Big Data – Data Processing and Machine Learning in the Cloud

Deenadhayalan Ravi
deenadhayalan.ravi@city.ac.uk

1. Introduction

This coursework focuses on exploring parallelisation and scalability in the cloud using Spark and TensorFlow/Keras. The aim is to parallelise pre-processing tasks and measure their performance in the cloud using Google Cloud Dataproc, and then perform evaluation and analysis on the results. All the tasks were performed on a set of image files from the public "Flowers" dataset. The coursework has three sections,

- The section 1: Involves data pre-processing using TensorFlow, which will be adapted for Spark in Task 1 of Section 1. Task 1 involves parallelising the pre-processing using Spark on Google Cloud Dataproc.
- In Section 2, the focus is on measuring the speed of data reading in the cloud, which will be parallelised using Spark in Task 2.
- The final section is a theoretical discussion, based on a paper.

2. Section 1: Data pre-processing

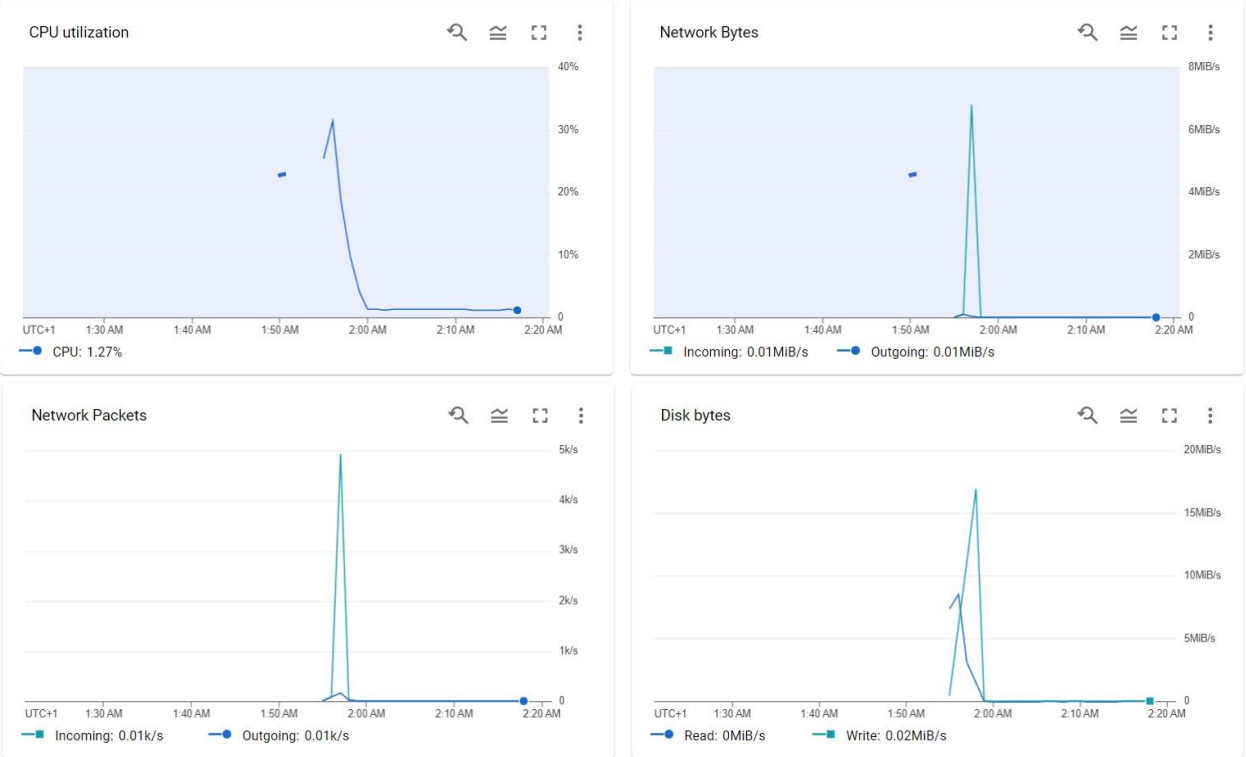
Task 1:

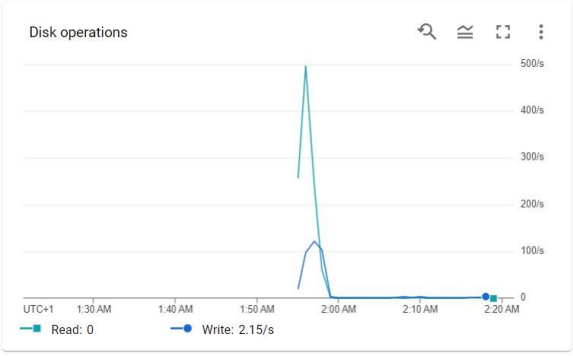
1.a) Created a script for the pre-processing using Spark mechanisms for distributing the workload over multiple machines.

1.b) Test the above by reading the TFRecord Dataset. And store the 1.a script in a python file named spark_write_tfrec.

1.c) Set up a cluster and run the script:

1.c.i) Create a single cluster in the virtual machine (single machine using the maximal SSD size (100) and 8 vCPUs) and run the script spark_write_tfrec. Results are as under,

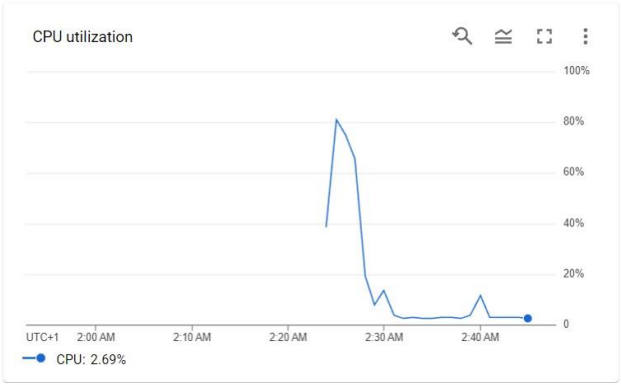


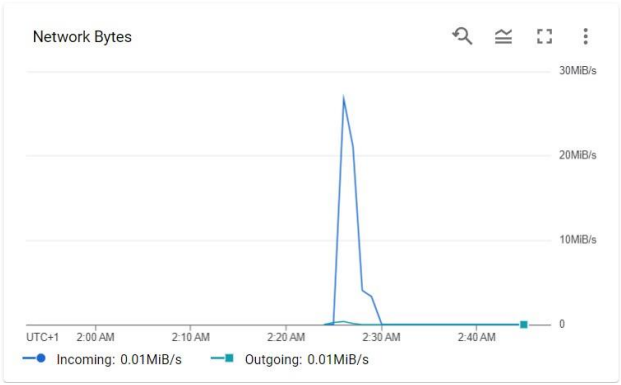


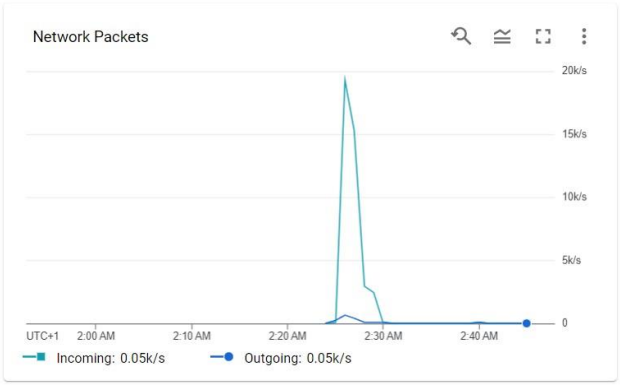
EQUIVALENT REST

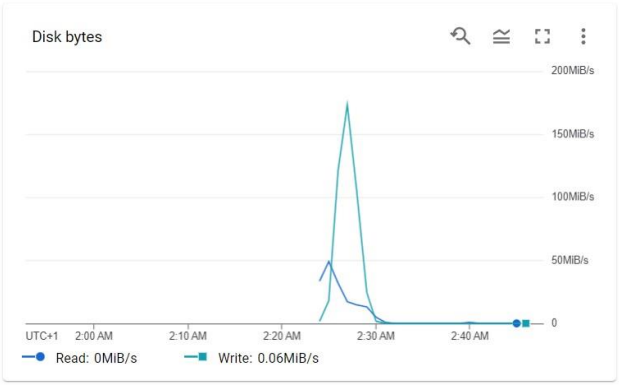
Table 1: 2 partition script on single machine cluster - 1 Master (8CPUs)					
Metrics	**Network Bytes**	**Network Packets**	**Disk Bytes**	**Disk Operations**	**CPU Utilization**
Max incoming	6.82 MiB/s	4.93 k/s	NA		31.46%
Max Outgoing	0.12 MiB/s	0.17 k/s			
Max Read	NA		8.49 MiB/s	497.33 /s	
Max Write			16.92 MiB/s	122.38 /s	

1.c.ii) Create largest possible cluster and run the script. The maximal cluster is with 1 master and 7 worker nodes, each of them with 1 (virtual) CPU. The master should get the full *SSD* capacity and the 7 worker nodes should get equal shares of the *standard* disk capacity to maximise throughput, and ran the 2-partition 'spark_write_tfrec' file and the results are as under,









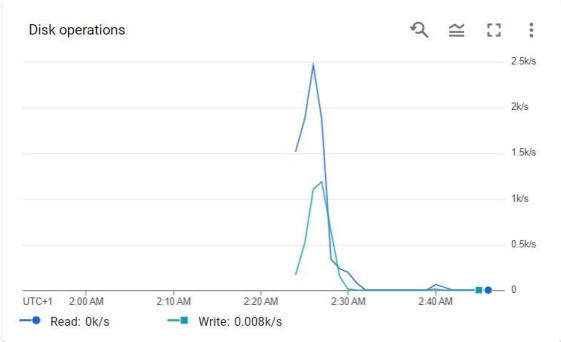
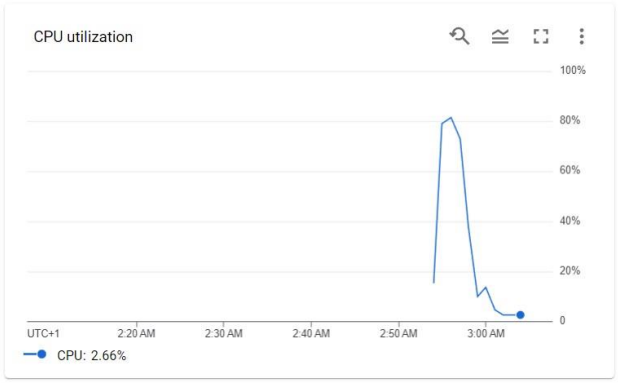
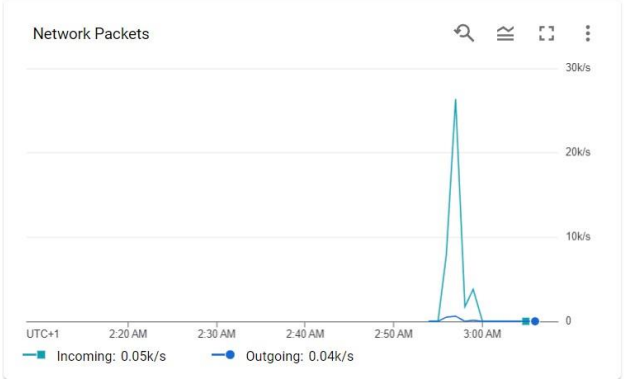


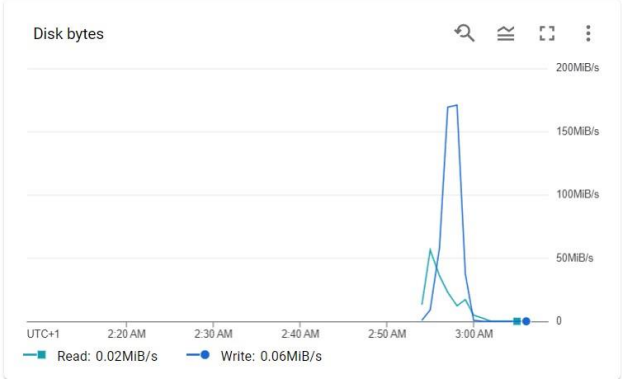
Table 2: 2 partition script on maximal cluster - 1 Master (1CPU) + 7Worker node (2CPUs/each)					
Metrics	**Network Bytes**	**Network Packets**	**Disk Bytes**	**Disk Operations**	**CPU Utilization**
Max incoming	26.72 MiB/s	19.29 k/s	NA		81.05%
Max Outgoing	0.43 MiB/s	0.7 k/s			
Max Read	NA		49.16 MiB/s	2.46 k/s	
Max Write			173.49 MiB/s	1.187 k/s	

Inference: Comparing the above two tables, we can observe that running the 2 partition script on a single machine cluster with 1 master (8CPUs) resulted in significantly lower utilization of CPU, network, and disk resources as compared to running the same script on a maximal cluster with 1 master (1CPU) and 7 worker nodes (2CPUs/each). In particular, the CPU utilization was much lower in the single machine cluster (31.46%) as compared to the maximal cluster (81.05%), indicating that the single machine was not able to take full advantage of the available CPU resources. Similarly, the maximal cluster was able to achieve much higher network and disk throughput as compared to the single machine cluster, indicating that distributed processing across multiple machines can result in significant performance improvements.

1.d) Optimisation, experiments, and discussion:
1.d.i) Improve parallelization: The codes were changed to parallelize the pre-processing task (i.e.16 partitions) and saved it as ‘spark_write_tfrec_task_1d.py’. ran this script in the above maximal cluster and the results are as under,







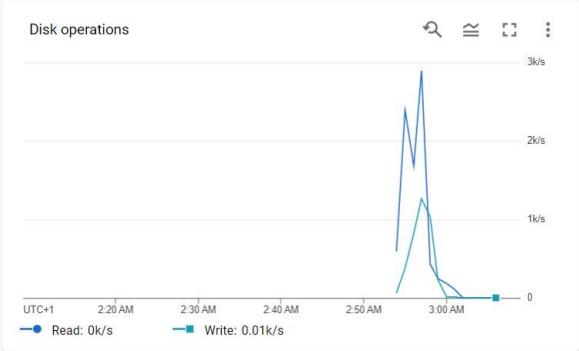
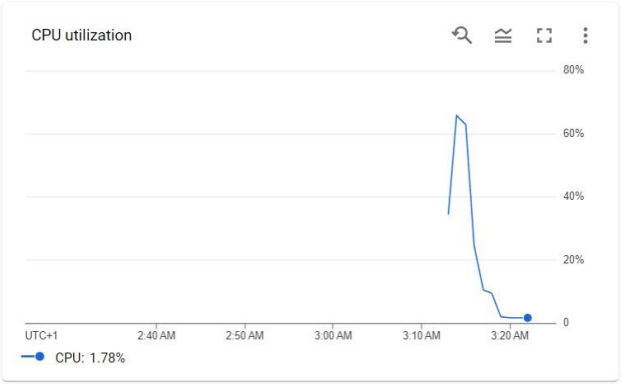


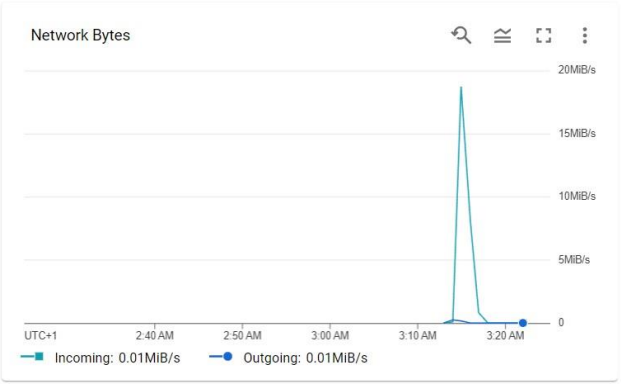
Table 3: 16 partition script on maximal cluster - 1 Master (1CPU) + 7Worker node (2CPUs/each)					
Metrics	**Network Bytes**	**Network Packets**	**Disk Bytes**	**Disk Operations**	**CPU Utilization**
Max incoming	36.64 MiB/s	26.45 k/s	NA		81.28%
Max Outgoing	0.56 MiB/s	0.68 k/s			
Max Read	NA		56.63 MiB/s	2.89 k/s	
Max Write			170.76 MiB/s	1.26 k/s	

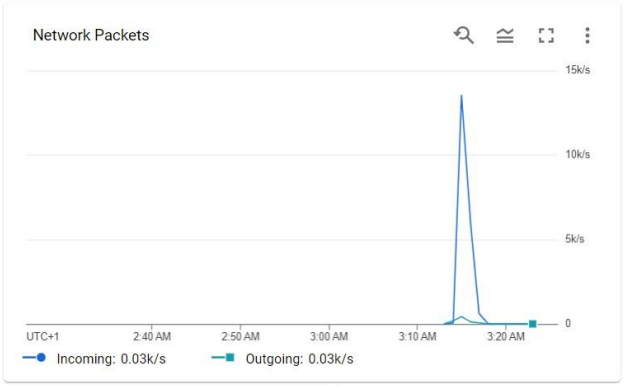
Inference: From the two tables 2 and 3, we can see that increasing the number of partitions from 2 to 16 has led to a slight increase in cluster utilization. Specifically, the maximum incoming network bytes and packets, as well as the maximum read and write disk bytes and operations have all increased in 16-partition compared to 2-partition.
This increase in cluster utilization can be attributed to the fact that with more partitions, the workload is distributed among more worker nodes, enabling parallel processing and reducing the time taken for each task. As a result, the overall processing time is reduced **from 60 seconds to 49 seconds.**

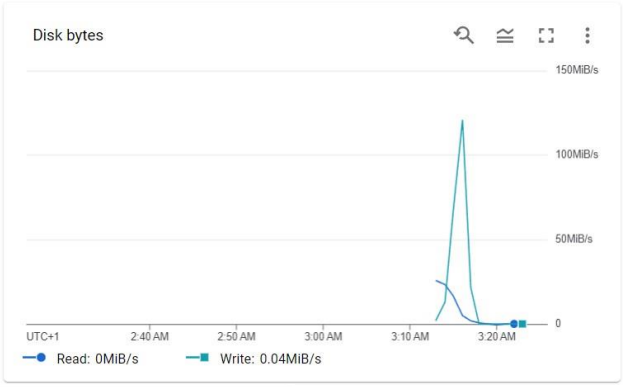
1.d.ii) Experiment with cluster configurations: test your program with 4 machines with double the resources each (2 vCPUs, memory, disk) and 1 machine with eightfold resources. Discuss the results in terms of disk I/O and network bandwidth allocation in the cloud.

Results of 16-partition script ran on cluster with 4 machines with double the resources each (2 vCPUs, memory, disk).









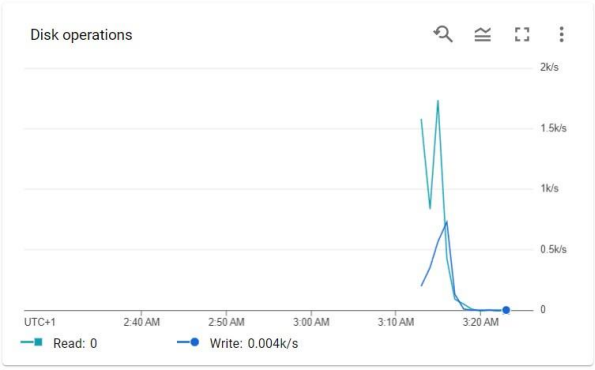
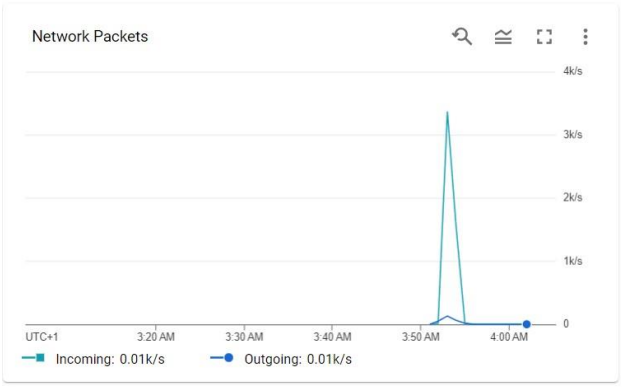
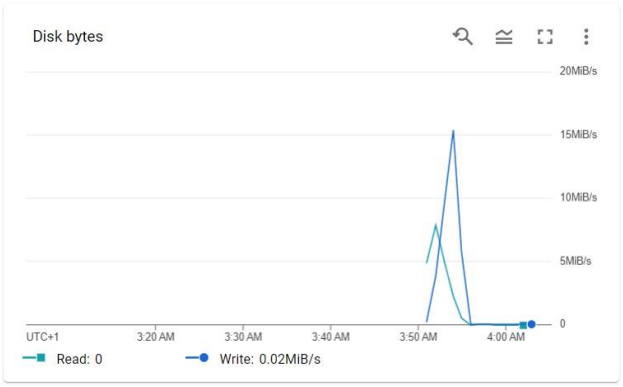
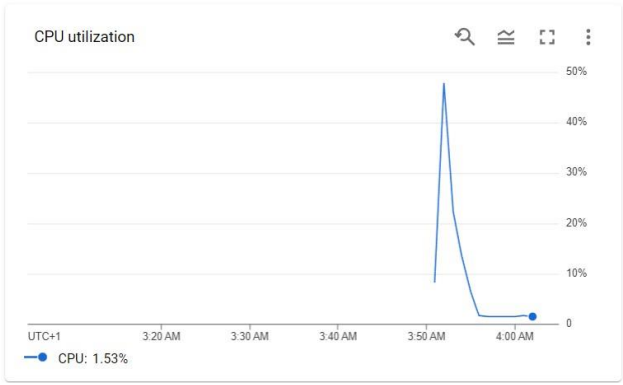


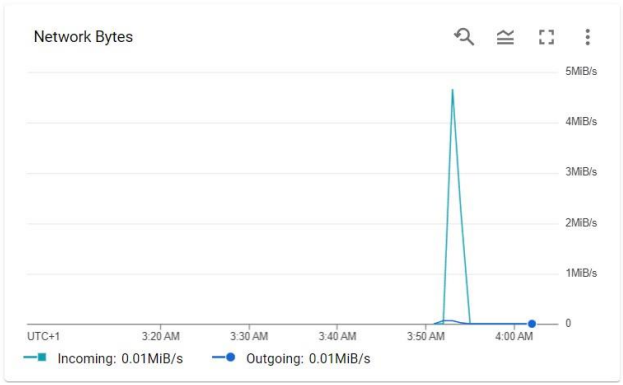
Table 4: 16 partition script on maximal cluster - 1 Master (2CPU) + 3Worker node (2CPUs/each)					
Metrics	**Network Bytes**	**Network Packets**	**Disk Bytes**	**Disk Operations**	**CPU Utilization**
Max incoming	18.74 MiB/s	13.54 k/s	NA		65.79%
Max Outgoing	0.21 MiB/s	0.45 k/s			
Max Read	NA		25.75 MiB/s	1.739 k/s	
Max Write			120.82 MiB/s	0.431 k/s	

Now, the results of 16-partition script ran 1 Master (8CPU) is as under,









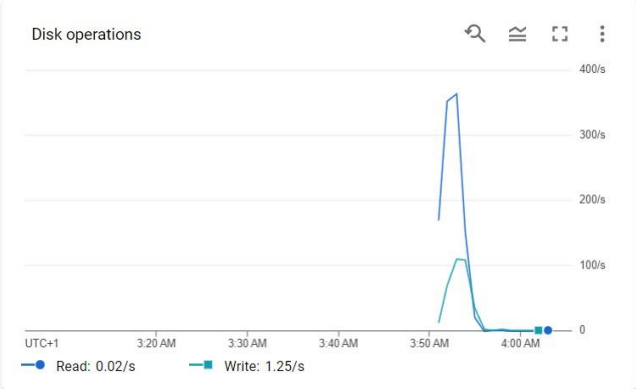


Table 5: 16 partition script on maximal cluster - 1 Master (8CPU)					
Metrics	**Network Bytes**	**Network Packets**	**Disk Bytes**	**Disk Operations**	**CPU Utilization**
Max incoming	4.67 MiB/s	3.38 k/s	NA		47.86%
Max Outgoing	0.08 MiB/s	0.14 k/s			
Max Read	NA		7.86 MiB/s	362.53 /s	
Max Write			15.42 MiB/s	109.77 /s	

Table 4 shows the performance of a 16-partition script on a maximal cluster with 1 Master node (2 CPUs) and 3 Worker nodes (2 CPUs each). The CPU utilization in this configuration was 65.79%, indicating that the available processing power was not fully utilized. The network bandwidth for incoming data was relatively high at 18.74 MiB/s, while outgoing data was low at 0.21 MiB/s. The disk operations for read and write were also high at 25.75 MiB/s and 120.82 MiB/s, respectively.

Table 5 shows the performance of the same script on a maximal cluster with 1 Master node (8 CPUs). The CPU utilization in this configuration was 47.86%, indicating that there was still processing power available for additional tasks. The network bandwidth for incoming data was lower at 4.67 MiB/s, while outgoing data was even lower at 0.08 MiB/s. The disk operations for read and write were also lower at 7.86 MiB/s and 15.42 MiB/s, respectively.

The results suggest that increasing the number of CPUs available to the system can lead to higher CPU utilization and potentially faster processing times. However, increasing the number of nodes in the cluster may not necessarily improve performance in all cases. In this particular case, the cluster with a single Master node and 8 CPUs actually had lower network bandwidth and disk I/O performance compared to the cluster with multiple Worker nodes.

1.d.iii) Explain the difference between this use of Spark and most standard applications like e.g. in our labs in terms of where the data is stored. What kind of parallelisation approach is used here?

Spark differs from most standard applications in terms of where the data is stored and the parallelization approach used. In our labs, data is usually stored locally, and the processing is also done locally on a single machine. However, with Spark, data is stored in a distributed file system, such as Hadoop Distributed File System (HDFS) or Amazon S3, which allows for efficient parallel processing across a cluster of machines. Spark's parallelization approach is based on the concept of Resilient Distributed Datasets (RDDs), which enable data to be partitioned across a cluster and processed in parallel on multiple machines. This approach allows Spark to handle large datasets and complex processing tasks more efficiently than traditional single-machine processing.

3. Section 2: Speed Tests

This task is about measuring this effect and parallelizing the tests with PySpark.

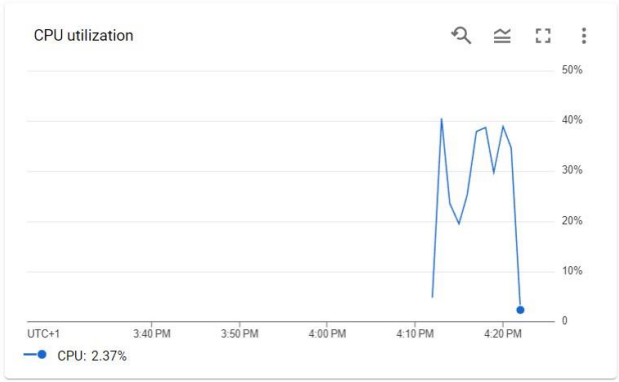
Task 2: Parallelising the speed test with Spark in the cloud.

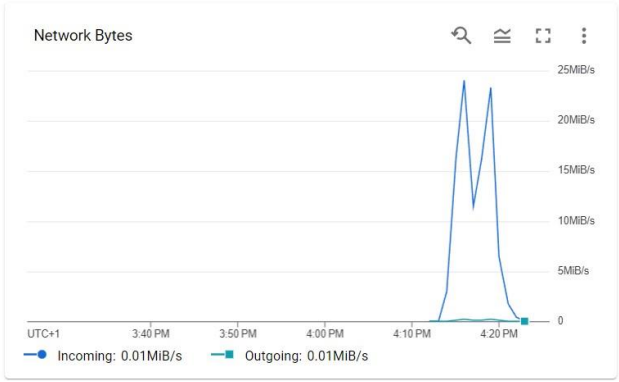
2a) Create the script: Adapt the speed testing as a Spark program that performs the same actions as above, but with Spark RDDs in a distributed way. The distribution should be such that each parameter combination (except repetition) is processed in a separate Spark task. Created Script

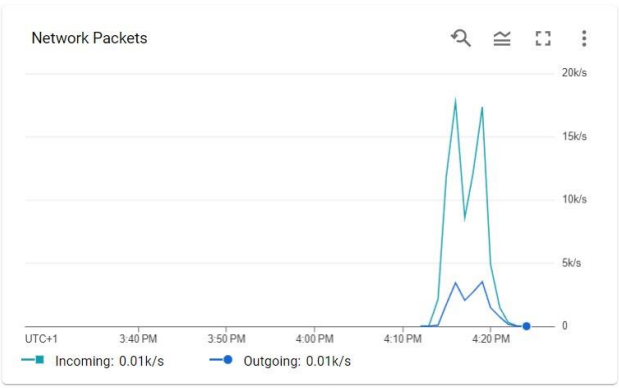
Now I have written a script to read the image files and tfrecord files in a file name spark_job.py
In that script, I have used the following combinations,
batch_sizes = [10, 20, 30, 40]
batch_numbers = [10, 20, 30, 40]
repetitions = [1, 2, 3]
2b) Testing the code and collecting results:

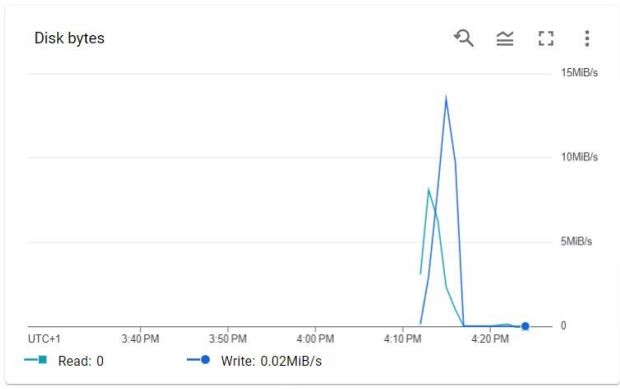
i) Ran spark_job.py locally

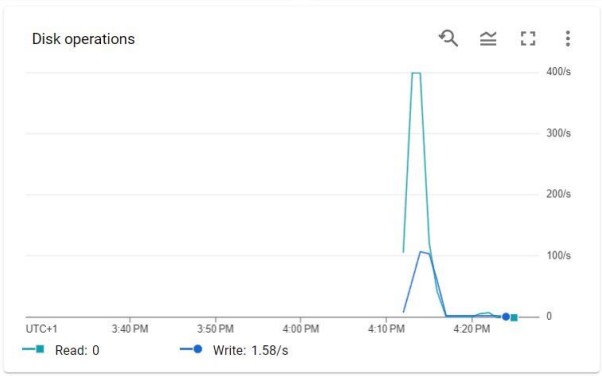
ii) Created spark_job_version2b.py to run it in cluster. Created 1M (8CPU) cluster ran the file. It has taken 686 seconds to run, and other metrics as under,





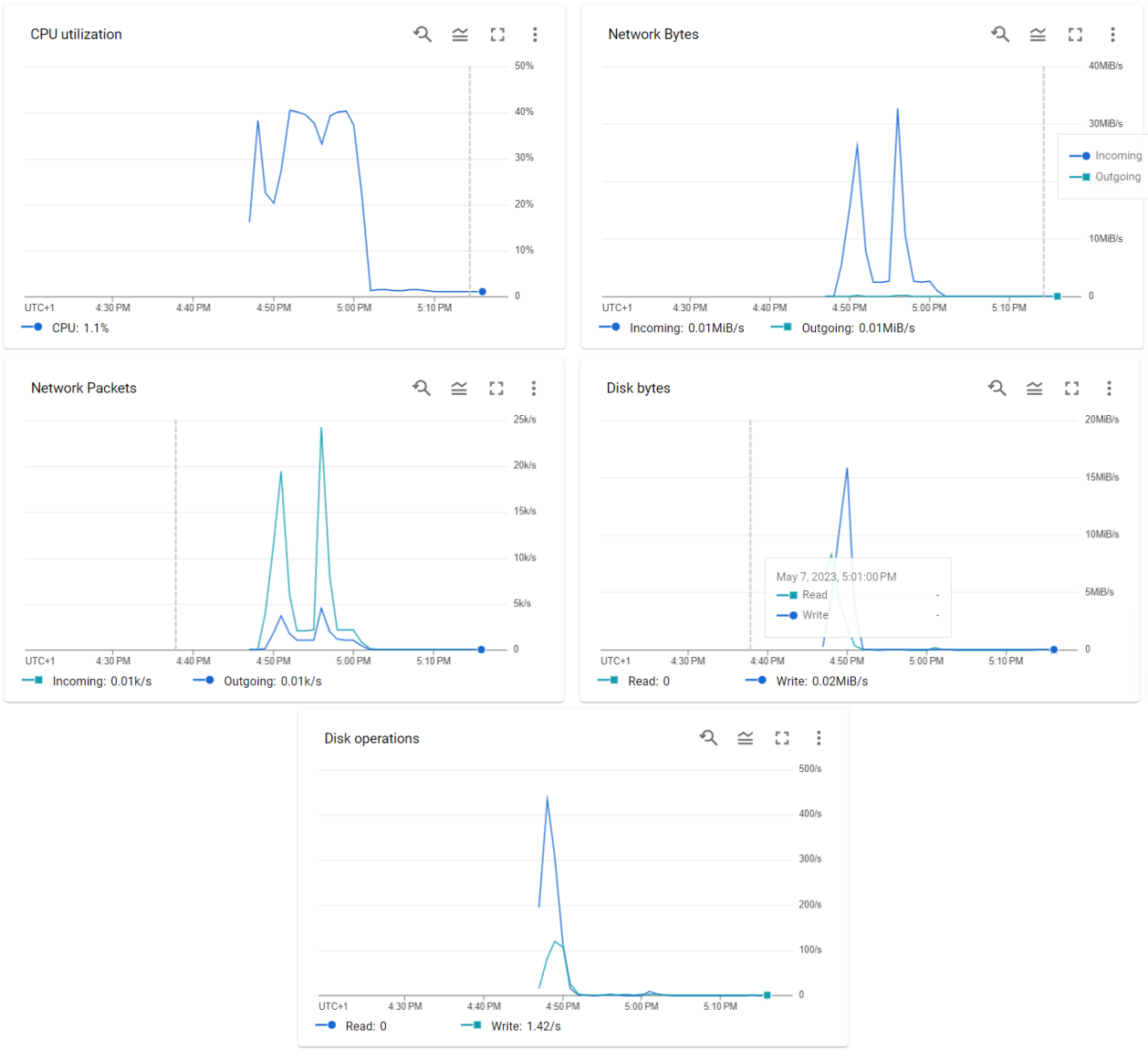






2C) Improve efficiency:
In order to improve efficiency, changed to add rdd.cache() and saved it as spark_job_version2c.py – **It has taken only 204 seconds which 30% of the time using the version 2b, which means in this case using cache has saved 70% of the time**. The other

parameters are captured and are as under,



2d) In order to find the correlation between the throughputs/reading speeds, we have analysed the results and compared the running speed with each and every parameter with both image files and TFrec files. Here are the Linear regression done of different variables,

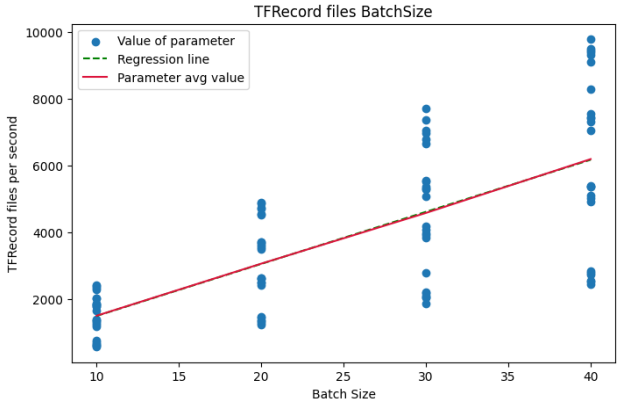


Fig. 1 Batch Size - TFRecord files

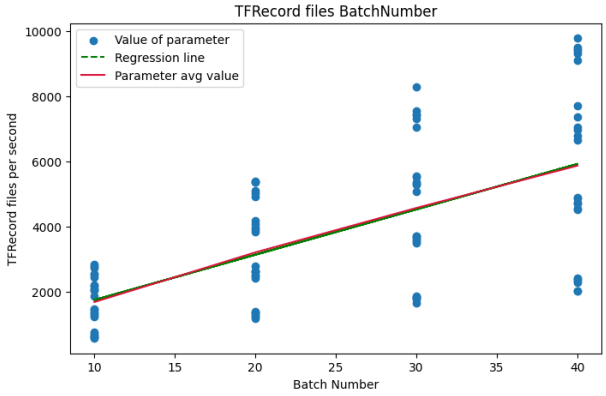


Fig. 2 Batch Number - TFRecord files

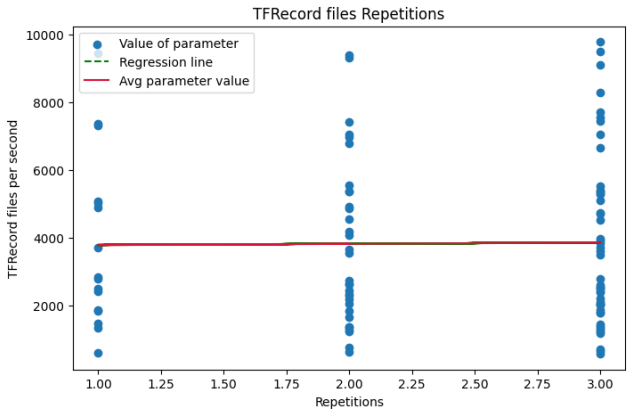


Fig. 3 Repetitions - TFRecord files

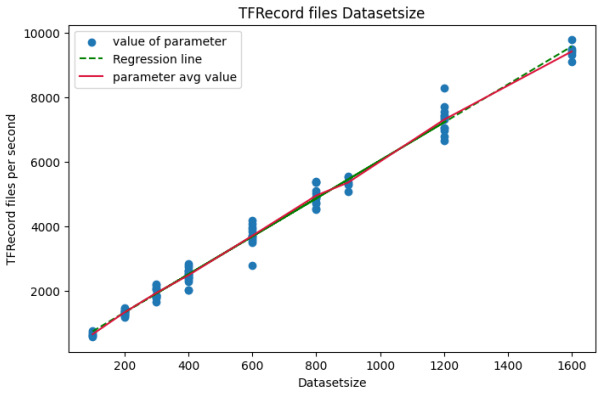


Fig. 4 Datasize - TFRecord files

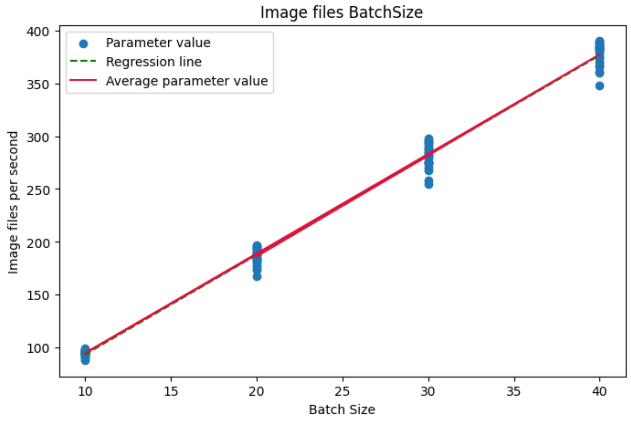


Fig. 5 Batch Size – Image files

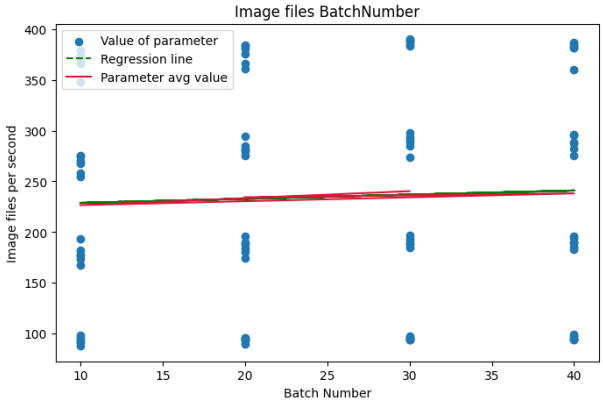


Fig. 6 Batch Number - Image files

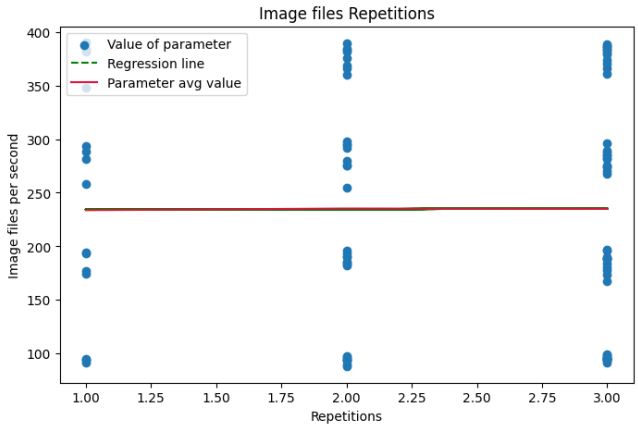


Fig. 7 Repetitions - Image files

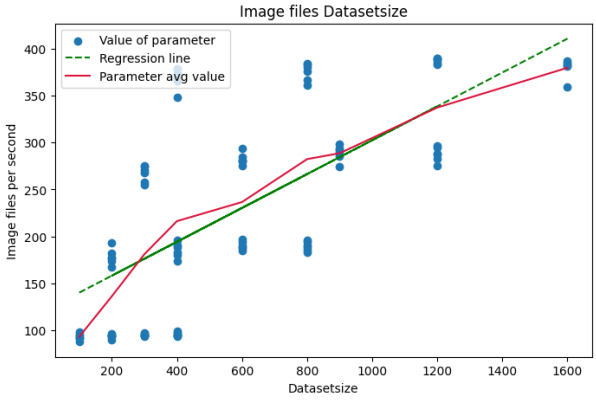


Fig. 8 Datasize - Image files

Parameters	slope	Intercept	R2-Value	correlation coefficient
TFRecord files Batch Size	156.24	-77.96	0.5	0.71
TFRecord files Batch Number	139.47	341.29	0.4	0.63
TFRecord files Repetitions	35.74	3744	0	0.00
TFRecord files Dataset size	5.9	140.09	0.98	0.99
Image files Batch Size	9.43	-0.97	0.99	0.99
Image files Batch Number	0.4	224.74	0	0.00
Image files Repetitions	0.44	233.85	0	0.00
Image files Dataset size	0.18	122.23	0.49	0.70

Table 6: Output of each parameter

As shown in the table above, in TF record files, the batch size and batch numbers are correlated with the running speed with the correlation coefficient of 0.71 and 0.63 respectively. Datasize is nothing but the combination of batch size and batch number, so the results are also in the same line. Whereas, repetitions is not at all correlated with the running speed.

4. Section 3: Theoretical discussion:

The paper describes the challenge of selecting the optimal cloud configuration for big data analytics jobs running in clouds due to the large number of possible VM instance types and cluster sizes to choose from. The wrong choice can lead to significant performance degradation and increased costs. The paper introduces CherryPick, a system that uses Bayesian Optimization to build performance models for various applications to identify the best or close-to-the-best configuration with only a few test runs. The experiments conducted on five analytic applications in AWS EC2 show that CherryPick has a high chance of finding optimal or near-optimal configurations, saving up to 75% search cost compared to existing solutions.

3.a Contextualise:

The paper discusses techniques for optimizing the performance of Apache Spark applications, which is a distributed computing framework for processing large-scale data sets. The techniques include partitioning, caching, and optimizing data transfer over the network.

In the context of the tasks in this coursework, the results from the tables the performance of different scripts on a Spark cluster with varying configurations. Based on the results, it is evident that changing the configuration of the Spark cluster has a significant impact on the performance of the scripts. For instance, increasing the number of worker nodes in the cluster and partitioning the data into smaller chunks significantly improved the performance of the scripts. Additionally, caching data in memory led to a considerable reduction in disk I/O operations, which improved the overall performance.

3.b Strategise:

Concrete strategies for different application scenarios can be defined based on the results obtained from the tables. For batch processing scenarios, partitioning the data into smaller chunks can significantly improve performance, as seen in the comparison of single machine cluster vs the maximal cluster (ref table 1 and 2). Additionally, increasing the number of worker nodes in the cluster can lead to improved performance, as seen when we fixed the maximal cluster and changed the scripts from 2 partition to 16 partition (ref table 2 and 3).

For stream processing scenarios, reducing the batch interval and ensuring data is cached in memory can significantly improve performance. This is because stream processing requires real-time processing, and reducing the batch interval ensures that data is processed as soon as it is received. Caching data in memory reduces the number of disk I/O operations, which can lead to a significant improvement in performance.

In general, the concepts discussed in the paper, such as partitioning, caching, and optimizing data transfer over the network, are critical for improving the performance of Spark applications. Therefore, it is essential to optimize these factors for any application scenario to ensure optimal performance.

Colab link to my codes: https://colab.research.google.com/drive/1-fcOkuRxZ-Tnz7xKxxo0Z6mFXfFgJP5X?usp=share_link