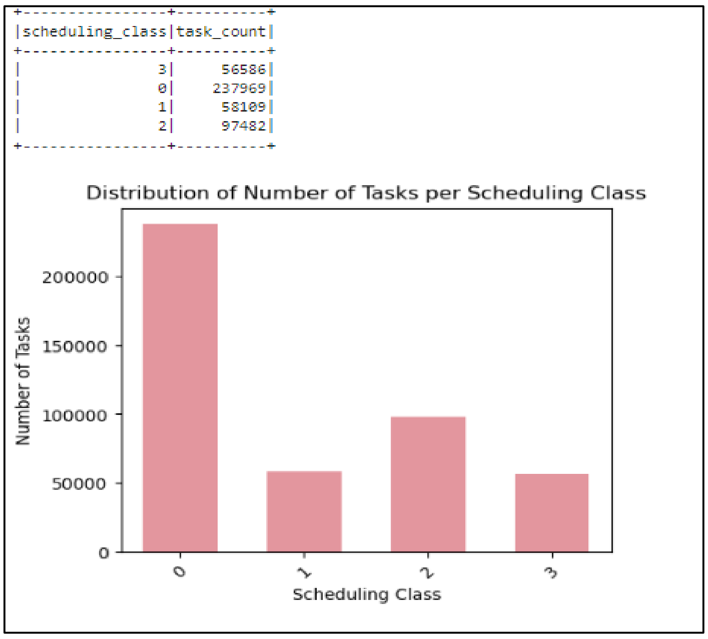
- ***Scheduling Class 0:*** Scheduling class 0 has a similar number of jobs (2179) as class 3, but the significantly **greater task count (237969)** indicates that tasks in this scheduling class are more yet may have **less - latency requirements**. These are tasks/jobs that can take time to accomplish **without having an impact** on the system's current performance.
- ***Scheduling Class 1:*** Scheduling Class 1 include task/job that **may require immediate attention (latency-sensitive) as well as** those that **may take slightly longer** to complete. The **highest job count (*3610*)** indicates that this class has a variety of jobs that address

different parts of the workload. Even if there are fewer tasks (*58109*) here, than in Class 0, Class 1 may be bit more urgent or different in terms of importance.

- ***Scheduling Class 2:*** Tasks/jobs in Class 2 could include a **combination** of things, providing a balance between those of **latency sensitive and those that are less urgent**, resulting in a workload that cover various aspects. In terms of job count (3030), Scheduling class 2 is between classes 0 and 3. The task count (97482), is lower than in classes 0 and 1, but nevertheless high. We could to understand that it’s a **balanced workload**, meaning this class contains a **diverse range of tasks/jobs.**
- ***Scheduling Class 3:*** Scheduling Class 3 represents **time-sensitive tasks**, which may include operations **require urgent attention**. Class 3 tasks/jobs are intended to be completed fast and are impactful to system's functioning. There is a moderate number of job count (1885) observed. The **relatively high number of tasks (56586)** show that the scheduling class 3 is **actively utilized**. The system may be handling an intense workload with a focus on low latency and on jobs/tasks that **need quick attention**.



```
+----------------+---------+
|scheduling_class|job_count|
+----------------+---------+
|               3|     1885|
|               0|     2179|
|               1|     3610|
|               2|     3030|
+----------------+---------+
```

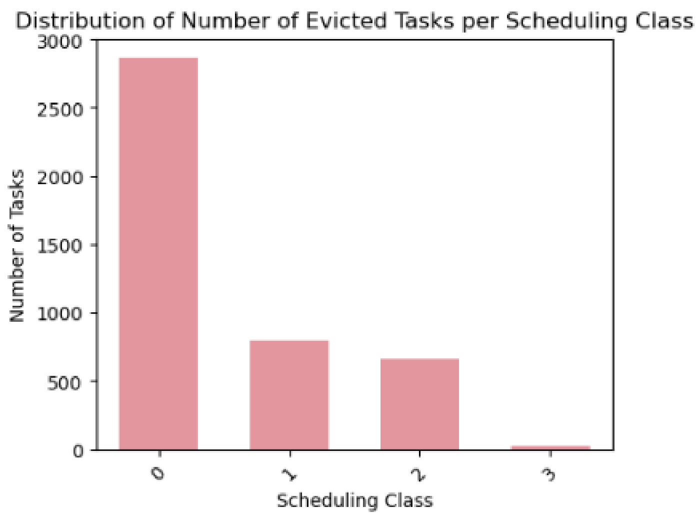


```
+----------------+----------+
|scheduling_class|task_count|
+----------------+----------+
|               3|     56586|
|               0|    237969|
|               1|     58109|
|               2|     97482|
+----------------+----------+
```

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

Yes, the analysis shows that tasks with **lower latency sensitivity (scheduling class 0) experience a significant number of evictions** (2864). As latency sensitivity increases (from scheduling class 0 to class 3), the number of evicted tasks decreases. We could interpret that this may be **due to various reasons**, like **system resource availability** (resource contention) and **scheduling** policies. The data provides insights into the relationship between scheduling class and eviction potential, which should be considered when **allocating resources** across different tasks.
We used **Spark.sql** to perform this analysis and used query to obtain the result, to explore the spark functionalities.

```
+----------------+------------------+
|scheduling_class|evicted_task_count|
+----------------+------------------+
|               0|              2864|
|               1|               792|
|               2|               662|
|               3|                20|
+----------------+------------------+
```



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

Count of tasks from the same job that run on the same machine: **2703**
Count of tasks from the same job that do not run on the same machine: **1814**

We observed that the **majority of tasks from the same job run on the same machine**, which is an expected behaviour in distributed computing environment. It supports the goal of improving performance by **reducing data transfer** and **communication overhead**. However, there is also close count in scenarios where tasks from the same job run on **different machines,** this could be due to **resource constraints, load balancing, scheduling policies**, etc.

6) Are the tasks that request the more resources the one that consume the more resources?

Percentage of tasks that request the more resources are the one that consume more resources: **28.09%**

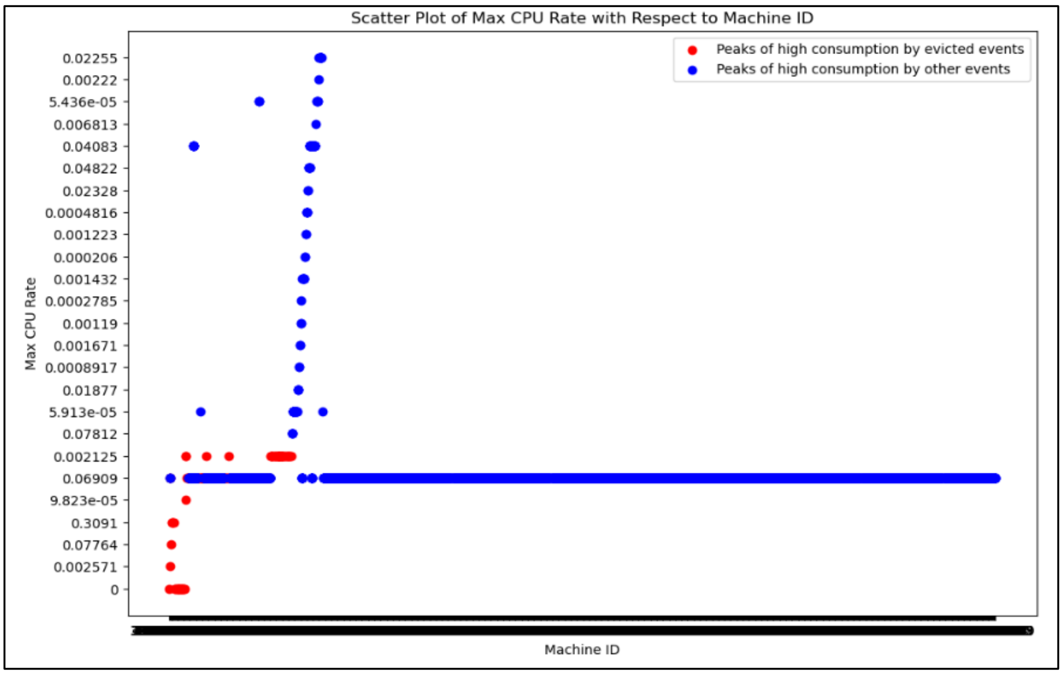
For our analysis, we assumed that **more resources** are those that are among the **top 10%** (calculated according to the dataset volume) of the highest resource request and the highest consumed resources. However, it was **challenging to comprehend** the criteria "more resources", and how to determine **how many of the top/highest** requests and consumption should be considered.

According to our data analysis, there were ***28.09%*** of the total tasks falling into this criteria. This shows whether the tasks are **resource-intensive.** The analysis also provides an insight that may be in some scenarios, **tasks may request resources but not completely use them**, resulting in **over-allocation** and considering this question can aid in identifying this issue.

7) Can we observe correlations between peaks of high resource consumption on some machines and task eviction events?

This analysis provided us with a more in-depth understanding of **how to identify high resource utilization peaks** by exploring the machine's task usage parameters. As we narrowed down our analysis to find the peaks of high resource consumption for the machine, we looked for task eviction events that occurred at that timestamp. Doing so, we may see if there is a correlation, which indicates that due to resource exhaustion on machines could be a factor driving to task evictions.

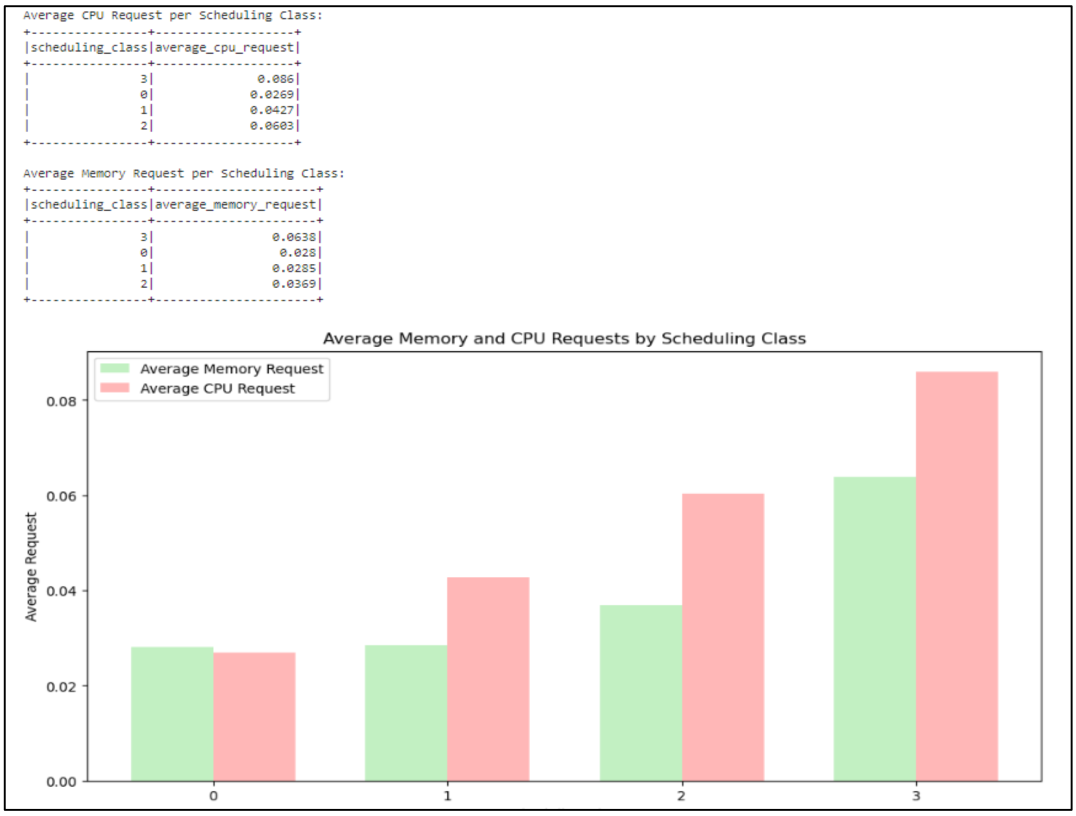
However, we **did not observe any such correlation in our data**. The eviction tasks did not occur at the same time or around the same time as the peaks of high resource consumption, showing **no direct correlation** in the case of our data. Here, maybe this indicates that **task evictions** are caused by **other factors than resource exhaustion** (peak of the high consumption).



4. Additional Questions Proposed:

1) Do tasks of latency sensitive scheduling class, have greater average memory and CPU requests than other scheduling classes?

- We were intrigued by our prior experiments of original questions on learning about schedule class levels, we delved into the question of whether the scheduling class associated with **higher latency sensitivity** tends to **request more resources** for **efficiently handling urgent operations.**
- Our analysis not only confirmed our expectations but also provided a clear depiction: the **scheduling class (3)** characterized by higher latency sensitivity consistently requests **more resources**. In contrast, other scheduling classes tend to request relatively fewer resources than their more latency-sensitive counterparts.
- This observed behaviour emphasizes the **wise allocation of resources to fulfil the specific demands of latency-sensitive jobs**, resulting in a focused and effective approach to **managing time-critical processes.**



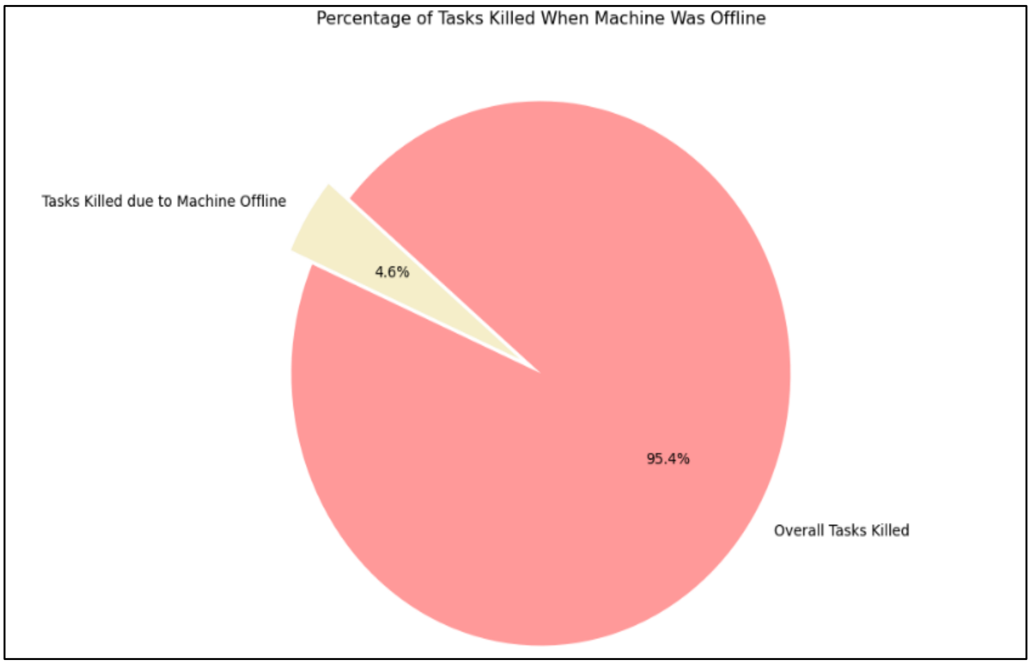
```
Average CPU Request per Scheduling Class:
+----------------+------------------+
|scheduling_class|average_cpu_request|
+----------------+------------------+
|               3|             0.086|
|               0|            0.0269|
|               1|            0.0427|
|               2|            0.0603|
+----------------+------------------+

Average Memory Request per Scheduling Class:
+----------------+---------------------+
|scheduling_class|average_memory_request|
+----------------+---------------------+
|               3|               0.0638|
|               0|                0.028|
|               1|               0.0285|
|               2|               0.0369|
+----------------+---------------------+
```

2) How much percentage of tasks are evicted simply because of the machine being unavailable (due to maintenance or repair)?

Total number of tasks killed: 42456
Total number of tasks killed when machine was offline: 1935
Percentage of tasks killed when machine was offline: 4.56%

We calculated the machines that went offline at the timeframe of when the task event was executed/was in usage. The considerable impact of machine unavailability on task performance was observed. The percentage observed (**4.56%**) indicates that a **small portion**

of task evictions are due to machine downtime. This may indicate that **maintenance or repair efforts interfere with the system's capacity** to execute tasks. On a large scale basis, the percentage may increase than the observed.
There is need to consider **improving maintenance schedules** or developing techniques to **reduce the impact of machine unavailability** on task execution.



We also had **other additional questions in mind**, such as the statements that were made in the Google documentation and how we could prove or analyse them.

1) Tasks with higher latency sensitivity tend to have higher task priorities.

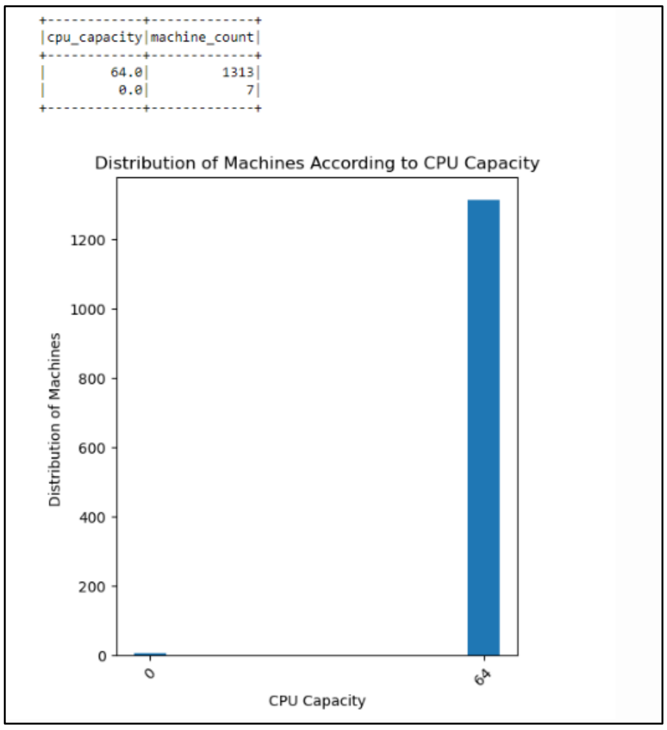
2) Tasks that exceed their resource limits may be throttled (e.g., CPU) or destroyed (e.g., memory).

5. Extended Work:

We chose to extend our project by analysing other dataset (Alibaba 2017 cluster traces) to have a deeper understanding and get comfortable on how to manage and analyse large datasets.

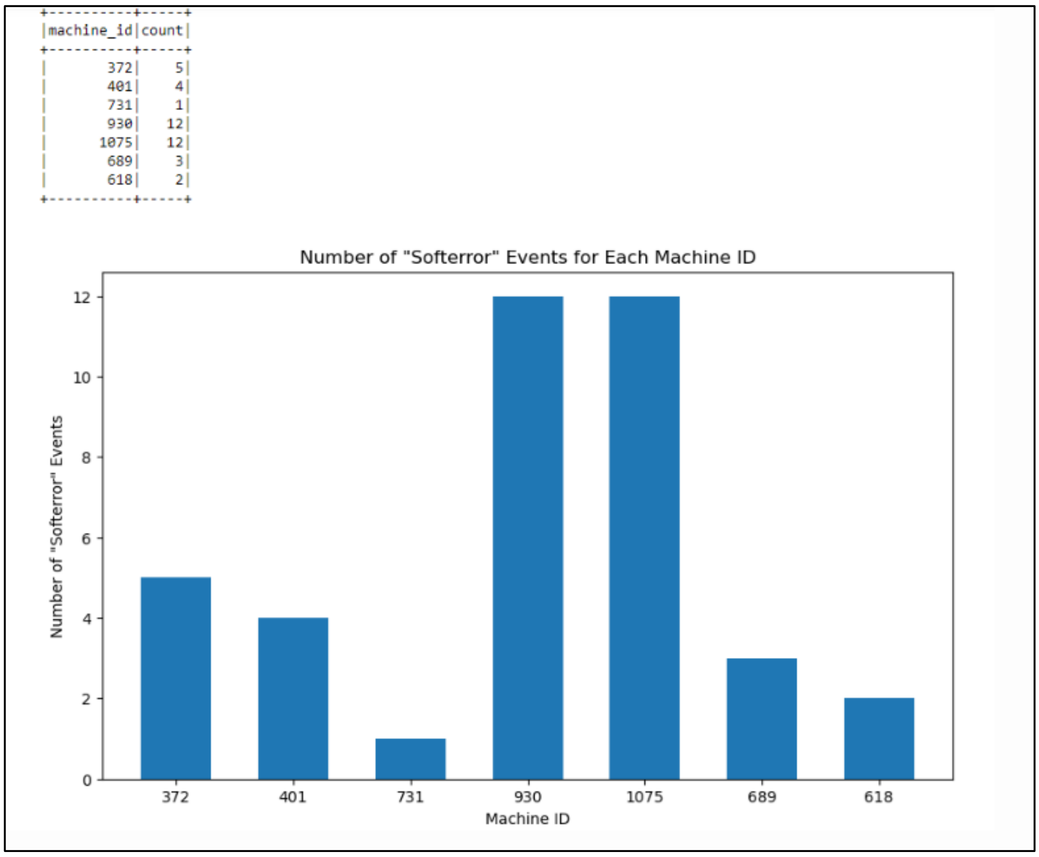
1) **How the machines are distributed according to the CPU capacity?**

   To answer this question, we will use the data table Machine events(server_event.csv) we get the machine_id and the CPU capacity then we filter the not null values, then count the number of machines for each CPU capacity. We observe that **contrary to the Google Cluster data 2011, all the machines in this case have the same CPU capacity**.



```
+------------+-------------+
|cpu_capacity|machine_count|
+------------+-------------+
|        64.0|         1313|
|         0.0|            7|
+------------+-------------+
```

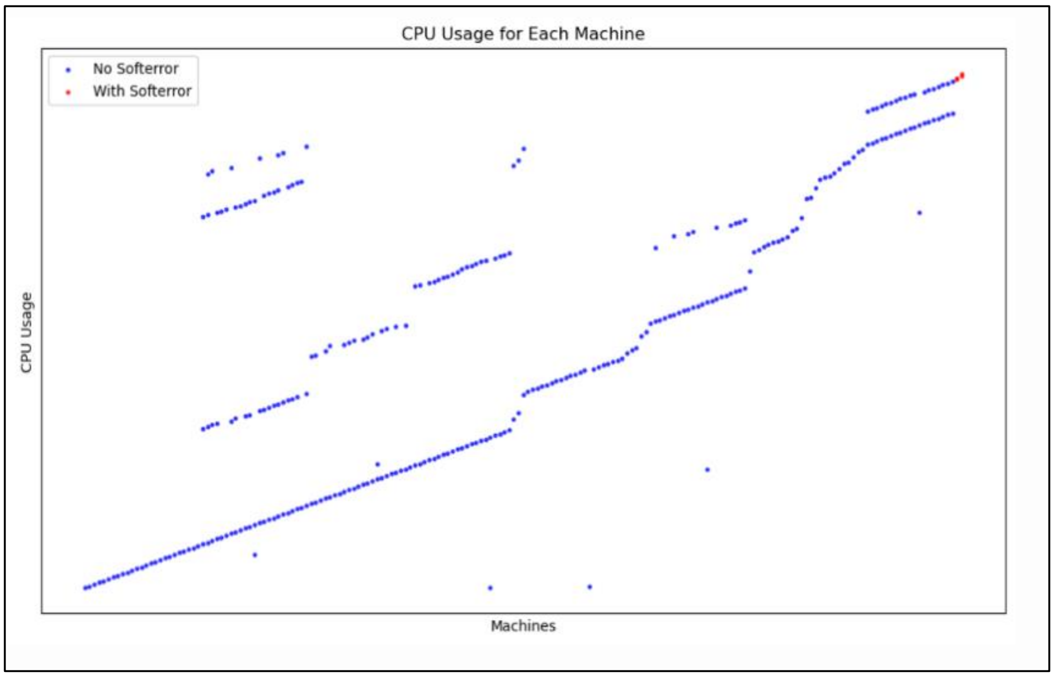
2) **Which machines might require prioritized maintenance?**
In order to continue exploring this dataset, we answer this question, by filtering the machines with event type = soft error (ones that went offline temporarily) and grouping them according to the machine id then counting it. We observe that **only 7 machines have become temporarily unavailable due to software failures**, those machines could be considered as troublesome and more prone to errors, which **might require prioritized maintenance**.

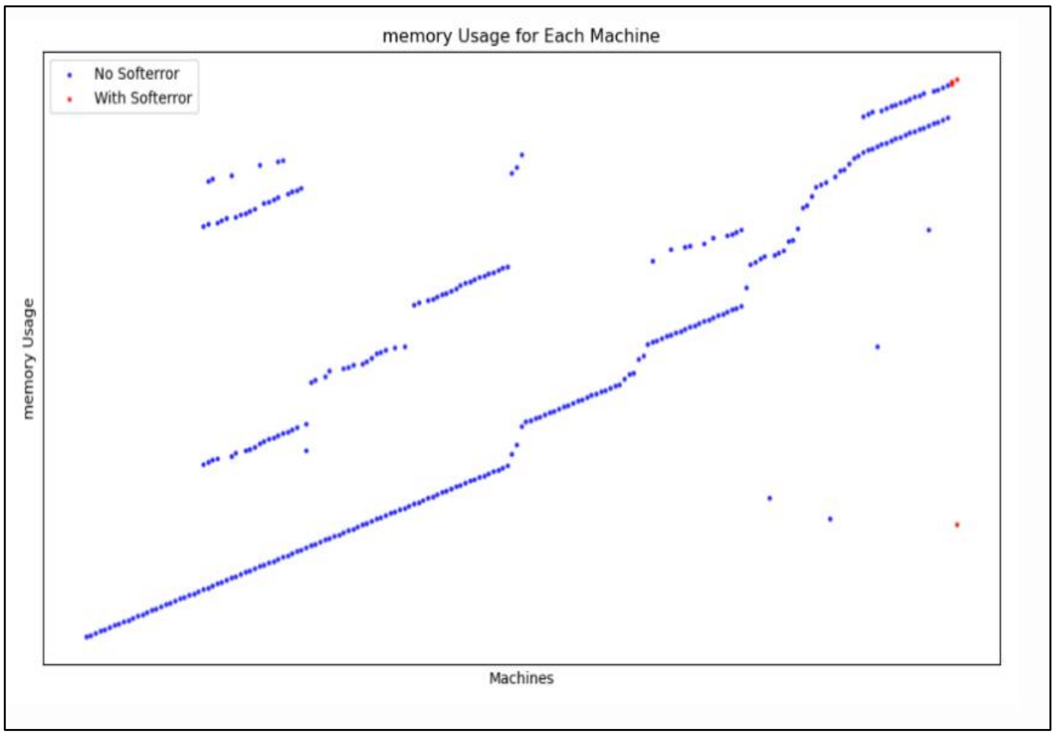


```
+----------+-----+
|machine_id|count|
+----------+-----+
|       372|    5|
|       401|    4|
|       731|    1|
|       930|   12|
|      1075|   12|
|       689|    3|
|       618|    2|
+----------+-----+
```

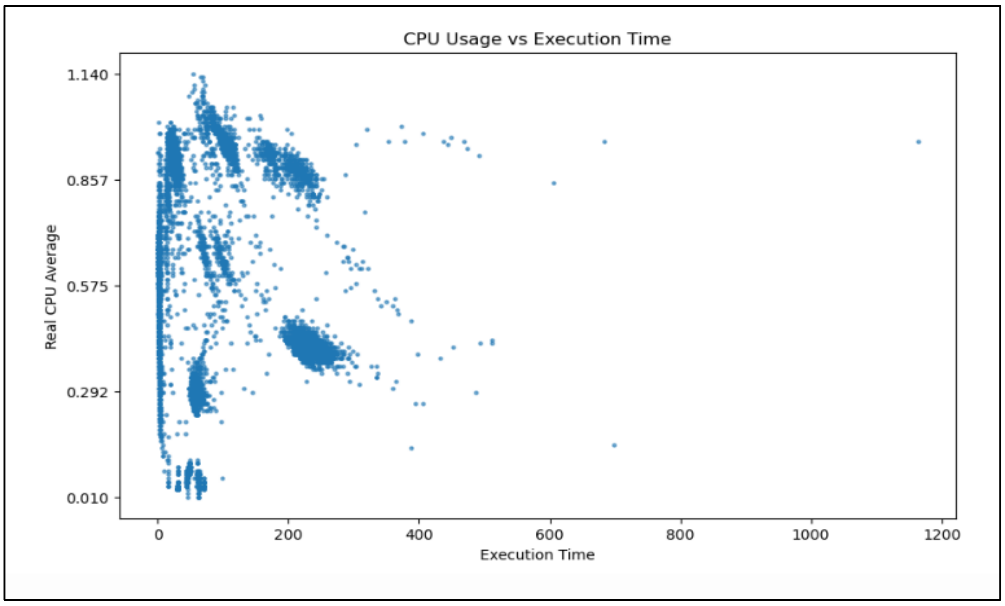
3) **Do machines with "Softerror" events exhibit distinct resource usage characteristics compared to those without events?**

To answer this question, we use the table machine usage and machine events, we get we start by filtering out the null values in the columns representing CPU usage and memory usage. Subsequently, we plot the results specifically for 10,000 machines. It becomes apparent that, concerning **CPU usage**, the **machines that encountered failures** are the **ones exhibiting higher CPU usage**. However, in the case of **memory**, **all machines that experienced failures utilized a significant amount of memory**, except for one. From this observation, we can infer that machines that go down frequently are typically those utilizing the most resources.





4) **Is there any correlation between execution time of tasks in a batch scenario and their CPU usage?**
To investigate the correlation between the execution time of tasks in a batch scenario and their CPU usage, we'll explore the table instance table. We filters out rows where the "real_cpu_avg" column has null values, then calculate the "execution_time" by subtracting "start_timestamp" from "end_timestamp". The plot shows that there is **apparently no correlation between the execution time of tasks in a batch scenario and their CPU usage**. There are tasks with short execution times but high CPU usage, while others exhibit the **opposite pattern**.



5) How we can adjust resource allocation between online service and batch jobs to improve throughput of batch jobs while maintain acceptable service quality for online service?

The average of used CPU over requested for online services: ***9.50%***
The average of used CPU over requested for batch jobs: ***1.22%***

- To address this question suggested in the Alibaba Dataset documentation, we propose calculating the **average percentage of CPU utilization over the requested CPU for both batch jobs and online services**. The intention is to compare these percentages and assess whether one of the two cases utilizes less CPU than originally requested. If such a scenario is identified, it suggests a potential opportunity to reallocate the unused CPU resources to the other case.
- To achieve this, we will work with the instance table, task table, and service instance event table. For online services, the percentage is available in the "cpu_util" column. To obtain the result, we use the AVG function to calculate the average.
- For batch jobs, we begin by selecting the "job_id" and "task_id" columns, as well as the "plan_cpu"(representing the requested CPU) and "real_cpu_avg" (representing the used CPU). After filtering out null values, we perform a join operation on the tables using the job and task IDs. Following the join, we calculate the percentage of used CPU over the requested CPU before computing the average. We observe that **online services, on average, utilize a significantly higher percentage of the requested CPU compared to batch jobs**. So as a proposition for resource **optimization will be reallocating the unused CPU resources from batch jobs to online services**, considering the observed differences in resource utilization.

6. Conclusion:

This experiment provided us with the understanding of how to **analyse and visualise enormous amounts of data**, particularly within a cluster management system. Even the seemingly small yet good practice/learning that we came know, such as the importance of verifying dataset integrity to detect corruption or malicious alterations, the importance of investigating non-mandatory attributes while exploring the dataset, as outliers such as NA or 0 could affect our analysis and using unique key identifier while joining large datasets. The **challenging part** that we faced was when considering **timestamps** to **accurately identify the event occurrence**.

We did an extended work on **Alibaba dataset** taking the oldest one were we analysed and got familiarized with the similar dataset environment as Google Cluster data traces. There **were contrary observations compared to Google cluster data** on its resource utilization and other aspects that we discussed.

Overall, we were able to **explore the Spark's data analysis capabilities on large scale data management**.

7. References:

1) Apache Spark 3.5.- Functions: https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/functions.html
2) Advanced PySpark for explanatory Data Analysis: https://www.kaggle.com/code/tientd95/advanced-pyspark-for-exploratory-data-analysis
3) Pyspark Tutorial: https://www.datacamp.com/tutorial/pyspark-tutorial-getting-started-with-pyspark
4) Pyspark SQL: https://spark.apache.org/docs/2.4.0/api/python/pyspark.sql.html