Big Data Coursework Part 2

Célia Detrez

20/05/2020

Abstract— This project has for objective to understand and analyse parallelisation and machine learning in the cloud. Different strategies are compared from experiments and literature results to define concrete strategies for different scenarios. A public "Flowers" dataset (3670 images, 5 classes) is used to produce results **[7]**. The link of the notebook can be found in the Appendix.

1 Data pre-processing

1.1 Speed test analysis (Task 1b)

The influence of the batch size, the batch number, the repetition on the throughput in images per second of the pre-processing task is analysed in Fig. 1. The dataset is batched in *batch_size* partitions and *batch_number* batches are selected. The files are then read. The mean line gives a better estimate of the relationship between the throughput and the parameters. It highlights the limitations of the linear regression.

The throughputs in images per second for the TFRecord files dataset (datasetDecoded) are significantly higher than the throughputs for the Image files dataset (dataset2) regardless of the parameters. For both datasets, it increases with the batch size, the batch number and the product of the batch size and batch number. The variation of throughputs for different repetition of the task reflects the effect of the noise for Image files when accessing files. As there are less TFRecord files, the test on TFRecord files is more robust than the test on Image files. The multilinear and linear regressions show that there is a linear relationship between the batch size and the time: a p-value inferior to 0.05 is observed for either the batch size or the product of the batch size and the batch number (see notebook).

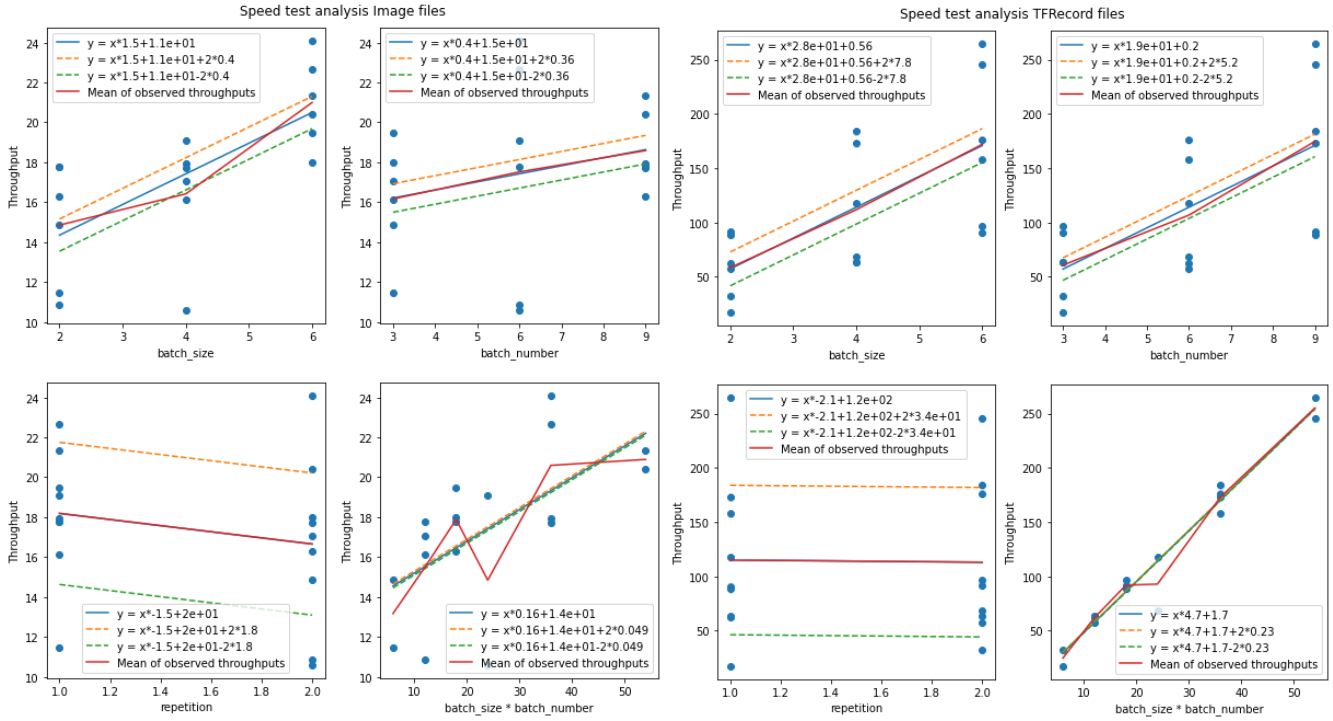


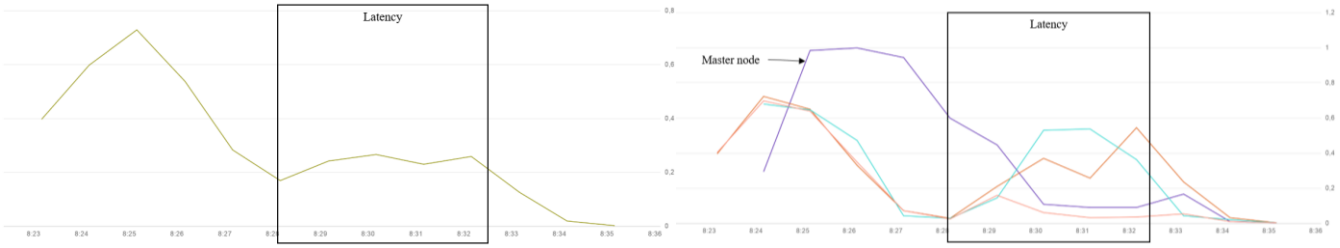
Fig. 1. Speed test analysis.

1.2 Parallelising the speed test with Spark in the cloud

1.2.1 Set up a cluster and run the script. (Task 2b)

A cluster with 4 machines of a single resource (1 vCPUs, memory, disk) has been set up. By default, the function parallelize sets the number of partitions to 2 **[3]**. Indeed, in Fig. 2. only two nodes use more than 0.2 of the CPUs available (in blue and

orange). In theory, each task should take the same time but in practice, running times can be different as it depends on the resources.

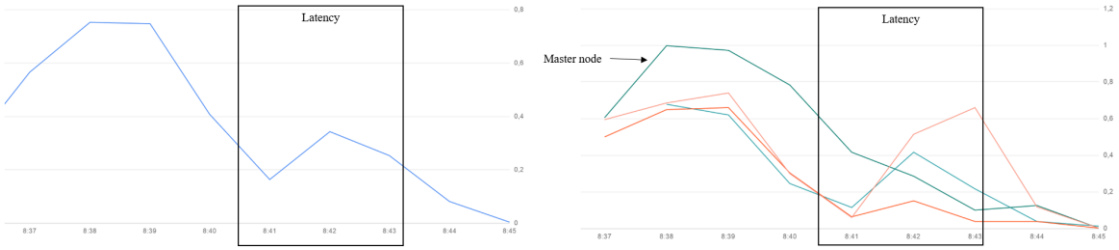


a. Mean CPU 1min window (% CPU). b. Network load (% CPU).

Fig. 2. Cluster performances.

1.2.2 Improve cluster efficiency (Task 2c)

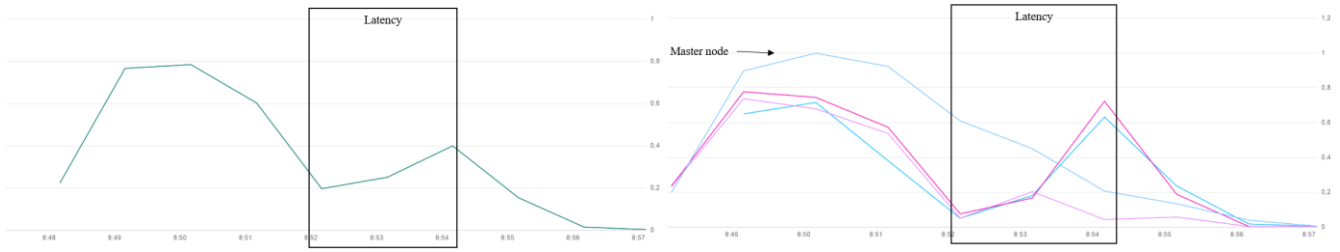
Caching is a technique that allows saving the previously read results to speed up the computation. It is useful if the same RDD is used several times. Caching saves the values after the first time that an RDD is collected. Fig. 3b underlines that the computation is done on 2 nodes. The latency is lower with caching than without caching.



a. Mean CPU 1min window (% CPU). b. Network load (% CPU).

Fig. 3. Cluster performances using caching.

As stated in 1.2.1, the computation is done on only two nodes, speeding up the computation is achievable executing the computation on more nodes. Adding a second parameter in the calling of the *parallelize* function allows splitting the task between all the available nodes. It is recommended to set this number to a high enough number to get at least 2-3 tasks per CPU core **[4]**. I set it to 16, so each node should have between 4 and 6 tasks to compute. Fig. 4b shows that the computation is done mostly on 2 nodes out of the 3 available worker nodes. Indeed, in practice tasks are distributed across nodes, but computational times are not equally distributed. At the end of a task, a new one can start. However, the running time of tasks can be different depending on resources and one node could compute two tasks while one has not finished its first one.



a. Mean CPU 1min window (% CPU). b. Network load (% CPU).

Fig. 4. Cluster performances using caching and parallelization value set to 16.

The gains of time are confirmed in Tab. 1, using caching and parallelize improves significantly the latency.

	Latency (s)
No caching, parallelize (3)	4 min 11 s
Caching, parallelize (3)	2 min 28 s
Caching, parallelize (16)	2 min 6 s

Tab. 1. Cluster efficiency.

1.2.3 Retrieve, analyse and discuss the output (Task 2d)

The speed test analysis in 1.1 is reproduced using the cloud. This result should show that the use of TFRecord is more efficient for large-scale machine learning. Splitting tasks between different nodes allows executing large-scale tests. This result could be explained by the fact that cloud data may be stored in distant physical locations. Theoretically, each access to a file takes 10ms for finding it and 0.2s per Mb to read it **[5]**. Using this reference, the throughput to read the 3670 images have been estimated in Tab. 2.

	Image Files	**TFRecord Files**
Dataset properties		
Number of files	3670	16
Maximum size of files	2Mb	56Mb
Times		
Disk seek (same datacenter)	37s	0.16s
Read from disk	115s	11s
Total		
Total time	152s	11s
Throughput estimation	24 Images/s	329 Images/s

Tab. 2. Throughput estimation.

These estimations are dependent on the disk type and the size of files and where the data is stored. Hence, the throughputs should be approximately 14 times higher for TFRecord files than for the Image files.
Fig. 5 highlights that the estimation of the ratio is close to the practice. The throughputs for TFRecord files are about 15 times higher than Image files. The difference may result from the value taken from the estimation and uncertainty caused by the noise when accessing the files.
For large-scale machine learning, a balance between the number of files and the size of files must be found to guarantee fast computation.
The observed behavior is similar to what I would expect from a single machine. This speed test depends on the number of files that have to be read, the capacity of fast seeking and reading from the disk. Cloud providers tie throughput to the capacity of disk resources as seeking and reading tasks are dependent on the disk resources: reading 1Mb from an SDD disk takes 1ms against 20ms from a standard disk **[5].**
Communications bottleneck in speed tests in parallel on the cloud needs to be considered for speed tests in the cloud. Indeed, the measured throughput should not be influenced by waiting times because the capacities of a node are reached. Moreover, the bucket reading speed should be the same across nodes. The default configuration is enough if the number of read requests per second is lower than 5000 **[6]**.

Fig. 5 demonstrates that in practice, it can be useful to choose a large enough batch size and batch number to allow fast reading.

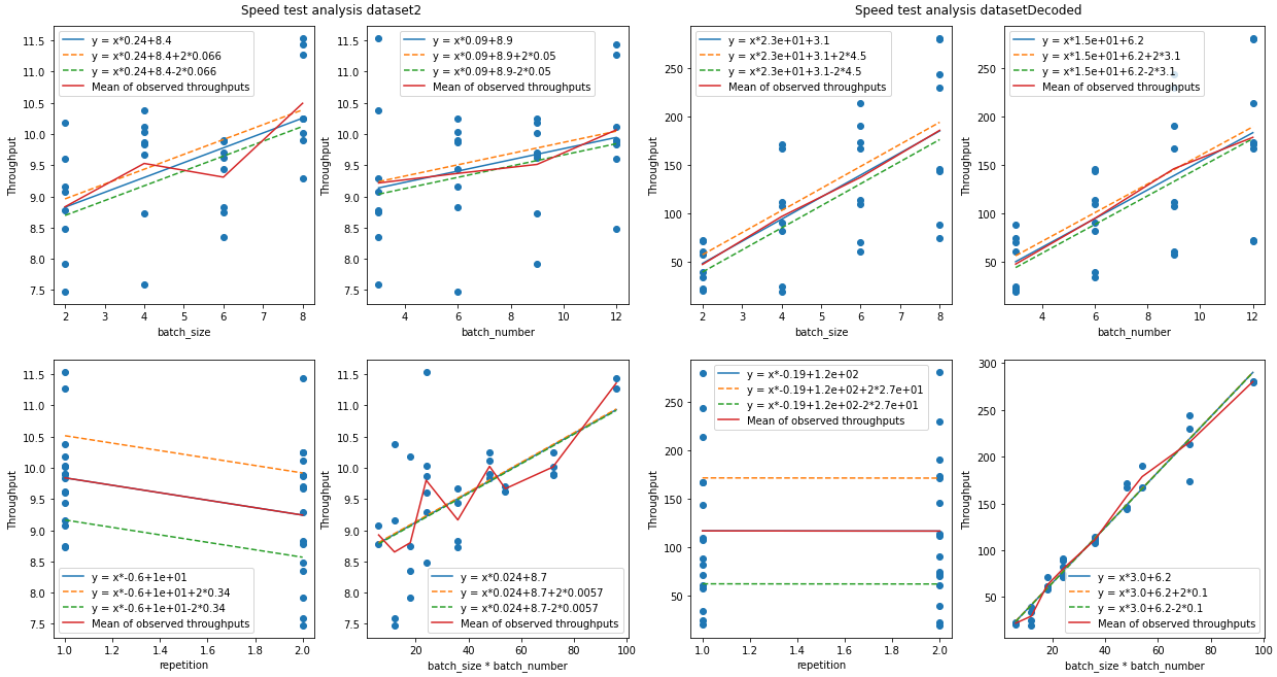


Fig. 5. Speed test analysis.

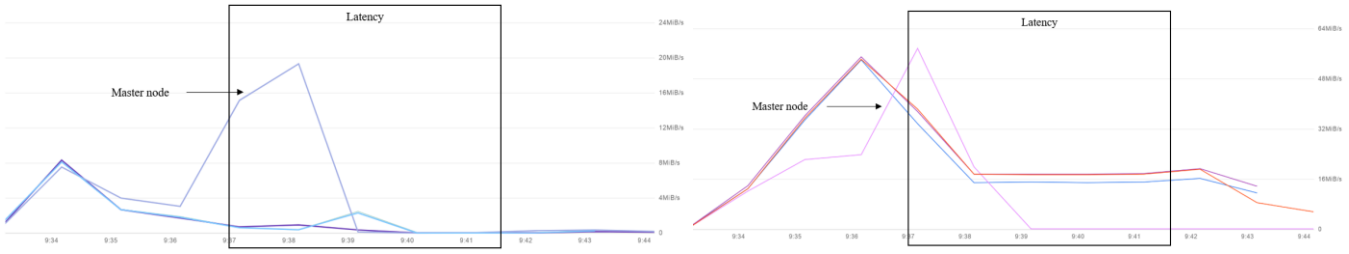
1.3 Write TFRecord files to the cloud with Spark (Task 3c)

1.3.1 Discuss the results in terms of disk I/O and network bandwidth allocation in the cloud (Task 3ci)

TFRecord files are written to the cloud using three different configurations for the clusters. The *4 machines* configuration stands for a cluster composed of a master node with an SDD disk limited to 100GB and three worker nodes with standard disk limited to 667GB. Each machine has one CPU. The *4 machines with double resources* configuration is similar to the *4 machines* configuration but each machine has two CPUs. The *1 machine with quadruple resources* configuration corresponds to a single node with an SDD disk limited to 100GB and 4 CPUs.
Tab. 3 underlines that the computation is faster on the cloud with Spark. The latency has been divided by almost 2 using 4 machines with single resource. Adding double resources divides the time by 2. The latency is almost inversely proportional to the number of vCPUs. The machine with quadruple resources is faster than the 4 machines with single resource.

	Latency (s)
Local machine	8 min 26s
4 machines	4 min 31 s
4 machines with double resources	1 min 53 s
1 machine with quadruple resources	2 min 45 s

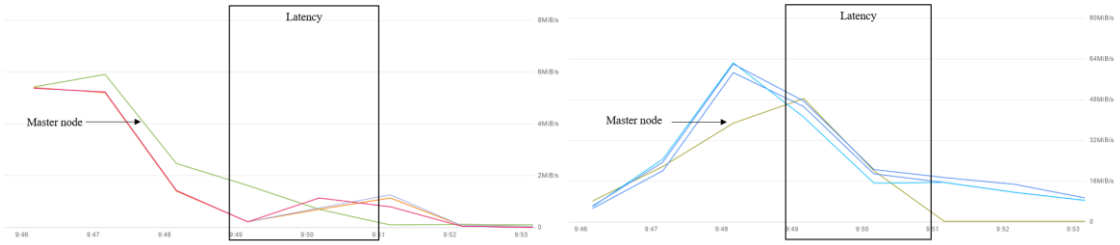
Tab. 3. Cluster efficiency.



a. Disk read bytes. b. Disk write bytes.

Fig. 6. Cluster performances on *4 machines*.

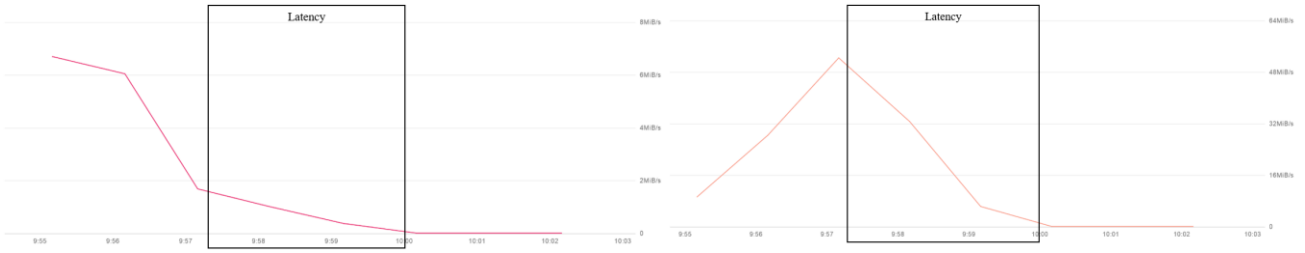
Fig. 6 highlights that none of the machines reaches the limit I/O speed proportional to the maximum size of disks. The master node is faster than the worker nodes. It can be explained by the high speed of reading from SDD disk compared to standard disk. It also puts under the spotlight that the bandwidth is adequate, and nodes could be added.



a. Disk read bytes. b. Disk write bytes.

Fig. 7. Cluster performances on *4 machines with double resources*.

Nodes are added increasing the number of CPUs per machine. The I/O speed is still below the limits. Worker nodes are almost as fast as the master node (Fig. 7). It explains the smaller latency time for this experiment than the one with the *4 machines configuration.*



a. Disk read bytes. b. Disk write bytes.

Fig. 8. Cluster performances on *1 machine with quadruple resources*.

The last experiment consists in testing a machine with fewer nodes but more powerful nodes. The *1 machine with quadruple resources* configuration does not reach the maximum memory allowed (Fig. 8). Its I/O speed is higher than the *4 machines* configuration. It can be explained by the adequate SDD memory available and the high I/O speed of SDD disks. However, its latency is smaller than the one of the *4 machines with double resources* configuration as the multiple nodes allow faster computation.

1.3.2 Explain the difference between this use of Spark and most standard applications like e.g. in our labs (Task 3cii)

The difference is that during labs, Spark was used to define a pipeline that worked on a single node. This use of Spark partition tasks between different workers to fasten computation.

2 Machine Learning in the Cloud (task 4c)

2.1.1 Expectation

Training should speed up adding a reasonable number of GPUs to keep a large enough training and validation dataset: each worker will use b/k samples where k is the number of GPUs and b is the size of the batch. Distributed learning should allow splitting learning between GPUs.

2.1.2 Results analysis

Parameters		Latency	
Machine, task	**Batch size**	**Epochs 50**	**Epochs 100**
standard_gpu	64	358s	
standard_gpu, distributed	64	355s	692s
standard_gpu, distributed	128	326s	666s
standard_gpu, distributed	352	321s	628s

Tab. 4. Model training on AI platform.

Tab. 4 highlights that distributed learning accelerates the learning. Increasing the batch size makes learning faster. The latency is linearly related to the number of epochs. Moreover, full GPU capacity (100%) is used in the 4 experiments (see Appendix). It highlights that adding nodes or using more powerful nodes could be used.

The number of K80 was limited to 1 and the request for increasing the quotas has not been accepted (see error example in Appendix). I expect that using more GPUs should have allowed not to reach the maximum of GPUs capacities and speed up training.

3 Theoretical discussion

3.1 Context (Task 5a)

Setting a good configuration of a cluster have been revealed challenging (tasks 2c, 3c and 4c). The CPU count, cluster size and disk type influence have been studied. Alipoufard and Yu showed that a near-optimal combination of these parameters can be found using a prediction **[1]**. They used a Bayesian Optimization framework to find the global optimum of the function of the cost, the CPU/RAM ratio, the CPU count, the cluster size and the disk type. The advantage of a such choice is that it does not assume any format of the function. It needs only a small number of samples to find a near-optimal solution and it works with uncertainty. Alipoufard and Yu made the choice to use a Gaussian process as prior. It does not reduce the scope of applications of their method, but it ensures that if the function is close to a Gaussian process, the search will be faster and require fewer points.

The second aspect of this project focused on the batch size influence. It has been shown that the higher the batch size is the higher the throughput is (tasks 1 and 2) and the smaller the latency is (task 4). However, a middle ground must be found between the amount of memory and the batch size. Smith et al. demonstrated that changing the batch size during the learning is equivalent to use an adaptative learning rate and allow optimizing and speeding up the learning **[2]**. They point out that the parallelism is better, and it avoids hyper-parameter tuning. The optimal batch size can be defined as proportional to the learning rate only when the batch size is much lower than the training size.

3.2 Strategise (Task 5b)

Batch learning consists in updating the model after each of the B batches of size b. Hence computations of gradients of a batch can be split into n nodes to fasten each update and need less memory on each node. If n nodes are used, the sub-batches would be of a size $\left\lfloor \frac{b}{3N} \right\rfloor$ to have about 3 jobs per node as recommended by spark documentation. Each node should have at least enough memory to store the size of a sub-batch.
Online learning updates are made after each data point. The amount of memory can be relatively small as each point are seen only once. A single node is enough as the update is made for each point.
Stream learning works on a fixed size stream. Each epoch, the first element of the stream is either dropped and a new one is added at the end or kept. Hence, the memory must be large enough to store the stream. As for batch learning, computations of gradients can be split into different nodes to fasten each update and require less memory on each node.

The number of nodes and CPUs/GPUs depends on the cost and time that the user can accept. Hence, choosing the right configuration is dependent on the application and requires high adaptivity as stated by Alipoufard and Yu **[1]**. Increasing the batch size during the learning is possible to fasten computation as prone by Smith et al. **[2]** but it requires the user to specify a maximum batch size to allow cluster configuration and have enough memory.

WORD COUNT: 1983 WORDS

The word count does not include titles and labels of figures.

REFERENCES

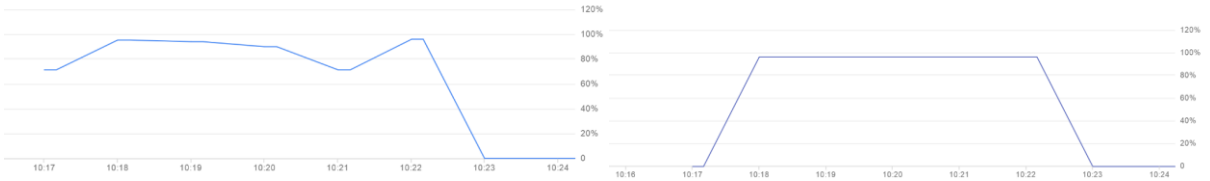
[1] O. Alipourfard and M. Yu, 'CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics', in *USENIX NSDI 17*, pp. 469–482, 2017.
[2] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, 'DON'T DECAY THE LEARNING RATE, INCREASE THE BATCH SIZE', in *ICLR*, 2018.
[3] "Configuration - Spark 2.4.5 Documentation", *Spark.apache.org*, 2020.
[4] "Tuning - Spark 2.4.5 Documentation", *Spark.apache.org*, 2020.
[5] "Latency numbers every programmer should know", *Gist*, 2020.
[6] "Request rate and access distribution guidelines | Cloud Storage", *Google Cloud*, 2020.
[7] "Flowers public | Google Cloud Platform", *Console.cloud.google.com*, 2020.

APPENDIX

1. Link to the notebook

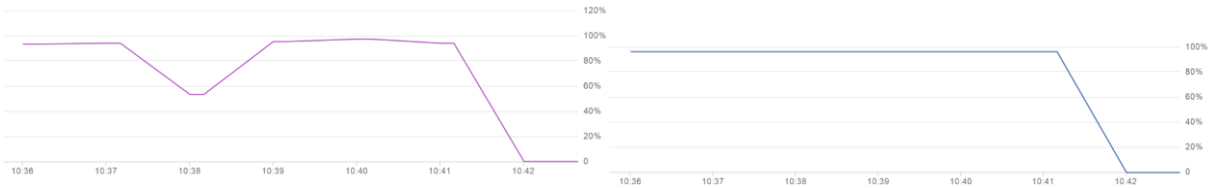
https://colab.research.google.com/drive/1DeC6CBjV2ezL_ufdTzExQHYMnC-IBFHn?usp=sharing

2. Figures for task 4c



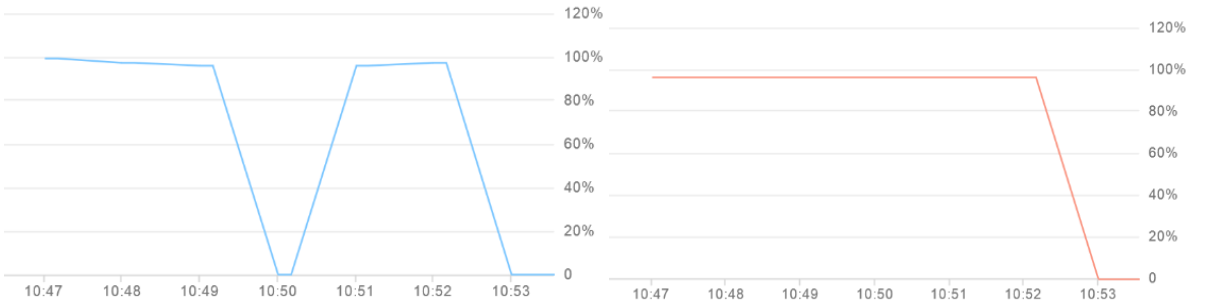
a. Mean GPU 1min window (% GPU). b. Mean Memory 1min window (% Memory).

Fig. 1. Standard_gpu.



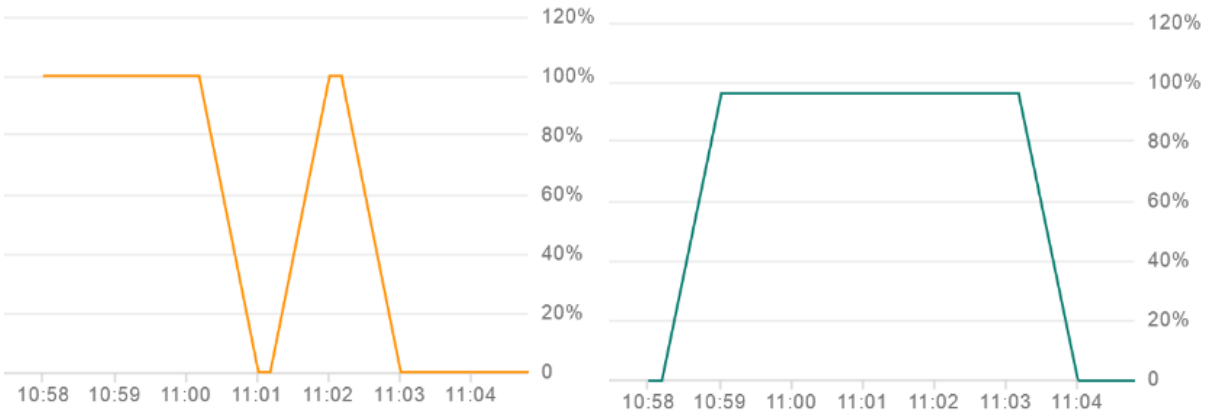
a. Mean GPU 1min window (% GPU). b. Mean Memory 1min window (% Memory).

Fig. 2. Distributed, standard_gpu, batch size 64.



a. Mean GPU 1min window (% GPU). b. Mean Memory 1min window (% Memory).

Fig. 3. Distributed, standard_gpu, batch size 128.



a. Mean GPU 1min window (% GPU). b. Mean Memory 1min window (% Memory).

Fig. 4. Distributed, standard_gpu, batch size 256.

a. Error observed in task 2

For *complex_model_m_gpu*:

```
"The request for 4 K80 accelerators exceeds the allowed maximum of 0 TPU_V2_POD, 0 TPU_V3_POD,
1 K80, 1 P100, 1 P4, 1 T4, 1 V100, 8 TPU_V2, 8 TPU_V3."
```

For *complex_model_l_gpu*:

```
"The request for 8 K80 accelerators exceeds the allowed maximum of 0 TPU_V2_POD, 0 TPU_V3_POD,
1 K80, 1 P100, 1 P4, 1 T4, 1 V100, 8 TPU_V2, 8 TPU_V3."
```