Big Data Coursework Report

Name: Thanawat Thanaponpaiboon
Email: Guy.Thanaponpaiboon@city.ac.uk

1d) Optimisation, experiments, and discussion (17%)

i) Improve parallelisation

```
firstRDD = sc.parallelize(filename, 16) ### TASK 1d ### (default setting = 2)
```

In order to improve parallelisation, we adjust the **second parameter** from the default (2) to (16) according to the example from section 1.2. This means we increase the amount of data into 16 partition and processes each partition in parallel across the nodes in the google cloud cluster.

- **Two nodes (parallelize = 2)**

Figure 1: Single machine maximal SSD size (100) + 8 vCPUs. (2 partition)

- **Sixteen nodes (parallelize = 16)**

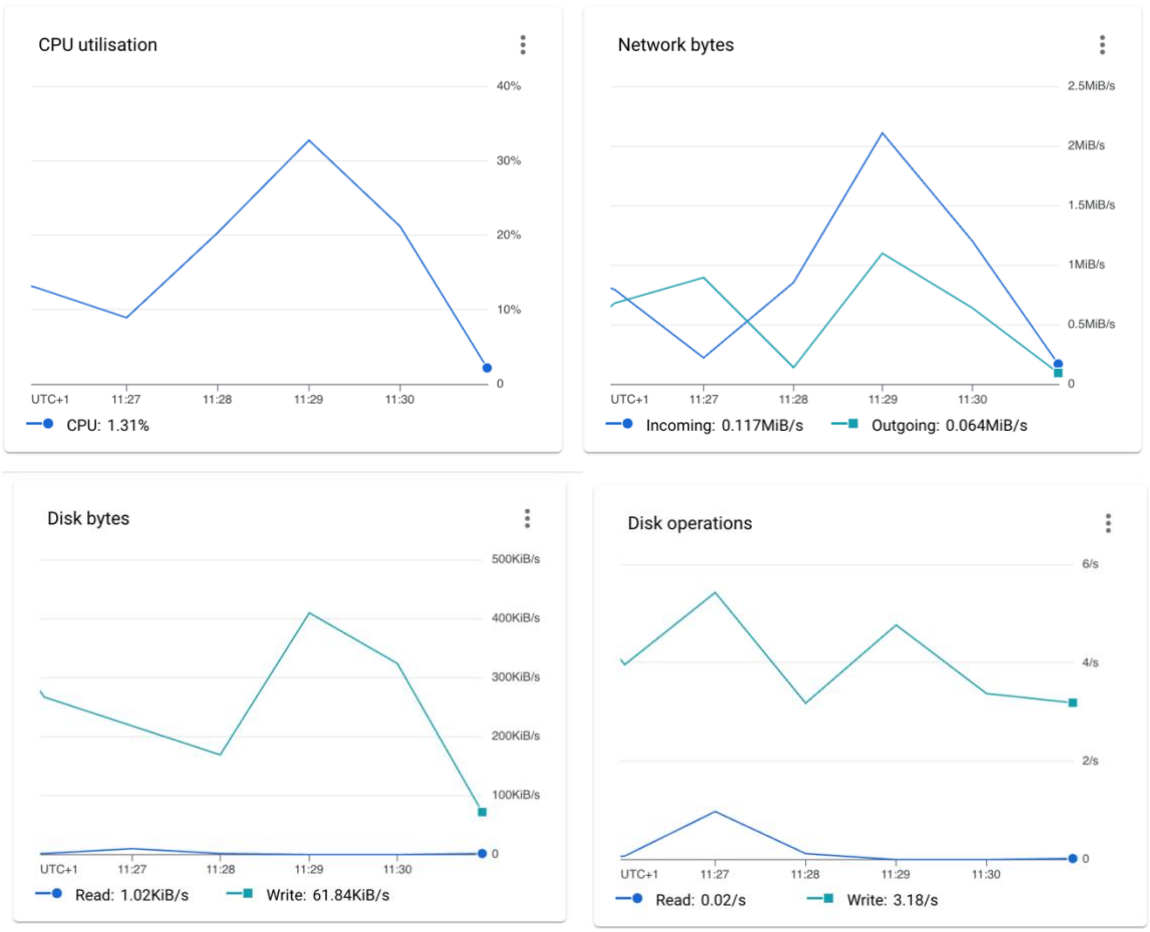


Figure 2: Single machine maximal SSD size (100) + 8 vCPUs. (16 partition)

In general, when the second parameter was added from the default (2) to (16) the CPU utilization went up from 19.63% to 32.71% at peak, increasing the number of multithreading may result in high CPU usages or increased CPU cycles. Although this costs some CPU time upfront, but network and disk bandwidth are saved. We can see from both figures, the lower number of disk bytes when we increase the number of partitions to 16 indicates that the disk is being used slightly compared to executing only 2 nodes.

Start time	Elapsed time	Labels
26 Apr 2023, 11:27:17	2 min 20 sec	nodes : 16
26 Apr 2023, 11:21:28	4 min 59 sec	nodes : 2

In terms of the **difference in the processing time,** increase the number of partitions from 2 to 16 increase the speed of computational time from around 4 minutes 59 seconds to 2 minute and 20 seconds. Hence, increasing the number of partitions improves processing speed by 2.1 times.

ii) Experiment with cluster configurations.

The largest possible cluster that we can create consists of 1 master and 7 worker nodes. Each of them with 1 (virtual) CPU. The master node gets a full 100GB capacity of SSD persistent disk, and the 7 worker nodes get equal shares of the standard disk capacity to maximize throughput which means 2000GB of standard persistent disk need to be divided by 7, 285GB will be the number of standard disk capacity per node approximately.

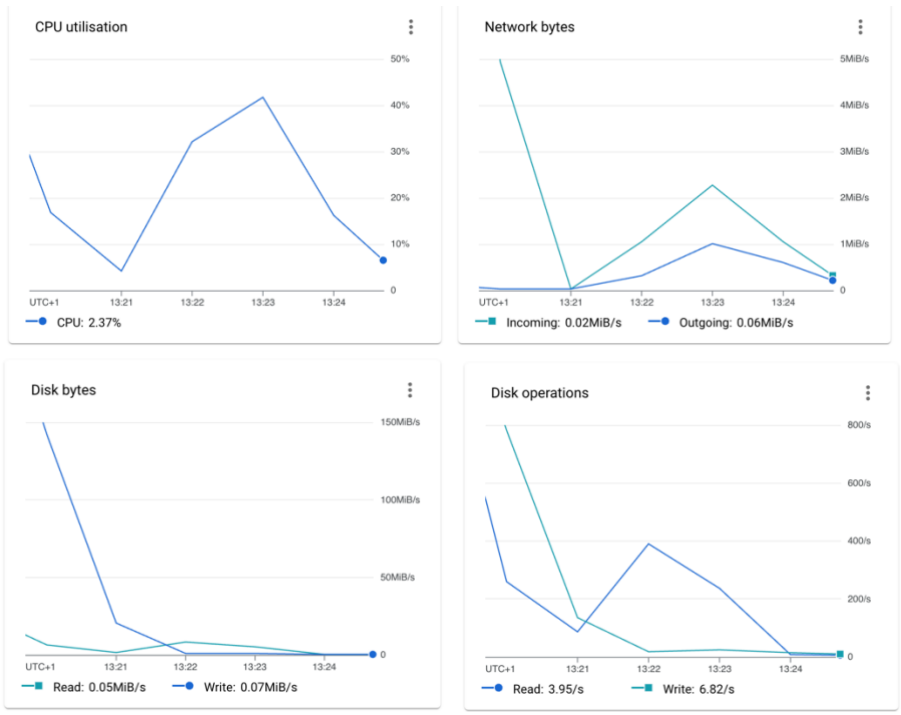


Figure 3: Largest possible cluster 8 VMs (1 master +7 worker nodes) (16 partition)

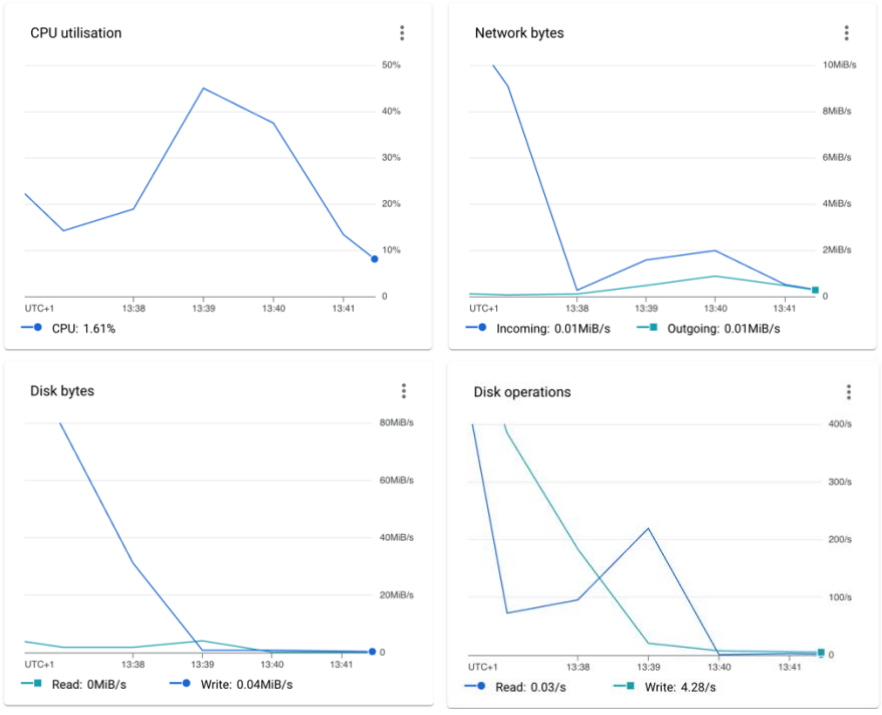


Figure 4: A machine with eightfold resources + 3 machines (2 vCPUs, memory, disk) (16 partition)

26 Apr 2023, 13:37:25	3 min 3 sec	machine : 4
26 Apr 2023, 13:20:57	2 min 29 sec	vm : 8

Creating the largest possible cluster can result in high CPU utilization. This can be the same scenario as when we created 4 machines with double virtual CPUs. Both clusters require a high CPU usage which lies around 41%-45% at the highest state. In the same way, a large cluster typically requires more disk I/O and network bandwidth than a smaller cluster, as you can see from the figure that after we create increase the worker machine network and disk also goes up to.

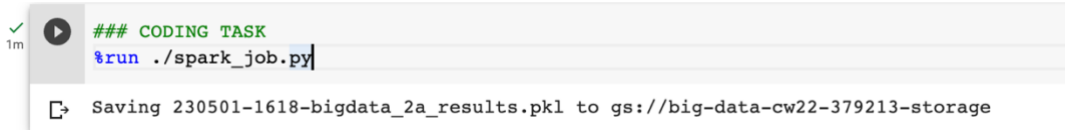
Large clusters can be beneficial for distributed computing such as Spark when we run across multiple nodes in parallel simultaneously. By distributing the workload across multiple nodes. This approach can reduce the computational time during the process and the risk of any hardware or software failures. However, creating a cluster that does not fit into the task can lead to poor performance, slow response times, and also end up wasting resources such as storage, and networking bandwidth. As can be seen from the experiment increasing the worker node in the cluster seem not to have a profound effect in terms of increasing the speed of processing time compared to the single master node. And reducing the number of worker nodes to 3 along with using a couple of Virtual CPUs in the working node produces a similar result as the experimentation with 8 virtual machines.

iii) Explain the difference between this use of Spark and most standard applications

In the lab session, data is usually stored in a single machine like pandas dataframe array, excel database or CSV file etc. It always executes locally in a platform such as Jupiter. In contrast, Spark use the technique called **parallelize** for distribute the data over partitions, which mean that the data is distributed across multiple machines or nodes in a cluster can be use simultaneously. The result is drastically increasing the speed of processing time when perform on large datasets. However, spark distribution needs to rely on extra settings and more external environments like google cloud which might affect the processing time as we mentioned. Poor settings can result in poor performance and wasteful resources. While most of the time standard applications proceed locally without any reinforcement.

2b) Testing the code and collecting results (4%)

I) First, test locally with %run: it takes a minute to run through all step locally



II) Cloud: **CPU and network load** over time after applying the code on GCP

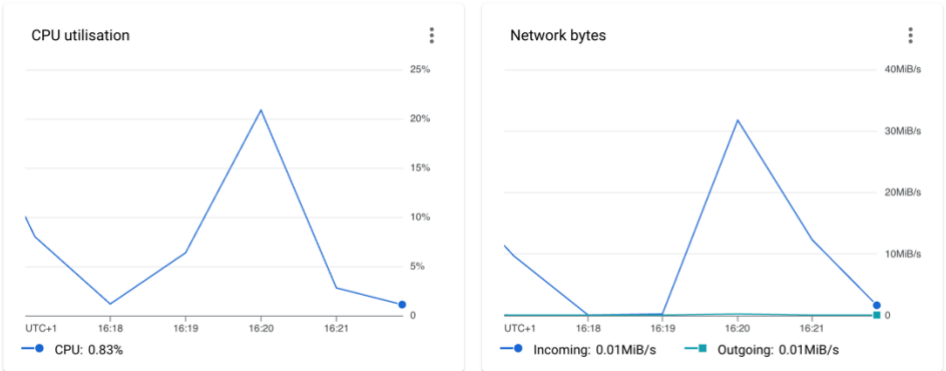


Figure 5: A single machine using the maximal SSD size (100) and 8 vCPUs (Speed test for Task 2b)

2c) Improve efficiency (6%)

We implemented a straightforward version into Clould and found **inefficiency in your code.** Running on Cloud takes quite a long time similar to running locally. Because we cannot fully utilize the power of the spark.

```
ds_comb = parameters_rdd.flatMap(new_time_configs_ds)
tf_comb = parameters_rdd.flatMap(new_time_configs_tf)

### TASK 2c ###
ds_comb.cache()
tf_comb.cache()

# create rdd_results for dataset : batch_size & images_per_second
rdd_dataset_bs_speed = ds_comb.map(lambda x: (x[0],x[5]))
dataset_bs_speed_result = rdd_dataset_bs_speed.collect()

# create rdd_results for dataset : batch_number & images_per_second
rdd_dataset_bn_speed = ds_comb.map(lambda x: (x[1],x[5]))
dataset_bn_speed_result = rdd_dataset_bn_speed.collect()

# create rdd_results for dataset : repetition & images_per_second
rdd_dataset_rep_speed = ds_comb.map(lambda x: (x[2],x[5]))
dataset_rep_speed_result = rdd_dataset_rep_speed.collect()

# create rdd_results for tf_dataset : batch_size & images_per_second
rdd_tfdataset_bs_speed = tf_comb.map(lambda x: (x[0],x[5]))
```

We apply **RDD.cache()** function to our RDD to improve efficiency. The first noticeable thing is that the computational time for compiling this code on cloud is faster. Due to this function catch the RDD in the memory or disk instead of remaking it every time you call that RDD. In task 2a, we often reuse the same RDD (ds_comb, tf_comb) to generate the new result RDD for each combination of parameters compared to speed. The result after applying cache function significantly increases the performance because caching the RDD reduce the amount of time spent on recomputing the RDD from its source data.

Start time	Elapsed time	Labels
1 May 2023, 17:11:51	47 sec	cache : true
1 May 2023, 16:18:37	1 min 30 sec	cache : non

However, cache function consumes a significant amount of memory or disk space. Hence, you should make sure when applying cache RDDs that the task will reuse the same RDD more than once.

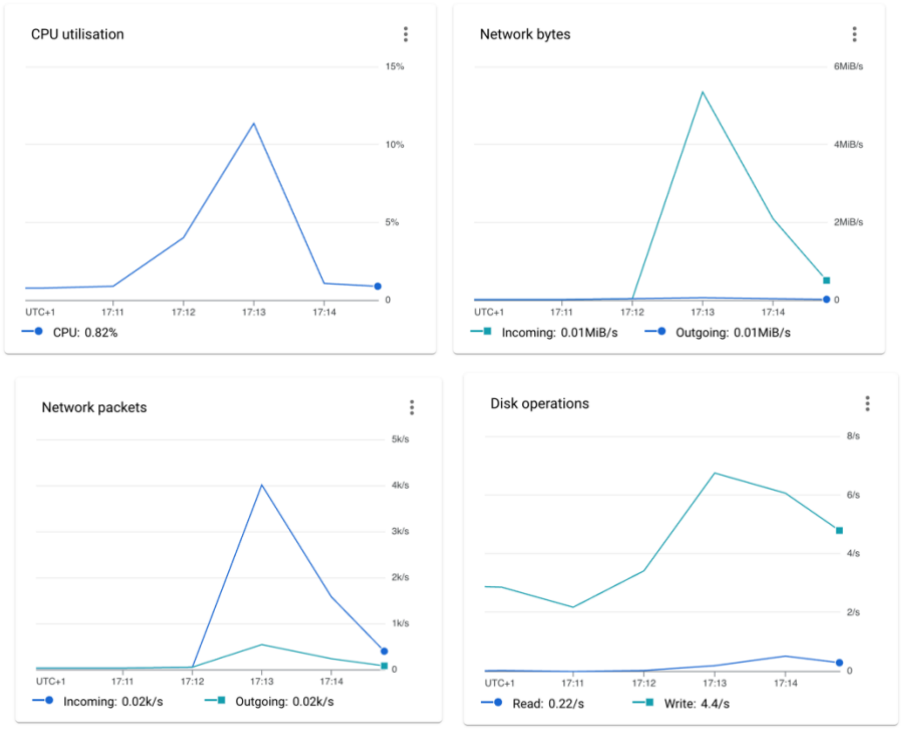


Figure 6: A single machine using the maximal SSD size (100) and 8 vCPUs after apply cache() (Speed test for Task 2c)

2d) Retrieve, analyse and discuss the output (12%)

After we retrieve the result of each combination from pickle file in GCP bucket, we perform linear regression to find a global trend and observe the correlation between each parameter compared with throughput (Image per second) in the time data points. Below are visualizations of the relationship between throughput across different parameters in each specific case which are image file and TFRecord file.

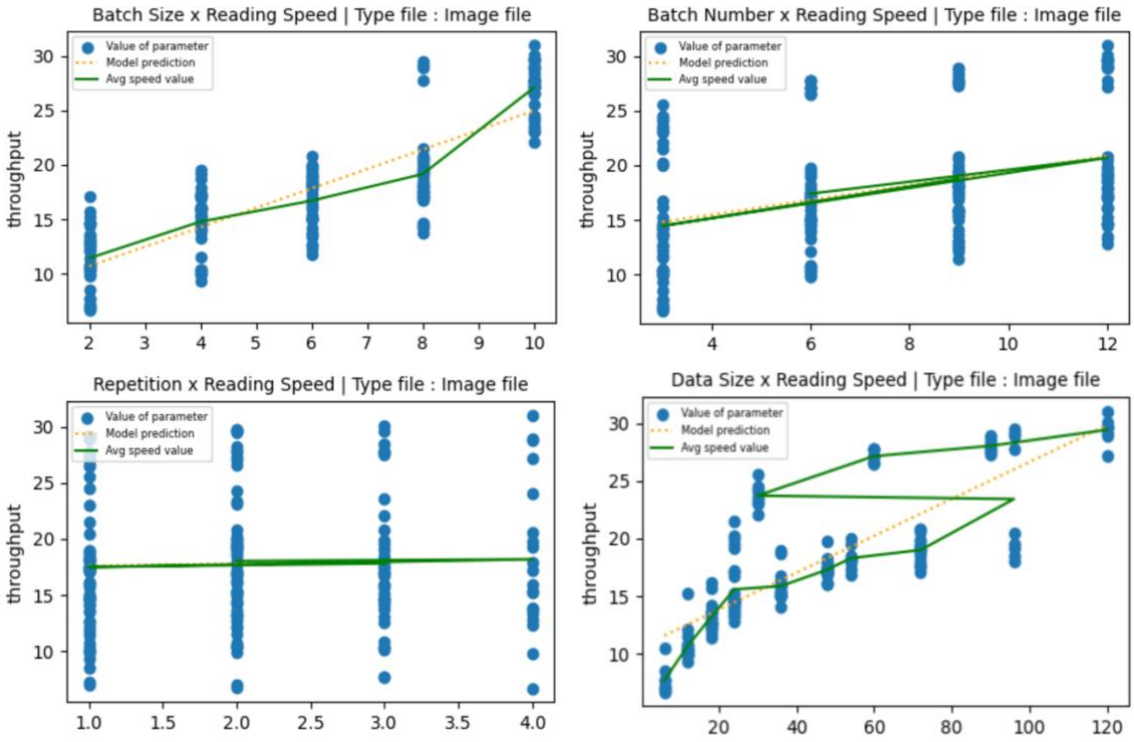


Figure 7: Regression plots of each parameter for image file case

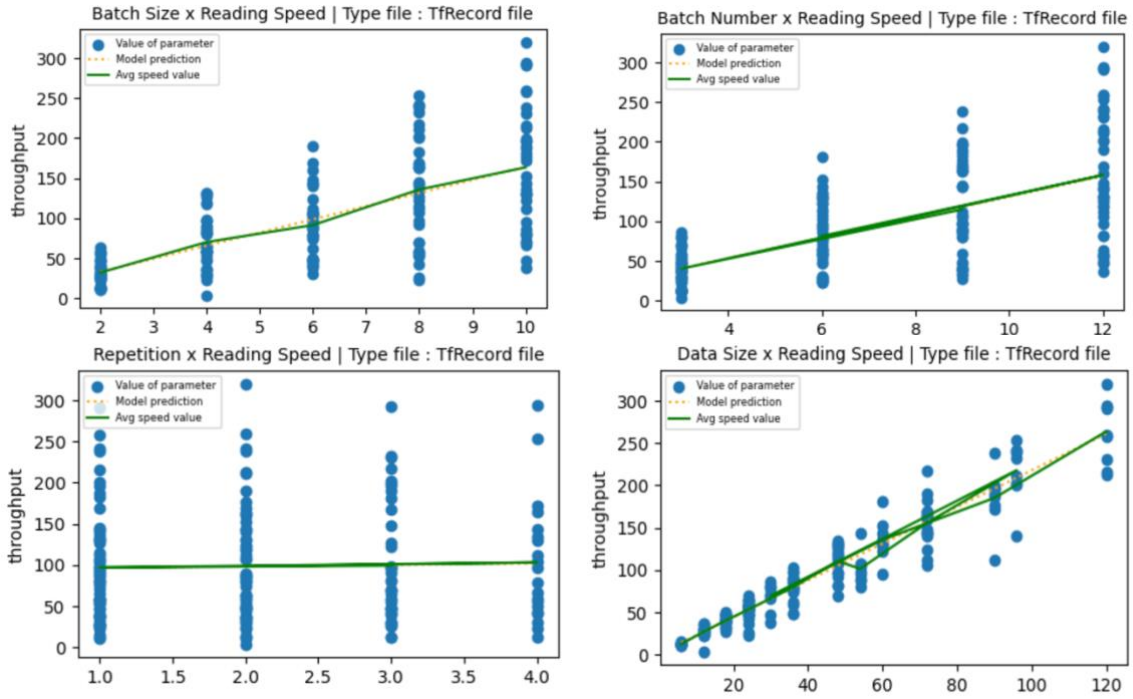


Figure 8: Regression plots of each parameter for TFRecord file case

Scale for two cases obviously is seen that TFRecord file has a larger amount of throughput around ten times than image file with the same parameter indicating that at the same given time, TFRecord files are more effective and faster at reading flower images.

Figures 7, 8 and the table below indicate that there is a strong correlation of throughput with batch size and batch number in both TFRecord file and image file case. We can see from the high value of coefficient and low value of p-value shows that the results from the model are replicable. Similar results were found in data size, which is the product of batch size and number, which revealed a linear positive trend. The result suggests that increasing batch parameters such as batch number and batch size lead to a significantly larger throughput per second as well. There is no relationship between repetition and throughput in both cases. It can be seen that the lines of the average and prediction from the model are flat on the horizontal line no matter how much the number of repetitions increases. In addition, high p-value demonstrate the result cannot conclude that the explanatory repetition affects the throughput. Therefore, repetition does not affect the amount of throughput.

Case Type	Parameter	Coefficient	Intercept	P-value
Image file	Batch Size	1.784	7.106	7.9e-49
	Batch Number	0.670	12.787	2.4e-7
	Repetition	0.188	17.417	0.679
	Data Size	0.160	10.607	1.3e-44
TFRecord file	Batch Size	16.406	0.124	1.7e-24
	Batch Number	12.902	1.799	1.2e-20
	Repetition	1.750	94.869	0.739
	Data Size	2.168	0.989	7.6e-98

Table 1: Result of the parameters

It is much more beneficial to run testing in local single machine before running on cloud to avoid technical errors. However, for **large-scale machine learning**, running on cloud with parallelization might be more effective in both performance and training time. Providing a better environment to run the model simultaneously when dealing with larger datasets. As a result, it will lead to higher accuracy of the linear model as well as reduce the amount of time for training model.

However, there are some circumstances that we might encounter with bottleneck problem with limits of the bucket reading speed even though we parallelize the data and model and run simultaneously we have to consider the amount of retrieved input that can feed into the model. Setting a higher number of parallels or maximum cluster without considering the retrieved data can result in wasteful of resources and cost.

Task 3: Discussion in context. (24%)

3a) Contextualise

Throughout the experiment on cloud with various setting of clusters from this coursework, we can observe the diversity of the results in both good and bad performance in terms of processing time, CPU, network, and disk bandwidth according to the parameter and type that we set to our cloud cluster machine. Setting the right variables leads to significant improvement in processing time as well as performance. In contrast, poor setting may result in a slow process or wasteful resources and costs or even worse than running locally.

Since there are dozens of possible VM instance types and even more cluster sizes to choose from, choosing the right cloud configuration may take time and resources. As we can see from the previous tasks setting cluster and execute on google cloud take an amount of time especially with the large cluster can take up to 5-10 minutes just only to create single cluster.

Cherry-pick can be applied to mitigate this problem and find optimal configurations in few times instead of running through all possible sets of clusters manually. Cherry-pick take advantage of Bayesian Optimization method to build accurate models to distinguish the near-best cloud configuration in a few runs. As a result, it can save time and searching costs for optimized cloud configuration and also guarantee that we will end up with the near-best likely optimal configuration.

3b) Strategise

The flower images dataset in this coursework uses the benefit of **batch** processing to break the workload into a smaller chuck of data and process it simultaneously. In spark programming we use method called parallelization to create Resilient Distributed Datasets (RDD) to perform parallel processing across a cluster or machines. This leverages the use of resources such as CPU, memory, and disk space significantly improving overall performance and processing time. Batch processing usually perform on large scale dataset which required a lot of processing power and storage to be processed efficiently. This method typically executes on existing datasets without the need for real-time processing such as in the lab datasets are normally maintained in the storage and ready to use which we focus on quality and performance in terms of accuracy and time processing.

On the other hand, **stream** processing might involve the real-time dataset a great example would it be continuous flow data that come from cameras or sensors, or sales transactions. This kind of data plays an important part in business nowadays providing tremendous insights and being able to process in real time. Giving the instant result or immediate feedback to the user. However, similarly to batch, stream processing also relies on scalable real-time processing services such as GCP to distribute the workload across multiple machines to increase the processing time and performance which is easier to control and limited the scale of resources. The result may benefit the end-user experience by giving instant responses and improving the computational time.

These two scenarios required different resources and configurations in order to reach the highest performance. As we discussed in section 3a to maximize the full potential of both resources and configurations. Cherry-Pick is a decent method to be applied for searching the best cloud configurations by improving the estimation for configurations that are closer to the optimal. Using Bayesian optimization technique, searching across different cloud configurations allows this algorithm to work well even with limited given information. Consequently, performance, reliability, and cost for these two scenarios can be improved by applying Cherry-Pick. We can ensure that we will end up with at least near-best optimize configuration which lead to an increase in the performance of the task and minimize the spending cost.

Word Count: 1898 words (not including Question, figures, and table)
Colab Link: https://colab.research.google.com/drive/13JvmvSEmvPOmuRUwCd0gJsJFQvdLz8V9?usp=sharing