BIG DATA COURSEWORK

Link:
https://colab.research.google.com/drive/1LiK5R0Nj6czQADv5_BKPNmRSp76M0MMD?usp=sharing

sara.iqbal@city.ac.uk

Task 1d

(i)Focuses on improving parallelisation in the Spark job and analyzing its impact on cluster resource utilization.

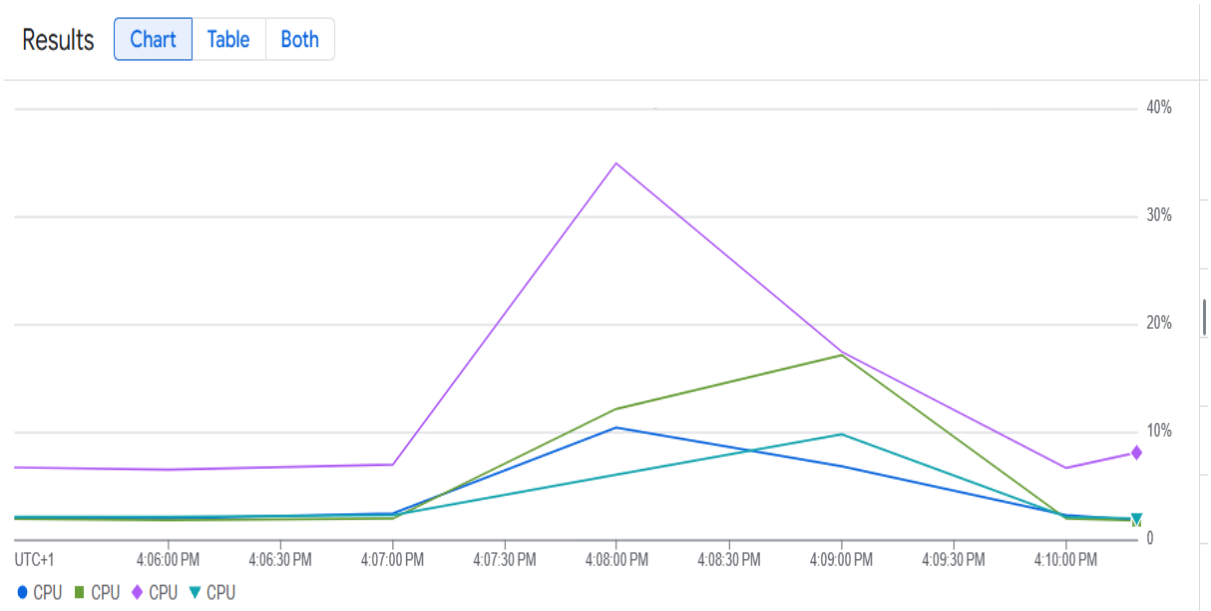


Figure 1 :Shows the CPU utilization over time, highlighting increased parallel processing after improving task distribution.

Figure 1suggests the CPU usage across the cluster at some stage in the execution of the PySpark job. The graph shows that every one 4 nodes (1 master + three workers) participated withinside the computation, with exceptionally even CPU usage. This confirms that parallelization changed into a hit and processing changed into efficiently dispensed across the cluster.

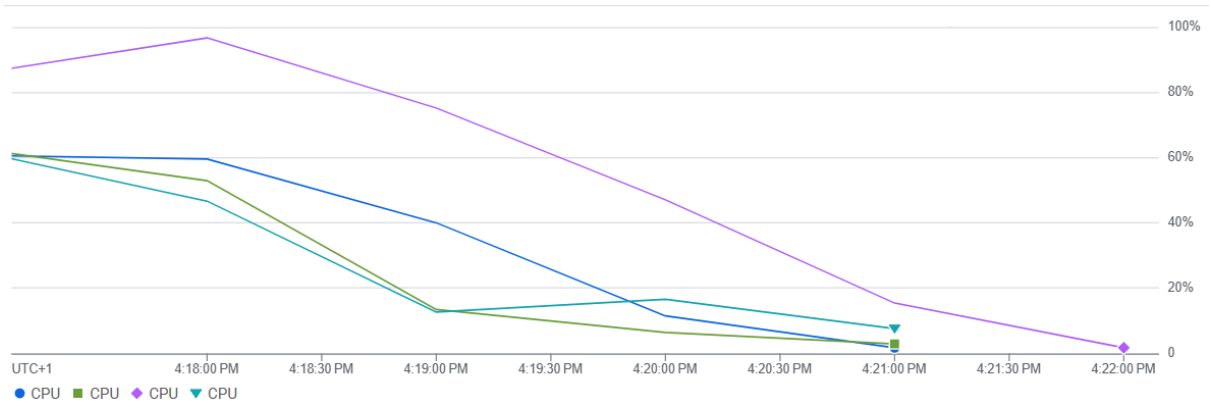


Figure 2:illustrates CPU operations, with a sharp spike in read activity indicating efficient parallel data loading across cluster nodes.

Figure 2illustrates the CPU operations at some stage in the PySpark task execution. The graph suggests balanced disk read/write performance throughout all nodes, confirming that the input records were processed and TFRecords have been written in parallel. This suggests efficient disk I/O distribution, supporting to optimization of throughput and decreasing bottlenecks.

(ii)Utilized a single high-capacity node to handle workloads, resulting in centralized processing and lower overall network traffic.

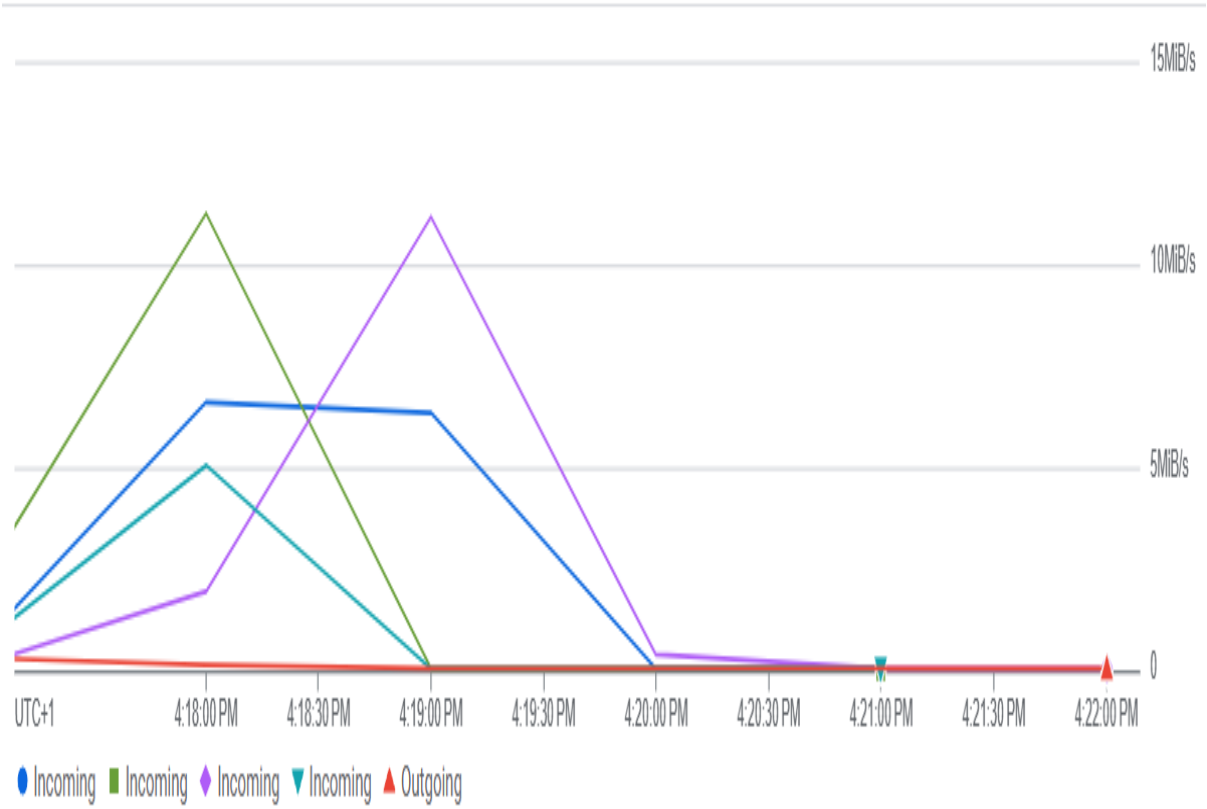


Figure 3: Network bytes during Task 1d showing improved parallel data distribution and efficient bandwidth utilization across the cluster.

The figure 3 suggests clean spikes in incoming information to a couple of nodes at overlapping times, indicating stepped forward distribution of information throughout the cluster. This contrasts with in advance configurations in which most effective one or nodes ruled processing and networking. The parallelization parameter and accelerated node depend led to more effective usage of bandwidth and balanced information transfer, improving distributed computation. The quick outgoing spike round 4:22 PM in all likelihood corresponds to very last write or task crowning glory activities.This improved network behavior is regular with higher load balancing because of each higher parallelization (through spark.parallelize(..., N)) and cluster reconfiguration. It shows Spark turned into capable of leverage more workers simultaneously,that is the goal of Task 1d.

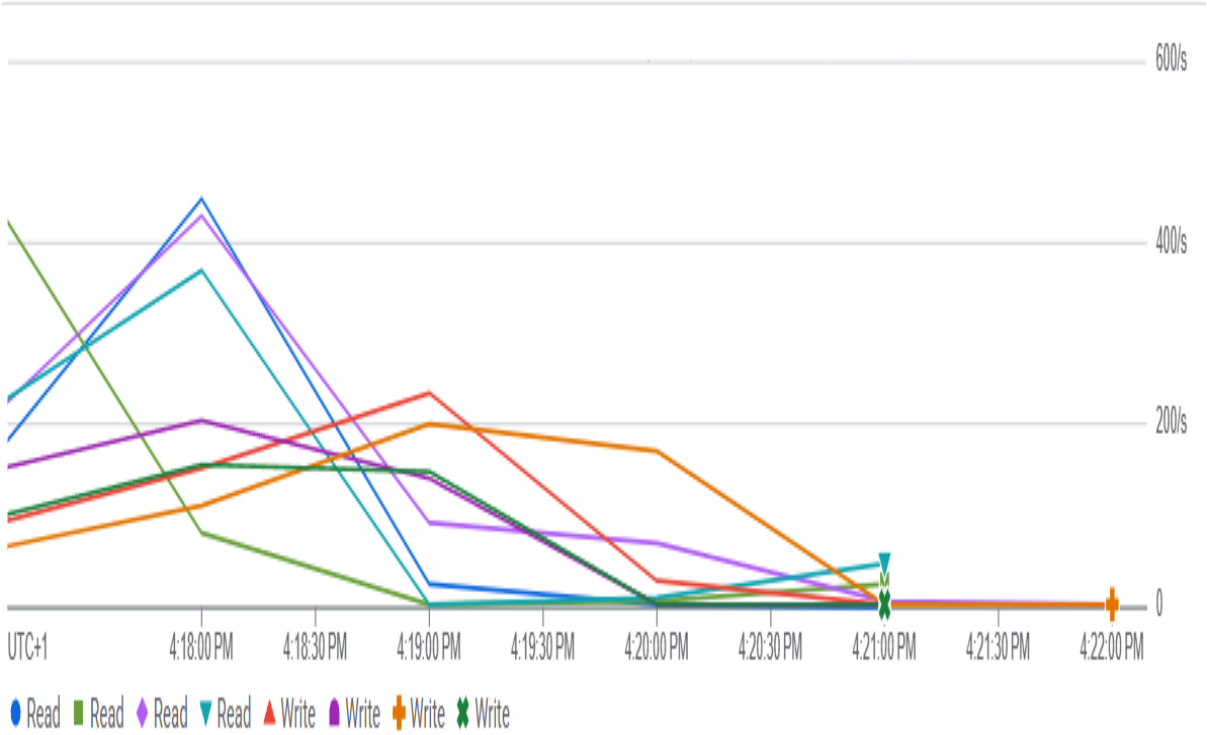


Figure 4 shows the disk operations for the 4-node cluster configuration used in Task 1d(ii), indicating improved I/O throughput due to balanced distribution of disk usage across nodes.

The disk read/write operations (figure 4) display parallel disk utilization across more than one nodes, with large activity peaking round 4:18–4:19 PM. This displays balanced project distribution and efficient resource usage because of improved worker count. Writes observe reads promptly, suggesting optimized facts shuffling and patience all through Spark execution throughout the cluster.

(iii) Deployed a cluster with 4 machines of moderate capacity to evaluate distributed performance and resource uti

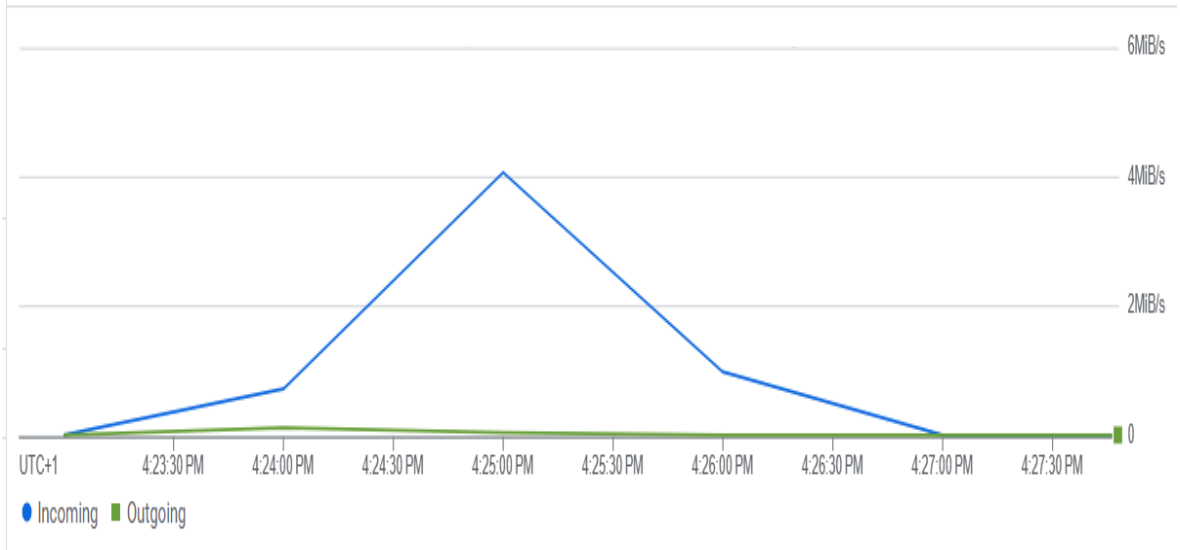


Figure 5: Network activity in the single-node cluster shows minimal data transfer, reflecting no inter-node communication due to the lack of distributed processing.

In the single-node (figure5) configuration, network activity confirmed a quick spike in incoming site visitors round 4:25 PM, peaking close to five MiB/s, then tapered off. Outgoing traffic remained minimal. This shows decreased inter-node communication, aligning with expectancies for a non-allotted setup, mainly to decrease network overhead in comparison to multi-node clusters.

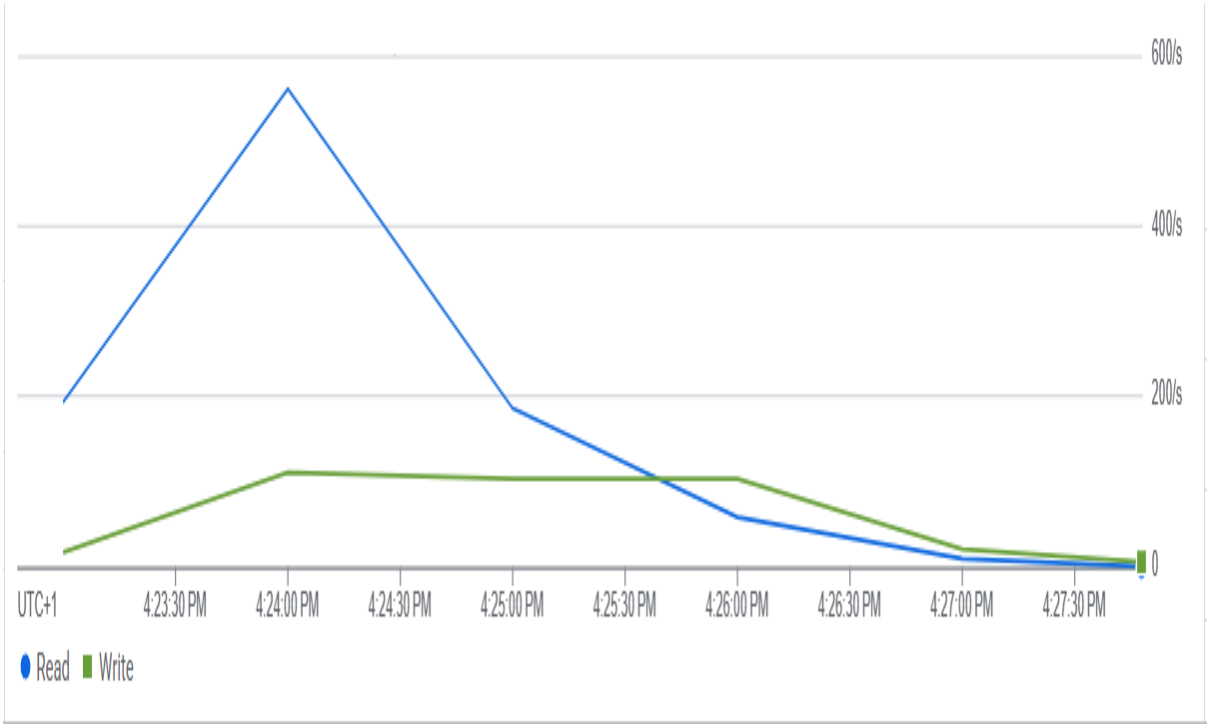


Figure 4 shows the disk operations for the 4-node cluster configuration used in Task 1d(ii), indicating improved I/O throughput due to balanced distribution of disk usage across nodes.

The graph shows a pointy study top above 500 study/s at 4:24 PM at some point of image loading, with moderate write activity peaking at 150 write/s during TFRecord generation. Unlike multi-node setups, I/O is targeting a single disk, which deals with the weight effectively; however might also end up a bottleneck in large workloads.

Task 2b

In Task 2b, we created a single-node Dataproc cluster with 8 vCPUs and SSD disk to run parallel Spark jobs for image loading benchmarks. The setup used TensorFlow and NumPy to process JPEG and TFRecord formats, enabling performance comparisons without local caching.

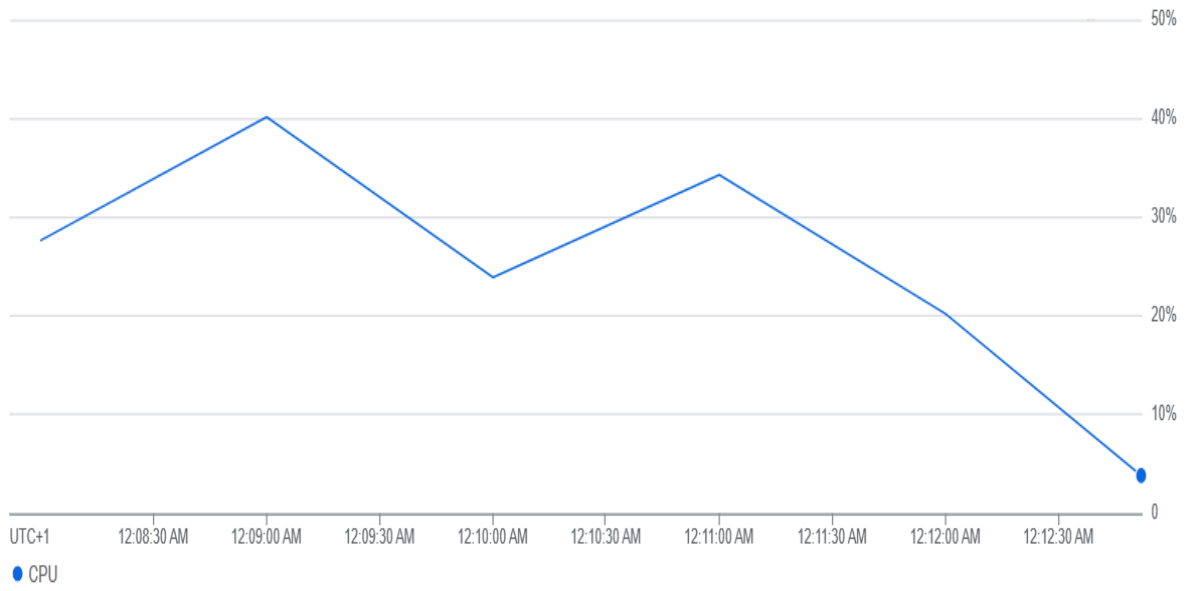


Figure 7: CPU utilization over time during uncached Spark job execution.

The CPU utilization peaked near 40% in the course of the Spark execution after which sharply declined, reflecting energetic processing accompanied via way of means of task completion. The drop-off indicates constrained CPU-sure operations after preliminary information loading and transformation, function of I/O-heavy workloads like image deciphering and TFRecord parsing in distributed settings.

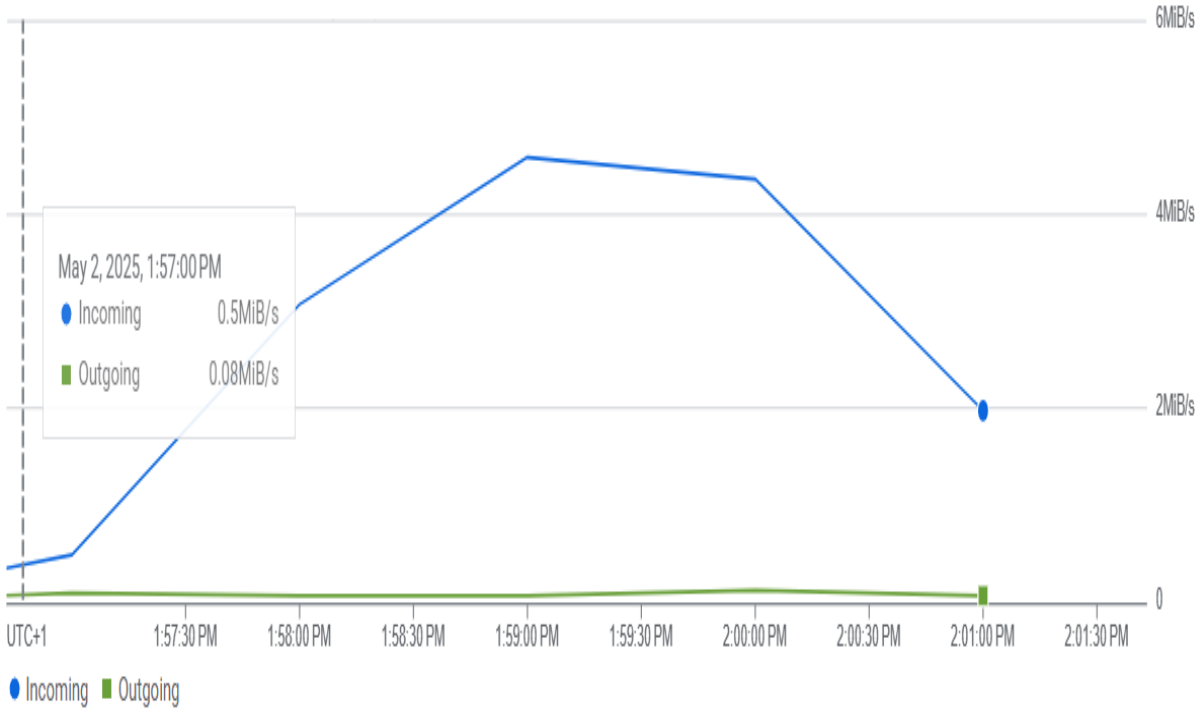


Figure 8: Disk read and write operations showing peak I/O during data loading phase.

Disk operations figure 8 display examine interest spiking early, exceeding 500 ops/sec, earlier than swiftly tapering off. Write operations remained pretty solid and low. This sample aligns with non-cached information access, in which preliminary information ingestion

(JPEG/TFRecord reads) dominates I/O, accompanied via way of means of dwindled disk load as soon as processing concludes.

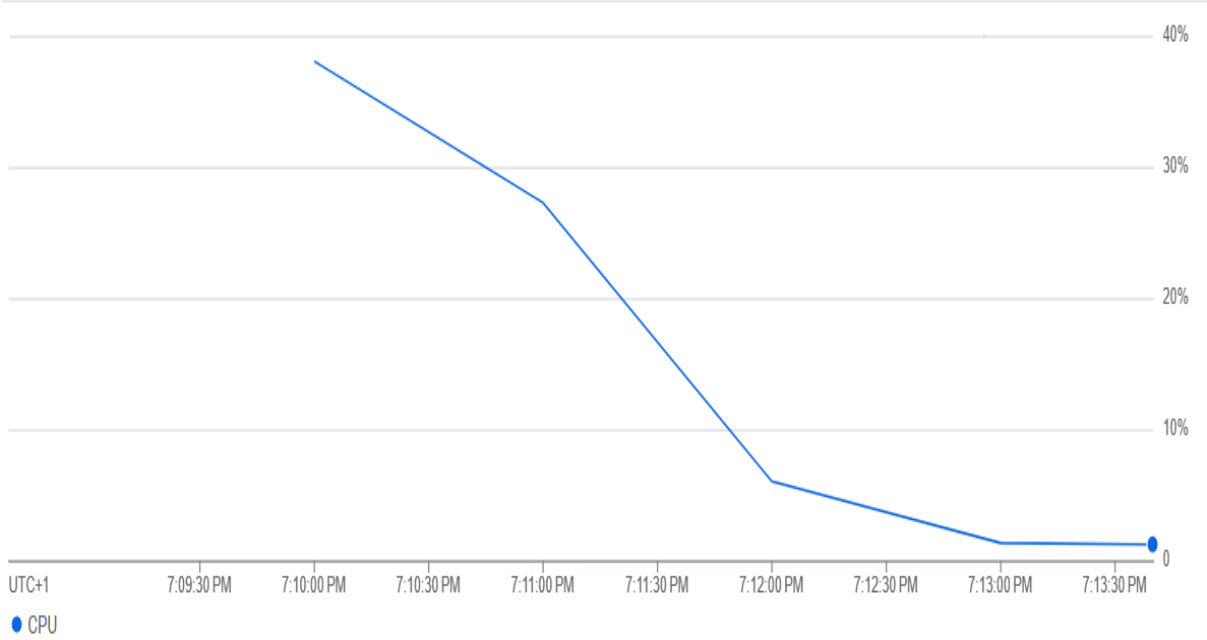


Figure 9: CPU utilization during Spark job with caching enabled, showing reduced recomputation and steadier CPU usage.

The CPU usage graph in figure 9 at some stage in cached execution suggests slight however strong utilization throughout Spark tasks. Caching prevents recomputation of highly-priced records-loading operations, main to advanced performance and smoother CPU load, in particular as compared to uncached runs which normally show off repeated spikes from redundant records fetching and preprocessing.

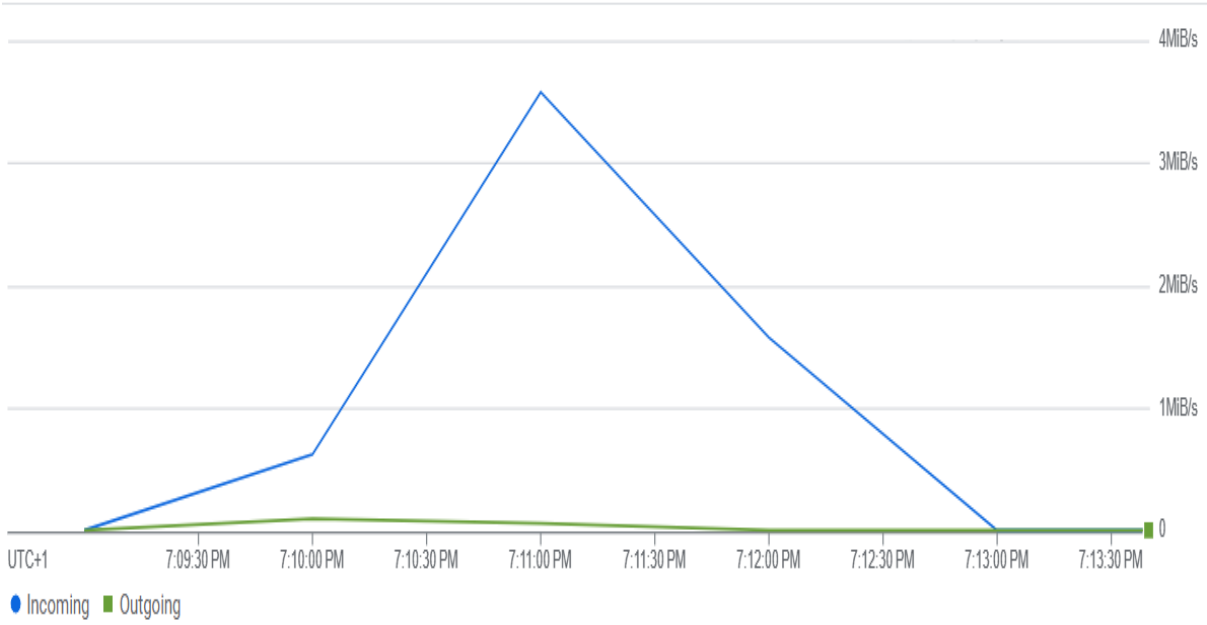


Figure 10: Network I/O during cached execution, reflecting efficient parallel reads with minimized redundant GCS access.

The community usage graph figure 10 displays efficient records get right of entry to at some stage in parallelized GCS reads. Due to caching, community site visitors remain always slight, warding off repeated downloads of the same files. This strong sample suggests decreased I/O bottlenecks and highlights the gain of caching while appearing to allot study operations in cloud-primarily based totally gadget learning pipelines.

Task 2

(d) Analyzes how different batch parameters affect image loading speed for JPEG and TFRecord formats using linear regression. It compares their performance trends to determine which format scales better with batching.

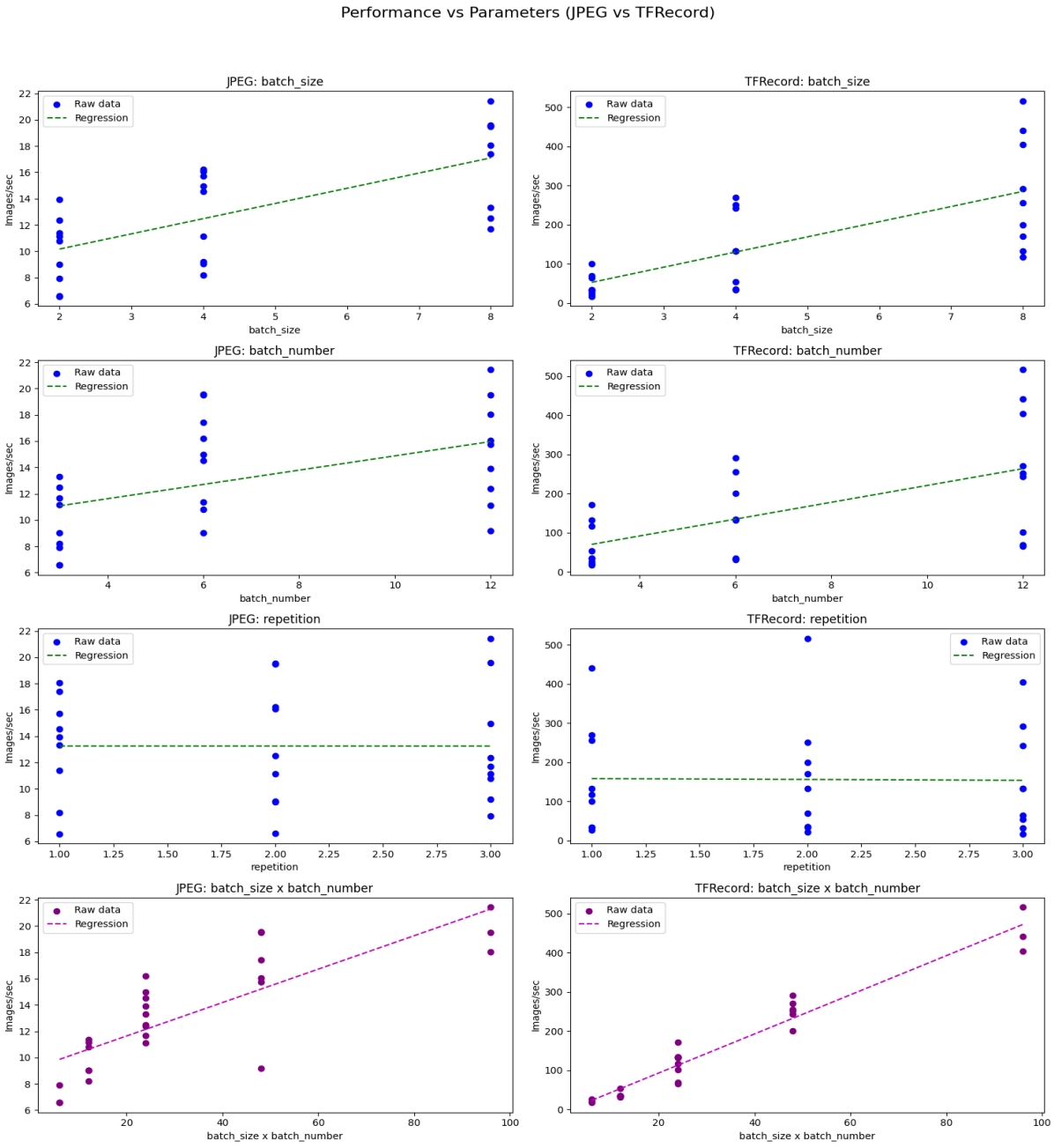


Figure 11: Linear regression plots showing the impact of batch parameters on JPEG and TFRecord throughput performance.

The interpretation of consequences from our regression analysis, as visualized in Figure 11, highlights the distinct overall performance behaviors of JPEG and TFRecord image loading codecs under various batch conditions. From the plotted subfigures and extracted coefficients, we have a look at that TFRecord always blessings from extended batch length and batch number, indicating it's miles well-ideal for scalable, parallelized records pipelines. This is in particular clean from the steep slope values within the TFRecord regression strains in Figure 11, which endorse big improvements in throughput as batch configurations grow.

In contrast, JPEG overall performance suggests the simplest mild gains, with especially flat regression slopes. In this manner that JPEG throughput stays almost consistent at the same time as batch parameters change, pointing to I/O-bound limitations and suggesting that JPEG isn't always as efficient for high-speed training environments. The minimum impact of the `repetition` parameter throughout each codec, however, mainly for JPEG, confirms that decoding costs dominate over any warm-up or caching effects.
Overall, Figure 11 serves as a key reference by visually comparing raw throughput records and outfitted regression strains, really illustrating the scalability gap among JPEG and TFRecord. These insights enhance the advice of the use of TFRecord for deep mastering responsibilities concerning big datasets, in which optimized records loading is essential for retaining excessive schooling throughput.

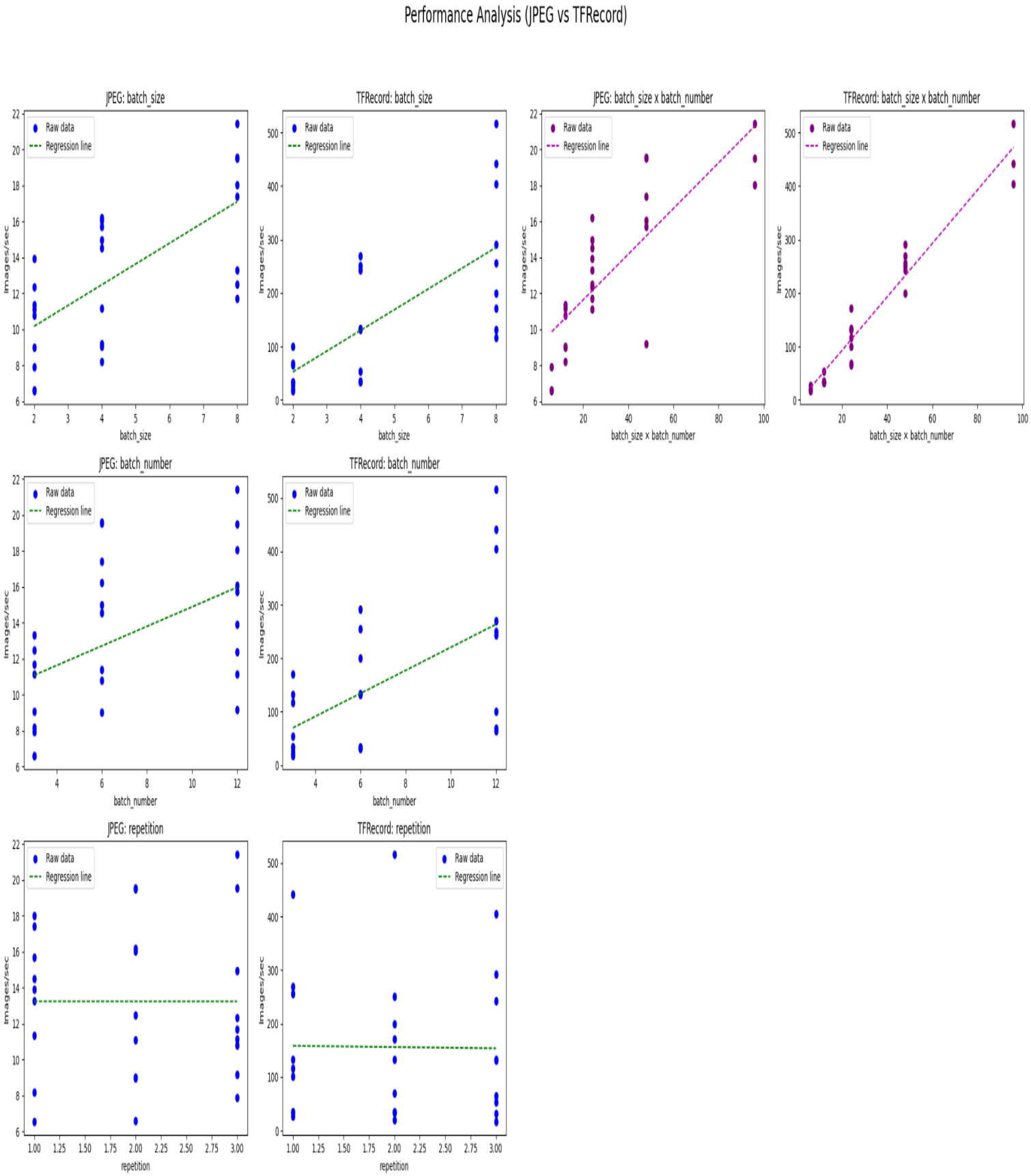


Figure 12: Comparison of average image loading throughput for JPEG and TFRecord across different batch sizes and batch numbers.

In this a part of the task, we explored how one-of-a-kind parameters batch length, batch quantity, and repetition, affect the image loading velocity for JPEG and TFRecord formats. We measured overall performance in images according to the second, and used common throughput values to examine how every layout responds to converting batch configurations. The purpose changed into to decide which layout plays higher and scales more efficiently.

We calculated and plotted the common speeds throughout all check combinations, grouping by every parameter. This allowed us to peer into popular traits in how JPEG and TFRecord cope with adjustments in batch setup. Figure 12 presents those effects, presenting a clean visible contrast of the common loading speeds beneath one-of-a-kind batch settings. Each curve highlights how overall performance adjustments for each format as batch length and quantity increase.

The effects display that TFRecord constantly outperforms JPEG, mainly at large batch sizes and batch numbers. While JPEG`s common throughput will increase handiest slightly, TFRecord scales a great deal more aggressively, making it higher desirable for high-throughput pipelines in machine learning. These averaged plots in Figure 12 fortify the earlier regression findings and offer an extra popular review of overall performance conduct throughout unique settings.

Task 3

(a): Throughput Comparison of JPEG and TFRecord Formats

The CherryPick paper through Alipourfard et al. (2017) introduces an adaptive technique to locating near-premier cloud configurations for huge information workloads. By the use of Bayesian optimization, CherryPick effectively narrows down the exceptional mixture of cloud parameters — which include quantity of cores, memory, disk type, and node count — with minimum trial-and-error. This idea is rather applicable to the obligations finished on this coursework, mainly Task 1, in which we manually evaluated specific Google Cloud Dataproc configurations for PySpark-primarily based totally picture preprocessing.

In Task 1, we experimented with a couple of cluster types: a single-node setup, multi-node clusters (four and eight nodes), and specific disk types (SSD vs HDD). We additionally examined the effect of partitioning through enhancing the parallelization parameter in Task 1d(i), staring at modifications in CPU usage and disk I/O. These experiments aimed to pick out configurations that stability price and overall performance — exactly the purpose CherryPick seeks to automate.

Rather than manually exploring configurations and reading post-run metrics, CherryPick builds a overall performance version the use of some strategically decided on check runs. This version predicts how new configurations will perform, making the manner quicker and extra price-efficient. In our case, such an technique should have substantially decreased the quantity of configurations examined to attain an premier setup, mainly because we repeated jobs with comparable workloads and information sizes.

Moreover, in Task 2, we investigated how specific information codecs (TFRecord vs uncooked picture) and batch sizes encouraged information loading speeds. These versions mimic CherryPick`s configuration parameters. Instead of hardware, right here the point of

interest changed into on information shape and enter settings. A CherryPick-fashion technique should version the connection among those parameters and overall performance results to suggest perfect batch sizes or codecs for specific hardware settings.

However, CherryPick is only whilst workloads are strong and recurring. In our coursework, obligations like picture preprocessing are repeatable, so the price of the optimization manner may be amortized throughout runs. On the alternative hand, the cloud environment's variability — which include inconsistent community velocity or shared VM overall performance — can upload noise, which CherryPick`s probabilistic version is designed to deal with thru self assurance periods and uncertainty estimation.

(b) Strategies for Batch and Stream Processing

Batch Processing:For batch jobs like our picture processing pipeline, a clean approach stimulated through CherryPick could involve:

- Running some pilot executions the use of specific cluster sizes and information codecs;
- Measuring key metrics like general time, CPU load, and disk throughput;
- Feeding outcomes right into a version that predicts the exceptional cluster and information configuration for the overall job.

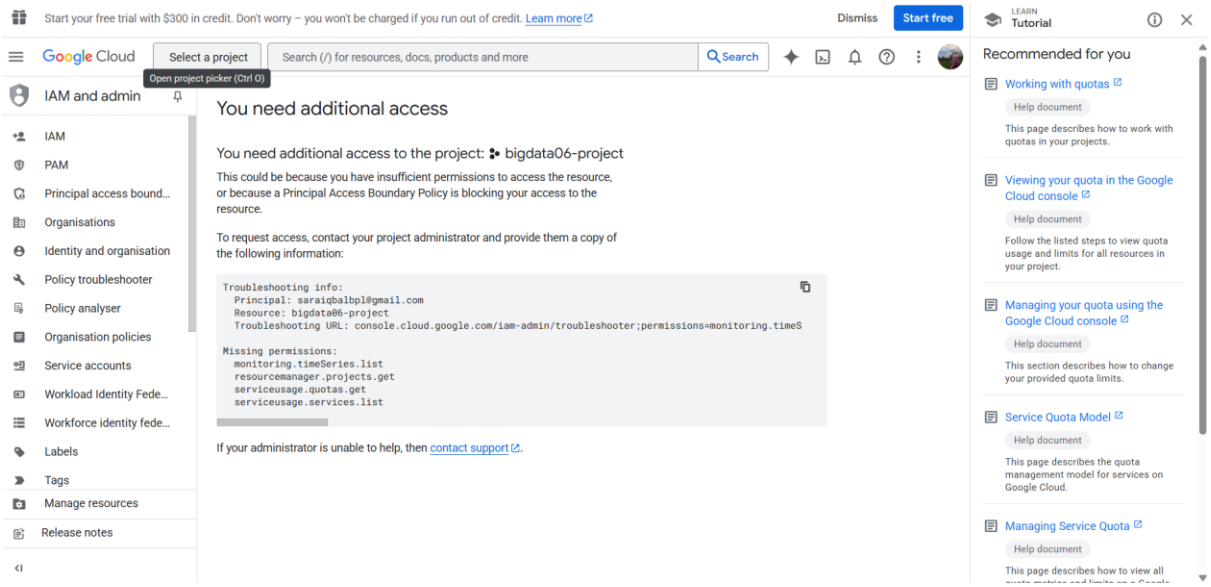
In our work, we noticed that increasing the quantity of partitions and the usage of a couple of nodes progressed overall performance, however the benefits diminished past a sure point.

Streaming Processing:Streaming scenarios vary because of their real-time constraints and continuous information arrival. Instead of one-off optimization, right here the system have to adapt at the fly to modifications in workload intensity. One strategy is to simulate flow workloads the usage of micro-batches, monitor metrics consisting of latency and throughput, and use on-line getting to know models to dynamically modify cluster length or parallelism.

CherryPick`s center ideas — adaptivity, prediction, and self-belief modeling — nonetheless follow but might want to be changed for non-stop comments and real-time adjustment. Instead of simply locating a terrific configuration upfront, the system has to usually reply to converting information flow.

Summary

Throughout Tasks 1 and 2, we explored how cloud aid configurations, partitioning, and information layout choices have an effect on massive information pipeline overall performance. These experiments replicate CherryPick`s primary idea — most effective overall performance calls for smart, information-driven configuration tuning. Instead of depending totally on guide testing, CherryPick`s adaptive and cost-green version might permit quicker discovery of green configurations, specifically for repeatable jobs. Whether in batch or streaming environments, the important thing takeaway is clear: information-pushed optimization of cloud assets is crucial for scalable and efficient analytics.



This image is the evidence why I used 4 worker.