

Analyzing Data with Spark

Amir Kroudir, Kseniya Shevchuk

January 19, 2021

Contents

1	Introduction	2
2	Data presentation	2
3	Data Analysis	3
3.1	Objective	3
3.2	Approach and tools	3
3.3	Results	3
3.3.1	Distribution of the machines according to their CPU and memory capacity .	4
3.3.2	The percentage of computational time lost due to maintenance	5
3.3.3	The percentage of computational power lost due to maintenance	8
3.3.4	Average number of tasks that compose a job	8
3.3.5	Relation between the scheduling class of a job, the scheduling class of its tasks, and their priority	9
3.3.6	Distribution of chance of eviction for different priorities	9
3.3.7	Percentage of jobs having tasks run on the same machine	10
3.3.8	Relation between the amount of resource consumed by tasks and their priority	11
3.3.9	Correspondence between request of resources and their real consummation . .	14
4	Performance Analysis	14
4.1	Approach	14
4.1.1	Spark	15
4.1.2	Pandas	15
4.1.3	Dask	15
4.2	Results	16
4.2.1	Spark vs Pandas vs Dask DataFrames	16
4.2.2	Spark lazy evaluation VS Spark non lazy evaluation	17
5	Using Google Cloud Platform	18
5.1	Approach	18
5.1.1	Creating DataProc Cluster	18
5.1.2	Connecting to the cluster Jupyter Notebook	20
5.1.3	Loading data into the cluster HDFS	21
5.1.4	Running jobs into the cluster	21
5.2	Results:	22
5.2.1	Performance Analysis on GCP	22
5.2.2	Running one Job on all the Data	24

1 Introduction

The data we studied in this lab have been released by Google in 2011. It represents 29 days of activity in a large scale Google machine (a cluster) featuring about 12.5k machines. It includes information about the jobs executed on this cluster during that period as well as information about the corresponding resource usage.

Our goal for this project was to learn how to work with Big Data. Our main achieved results are:

- We got acknowledged with Spark instrument and studied how it functions.
- We tried different approaches of data analysis provided by Spark (RDD, Dataframes).
- We explored the Google dataset while answering questions.
- We created representative visualisations in order to better demonstrate correlations between different types of data.
- We extended our work by using different tools to work with Big Data (such as Spark Dataframes, Pandas DataFrames and Dask Dataframes).
- We compared performance results of the tools.
- We also extended our work by using Google Cloud platform to compare between the tools.

Detailed description of the results is presented in this report and in the code provided alongside.

2 Data presentation

Our study was done on the data collected by Google about usage traces of a Google compute cell. This data concerns more than 12500 machines during the time lapse of 29 days. Data is stored in Google Storage public bucket accessible by the link [gs://clusterdata-2011-2/](https://clusterdata-2011-2/). Data can be downloaded using `gsutil`. Data is divided into subfolders, each folder contain files that represent a part of one of the following tables:

- Machine events
- Machine attributes
- Job events
- Task events table
- Task constraints
- Task usage

Due to limitations of our own computers we weren't able to run the code on all data provided. We had to limit final runs on small part of data. E.g. for **Job events** we downloaded and used only 100 files out of 500 and for **Task usage** only 10 files out of 500. Unfortunately our attempts to run the code on larger dataset led to a crush of system.

3 Data Analysis

3.1 Objective

Our work focused on using the previous described data to answer some key interesting questions :

- What is the distribution of the machines according to their CPU and memory capacity?
- What the percentage of computational time lost due to maintenance (a machine went offline and reconnected later)? what is the distribution of this lost over CPU capacity and memory capacity?
- What the percentage of computational power lost due to maintenance (a machine went offline and reconnected later)?
- On average, how many tasks compose a job?
- What can you say about the relation between the scheduling class of a job, the scheduling class of its tasks, and their priority?
- Do tasks with low priority have a higher probability of being evicted?
- In general, do tasks from the same job run on the same machine?
- Are the tasks that request the more resources the one that consume the more resources?
- Is there a relation between the amount of resource consumed by tasks and their priority?

3.2 Approach and tools

In order to answer the questions cited above, we used **Jupyter Notebooks** and **PySpark** as main tools, with the support of some python general purpose libraries and standard libraries used for Data Analysis such as **Pandas** for handling data in the form of DataFrames, **matplotlib** and **seaborn** for plotting Data into visualisations to better answer our questions.

Generally, our approach to the our data analysis is :

- We read the data into RDD or a Spark DataFrame to be able to handle it.
- Do some exploratory data analysis to gain some insights about the data sets
- Manipulate the data-set using filtering, merging, joining and other operations in order to create new data-sets that we can use to answer our questions.
- Use the new data-sets to create insightful visualisations that enable us to answer our questions.

3.3 Results

In this part we detail the results of our analysis with the appropriate visualisations and comments to answer the questions [3.1](#). Description of how we we did each of our analysis including the code is given in the notebook associated with this submission. For more information regarding the code presentation, please consult `README.md`.

3.3.1 Distribution of the machines according to their CPU and memory capacity

According to the documentation: “*The CPU capacity, and Memory capacity are normalized and divided into categories: for example 1 corresponds to the highest CPU/memory capacity*”.



Figure 1: Distribution of machines over CPU capacity

For CPU capacity (figure 1), we observe that 92.66% of the machines have medium CPU capacity (CPU capacity = 0,5) compared to machine with highest computing capacity (CPU capacity = 1), there latter represent 6% of the machines, and the rest in between the machines we don't have information about and the slow machines (CPU capacity = 0.25).

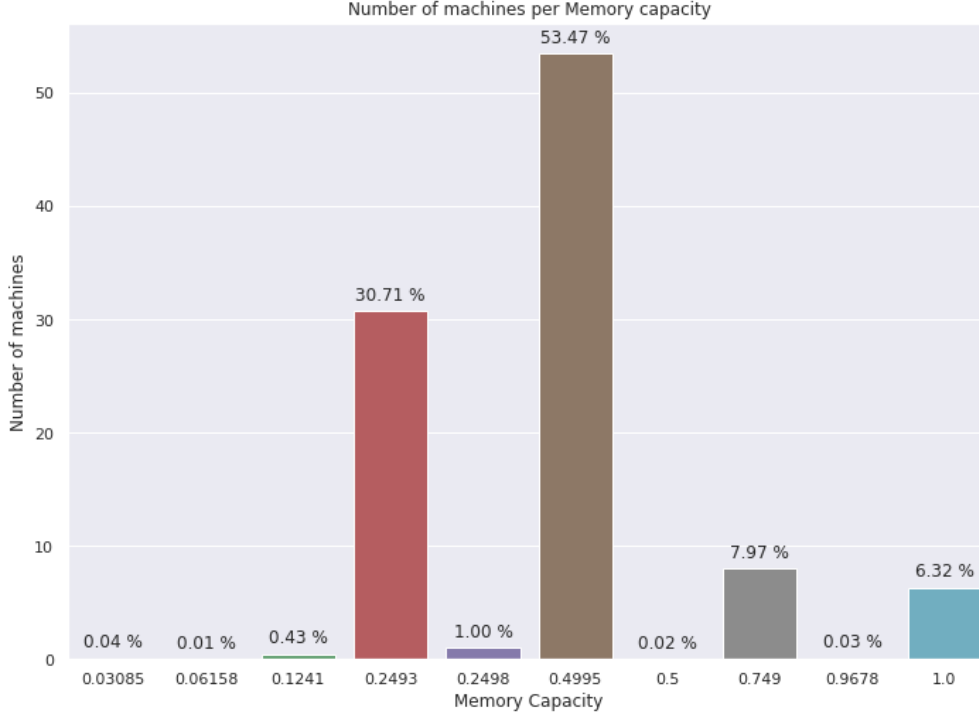


Figure 2: Distribution of machines over Memory capacity

For the memory capacity (figure 2), we notice that 53.47% of the machines are with medium memory capacity (memory capacity = 0.4995) compared to the high end machines, while machines with around 25% memory capacity compared to the high end machines represent 30.6%.

Over all, we notice that Google Cluster has a majority of medium capacity machines in term of CPU and memory capacity which can be used for multiple purposes and serve multiple functions in Google cluster, with some high end machines that we guess have special tasks that require special capacities. There is also some low end machines that we guess are old machines that are still being used in the cluster.

3.3.2 The percentage of computational time lost due to maintenance

Before elaborating on the results, we would like to explain briefly what do we consider as lost computing time. The lost computing time is the sum of all the differences between the time that the machine was deleted from the cluster for maintenance and then added again. In case the machine was deleted and never added again in our trace period, we add the time between the last event of deletion of the machine and the theoretical maximum time that the machine should've been operating.

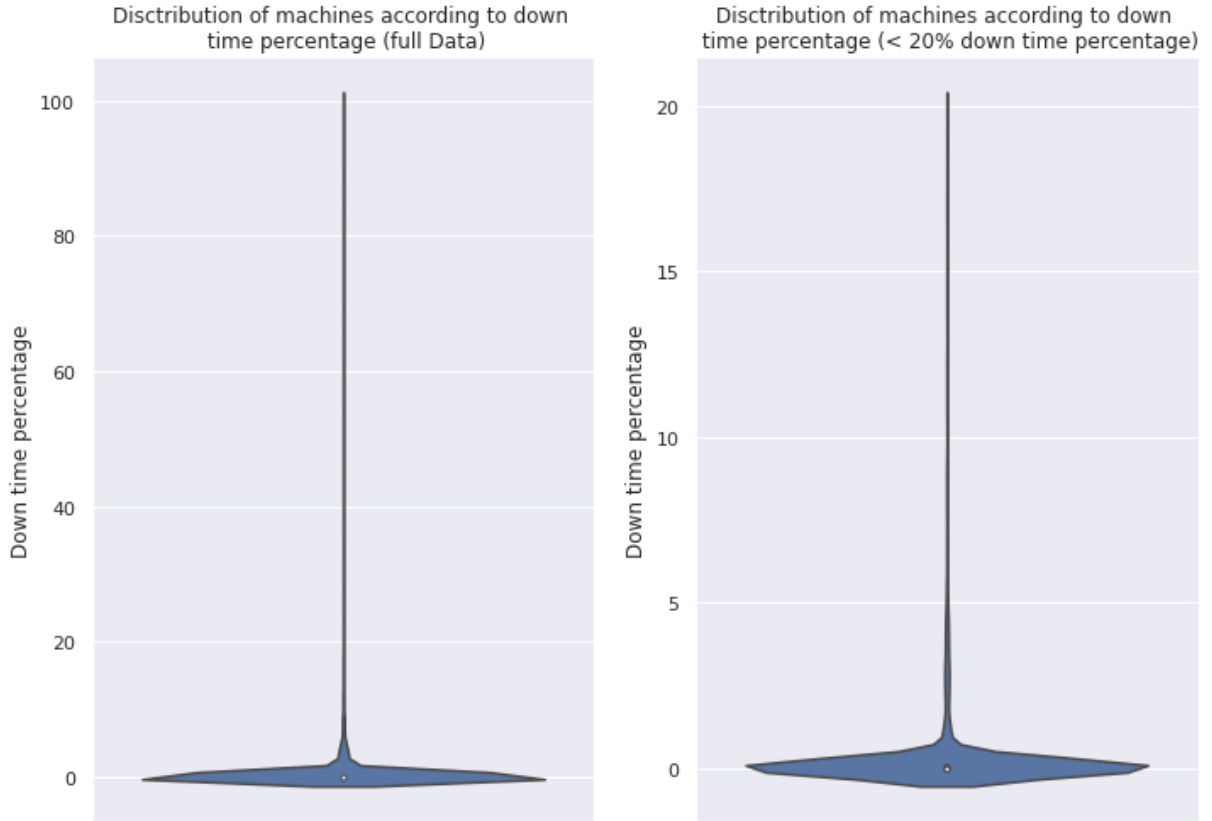


Figure 3: Distribution of lost time

Using those two violin plots (figure 3), we see that most of machine in the cluster has a lost time near 0, while there is some machines that have a lost time near 100%.

After seeing the global trend in the down time, we thought it would be interesting to see how the down time is distributed over CPU capacities and Memory capacities.

Results of distribution of the down time over CPU capacities (figure 4):

- We notice that the low end machines of 25% CPU capacity are more sparsed around 0 which means that they are the ones with more lost time due to maintenance, which makes sense and support our hypothesis that they are old machines still operating.
- We notice also that mid-range machines are the ones that has less lost time, compared to the high end machines.

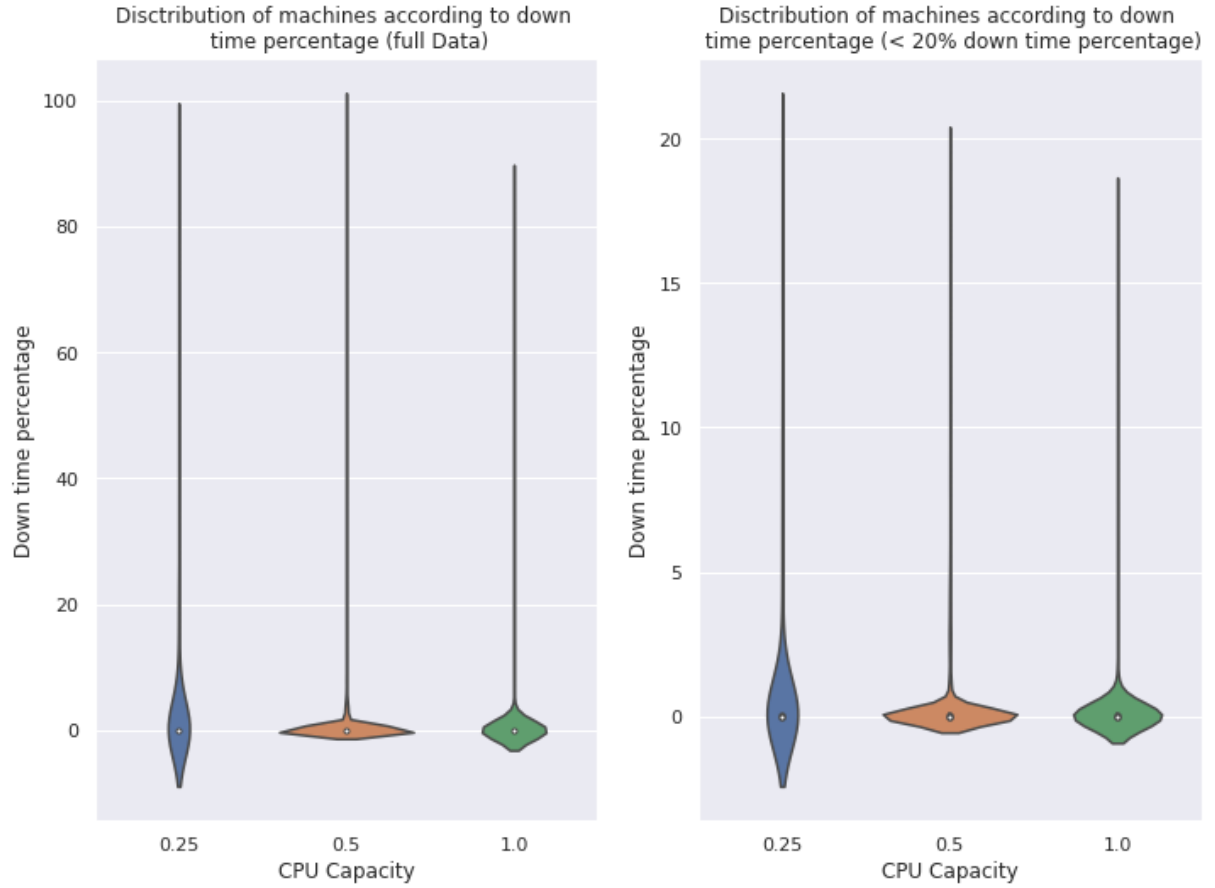


Figure 4: Distribution of lost time over CPU capacity

Results of distribution of the down time over Memory capacities (figure 5):

- From the sparcity of the data, this graph is hard to read, but we can tell that machines with lower Memory capacity have higher lost computation times compared to machines with larger Memory capacity. Comparison between machines with 12% Memory capacity and machines with near 50% of capacity shows that the first have lost time between 15% and 17.5% while the second have lost time between 5% and 14%.

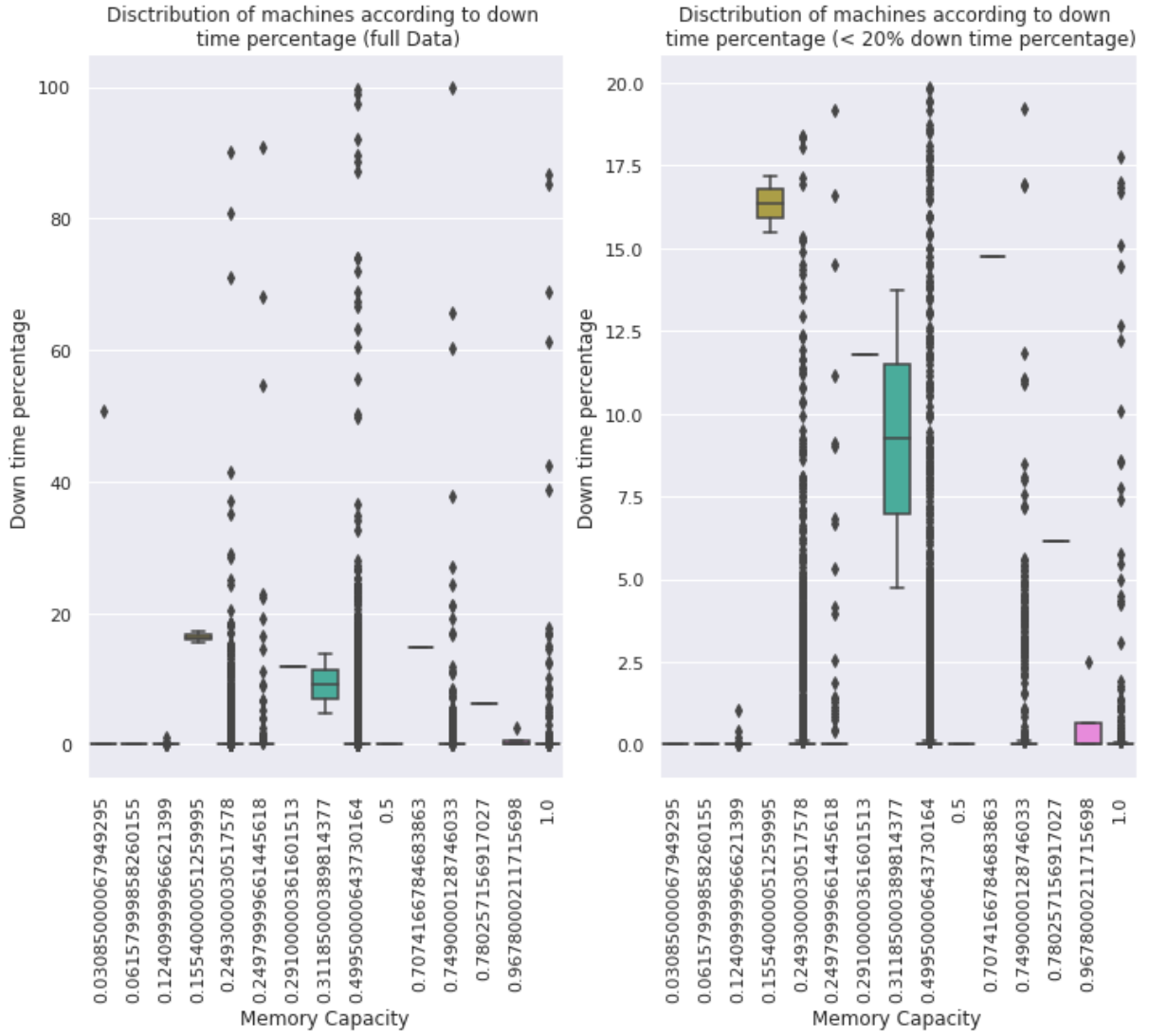


Figure 5: Distribution of lost time over Memory capacity

Overall, the average percentage of lost time per machine is **0.71%**.

3.3.3 The percentage of computational power lost due to maintenance

The average percentage of the computational power lost per machine is **0.37%**. This percentage is calculated by weighting the list time percentages by the CPU capacity (more details in the notebook).

3.3.4 Average number of tasks that compose a job

Working the `task_events` tables let us collect number of tasks composing each job. Which allowed us to calculate the average of this number. The calculations were run on 20 files of `task_events` tables and the result is **403** tasks in average compose one job.

3.3.5 Relation between the scheduling class of a job, the scheduling class of its tasks, and their priority

According to the documentation each job consists of tasks and it is logical to suppose that tasks of a job have the same scheduling class. However this hypothesis was firstly verified which you can find in the source code provided alongside with our report.

Making sure that the scheduling class of a job doesn't change let us work straightly with tasks. Another attribute of a task is its priority. To see if there is a relation between scheduling class and priority we gathered all tasks of the same scheduling class and then calculated the mean of priority. The results can be observed on the figure 6.

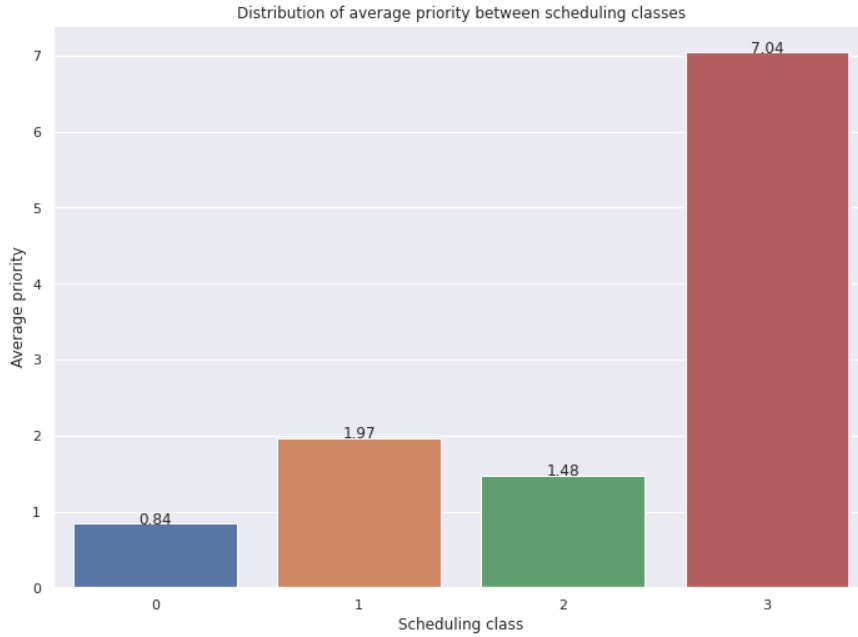


Figure 6: Distribution of average priority for each scheduling class

While observing the graphic we would like to cite the documentation:

“Note that scheduling class is not a priority, although more latency-sensitive tasks tend to have higher task priorities.”

The results seem to prove this point completely. The most latency-sensitive tasks (scheduling class of which is 3) in average are given a priority of 7, which is way more than for any other scheduling class. Also we observe that in average priority assigned to scheduling class 1 is slightly higher than priority assigned to scheduling class 2. This observation also proves the citation that having higher scheduling class does not necessarily guarantees having higher priority. Also as expected the least latency-sensitive tasks are not prioritized which indicates us that the tasks of this scheduling class tend to wait their turn longer.

3.3.6 Distribution of chance of eviction for different priorities

One of possible events that can happen to a task is being evicted. A task or job may be descheduled because of a higher priority task or job, because the scheduler overcommitted and the actual demand exceeded the machine capacity, because the machine on which it was running became unusable (e.g. taken offline for repairs), or because a disk holding the task's data was lost.

In order to observe if there's a correlation between a task priority and its chance to be evicted we gathered tasks by priority and calculated how many times they were evicted. And then calculated what percentage this number makes of all times any task of any priority was evicted (sum of all lines with `event_type = 2`).

The result can be observed on figure 7.

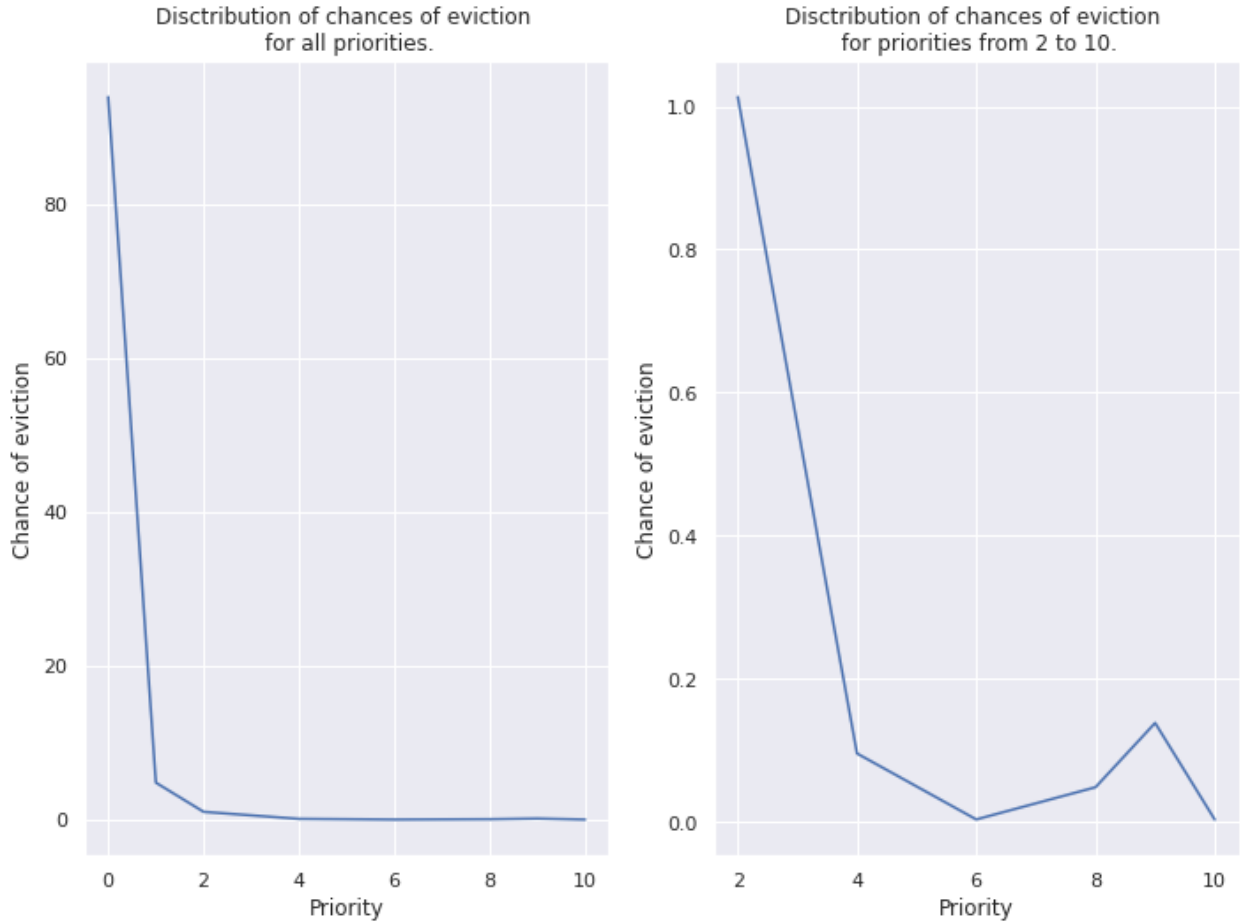


Figure 7: Distribution of chance of eviction (both graphics in %) for each priority

The research has shown that tasks of priority 0 got evicted 93% of times which is not surprising. Also quite often tasks of priority 1 got evicted as well (around 4%). The rest of priorities are in average evicted less often. This is represented on the left graphic.

To be able to zoom on higher priorities and to see the distribution of their chance of eviction we excluded from the resulting calculation values of priorities 0 and 1 (see the right graphic). For some reason tasks with priority 9 are evicted more often then tasks with lower priority.

Still, looking at overall results we can conclude that the lowest priorities are evicted very often.

3.3.7 Percentage of jobs having tasks run on the same machine

To answer this question it is sufficient to gather all `machine_id`'s for the same `job_id` to see if there's multiple machines for tasks of the same job. The final distribution of percentage of jobs having tasks run on the same or on multiple machines is represented on the pie chart 8.

Distribution of jobs that run on the same machine and jobs that run on multiple machines

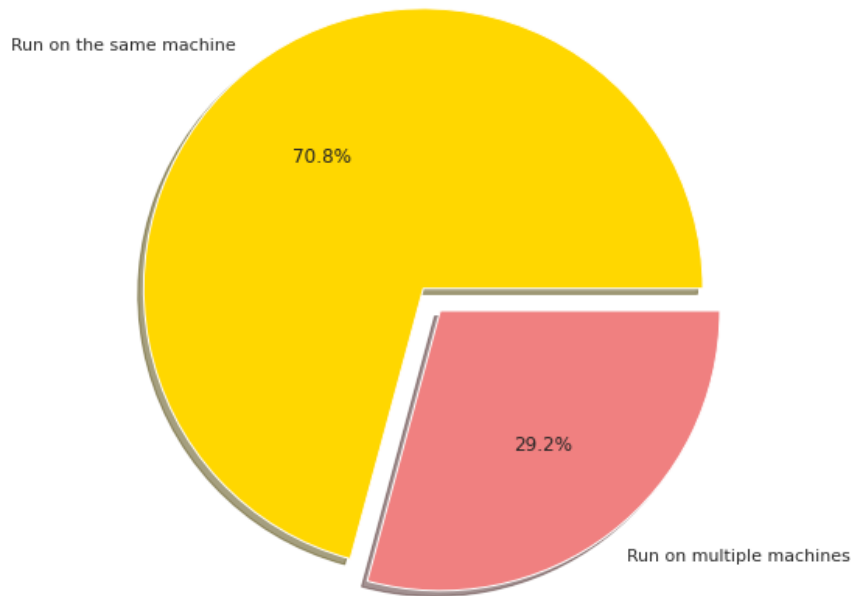


Figure 8: Distribution of percentage of jobs having tasks run on the same machine

As we can see, around **71%** of jobs have their tasks running on the same machine.

3.3.8 Relation between the amount of resource consumed by tasks and their priority

In the `task_usage` table we may observe different types of resource consumption such as memory, disk space, CPU usage. To have an answer we made a distribution of average of each of these resources usage for each priority. Corresponding graphics are [9](#), [10](#), [11](#), [11](#), [12](#), [13](#).

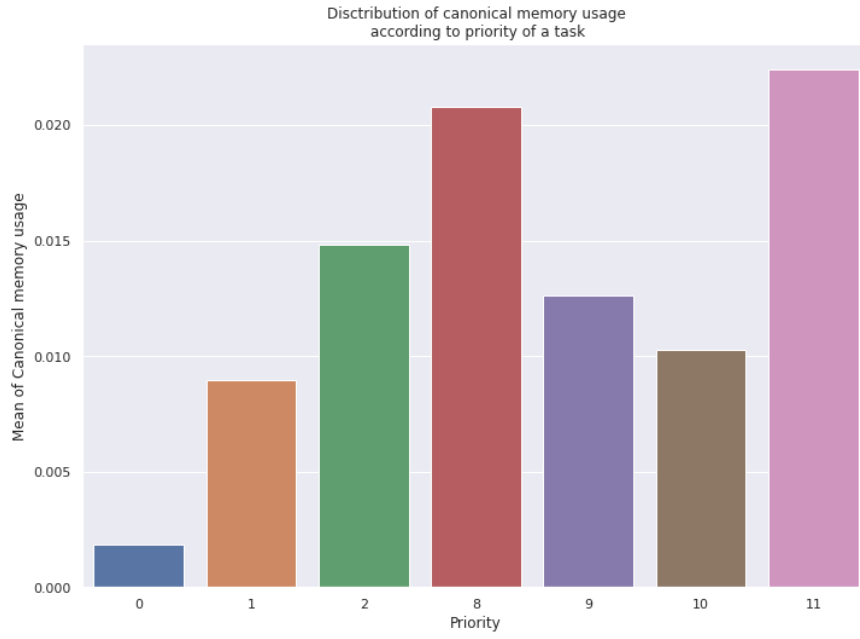


Figure 9: Distribution of canonical memory between priorities

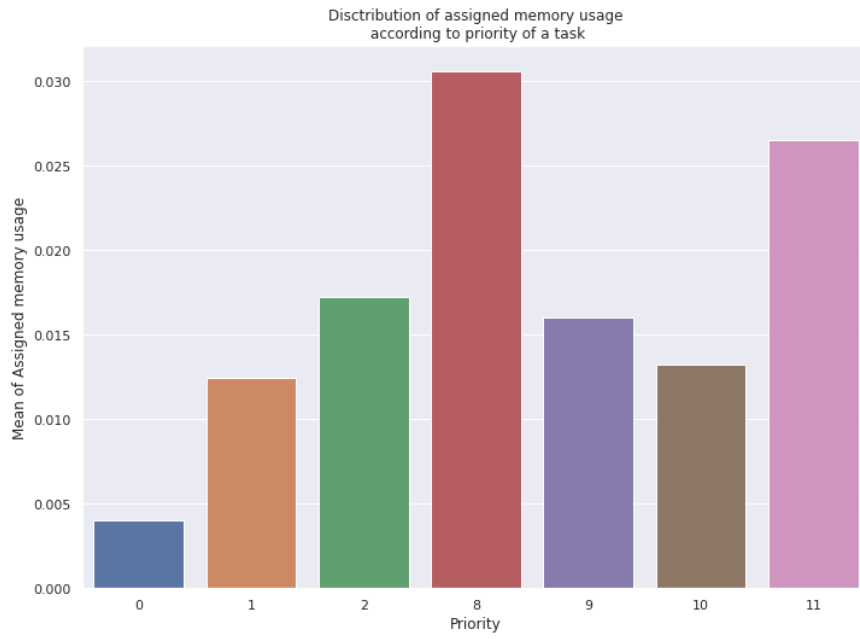


Figure 10: Distribution of assigned memory between priorities

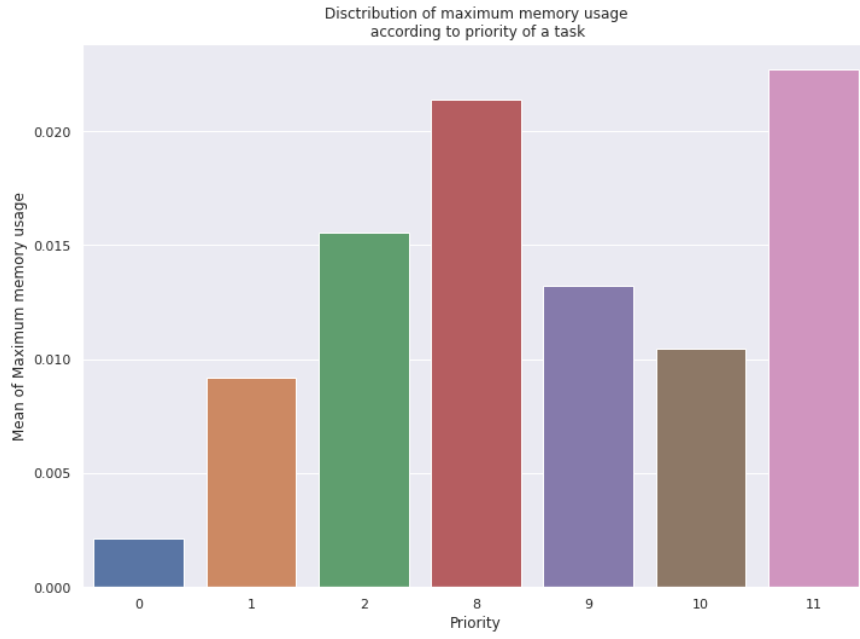


Figure 11: Distribution of maximum memory between priorities

In average, tasks of priority 8 and 11 consume most of canonical, assigned and maximum memory while execution.

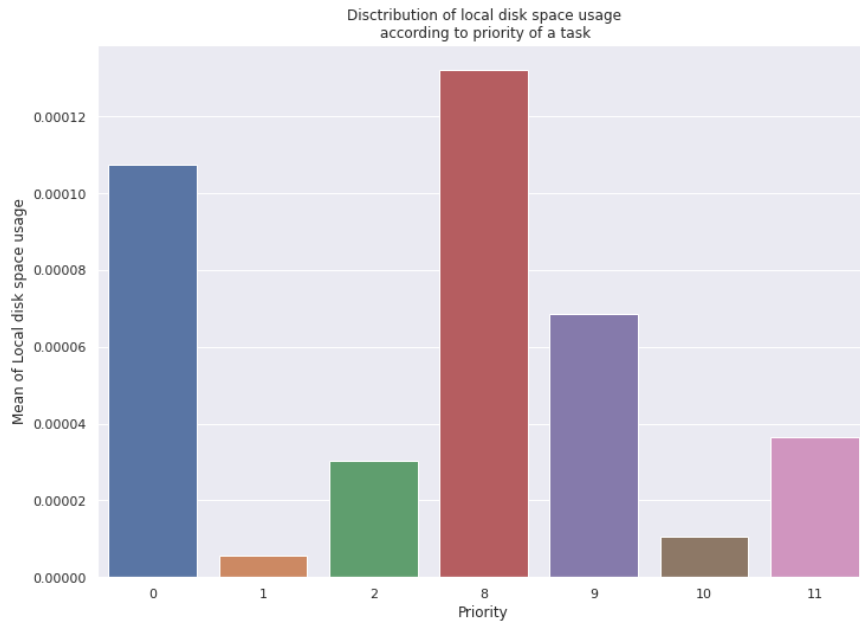


Figure 12: Distribution of disk space usage between priorities

In average, tasks of priority 1 and 8 consume the most of local disk space while execution.

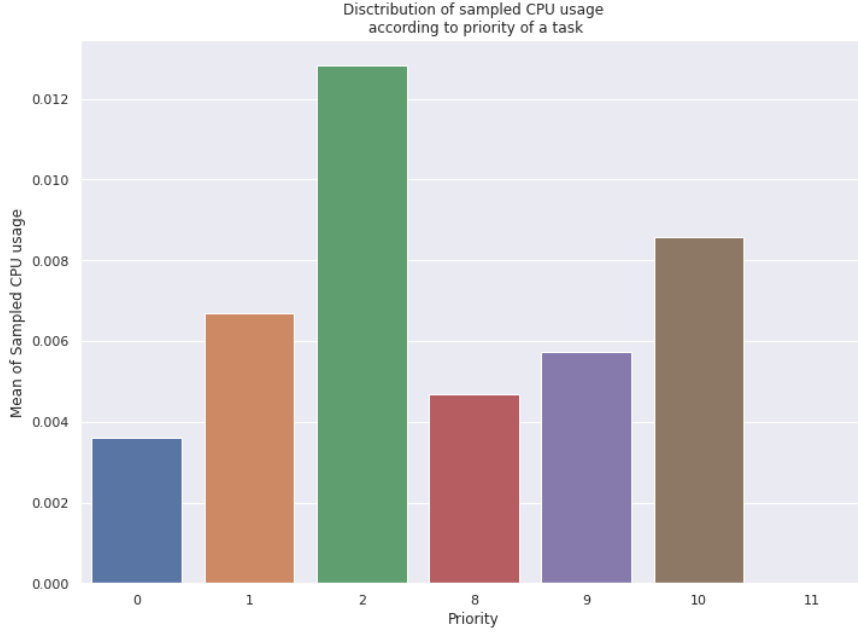


Figure 13: Distribution of sampled CPU usage between priorities

In average, tasks of priority 2 and 10 consume the most of sampled CPU while execution.

Overall we may conclude that even if in specific groups of resource usage (e.g. memory usage) we can observe a certain correlation, it does not necessarily repeats with other type of resources.

3.3.9 Correspondence between request of resources and their real consummation

We chose this question to extend our work and to run performance analysis. For this reason we placed our approach and conclusions in the next section.

4 Performance Analysis

To extend our work, we decided first to do a performance analysis to compare between different tools such as Spark Dataframes, Pandas DataFrames and Dask Dataframes, and then to focus on Spark and compare performances with using lazy evaluation and without using it.

While doing our comparison, we answer the question:

- Are there tasks that consume significantly less resources than what they requested?

4.1 Approach

To do our analysis, we created 4 scripts to track the execution time of 6 operations (reading Data, Selecting subset of the data, join operation, Simple filtering, Filtering on multiple conditions, Getting the final result) while answering the already mentioned question.

Before listing the 4 scripts and what they do, we first introduce the three used tools :

4.1.1 Spark

Apache Spark is an open-source distributed general-purpose cluster-computing framework. Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance.

4.1.2 Pandas

Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

4.1.3 Dask

Dask is an open source library for parallel computing written in Python

In order to compare between the three libraries, we use 4 scripts :

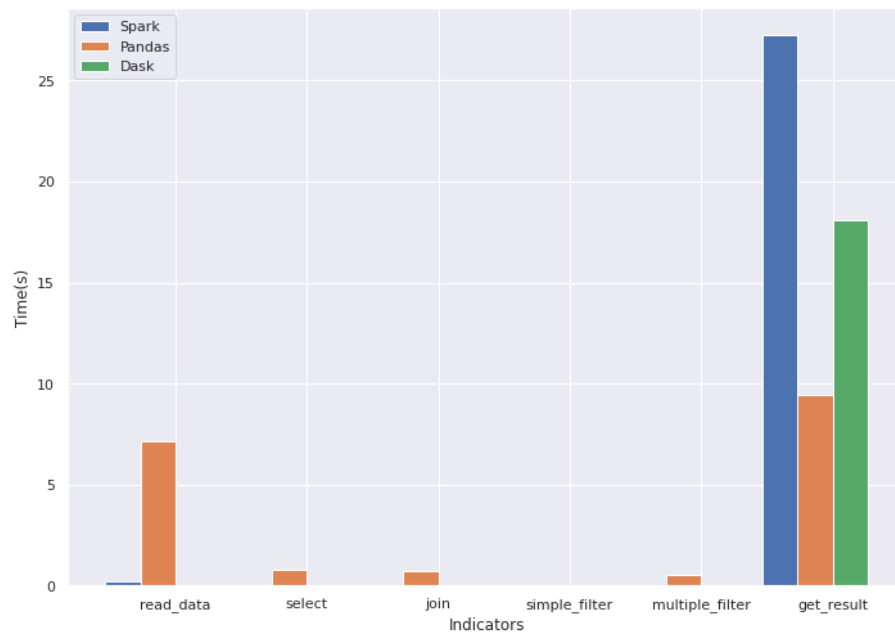
<code>performance_analysis_pandas.py</code>	Using Pandas
<code>performance_analysis_dask.py</code>	Using Dask
<code>performance_analysis_spark.py</code>	Using Spark with lazy evaluation
<code>performance_analysis_spark_not_lazy.py</code>	Using Spark without lazy evaluation

Note

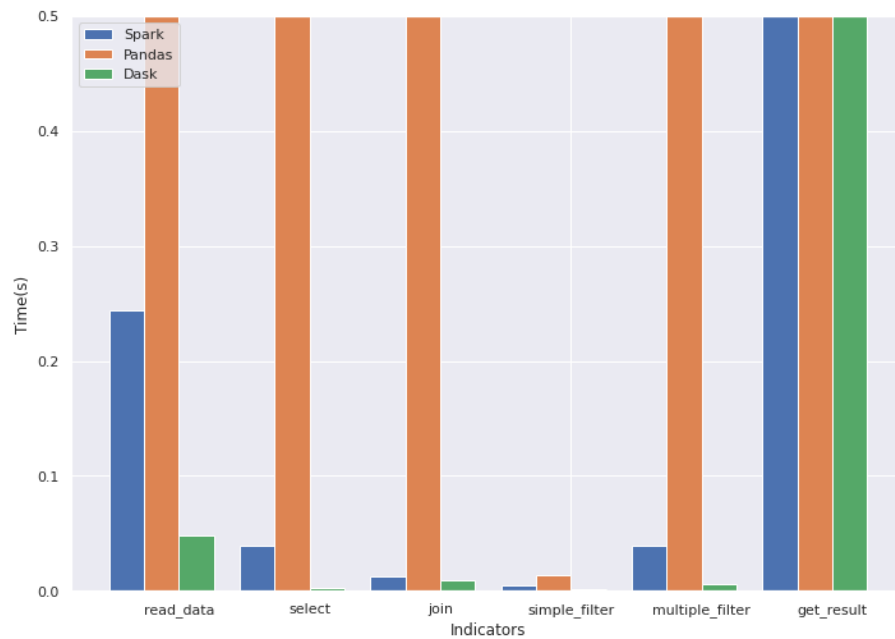
Tests are executed 20 times, and then we use the average of the 20 tests for each scenario to get the times we use in our plots. For this part we used only one file for each table since it takes a while to get results.

4.2 Results

4.2.1 Spark vs Pandas vs Dask DataFrames



(a) Full Y axis



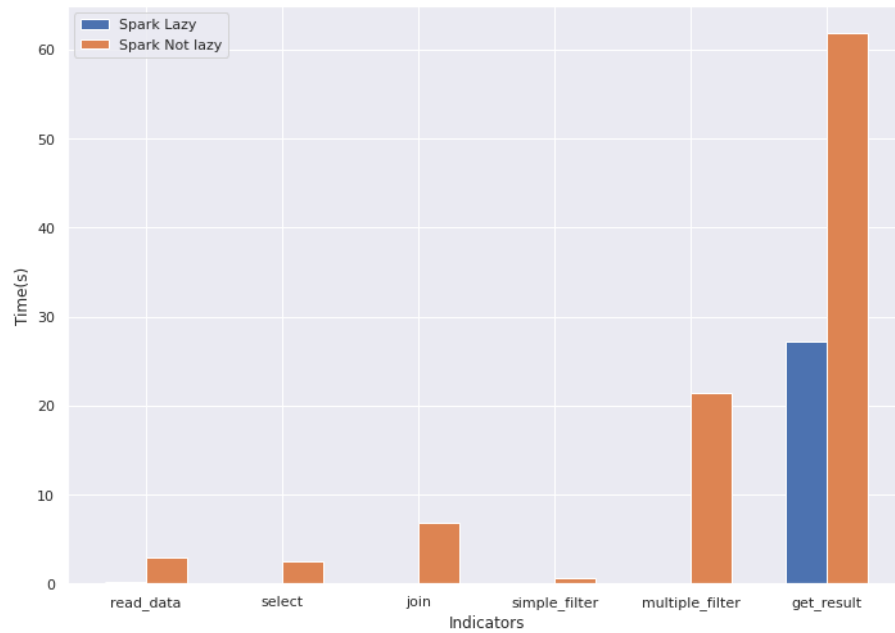
(b) Limited Y axis to 0.5

Figure 14: Spark VS Pandas Vs Dask

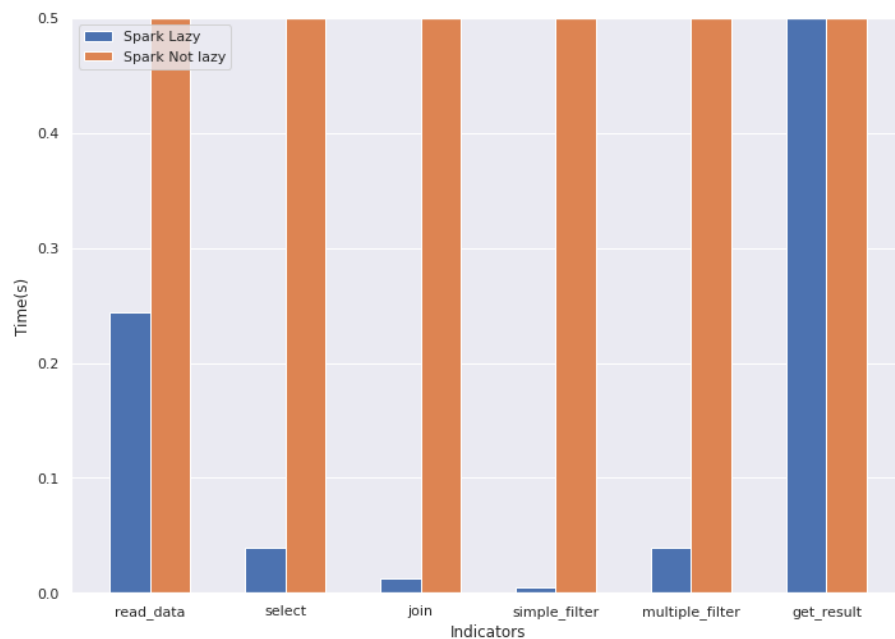
After running the three scripts on a local machine the results show the superiority of Pandas compared to Spark and Dask (figure 14). This can be explained by the fact of using one local machine, and hence we can't see the impact of parallelism on this scale. We will see in the last

part of using google cloud if the results will change when using a cluster.

4.2.2 Spark lazy evaluation VS Spark non lazy evaluation



(a) Full Y axis



(b) Limited Y axis to 0.5

Figure 15: Spark Lazy VS Spark not Lazy

Analyzing figure 15 we notice that the Lazy evaluation approach performs better than the approach that doesn't use the Lazy evaluation. This can be explained by the fact that in the no lazy evaluation

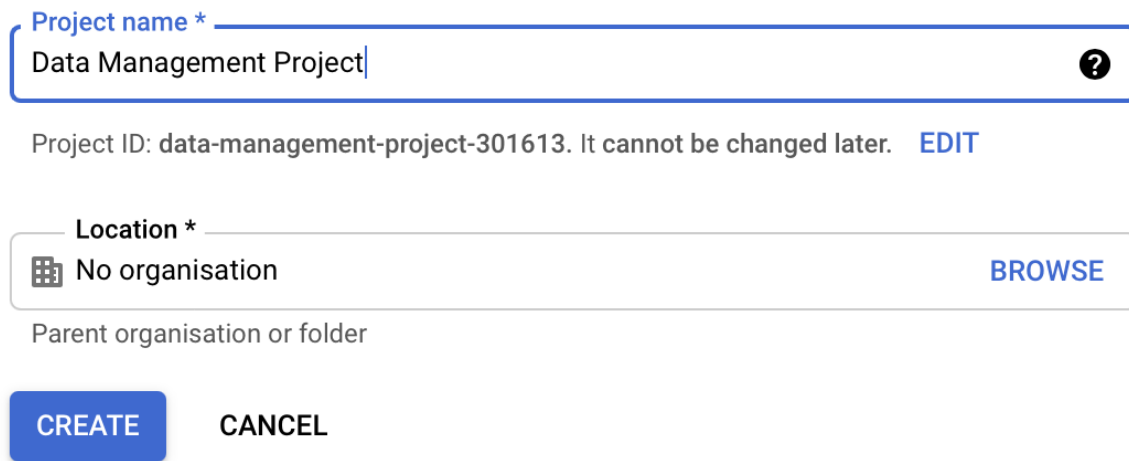
approach we do not have an optimization engine like **Catalyst** optimizer and **Tungsten** execution engine. In the contrary, when using lazy optimisation we use **Catalyst** to generate optimized logical and physical query plan.

5 Using Google Cloud Platform

After testing the performance on the local machine, we decided to take it a step further and extend our work once again by using Google Cloud platform to compare between Pandas, Dask and Spark DataFrames.

5.1 Approach

In order to do our analysis we have to follow multiple steps, starting with creation a Google Cloud platform project called *Data Management Project* (figure 16).



Project name *
Data Management Project ?

Project ID: data-management-project-301613. It cannot be changed later. [EDIT](#)

Location *
No organisation [BROWSE](#)

Parent organisation or folder

CREATE **CANCEL**

Figure 16: Creating GCP Project

5.1.1 Creating DataProc Cluster

Before running our Analysis we started by creating a Dataproc ¹ cluster. In order to do that, we used Google Cloud Shell and ran the command :

```
gcloud beta dataproc clusters create dm-spark-cluster --region us-central1 \
  --zone us-central1-f \
  --master-machine-type n1-standard-4 \
  --master-boot-disk-size 500 \
  --num-workers 2 \
  --worker-machine-type n1-standard-4 \
  --worker-boot-disk-size 500 \
  --image-version 1.5-debian10 \
```

¹Dataproc : Big data platform for running Apache Hadoop and Apache Spark jobs.

```
--optional-components ANACONDA,JUPYTER,ZOOKEEPER \  
--project data-management-project-301613 \  
--enable-component-gateway
```

Explanation the flags used:

- **dm-spark-cluster:** The name of the cluster.
- **-region :** the region where the cluster will be created.
- **-zone :** the zone where the cluster will be deployed.
- **-master-machine-type :** the type of the master machine.
- **-master-boot-disk-size :** the size of the disk of the master machine.
- **-num-workers :** the number of workers, in our case it is 2, so our cluster will have a total of 3 machines, two workers and one master.
- **-worker-machine-type :** the type of the worker node.
- **-worker-boot-disk-size :** the size of the disk of the worker machines.
- **-image-version :** the image version that will be installed on the cluster, we choose the version Debian 1.5 since it comes with `Python3`
- **-optional-components :** optional components that will be installed, in our case we used ANACONDA, JUPYTER and ZOOKEEPER.
- **-project :** the project ID to which this cluster will be assigned.
- **-enable-component-gateway :** Enabling external connections to Jupyter Notebook from a server. The connection will be secured using `SSH` ²

²Secure Shell is a cryptographic network protocol for operating network services securely over an unsecured network

Dataproc

Clusters

Jobs

Workflows

Auto-scaling policies

Component exchange

Metastore

Notebooks

Clusters

+ CREATE CLUSTER

REFRESH

DELETE

REGIONS

Search clusters, press Enter

<div></div>	Name <div></div>	Region	Zone	Total worker nodes	Scheduled deletion	Cloud Storage staging bucket
<div></div>	<div></div> dm-cluster	global	europa-west1-b	2	Off	dataproc-staging-e7543397537445-e754339753744

(a) Dataproc Dashboard

VM instances	+ CREATE INSTANCE	↓	↺	▶	■	⏸	🔄	🗑	MANAGE ACCESS
Filter VM instances									Columns ▾
<input type="checkbox"/>	Name ^	Zone	Recommendation	In use by	Internal IP	External IP	Connect		
<input type="checkbox"/>	dm-spark-cluster-m	us-central1-f			10.128.0.19 (nic0)	34.68.84.211	SSH ▾	⋮	
<input type="checkbox"/>	dm-spark-cluster-w-0	us-central1-f			10.128.0.21 (nic0)	34.67.81.184	SSH ▾	⋮	
<input type="checkbox"/>	dm-spark-cluster-w-1	us-central1-f			10.128.0.20 (nic0)	23.251.145.145	SSH ▾	⋮	

(b) Compute Engine Dashboard 0.5

Figure 17: GCP dashboards after creating the cluster

On the figure 17, we see Dataproc dashboard after creating the cluster as well as Compute Engine³ dashboard. As we can see in the Compute Engine dashboard, our script created 3 virtual machines, one as the master node and two worker nodes.

5.1.2 Connecting to the cluster Jupyter Notebook

After creating the cluster, we connect to Jupyter Notebook using the web interface (figure 18).

³Compute Engine: Infrastructure as a Service to run Microsoft Windows and Linux virtual machines.

SSH tunnel

[Create an SSH tunnel to connect to a web interface](#)

Component gateway

Provides access to the web interfaces of default and selected optional components on the cluster. [Learn more](#)

Disabled

Equivalent [REST](#)

Figure 18: Connecting to the cluster notebook

After that we launch a terminal on this notebook to progress to the next step.

5.1.3 Loading data into the cluster HDFS

Before loading the data into the HDFS, first we load it into the cluster by using `gsutil`:

```
gsutil cp -R gs://clusterdata-2011-2/task_events ./data/task_events
gsutil cp -R gs://clusterdata-2011-2/task_usage ./data/task_usage
```

We load the data into the HDFS using the following commands:

```
hadoop fs -put ./data/task_events /task_events
hadoop fs -put ./data/task_usage /task_usage
```

After running those commands, we check that we successfully loaded the data into HDFS by running

```
hadoop fs -ls /
```

```
root@dm-spark-cluster-m:/# hadoop fs -ls /
Found 5 items
drwxr-xr-x   - root hadoop          0 2021-01-15 01:38 /data
-rw-r--r--   2 root hadoop    4139742 2021-01-15 01:39 /task_events
-rw-r--r--   2 root hadoop    91723415 2021-01-15 01:39 /task_usage
drwxrwxrwt   - hdfs hadoop          0 2021-01-15 01:31 /tmp
drwxrwxrwt   - hdfs hadoop          0 2021-01-15 01:31 /user
root@dm-spark-cluster-m:/#
```

Figure 19: Listing HDFS files

5.1.4 Running jobs into the cluster

After loading the data into HDFS, all what was left to do is to adapt the python scripts to read the data from HDFS, load them into the Master node in the cluster, and run them to get results using command line :

```
python scriptFileName.py
```

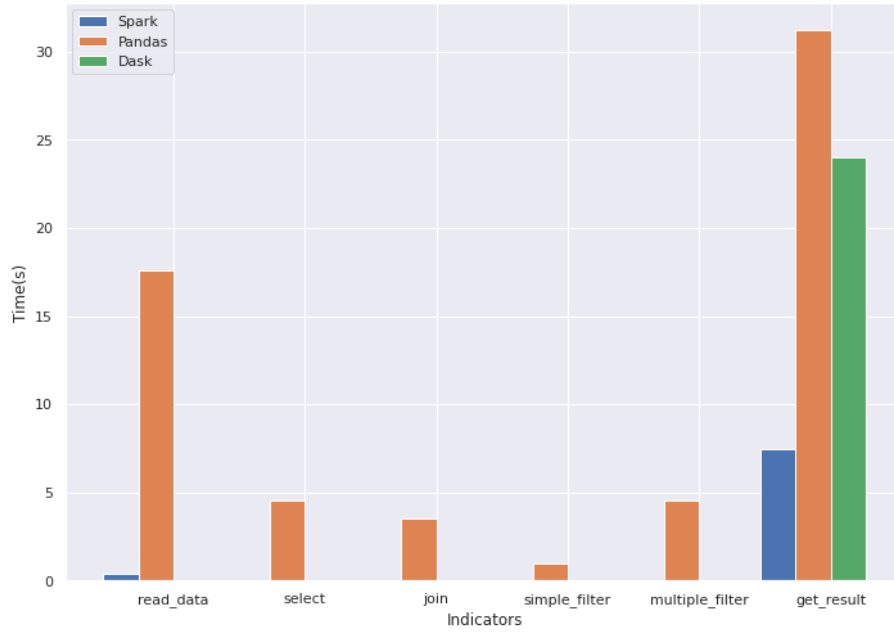
The scripts to we used are the following:

<code>performance_analysis_pandas_gcp.py</code>	Using Pandas
<code>performance_analysis_dask_gcp.py</code>	Using Dask
<code>performance_analysis_spark_gcp.py</code>	Using Spark with lazy evaluation

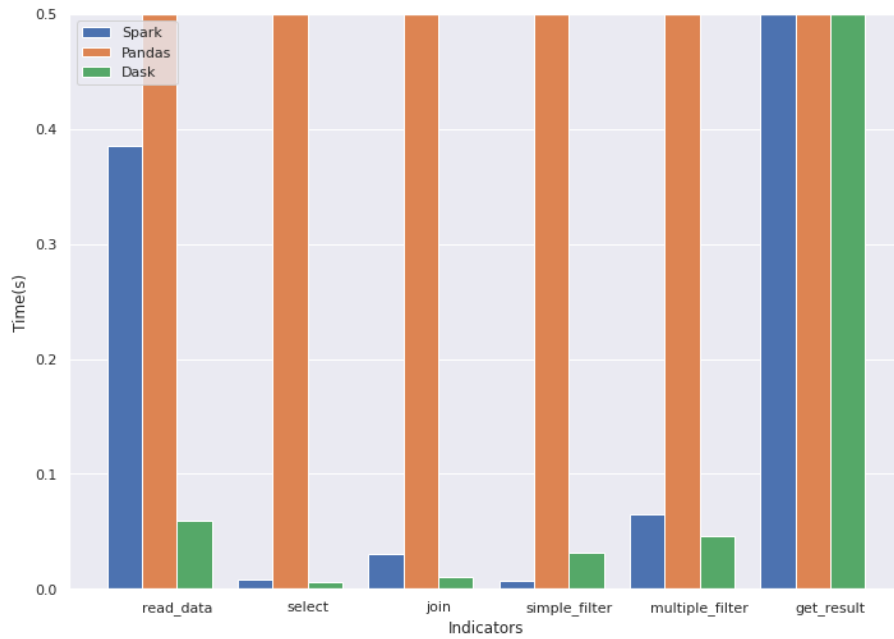
5.2 Results:

5.2.1 Performance Analysis on GCP

After following the approach detailed above, we managed to run our scripts in the cluster, and the most interesting result was the fact that when comparing between Spark, Dask and Pandas, the results were different this time.



(a) Full Y axis



(b) Limited Y axis to 0.5

Figure 20: Spark VS Dask VS Pandas on GCP

As figure 20 illustrates, we see that Spark take the lead when excuting in the cluster compared to Dask and Pandas. This illustrates the added value on Spark and its robustness while dealing with large Datasets. We notice also that Dask improves compared to Pandas as a result on the parallelism executed in the Cluster.

5.2.2 Running one Job on all the Data

Taking advantage of the computational power provided by our cluster, we decided to run a python script `full_dataset_spark_gcp.py` on the full dataset that has the size of near **42 GB**.

After running this script we got the result that **27.3%** of the tasks request less resources than needed. The exclusion time for this job was over **one hour**.