Big Data Report

Mussa Yousef

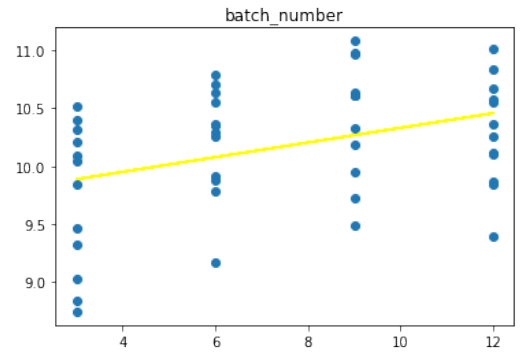
Mussa.yousef.1@city.ac.uk

Share Link for Google Colab document: https://colab.research.google.com/drive/121Gwb-OBAabqUcbXlkT-sFV_ohuDo2dh?usp=sharing

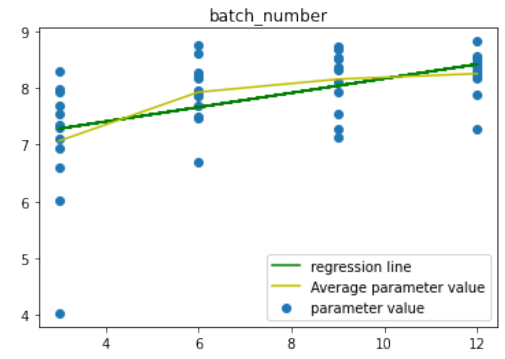
2c: Our initial speed test locally saw it take some time. It needed to be executed to be stored locally which would allow easy retrieval to be made when we built our regression line in 1b. Here we are carrying out the same experiment on the cloud. However, executing the speed test without cashing would mean the function would rerun when the argument is called. Therefore, we cached our RDD and speed test results for each dataset results to save this in the short-term memory feed. This enhanced speed to which the data is retrieved by reducing the need to access the underlying storage layer. Furthermore, we utilised parallelisation when calculating our reading speed by initialising our RDD. This allows us to further enhance efficiency by running speed tests in parallel on each dataset as they are both independent. We then This could be practically seen in Q2b where we set up our cluster and ran the script, we saw the script took 1hr 34 minutes to complete whereas when running the same cluster with caching and parallelisation of branches we see a significant decrease of time to 13 minutes.

2d:

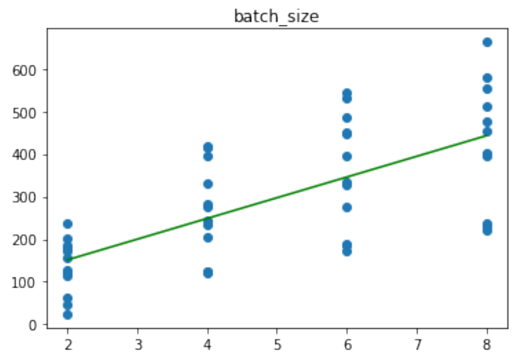
Dataset2 Local Test



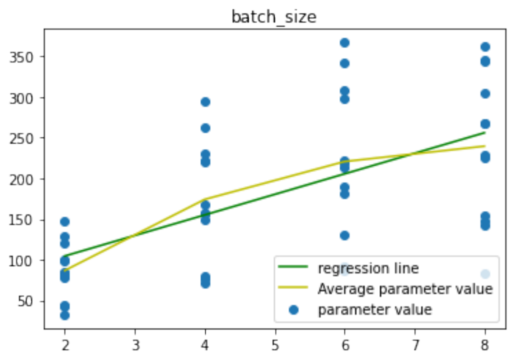
Dataset2 Cloud Test



DatasetDecoded Local Test



DatasetDecoded Cloud Test



First and foremost, comparing our Local and cloud tests we immediately find our model has predicted faster results would be obtained when running on the cloud. However, when looking at the retrieval time taken for the results to be calculated. Our Cloud took approximately 7799ms to connect to spark servers which is extremely high latency, considering external factors such as network, bandwidth, and essentially the cloud service provider this is still extremely high. As we are using a SaaS, this is inevitably a potential due to multi-tenancy and limited client bandwidth. The behaviour here was different from a single machine, we ran the tests in parallel to be hence cost is limited due to processors don't typically need to wait for the previous to be executed. Systems that are optimised for throughput usually have higher latency for small tasks. Therefore, although efficient in big memory tasks. When carrying out small tasks it would actually be more efficient to utilise disk space to minimise latency. Hence Cloud providers would tie throughput to the capacity of the disk to generally manage efficiency. A bottleneck we have seen in this experiment is that the logistic regression doesn't scale well by default, this is because they need to read and share data globally between nodes. We didn't find this to be a significant issue for us as we localised our logistic regression using batches. Furthermore, using loops hindered the time taken in our experiment, particularly when experimenting on the cloud this hindered our efficiency, therefore in the future would attempt to reduce loops.

3ci) Here we are running our experiments in parallel in the cloud while distributing the workload. The majority of our labs worked locally, however when we utilised the cloud, we usually ran one job on one machine compared here where we have partitioned essential several jobs to be running at the same time on the cloud using a number of workers.

3cii) We find our first experiment although 75% Slower in regard to latency we find that both times taken to initialise and connect to the server slightly more efficient. The table below shows our results.

CPU type	**Total latency**	**Time taken to Initialise**	**Server connect time**
2 vCPUs +4 workers	7 Minutes	5652 ms	5747 ms
8 vCPUs	4 Minutes	5997 ms	6089 ms

Table 1: CPU Configurations

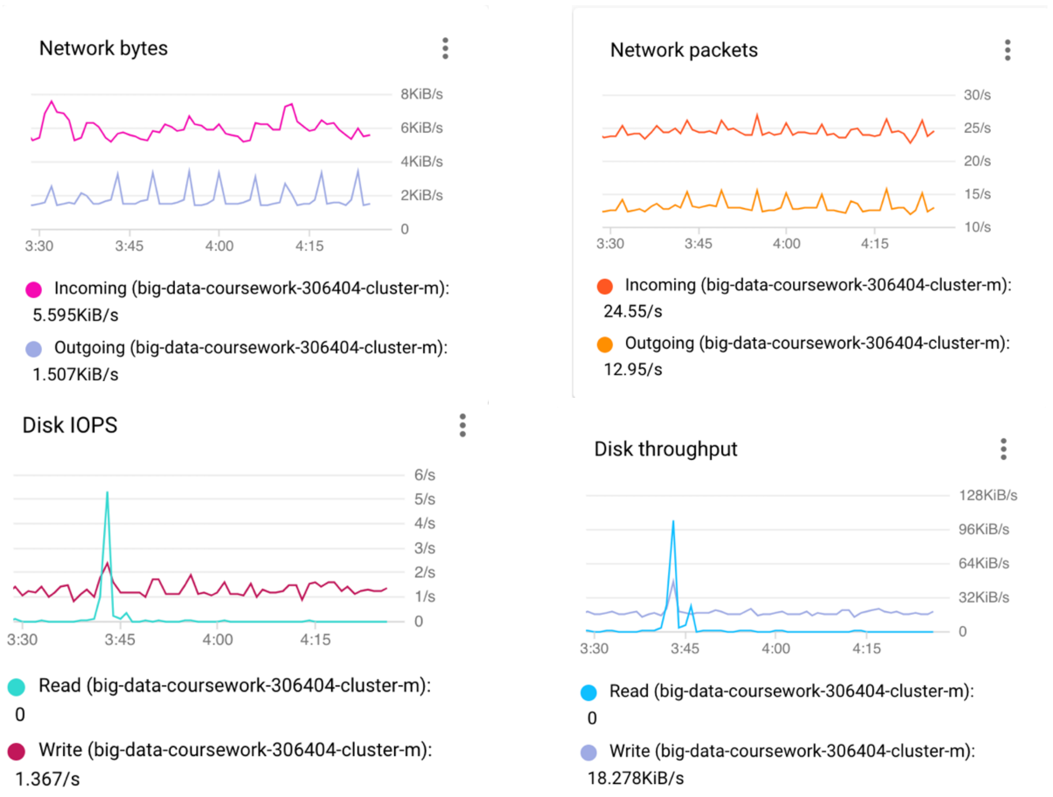


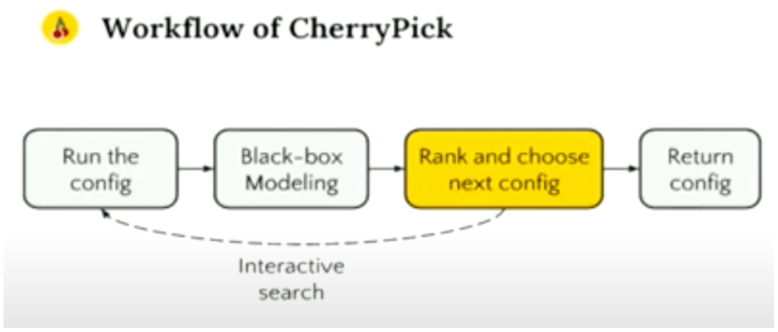
Figure 2: VM graphs which shows the CPU and Network Load over time

5a) Throughout this experiment, we interchange working locally to working in the cloud. Our first task was solely local however we were able to transition to the cloud by setting up clusters. Our third and fourth tasks were purely in the cloud, Writing TFRecord and Machine learning respectively. Hence whenever setting up a cloud interface, the user must choose several different configurations. Using Google Cloud, we understand the following restrictions are in place:

- max 100GB of *SSD persistent disk*

- max 2000GB of *standard persisten disk*
- max 8 *vCPUs*

Therefore, when setting up our cloud we trailed two configurations that worked reasonably. This is called searching or coordinate descent, where we use different combinations of RAM, CPU, disk. etc, to minimise cost while bettering performance. However, although a better solution may be found when searching, it's easily possible we may get stuck in a local minimum, suggesting inaccuracy to an optimal configuration. Modelling and Exhaustive are also exiting solutions when choosing the optimal configurations, however, there is always a major trade-off like the configurations having no adaptivity or high overheads respectively. Alipourfard et al (2017), suggest an alternative method to finding the most efficient configuration. The workflow suggested is called Cherry-pick



Workflow:

1) Run config
   - We first run the job on any user interface with the configurations inputted
2) Black-box modelling
   - Based on results It updates and captures the relationship between the cloud config their cost.
3) Rank and choose next config
   - Based on the black box model we rank the different combinations of config; we choose a config which minimizes the cost while satisfying the performance constraints specified by the user
4) Return Config
   - We do the 3 steps above iteratively with a feedback loop until we are satisfied then output that config as the result

Cherry-pick allows us to not only maximise efficiency but allows us to analyse the potential confidence in other configurations. Using this method here would have potentially been more efficient. But our biggest gain would be the effect cherry-picking has on noise. When going through this experiment on occasions simple cloud tasks took considerably longer than when previously carried out. We realized this was due to noise, noise is common in the cloud. However, this workflow considers the potential noise hence by using adaptive techniques to multiply a cost function in our black box.

5b) Batch processing like our cloud speed test allowed us to analyse essentially static data therefore analysis obtained is historic. This allows us to tailor our CPU to the size of the data hence speed would be solely reliant on factors we can manipulate and change. This allows Cherry picking to be carried out particularly easily as we can adapt our ranking using Bayesian optimisation with knowing the full extent of our capacity. Furthermore, our stopping function would be able to be calculated as we can assess the efficiency of our configuration hence our coordinate decent considering RAM, CPU, DISK, and Cluster size would have attained a high accuracy in reaching a global minimum.

On the other hand, online processing influences our configuration in real-time. To an extent, the logic here is relatable to unsupervised machine learning, where our real-time data is first classified to a certain time then clustered. For example, data on online sales; we are retrieving data at a rate of 4 sales per minute, and we have set our configuration to retrieve and analyse our data every 5 minutes. All of a sudden, our web traffic increases, and we now are receiving 50 sales per minute, due to the current configurations set to tailor our 4 sales per minute, immediately we may experience some latency due to our configuration of RAM and CPU.

Considering the previous novel approaches we could either change our batch initialisation rate to potentially capture data every 10 seconds, increasing the batch size, therefore prioritising CPU for the job. Cherry-picking would be more apprehensive as the iterative search for configuration would consider the implications of increased retrieval of data. Therefore, the configuration would be able to be tailored. However, this may cause a potential waste of resources by increasing overhead. Therefore, we can change the rate at which we collect data to enable micro-batch processing this would involve a combination of both, an increase of batch rate hence batch size which therefore would assist the cherry-pick workflow to be initiated at a more frequent rate. This would allow configuration to be tailored to the time frame to which an increase of data retrieval, therefore, minimising cost.

Number of words
1377