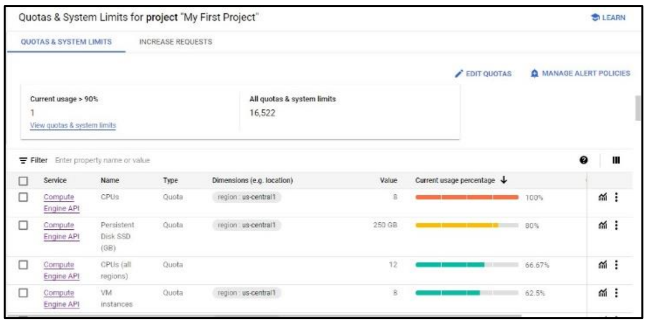
BIG DATA REPORT

Fatema Fidvi (Student ID – 230024865)

Colab link: https://colab.research.google.com/drive/1dX1M4aA4PEr5T-97HxUE3V2jFCoRbavs?usp=sharing

I. SECTION 1

➢ 1c ii) Setting up maximal cluster



Quotas & System Limits for **project** "My First Project"

QUOTAS & SYSTEM LIMITS | INCREASE REQUESTS

Current usage > 90%: 1 — View quotas & system limits

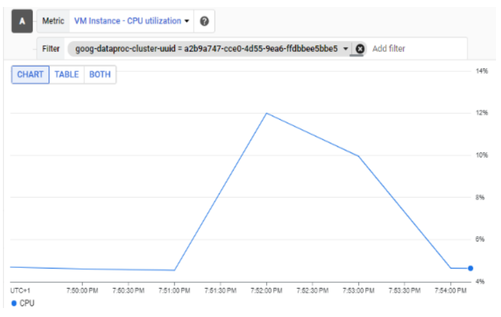
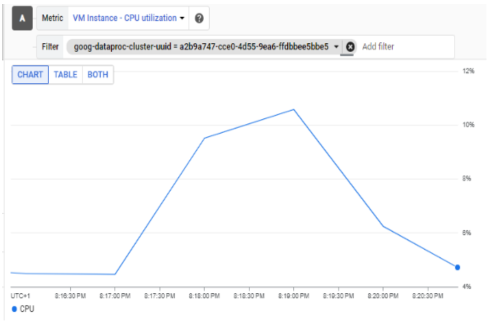
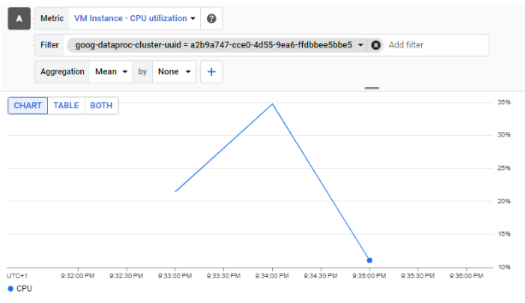
All quotas & system limits: 16,522

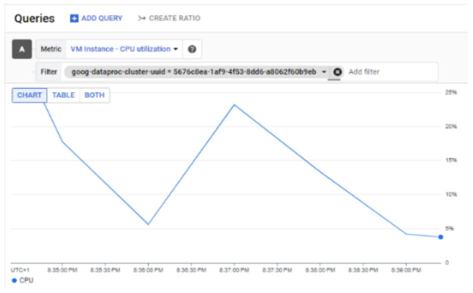
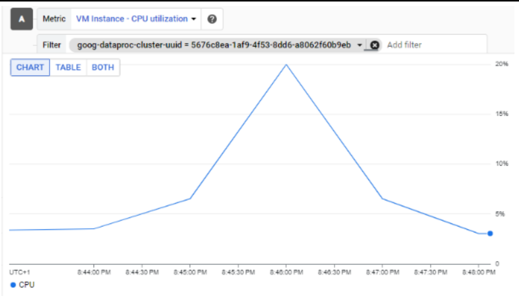
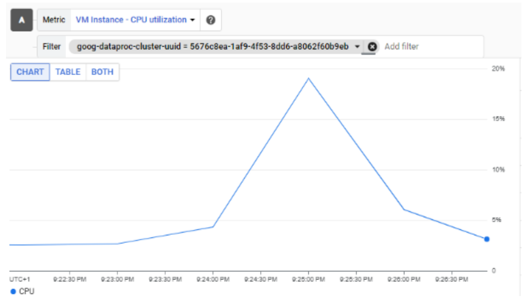
Service	Name	Type	Dimensions (e.g. location)	Value	Current usage percentage
Compute Engine API	CPUs	Quota	region : us-central1	8	100%
Compute Engine API	Persistent Disk SSD (GB)	Quota	region : us-central1	250 GB	80%
Compute Engine API	CPUs (all regions)	Quota		12	66.67%
Compute Engine API	VM instances	Quota	region : us-central1	8	62.5%

While setting up a maximal cluster for experiment 1c ii, a quota error occurred despite attempts to resolve it by deleting existing clusters and trying different regions. This issue, documented in the Colab notebook and shown in a screenshot, led to the creation of the maximal cluster with 3 worker nodes and 1 master node due to quota constraints.

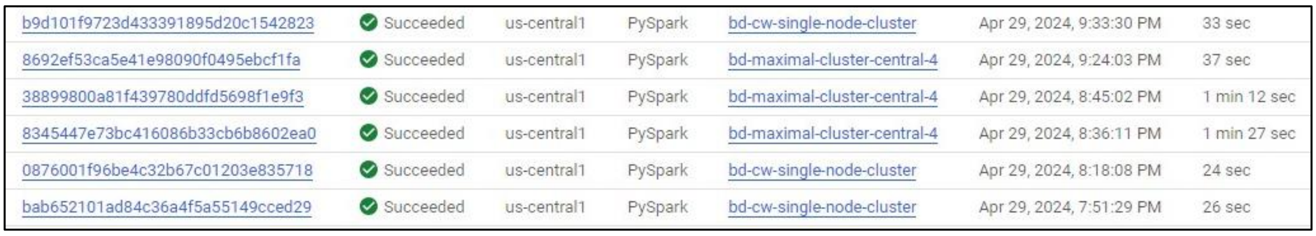
➢ 1d i) Improve Parallelization

The choice of 18 slices aimed to enhance task distribution across the maximal cluster's nodes, optimizing core usage with 2- 4 tasks per core, given its multi-core setup. Increasing the slices to 40 was intended to test further parallelism enhancements, targeting reduced job times by leveraging the full computational power of the cluster. The strategy showed that fine-tuning 'numSlices' is crucial in Spark jobs for achieving peak performance and efficiency. Below is the table showing the performance of single node cluster and maximal number of clusters with and without parallelization with different number of slices.

SINGLE NODE CLUSTER	GRAPH OF CPU UTILIZATION	Peak CPU Utilization & Elapsed Time	EXPLANATION
WITHOUT PARALLELIZATION (Job id – bab652101ad84c36a4f5a55149cced29)		CPU utilized = 12% Time = 26 sec	**numSlices = 2** (default) The minimal CPU utilization reflects the limited parallel processing capability of the single-node configuration, resulting in longer task execution times due to sequential processing within the two default slices
WITH PARALLELIZATION (Job id – 0876001f96be4c32b67c01203e835718)		CPU utilized = 10.5% Time = 24 sec	**NumSlices =18:** shows only a slight improvement in execution efficiency, highlighting the limitations of a single node to capitalize fully on added parallelism. The reduced CPU utilization indicates that the node may be reaching its capacity to handle parallel tasks effectively, leading to diminishing returns despite the increased partitioning.
WITH PARALLELIZATION (Job id – b9d101f9723d433391895d20c1542823)		CPU utilized = 35 % Time = 33 sec	**numSlices = 40:** resulted in higher CPU utilization and longer execution times, indicating that excessive partitioning on a single node can introduce significant management overhead, which detracts from overall performance efficiency.

MAXIMAL NODE CLUSTER (3 +1)	GRAPH OF CPU UTILIZATION	Peak CPU Utilization & Elapsed Time	DESCRIPTION
WITHOUT PARALLELIZATION (Job id – 8345447e73bc416086b33cb6b8602ea0)		CPU utilized = 23% Time = 87 sec	**numSlices = 2** (default): increased CPU utilization and longer execution times compared to a single-node setup indicate that limited parallelism prevents effective use of the expanded resources, resulting in inefficient performance.
WITH PARALLELIZATION (Job id – 38899800a81f439780ddfd5698f1e9f3)		CPU utilized = 20% Time = 72 sec	**NumSlices =18:** there was an improvement in execution time and a slight reduction in peak CPU utilization. This configuration suggests a better task distribution across the cluster's nodes, yet it still falls short of fully optimizing the available resources
WITH PARALLELIZATION (Job id – 8692ef53ca5e41e98090f0495ebcf1fa)		CPU utilized = 20% Time = 37 sec	**numSlices = 40:** execution time dramatically decreased, highlighting a much more efficient use of the cluster. This setup shows a substantial enhancement in task distribution across the nodes, achieving improved throughput while maintaining moderate CPU utilization, which underscores the effective optimization of available resources

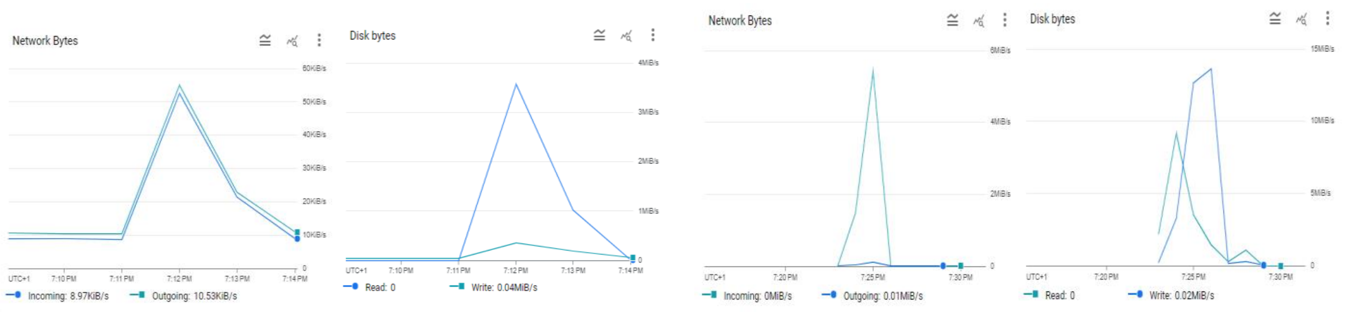
This is a screenshot of the timings each job took along with their job ids and names of single or maximal cluster used.



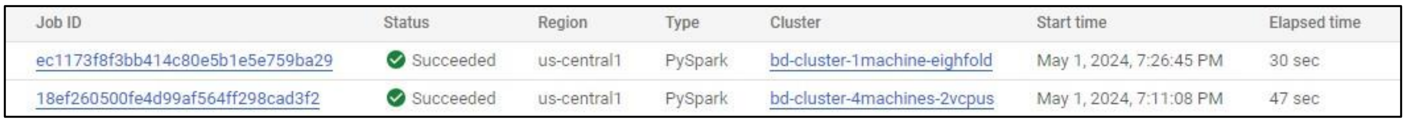
b9d101f9723d433391895d20c1542823	✓ Succeeded	us-central1	PySpark	bd-cw-single-node-cluster	Apr 29, 2024, 9:33:30 PM	33 sec
8692ef53ca5e41e98090f0495ebcf1fa	✓ Succeeded	us-central1	PySpark	bd-maximal-cluster-central-4	Apr 29, 2024, 9:24:03 PM	37 sec
38899800a81f439780ddfd5698f1e9f3	✓ Succeeded	us-central1	PySpark	bd-maximal-cluster-central-4	Apr 29, 2024, 8:45:02 PM	1 min 12 sec
8345447e73bc416086b33cb6b8602ea0	✓ Succeeded	us-central1	PySpark	bd-maximal-cluster-central-4	Apr 29, 2024, 8:36:11 PM	1 min 27 sec
0876001f96be4c32b67c01203e835718	✓ Succeeded	us-central1	PySpark	bd-cw-single-node-cluster	Apr 29, 2024, 8:18:08 PM	24 sec
bab652101ad84c36a4f5a55149cced29	✓ Succeeded	us-central1	PySpark	bd-cw-single-node-cluster	Apr 29, 2024, 7:51:29 PM	26 sec

➢ **1d ii) Experimenting with 4 machines with double the resources and 1 machine with 250 SSD. Discuss the results in terms of disk I/O and network bandwidth allocation in the cloud.**

Experimenting with different cluster configurations revealed insights into disk I/O and network bandwidth allocation in the cloud. The first setup, with four machines having double the resources, showed potential network underutilization and disk write inefficiencies. Despite ample resources, network bandwidth peaked modestly at 55 KiB/s, while disk write operations lagged behind with a peak of only 0.2 MiB/s. Conversely, the second setup, featuring a single machine with limited SSDs (250) due to quota constraints, demonstrated efficient performance. Network bandwidth peaked at 6 MiB/s, indicating effective utilization, while disk I/O operations were notably efficient, with peak read and write rates of 10 MiB/s and 14 MiB/s, respectively. These findings underscore the importance of resource optimization and efficient allocation in maximizing cloud performance. The job ids of the two clusters are shown down in the screenshot, where the first cluster took 47 seconds and the second cluster took just 30 seconds to run.



Network and Disk bytes from Running on 4 machines with double resources (left) & on single machine and 250 SSDs (right)



Job ID	Status	Region	Type	Cluster	Start time	Elapsed time
ec1173f8f3bb414c80e5b1e5e759ba29	Succeeded	us-central1	PySpark	bd-cluster-1machine-eighfold	May 1, 2024, 7:26:45 PM	30 sec
18ef260500fe4d99af564ff298cad3f2	Succeeded	us-central1	PySpark	bd-cluster-4machines-2vcpus	May 1, 2024, 7:11:08 PM	47 sec

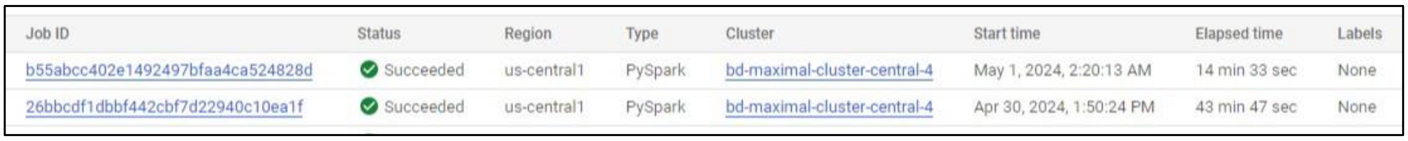
- **1d iii) Explain the difference between this use of Spark and most standard applications like e.g. in our labs in terms of where the data is stored. What kind of parallelisation approach is used here?**

In the labs, standard applications often store data on the local file systems or Hadoop Distributed File System (HDFS). Additionally, databases like MongoDB and cloud services such as Azure and Amazon S3 are generally used for structured and unstructured data storage. While these systems are capable, they generally lack optimal integration with Spark for distributed computing.
In contrast, our Spark application utilizes Google Cloud Storage (GCS), a cloud-based distributed object storage service. GCS enhances accessibility and scalability by managing data across a globally distributed infrastructure. This enables more effective data partitioning and parallel processing with Spark across multiple nodes, which is not as seamlessly achievable with MongoDB, Azure, or Amazon S3. Thus, GCS allows our Spark setup to handle larger datasets and scale more efficiently than the storage solutions typically used in our labs.
In this Spark application, we employ a data-parallelism approach by using the 'numSlices' parameter with the 'parallelize' method. This approach partitions data into multiple slices that are processed concurrently across different nodes in the cluster, optimizing data distribution and workload balance. This method not only improves processing speeds but also enhances fault tolerance and scalability, making it ideal for large-scale data processing in distributed computing environments.

II. <u>SECTION 2</u>

- **2c: Explain in your report what the reasons for this change are and demonstrate and interpret its effect**

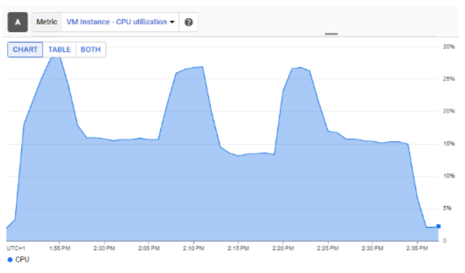
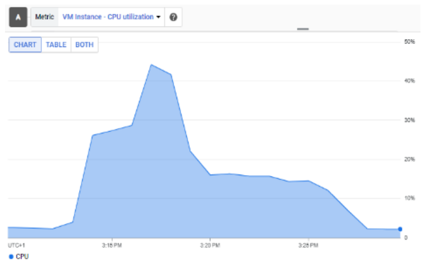
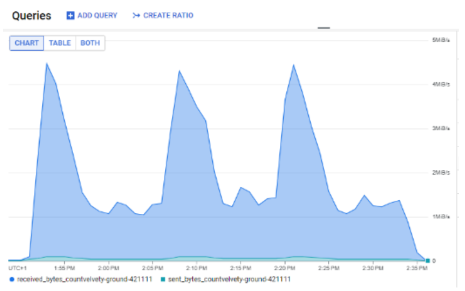
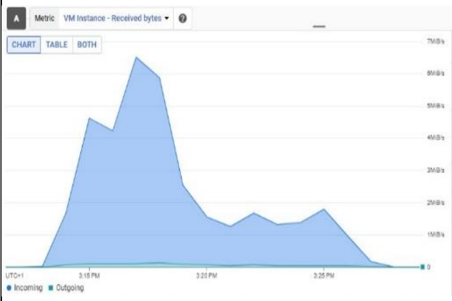
In the Spark job code for 2a, caching was implemented in 2c to enhance performance by preventing the repeated computation of the same RDD (results_rdd). This RDD is utilized multiple times for collecting performance metrics, aggregating data, and calculating averages, each of which would otherwise trigger a complete re-evaluation of the RDD from the beginning, including expensive read and map operations. Caching `results_rdd` immediately after its creation and before any operations like `collect()` optimizes the process by ensuring that heavy computations are done just once and subsequent operations access this precomputed data quickly, enhancing overall efficiency.



Job ID	Status	Region	Type	Cluster	Start time	Elapsed time	Labels
b55abcc402e1492497bfaa4ca524828d	Succeeded	us-central1	PySpark	bd-maximal-cluster-central-4	May 1, 2024, 2:20:13 AM	14 min 33 sec	None
26bbcdf1dbbf442cbf7d22940c10ea1f	Succeeded	us-central1	PySpark	bd-maximal-cluster-central-4	Apr 30, 2024, 1:50:24 PM	43 min 47 sec	None

These are the two job clusters showing the time difference in the elapsed time before and after caching

Metric	Without Caching	With Caching	Improvement
Total Runtime	43m 47s	14m 33s	70%
Peak CPU Utilization	30% (multiple)	45% (once)	More focused
Network Load Peaks	4 MiB/s (multiple)	6.5 MiB/s (once)	More efficient

CPU Utilization graph screenshot			Indicates a shift from multiple smaller peaks to a single, higher peak (45%), suggesting that while the job may temporarily use more resources, it completes quicker, reducing total resource consumption.
Network bytes (Network load) graph screenshot			Demonstrates a similar pattern with network traffic, where a single high peak (6.5MiB/s) replaces multiple smaller peaks, indicating more efficient data handling after caching.

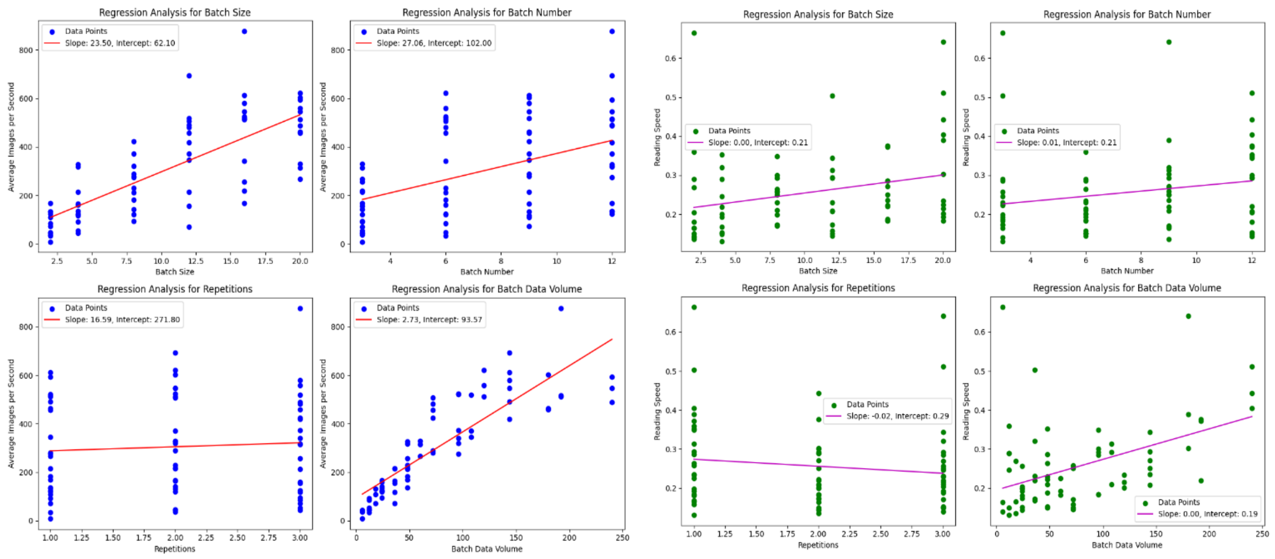
➢ **2d: Retrieve, Analyse and discuss the output**

Discuss the implications of this result for applications like large-scale machine learning. Keep in mind that cloud data may be stored in distant physical locations. How is the observed behaviour similar or different from what you'd expect from a single machine? Why would cloud providers tie throughput to capacity of disk resources? By parallelising the speed test we are making assumptions about the limits of the bucket reading speeds. Discuss, what we need to consider in speed tests in parallel on the cloud, which bottlenecks we might be identifying, and how this relates to your results. Discuss to what extent linear modelling reflects the effects we are observing. Discuss what could be expected from a theoretical perspective and what can be useful in practice.

The linear regression analysis of both the TFRecord and JPEG datasets (shown below in table format and scatter plot) reveals significant insights into their throughput and reading speed behaviors. In the TFRecord dataset, positive slopes across parameters generally indicate that higher values lead to increased throughput, while reading speed trends vary, with Repetitions showing a negative slope, potentially indicating diminishing returns. Similarly, the JPEG dataset exhibits positive throughput slopes across parameters, although reading speed trends differ. These findings underscore the critical role of disk I/O performance in large-scale machine learning applications, where cloud providers tie throughput to disk capacity. The observed behavior aligns with expectations of improved throughput with larger batch sizes and numbers, but the negative slope for Repetitions suggests potential overhead. Additionally, the provided latency numbers highlight the importance of minimizing latency between data storage locations, as cloud environments introduce complexities such as network latency and storage variability across regions. In conclusion, optimizing disk I/O performance and considering latency implications are vital for maximizing efficiency and resource utilization in large-scale machine learning deployments on cloud infrastructure.

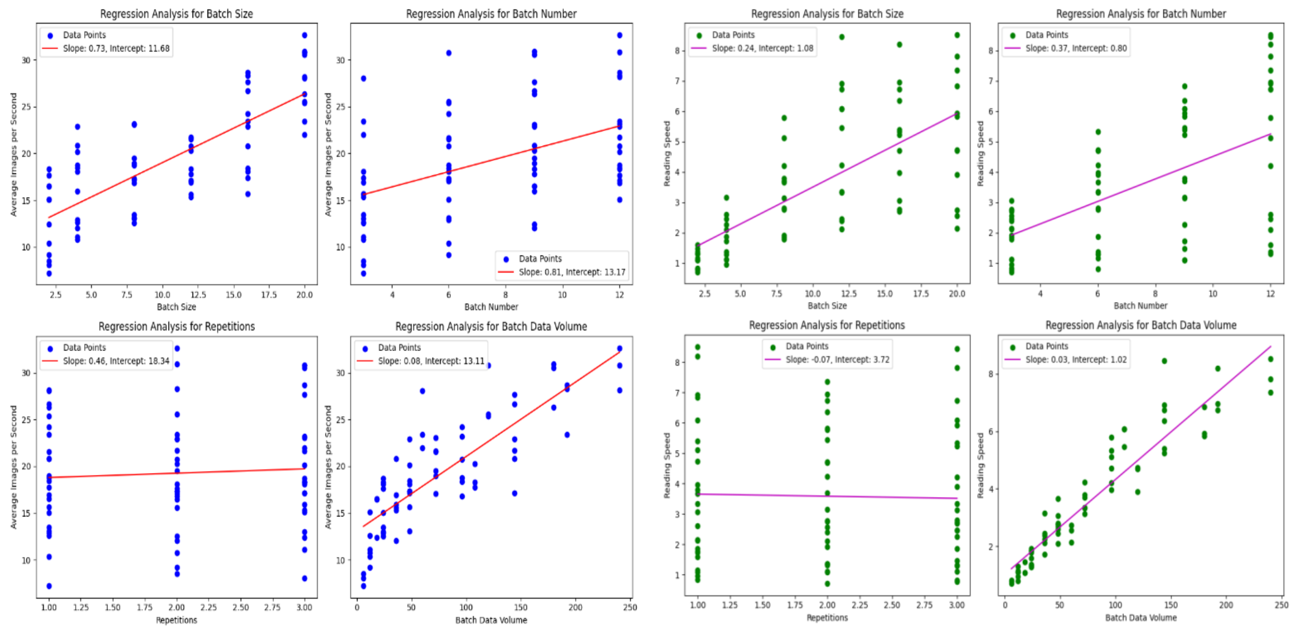
When conducting speed tests in parallel on the cloud, it's crucial to consider potential bottlenecks such as network latency, disk I/O limitations, and resource contention. Linear modeling provides insights into throughput and reading speed trends, but it may oversimplify complex interactions. While theoretical expectations align with certain trends like increasing throughput with larger batch sizes, practical considerations like network latency can influence real-world outcomes. Therefore, combining theoretical insights with empirical observations allows for a more nuanced understanding and informed decision-making in large-scale machine learning deployments on the cloud.

Linear Regression table for TFRECORD dataset				
Parameters	**Slope (Throughput)**	**Intercept (Throughput)**	**Slope (Reading speed)**	**Intercept (Reading speed)**
Batch Size	23.503925	62.101497	0.004626	0.207935
Batch Number	27.063281	102.000783	0.006558	0.206559
Repetitions	16.590107	271.795177	-0.017965	0.291672
Batch Data Volume	2.727811	93.570063	0.000784	0.194977



Plot of the output values, the averages per parameter value and the regression lines (**FOR TFRECORD DATASET**) for average images per sec (blue) and reading speeds (green)

Linear Regression table for JPEG dataset				
Parameters	**Slope (Throughput)**	**Intercept (Throughput)**	**Slope (Reading speed)**	**Intercept (Reading speed)**
Batch Size	0.733813	11.683522	0.242281	1.079661
Batch Number	0.812938	13.169223	0.370798	0.802244
Repetitions	0.463982	18.338293	-0.069885	3.723000
Batch Data Volume	0.079420	13.111228	0.033031	1.023303



Plot of the output values, the averages per parameter value and the regression lines (**FOR JPEG DATASET**) for average images per sec (blue) and reading speeds (green)

III. <u>SECTION 3</u>

- **3a) Contextualise: Relate the previous tasks and the results to this concept. To what extent and under what conditions do the concepts and techniques in the paper apply to the task in this coursework?**

In the context of our coursework, the concept of predicting an optimal or near-optimal cloud configuration for a given compute task, as introduced by Alipourfard *et al.* [1] in the CherryPick paper, finds practical application and relevance. The core of CherryPick's methodology is utilizing Bayesian Optimization to dynamically select cloud configurations that optimize for both cost and performance. This aligns closely with the challenges and objectives encountered in the tasks of Section 1 and Section 2 of the coursework.

In Section 1, particularly during the task to improve parallelization (1d i), I strategically increased the 'numSlices' to test the impact on task distribution and computational efficiency across single node and maximal node clusters. This task was directly aimed at identifying a more effective utilization of cloud resources, mirroring the objective of CherryPick to adaptively find the best configurations. The results showed that adjusting 'numSlices' could significantly influence CPU utilization and job execution time. For instance, increasing 'numSlices' from 18 to 40 on a single node cluster initially showed limited improvement in execution efficiency due to the inherent limitations of single-node processing. However, within a maximal node cluster, this adjustment drastically reduced execution time from 72 seconds to 37 seconds while maintaining moderate CPU utilization, thereby optimizing resource use.

In Section 2, during the speed tests (2c and 2d), we employed caching strategies to enhance performance by avoiding redundant computation across repeated data reads. This mirrors the efficiency gains that could be leveraged through CherryPick's approach to configuration optimization. By caching the results, I saw a reduction in total runtime and a more efficient peak CPU and network load, which indicates a more focused and effective use of resources.

The relevance of CherryPick's techniques in my coursework is particularly pronounced under conditions where cloud resource configurations and their optimization play a pivotal role in performance and cost. The tasks involved a complex interplay of computational resources, where the choice of configuration, such as the number of slices or the decision to cache data had substantial impacts on the efficiency and cost-effectiveness of cloud operations. This is similar to what CherryPick aims to achieve but in a more automated and theoretically informed manner using Bayesian Optimization.

Furthermore, the need for such adaptive configuration is especially critical when dealing with big data tasks that are resource-intensive and sensitive to execution time and resource allocation efficiency. The environments I worked with, characterized by variable task demands and differing data processing requirements (as evident in the differing outcomes when modifying 'numSlices' and using caching), are precisely the scenarios where CherryPick's configuration prediction mechanisms would be most beneficial.

In conclusion, the techniques and concepts from the CherryPick paper are not only applicable but potentially transformative for the tasks in the Big Data coursework, providing a theoretical and practical framework to optimize cloud configurations dynamically. This could lead to better performance, lower costs, and more efficient resource utilization in cloud-based big data processing tasks.

➢ **3b) Strategise: Define - as far as possible - concrete strategies for different application scenarios (batch, stream) and discuss the general relationship with the concepts above.**

- Batch Processing Strategies:
  a. Dynamic Resource Allocation:

In Section 1, where I experimented with adjusting the 'numSlices', it was clear that varying computational resources could drastically affect performance. For batch processing tasks, which often handle large datasets in non-real-time, implementing algorithms that dynamically adjust resources based on workload predictions can significantly enhance processing efficiency. This strategy aligns with the principles of dynamically selecting cloud configurations as seen in CherryPick, optimizing resource usage during computation-heavy periods.

  b. Optimized Task Scheduling:

Considering the flexibility in timing for batch processing tasks, I propose strategically scheduling these processes during periods when network traffic is low, which could potentially allow for more efficient data handling and processing speed without the constraint of competing resources. This scheduling optimization ensures that computational power is allocated where and when it's most needed, reflecting an operational adaptation of CherryPick's optimization approach.

- Stream Processing Strategies:
  a. Real-Time Resource Scaling:

For stream processing, which demands high responsiveness, dynamically scaling resources in real-time is crucial. Inspired by the gains observed from applying adaptive strategies in this coursework, this approach would ensure that resources are appropriately scaled based on the actual processing demand, maintaining system responsiveness and processing integrity without over-provisioning.

b. Robust Fault Tolerance:

Given the continuous operation in stream processing, developing robust fault tolerance mechanisms is essential. Techniques such as redundant data paths and automated failover protocols can enhance system reliability. This strategy ensures continuity and data integrity, which is vital for real-time processing applications like monitoring systems or online transaction processing.

- General Relationship with Above Concepts:

These strategies for batch and stream processing directly utilize the optimization principles discussed in CherryPick and observed during this coursework experiments. Adjusting 'numSlices' for batch tasks or dynamically scaling resources for stream tasks exemplifies practical applications of predictive and adaptive optimization techniques in managing cloud resources effectively.

The relationship with CherryPick is evident as these strategies implement a practical framework for dynamically adjusting cloud configurations based on actual data processing needs. By enhancing computational efficiency and processing speed, these strategies go beyond cost concerns to directly address the core performance challenges encountered in different data processing scenarios.

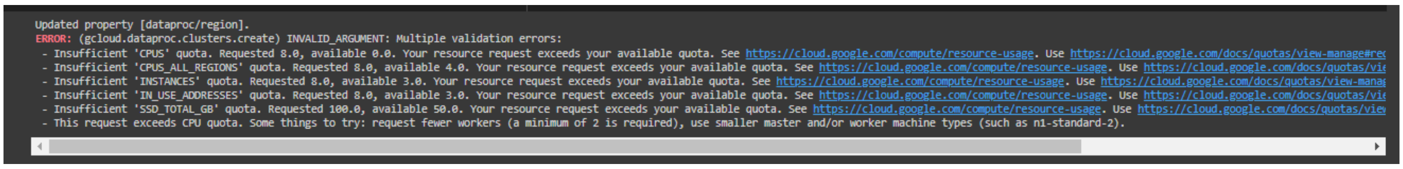
This tailored approach ensures that both batch and stream processing tasks are optimized for maximum efficiency, leveraging theoretical insights from optimization models and empirical evidence from my coursework to create robust, performance-focused cloud computing strategies.

Reference:

[1] Alipourfard, O., Liu, H. H., Chen, J., Venkataraman, S., Yu, M., & Zhang, M. (2017). Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In USENIX NSDI 17 (pp. 469-482).

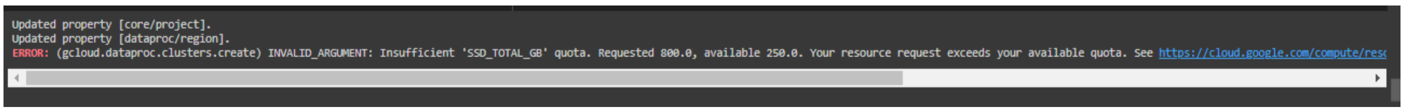
Abstract:

1) This is the error message of when I was trying to create maximal cluster even after deleting all the existing clusters, the error stayed showing the quota available is only 4 and not 8.



```
Updated property [dataproc/region].
ERROR: (gcloud.dataproc.clusters.create) INVALID_ARGUMENT: Multiple validation errors:
 - Insufficient 'CPUS' quota. Requested 8.0, available 0.0. Your resource request exceeds your available quota. See https://cloud.google.com/compute/resource-usage. Use https://cloud.google.com/docs/quotas/view-manage#rec
 - Insufficient 'CPUS_ALL_REGIONS' quota. Requested 8.0, available 4.0. Your resource request exceeds your available quota. See https://cloud.google.com/compute/resource-usage. Use https://cloud.google.com/docs/quotas/vie
 - Insufficient 'INSTANCES' quota. Requested 8.0, available 3.0. Your resource request exceeds your available quota. See https://cloud.google.com/compute/resource-usage. Use https://cloud.google.com/docs/quotas/view-manag
 - Insufficient 'IN_USE_ADDRESSES' quota. Requested 8.0, available 3.0. Your resource request exceeds your available quota. See https://cloud.google.com/compute/resource-usage. Use https://cloud.google.com/docs/quotas/vie
 - Insufficient 'SSD_TOTAL_GB' quota. Requested 100.0, available 50.0. Your resource request exceeds your available quota. See https://cloud.google.com/compute/resource-usage. Use https://cloud.google.com/docs/quotas/view
 - This request exceeds CPU quota. Some things to try: request fewer workers (a minimum of 2 is required), use smaller master and/or worker machine types (such as n1-standard-2).
```

2) This error showed up in 1c ii) when I was trying to create cluster with single machine with eightfold resources but the quota issue, even after deleting all the pre-existing clusters, showed that only 250 SSD is available and not 800 like we are asked to create.



```
Updated property [core/project].
Updated property [dataproc/region].
ERROR: (gcloud.dataproc.clusters.create) INVALID_ARGUMENT: Insufficient 'SSD_TOTAL_GB' quota. Requested 800.0, available 250.0. Your resource request exceeds your available quota. See https://cloud.google.com/compute/resc
```

Number of words in the Report (Excluding the questions and headings)	**2084**