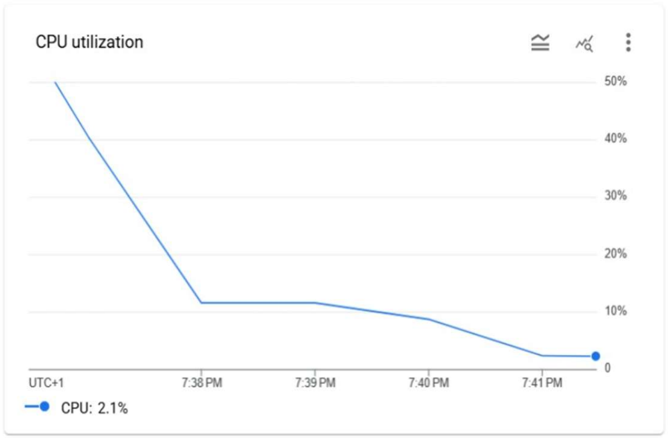
Notebook link:
https://colab.research.google.com/drive/17ZuVaNWnAmQPrMXyYcSr1Z1HeIfTwWb5?usp=sharing

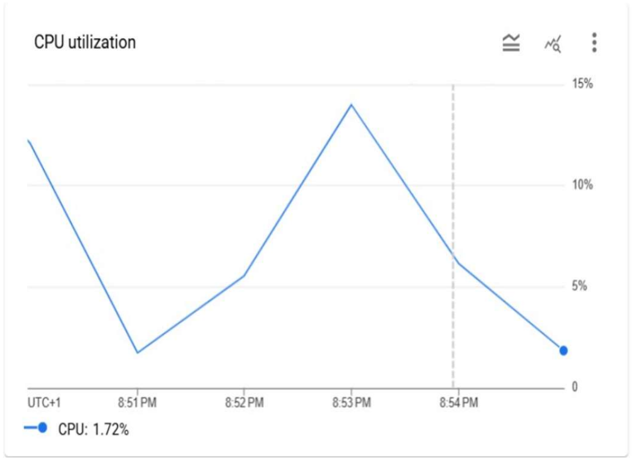
1d: Optimisation, experiments, and discussion

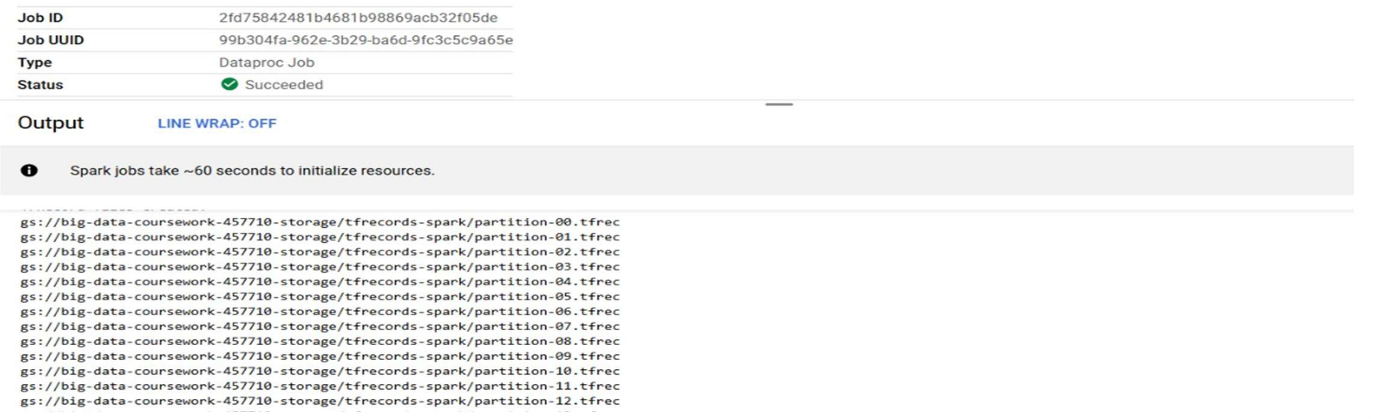
i. Improve parallelisation:
In task 1c-ii, The Spark preprocessing script used two partitions, showing CPU utilization of 2.1% and 2 TFRecord files.
After modifying with sc.parallelize(filenames, PARTITIONS), setting PARTITIONS = 16.

- 16 TFRecord files were created.
- Processing time improved from 44.89 seconds to 35.11 seconds.

Although, post-partitioning CPU usage was measured at 1.72%, the decrease was due to the small dataset size as we used factor 0.02.





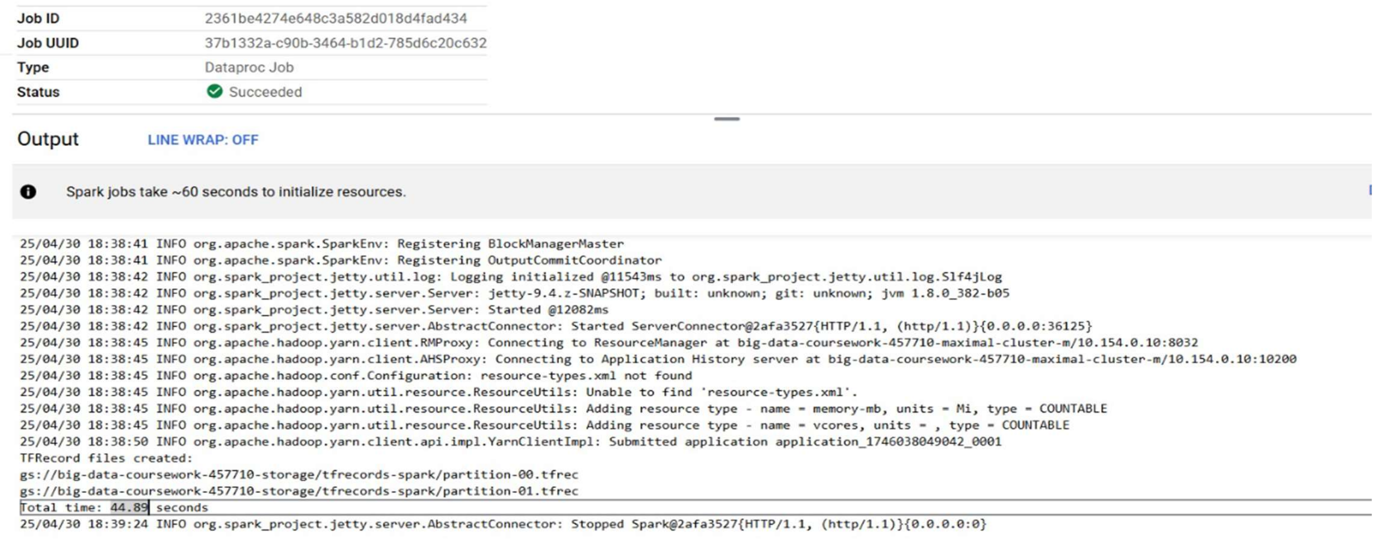


Job ID	2fd75842481b4681b98869acb32f05de
Job UUID	99b304fa-962e-3b29-ba6d-9fc3c5c9a65e
Type	Dataproc Job
Status	Succeeded

Output LINE WRAP: OFF

Spark jobs take ~60 seconds to initialize resources.

```
gs://big-data-coursework-457710-storage/tfrecords-spark/partition-00.tfrec
gs://big-data-coursework-457710-storage/tfrecords-spark/partition-01.tfrec
gs://big-data-coursework-457710-storage/tfrecords-spark/partition-02.tfrec
gs://big-data-coursework-457710-storage/tfrecords-spark/partition-03.tfrec
gs://big-data-coursework-457710-storage/tfrecords-spark/partition-04.tfrec
gs://big-data-coursework-457710-storage/tfrecords-spark/partition-05.tfrec
gs://big-data-coursework-457710-storage/tfrecords-spark/partition-06.tfrec
gs://big-data-coursework-457710-storage/tfrecords-spark/partition-07.tfrec
gs://big-data-coursework-457710-storage/tfrecords-spark/partition-08.tfrec
gs://big-data-coursework-457710-storage/tfrecords-spark/partition-09.tfrec
gs://big-data-coursework-457710-storage/tfrecords-spark/partition-10.tfrec
gs://big-data-coursework-457710-storage/tfrecords-spark/partition-11.tfrec
gs://big-data-coursework-457710-storage/tfrecords-spark/partition-12.tfrec
```



Job ID	2361be4274e648c3a582d018d4fad434
Job UUID	37b1332a-c90b-3464-b1d2-785d6c20c632
Type	Dataproc Job
Status	Succeeded

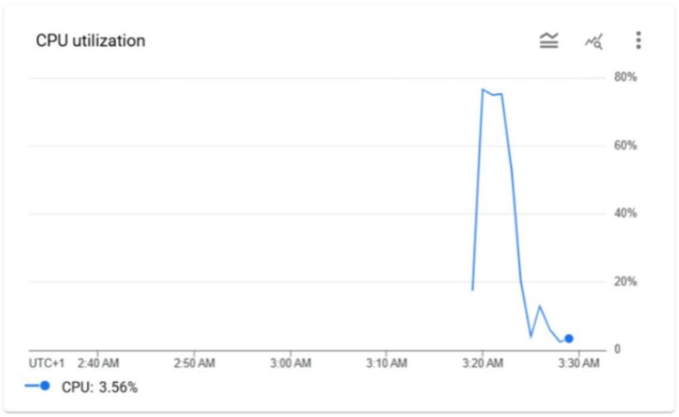
Output LINE WRAP: OFF

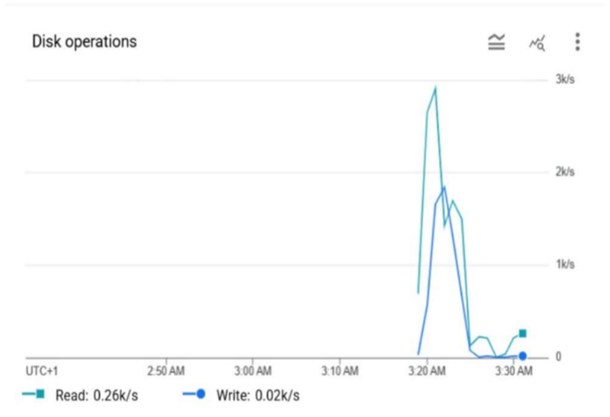
Spark jobs take ~60 seconds to initialize resources.

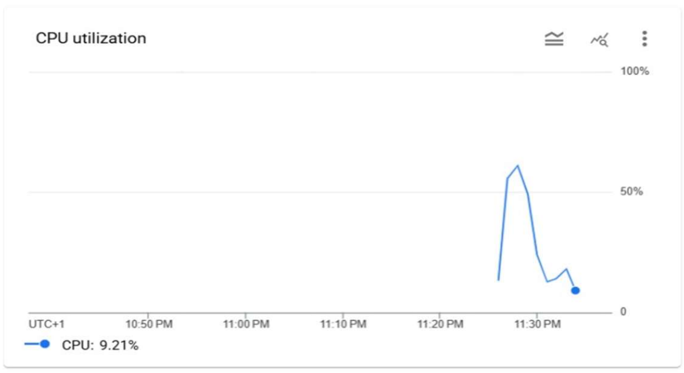
```
25/04/30 18:38:41 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
25/04/30 18:38:41 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
25/04/30 18:38:42 INFO org.spark_project.jetty.util.log: Logging initialized @11543ms to org.spark_project.jetty.util.log.Slf4jLog
25/04/30 18:38:42 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_382-b05
25/04/30 18:38:42 INFO org.spark_project.jetty.server.Server: Started @12082ms
25/04/30 18:38:42 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@2afa3527{HTTP/1.1, (http/1.1)}{0.0.0.0:36125}
25/04/30 18:38:45 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at big-data-coursework-457710-maximal-cluster-m/10.154.0.10:8032
25/04/30 18:38:45 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at big-data-coursework-457710-maximal-cluster-m/10.154.0.10:10200
25/04/30 18:38:45 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
25/04/30 18:38:45 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-types.xml'.
25/04/30 18:38:45 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = memory-mb, units = Mi, type = COUNTABLE
25/04/30 18:38:45 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = vcores, units = , type = COUNTABLE
25/04/30 18:38:50 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application_1746038049042_0001
TFRecord files created:
gs://big-data-coursework-457710-storage/tfrecords-spark/partition-00.tfrec
gs://big-data-coursework-457710-storage/tfrecords-spark/partition-01.tfrec
Total time: 44.89 seconds
25/04/30 18:39:24 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@2afa3527{HTTP/1.1, (http/1.1)}{0.0.0.0:0}
```

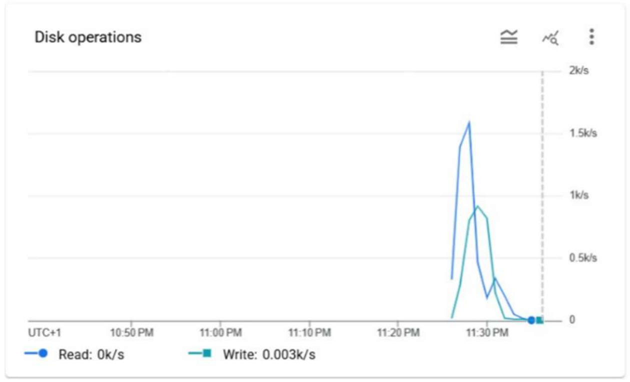
ii. Experiment with cluster configurations**:**

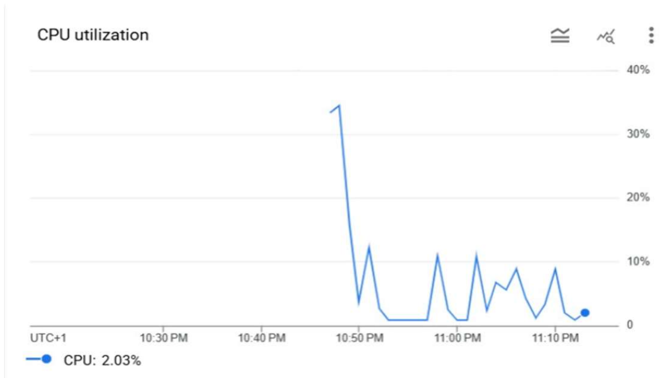
	8 VMs (n1-standard-1)	**4 VMs (n1-standard-2)**	**1 VM (n1-standard-8)**
CPU Utilization	3.56%	9.21%	2.03%
Disk Operations	R:0.26k/s,W:0.02k/s	R: 0/s,W:0.003k/s	R:0.98/s,W:2.43/s
Network Bytes	0.13/0.05 MiB/s	0.01/0.05 MiB/s	0.01/0.01 MiB/s
Network Packets	0.14k/0.14k/s	12.01k/0.65k/s	0.01k/0.02k/s

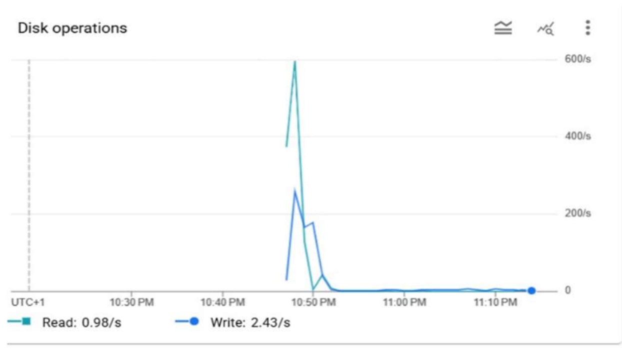












iii. Difference between the use of Spark and standard applications:

The Spark application significantly different from the lab-based exercises in terms of data handling and execution. Primarily used structured local files and ran in a local Spark session, with limited parallelism and single-node execution. In contrast, the Spark application processed image files stored in Google Cloud Storage, serialized them into TFRecords, and ran on a multi-node Dataproc cluster.

Data loading in labs was typically done using textFile(), while the Spark application used tf.io.gfile.glob() to read from GCS.

Parallelisation approach used in task:

- Data Parallelism: The preprocessing operation(decode, resize, recompress) is applied to different files across partitions.
- Task Parallelism with RDD Partitions: Each image is processed independently within an RDD partition, allowing multiple Spark executors to work concurrently.
- mapPartitionsWithIndex was used to write files, one TFRecord per partition, improving performance in distributed environments.
- parallelism using partitions sc.parallelize(filenames, 16)

2c) Improve efficiency

In the Task 2a, performed two separate actions in Spark RDD:

1. result_rdd.collect() - to retrieve results.
2. result_rdd.groupByKey().mapValues() - to calculate the average.

Here, result_rdd was recomputed twice, causing the dataset to be processed again during both actions.

To avoid this inefficient recomputation, use the cache() method in the result_rdd. This tells Spark to keep the RDD in memory after the first computation for faster performance.

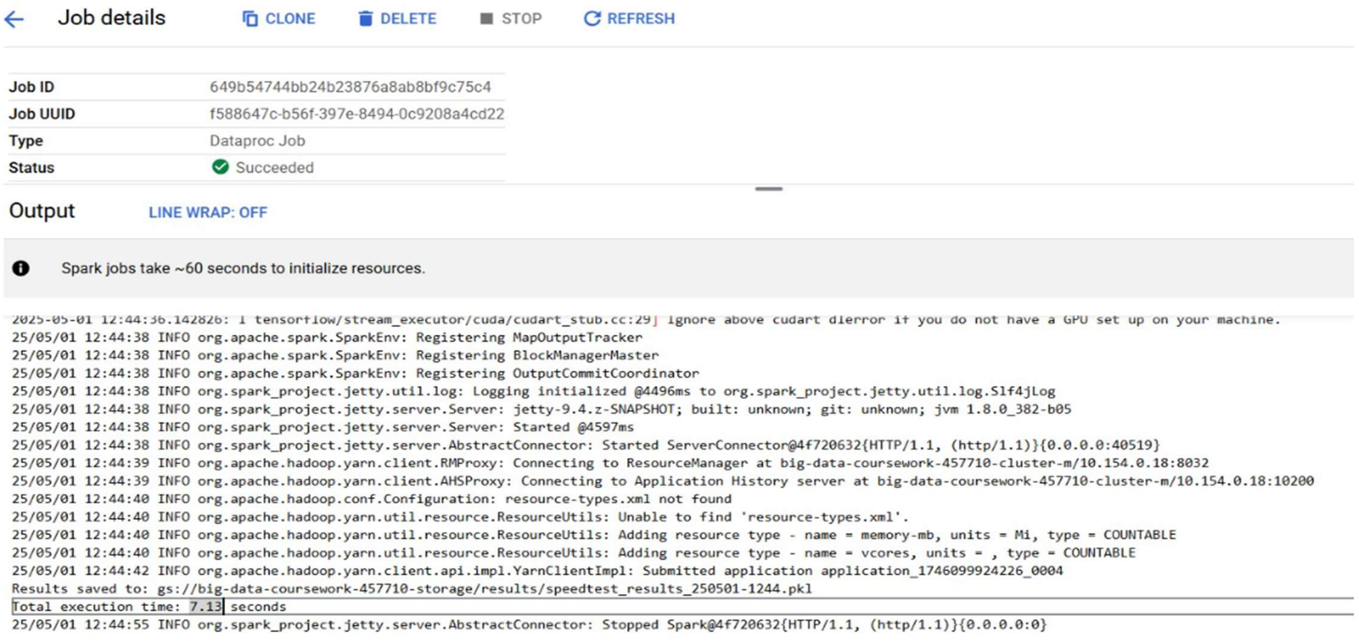
result_rdd = param_rdd.map(time_one_config).cache()

result_rdd.count() - Triggers caching

observation:

- Without caching: execution time 9.62 seconds
- With caching: execution time 7.13 seconds

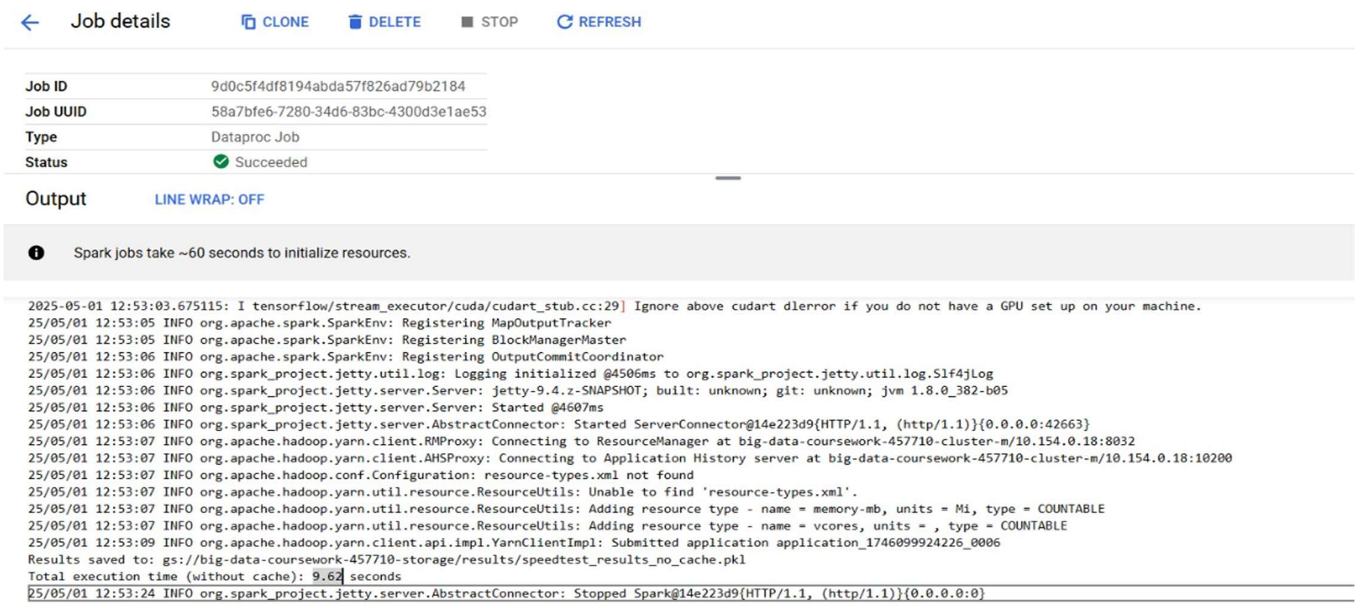
This is a 26% performance improvement, achieved by reducing redundant computation through caching.



Job ID	649b54744bb24b23876a8ab8bf9c75c4
Job UUID	f588647c-b56f-397e-8494-0c9208a4cd22
Type	Dataproc Job
Status	Succeeded

Spark jobs take ~60 seconds to initialize resources.

```
2025-05-01 12:44:36.142826: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
25/05/01 12:44:38 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
25/05/01 12:44:38 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
25/05/01 12:44:38 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
25/05/01 12:44:38 INFO org.spark_project.jetty.util.log: Logging initialized @4496ms to org.spark_project.jetty.util.log.Slf4jLog
25/05/01 12:44:38 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_382-b05
25/05/01 12:44:38 INFO org.spark_project.jetty.server.Server: Started @4597ms
25/05/01 12:44:38 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@4f720632{HTTP/1.1, (http/1.1)}{0.0.0.0:40519}
25/05/01 12:44:39 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at big-data-coursework-457710-cluster-m/10.154.0.18:8032
25/05/01 12:44:39 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at big-data-coursework-457710-cluster-m/10.154.0.18:10200
25/05/01 12:44:40 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
25/05/01 12:44:40 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-types.xml'.
25/05/01 12:44:40 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = memory-mb, units = Mi, type = COUNTABLE
25/05/01 12:44:40 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = vcores, units = , type = COUNTABLE
25/05/01 12:44:42 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application_1746099924226_0004
Results saved to: gs://big-data-coursework-457710-storage/results/speedtest_results_250501-1244.pkl
Total execution time: 7.13 seconds
25/05/01 12:44:55 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@4f720632{HTTP/1.1, (http/1.1)}{0.0.0.0:0}
```



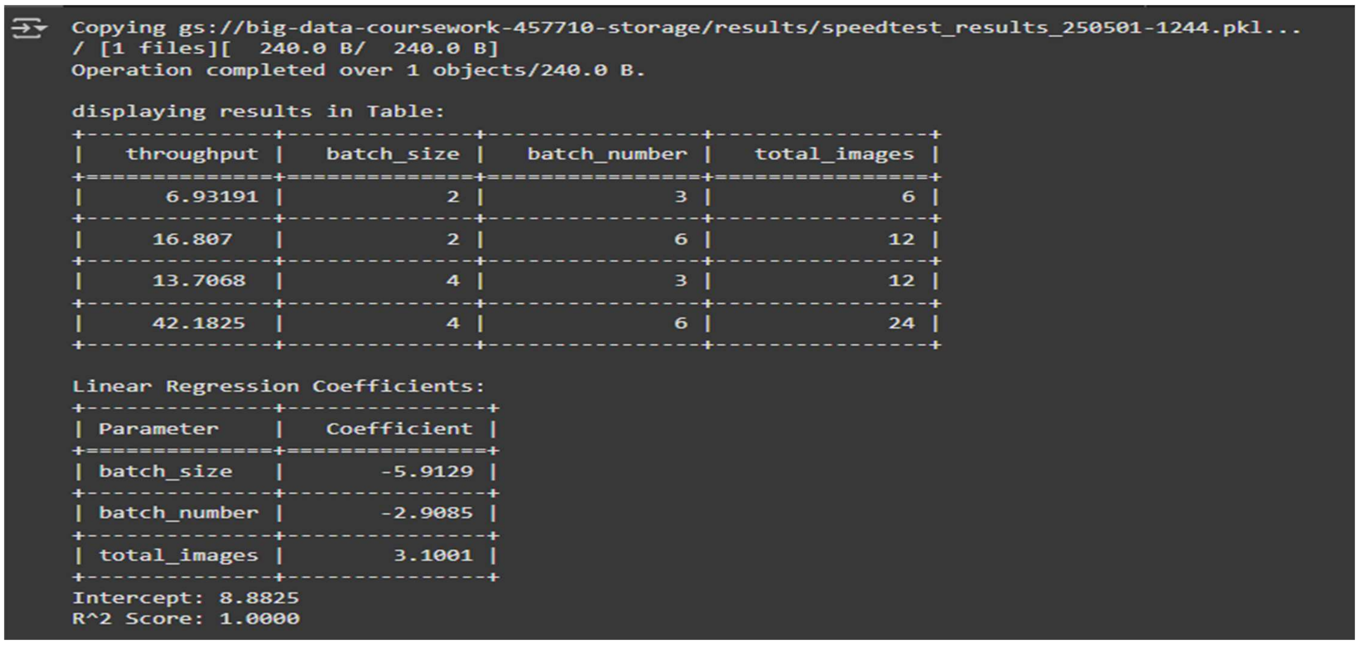
Job ID	9d0c5f4df8194abda57f826ad79b2184
Job UUID	58a7bfe6-7280-34d6-83bc-4300d3e1ae53
Type	Dataproc Job
Status	✓ Succeeded

Spark jobs take ~60 seconds to initialize resources.

```
2025-05-01 12:53:03.675115: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
25/05/01 12:53:05 INFO org.apache.spark.SparkEnv: Registering MapOutputTracker
25/05/01 12:53:05 INFO org.apache.spark.SparkEnv: Registering BlockManagerMaster
25/05/01 12:53:06 INFO org.apache.spark.SparkEnv: Registering OutputCommitCoordinator
25/05/01 12:53:06 INFO org.spark_project.jetty.util.log: Logging initialized @4506ms to org.spark_project.jetty.util.log.Slf4jLog
25/05/01 12:53:06 INFO org.spark_project.jetty.server.Server: jetty-9.4.z-SNAPSHOT; built: unknown; git: unknown; jvm 1.8.0_382-b05
25/05/01 12:53:06 INFO org.spark_project.jetty.server.Server: Started @4607ms
25/05/01 12:53:06 INFO org.spark_project.jetty.server.AbstractConnector: Started ServerConnector@14e223d9{HTTP/1.1, (http/1.1)}{0.0.0.0:42663}
25/05/01 12:53:07 INFO org.apache.hadoop.yarn.client.RMProxy: Connecting to ResourceManager at big-data-coursework-457710-cluster-m/10.154.0.18:8032
25/05/01 12:53:07 INFO org.apache.hadoop.yarn.client.AHSProxy: Connecting to Application History server at big-data-coursework-457710-cluster-m/10.154.0.18:10200
25/05/01 12:53:07 INFO org.apache.hadoop.conf.Configuration: resource-types.xml not found
25/05/01 12:53:07 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Unable to find 'resource-types.xml'.
25/05/01 12:53:07 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = memory-mb, units = Mi, type = COUNTABLE
25/05/01 12:53:07 INFO org.apache.hadoop.yarn.util.resource.ResourceUtils: Adding resource type - name = vcores, units = , type = COUNTABLE
25/05/01 12:53:09 INFO org.apache.hadoop.yarn.client.api.impl.YarnClientImpl: Submitted application application_1746099924226_0006
Results saved to: gs://big-data-coursework-457710-storage/results/speedtest_results_no_cache.pkl
Total execution time (without cache): 9.62 seconds
25/05/01 12:53:24 INFO org.spark_project.jetty.server.AbstractConnector: Stopped Spark@14e223d9{HTTP/1.1, (http/1.1)}{0.0.0.0:0}
```

2d) Retrieve, analyse and discuss the output

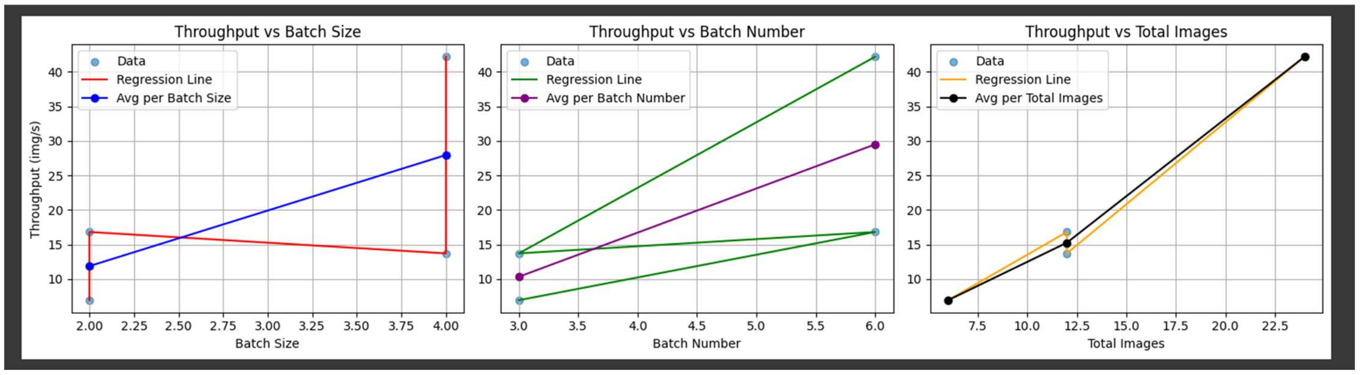
Output:



```
Copying gs://big-data-coursework-457710-storage/results/speedtest_results_250501-1244.pkl...
/ [1 files][  240.0 B/  240.0 B]
Operation completed over 1 objects/240.0 B.

displaying results in Table:
+--------------+---------------+-----------------+----------------+
|   throughput |    batch_size |    batch_number |   total_images |
+==============+===============+=================+================+
|      6.93191 |             2 |               3 |              6 |
+--------------+---------------+-----------------+----------------+
|     16.807   |             2 |               6 |             12 |
+--------------+---------------+-----------------+----------------+
|     13.7068  |             4 |               3 |             12 |
+--------------+---------------+-----------------+----------------+
|     42.1825  |             4 |               6 |             24 |
+--------------+---------------+-----------------+----------------+

Linear Regression Coefficients:
+--------------+---------------+
| Parameter    |   Coefficient |
+==============+===============+
| batch_size   |       -5.9129 |
+--------------+---------------+
| batch_number |       -2.9085 |
+--------------+---------------+
| total_images |        3.1001 |
+--------------+---------------+
Intercept: 8.8825
R^2 Score: 1.0000
```



Total_images(+3.10 img/s per extra image): each additional image in total batch pool adds ~3.1 img/s.

Negative single-term coefficients: because batch_size and batch_number are collinear, the model subtracts out part of each once total_images

$R^2=1$: with only four points and three highly correlated predictors, the regression line exactly interpolates measurements.

Implications of result for large-scale machine-learning on cloud storage:

Increasing batch sizes or the number of batches reduces throughput due to significant fixed delays in reading from cloud storage. For instance, Google Cloud Storage retrieval can take 10-100 milliseconds per call, compared to local SSD reads at 50 microseconds. At the same time, the large number of images per request helps mitigate per-call costs, increasing processing speed until network capacity limits or bandwidth delays become bottlenecks. Optimize batch sizes to balance payload and request overhead to run on cloud storage effectively, utilize local SSDs or caches for storing large data shards, and co-locate computing with storage to minimize latency. Caches can help reduce the impact of remote latency and recomputation.

Observed Behavior with compare to a Single Machine:

Aspect	**Cloud Storage**	**Single Machine**	**Key Difference**
latency	0.5 ms (LAN RTT), 50 ms (WAN RTT), 10–100 ms per GCS GET	~50 µs per SSD read, ~10 ms per HDD read	Cloud calls pay 100–1 000 more latency.
Throughput scaling	Linear rise until VM egress or QPS limits are hit	Near-linear scaling with IOPS/bandwidth	Cloud needs batching to offset high per-call costs.
Variability	High jitter	Low jitter	Cloud I/O timing is unpredictable. local I/O is consistent.

Why would cloud providers tie throughput to disk-resource capacity?

Fairness and isolation: Reviewing each bucket bandwidth prevents a single tenant from starving others.

Resource planning: Predictive I/O limits, providers should accurately size and bill network and storage backends.

Hardware constraints: Object stores share physical disks, networks, and load balancers, and each bucket can be mapped to real hardware saturation points.

Parallel speed tests on cloud storage must account for several potential bottlenecks:

VM network-egress bandwidth(e.g. a 10 Gbps NIC)

Bucket QPS limits(requests/sec per prefix)

Per-request TCP/TLS setup overhead

Client-side CPU/GIL limits when deserialising data linear modelling reflect

Doubling both batch size and batch count raised throughput from about 17 to 42 img/s, indicating the latency-amortisation regime and had not yet saturated any of these resources.

By increasing the number of parallel readers or using larger batch sizes, measure network bandwidth, bucket QPS limits, or CPU capacity is the factor that ultimately caps throughput.

The small range of data points**,** A multivariate linear model will interpolate perfectly($R^2=1$), yielding “img/s per extra image” estimates.

relationship: throughput follows a **saturating curve**

throughput(s,n) ≈ $\frac{s.n}{L+\frac{s.n}{B_{max}}}$

If need to predict performance for much larger batches or different network and storage setups, must use a non-linear, saturating model to achieve the exact result.

Task 3: Discussion in context

3a) Contextualise

In Tasks, Process has been done manually measured end-to-end performance specifically TFRecord write time and TFRecord read throughput under varying cloud and Spark configurations. These configurations included differences in cluster type for example, single-node, 4-node etc. clusters, Spark parallelism, and batching parameters (batch size and number of batches). As per the observation larger clusters and increased batch sizes generally improved throughput, However, improvements were reduced when I/O, network, or cloud limitations were encountered. Utilized caching to avoid redundant reads and highlighted the impact of cloud storage latency as a significant performance constraint.

The **CherryPick paper** focuses on optimizing data retrieval and distribution strategies for improving data flow. These concepts are highly relevant to the tasks which completed in this coursework, especially in relation to performance optimization and bottleneck identification during data processing in cloud-based environments.

It addresses same problem but automates it using **Bayesian Optimization** instead of manual work:

1. **Black-box performance modeling**
   objective (e.g. images/sec or total write time) as an unknown function C(x) of configuration (cluster size, vCPUs, disk type, partitions, batch size, etc.). When the problem scale becomes large then CherryPick accepted Gaussian Process is the only choice which is computationally tractable.

2. **Acquisition-driven search**
   Rather than trying every combination, CherryPick uses an **Expected Improvement** acquisition function to select the next configuration that is most likely to improve performance.

3. **Automatic stopping under noise**
   Cloud measurements are noisy due to shared networks but CherryPick's model resolve that noise. It stops when, for example, it is 95 % confident that the best observed configuration is within 5 % of the true optimum, thereby avoiding wasted trials.

   Relating below task with CherryPick paper:

- **Recurring Pipelines**: If TFRecord-based data processing pipeline is run regularly, then investing in CherryPick's few initial trials is justified, as the optimal configuration would be reused across many runs.
- **Moderate Search Space**: In configuration space comprising cluster size, CPU count, disk type, partitions, and batch parameters is relatively moderate, making it ideal for Bayesian Optimization techniques like CherryPick.
- **Scalar, Noisy Objective**: Throughput and write time are easily measurable but subject to variability in cloud environments. CherryPick's probabilistic modeling is particularly effective in handling this noise and making robust decisions.

Configuration	Manual	Cherrypick BO
Cluster size	1, 4, 8-core setups and compared runtimes	Models as part of vector
Disk type	separately	Uses pre-defined acquisition function
partitions	16, 32 partitions	Gaussian Process handles continuous or discrete features

Limitations

- **One-off Runs**: For all tasks, manual tuning done in tasks that may be more practical due to the overhead of setting up the BO process.
- **High-Dimensional Configuration Spaces**: If the system included many additional Spark parameters, CherryPick modelling may require dimensionality reduction or strong priors to remain effective.
- **Provisioning Time Constraints**: If cluster spin-up times are long (>10 minutes), the number of feasible BO iterations is reduced, which may diminish the optimization benefit.

Overall, The techniques from the CherryPick paper are relevant to the tasks, particularly in optimizing data retrieval, parallel processing, and addressing bottlenecks. Experiments with TFRecord files and parallelized Spark processing validated the paper's strategies for improving throughput and reducing latency in cloud environments. However, challenges such as cloud storage bottlenecks and data transfer latencies still exist. These insights highlight the importance of optimizing data formats, leveraging parallel processing, and understanding cloud infrastructure constraints to achieve efficient data processing.

3b) Strategise

In this section, discussing **batch** and **stream processing** strategies and relationship with the concepts from our previous tasks.

Batch Processing

Batch processing involves collecting a large amount of data over a period and processing it in chunks. This is efficient for real-time processing not required. Batch processing can be applied to scenarios like **training models**, **data aggregation**, **image classification tasks** where the system is designed to handle large datasets in a single processing cycle.

Essentials

1. **Data Size**: Batch processing's effectiveness depends on batch size. Using larger batches in task reduces processing time. A larger **batch size** allows more images to be processed together but it can also lead to higher memory usage.
2. **System Resource Allocation**: **Parallelization** and **caching** in Spark-based environments can help improve the efficiency of batch processing. **Caching** reduces redundant computation, especially when performing the same operations on different subsets of data.

Strategy:

- **Use TFRecord Files**: Since TFRecord files were shown to provide better performance in throughput compared to raw image files, using them for batch processing in image classification tasks would be beneficial. By reading large volumes of pre-processed image data in batch, we can achieve better efficiency.
- **Optimize Batch Size**: Based on the performance from tasks, fine-tuning the **batch size** and **number of batches** is crucial to finding the optimal throughput for system.

- **Parallel Processing:** In distributed environment like Google Dataproc, Spark enables parallel batch job execution across multiple nodes. This division of workload, highlighted in Tasks, enhances throughput.

Stream Processing

Stream processing involves continuous data input and processing in real time. This strategy is ideal for applications that require **real-time insights**, such as **live image recognition**, **sensor data processing**. It requires the system to process data continuously and with minimal delay.

Essentials:

- **Latency**: D**ata locality** becomes crucial in stream processing, especially when dealing with cloud environments. Data needs to be processed as it is received, without waiting for a large batch of data to accumulate. This requirement aligns with the **real-time insights** discussed in the CherryPick paper.
- **Data Flow**: Involves handling a **continuous flow of data**. To process real-time data **AWS Lambda**, **Google Cloud Functions** are use and **Apache Kafka** for handling stream data.

Strategy:

- **Event-Driven Architecture**: Implementing an **event-driven architecture** where each image processed triggers an event to the processing system ensures real-time processing.
- **Data Preprocessing**: These systems require data to be processed **immediately**. Preprocessing should be integrated into the pipeline using tools like **Apache Kafka**, ensuring minimal delay.
- **Optimizing Throughput**: To prevent bottlenecks, ensure that the streaming system is capable of handling small batches or chunks of data at a time. For example, using **TFRecord files**

Relationship

1. **Performance**:
    - Both **batch and stream processing** benefit from optimized data formats. **TFRecord** files were found to outperform raw image files in terms of throughput, which is a critical aspect in both batch and stream processing.
    - In **stream processing**, minimizing latency is key, and **real-time data processing** using smaller batches or chunks, it ensures low-latency responses.
2. **Parallelization and Scalability**:
    - The **parallelization** techniques from tasks demonstrate how batch processing can be made more efficient by splitting the task into smaller chunks and distributing them across multiple nodes.
    - In **stream processing**, scalability technique such as **horizontal scaling** as discussed in the CherryPick paper would be necessary to handle increasing data flow.
3. **Cloud-Based Systems**:
    - Cloud services like **AWS Lambda**, **Google Cloud**, and **Dataproc** used for both. These services are ideal for handling the scalability and dynamic resource allocation needed for large data processing.

Based on previous findings and CherryPick principles, optimizing batch sizes, leveraging cloud resources, and using TFRecord formats ensure efficient processing for batch and stream scenarios.

Number of words: 1999