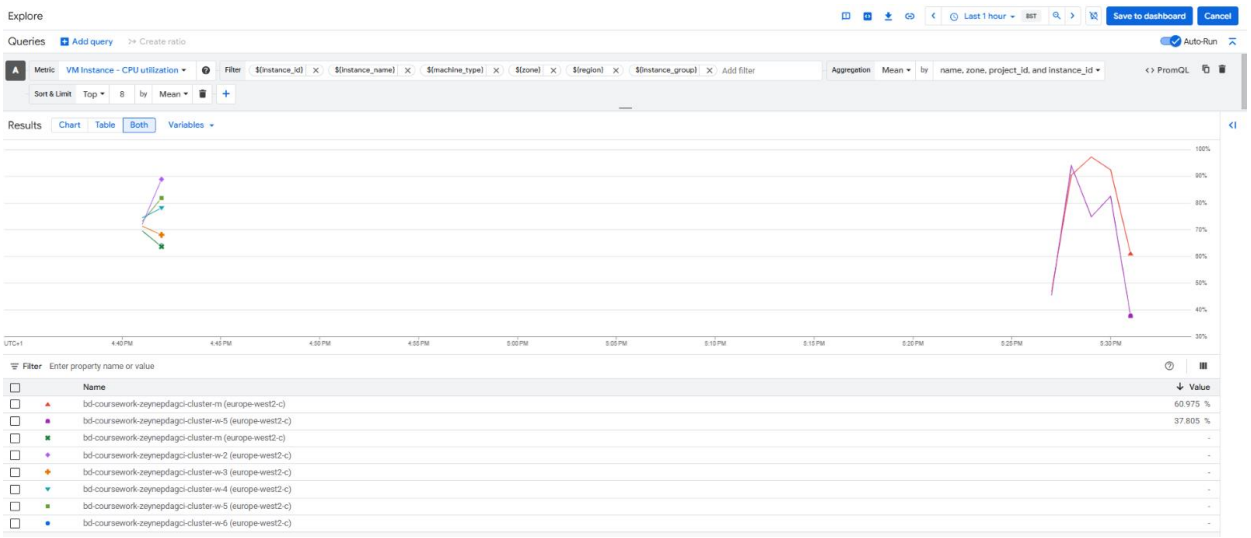
## Big Data Report

Colab Shareable Link:
https://colab.research.google.com/drive/1n8_oC2WLnsknCthNiSLQjBoSS8tcrsG_?usp=sharing

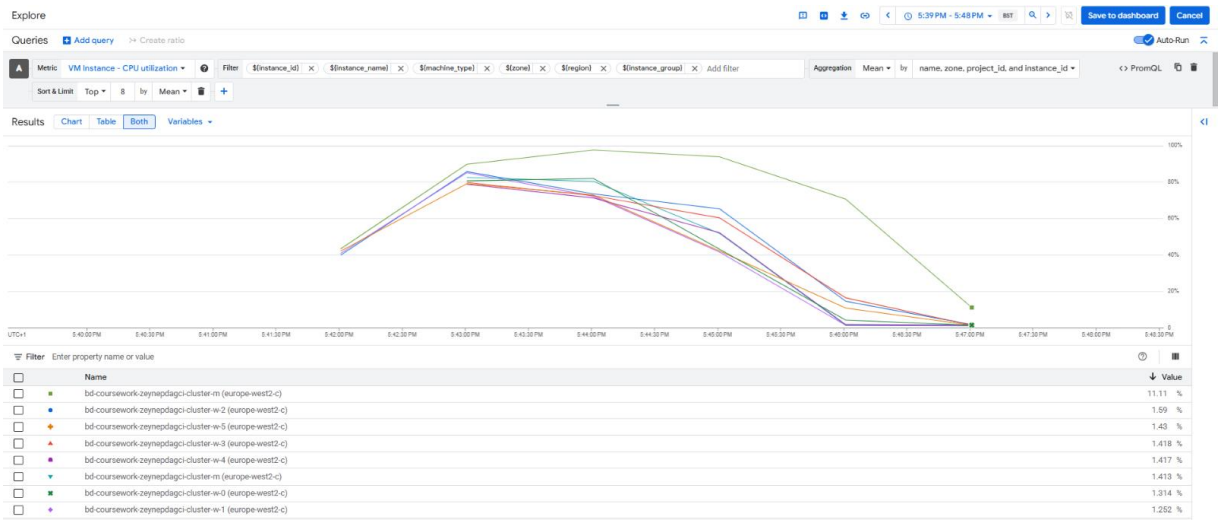
### Task 1d – Part i

Before the second parameter (2 partitions):




- 8 node maximal cluster running with 2 partitions
- Only 2 VMs (master and one worker node) have non-zero values (~61, ~38) while the other 6 are at 0%.
- Elapsed time: 41.38

After the second parameter (16 partitions):



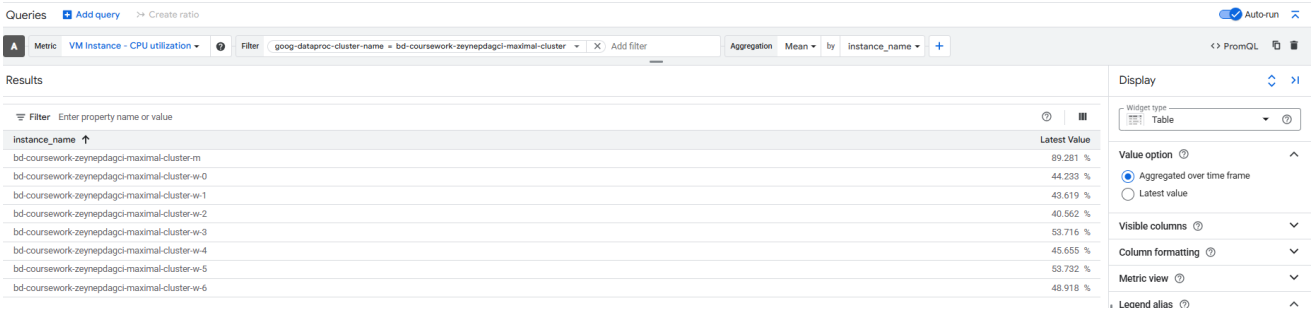

- 8 node maximal cluster running with 16 partitions (2 tasks per vCPU x 8 vCPUs)
- 8 of the VMs have non-zero values, which means Spark scheduled work on each of them.

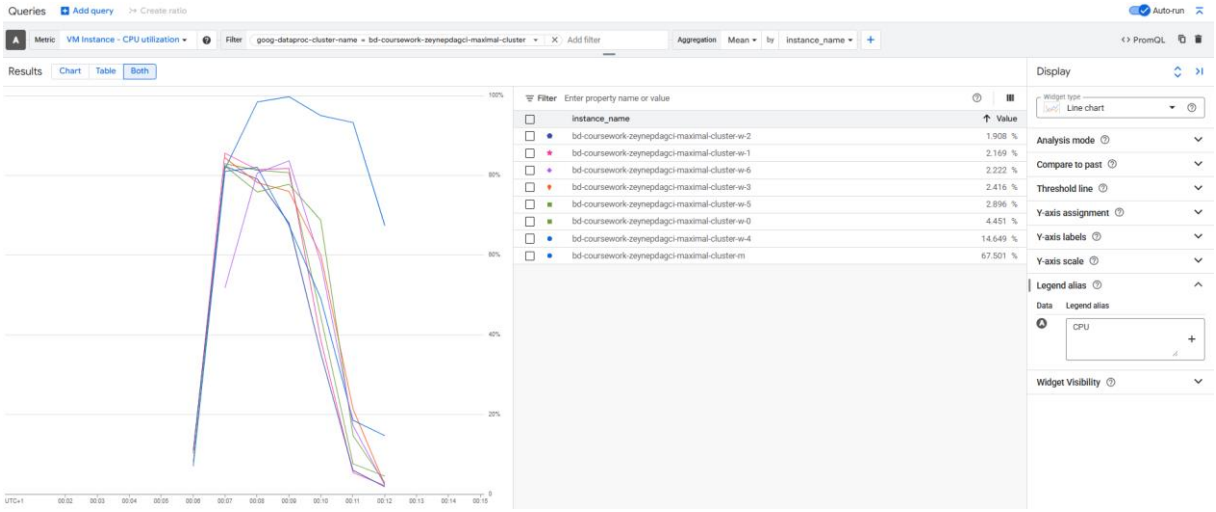
- Elapsed time: 32.43

## Task 1d – Part ii

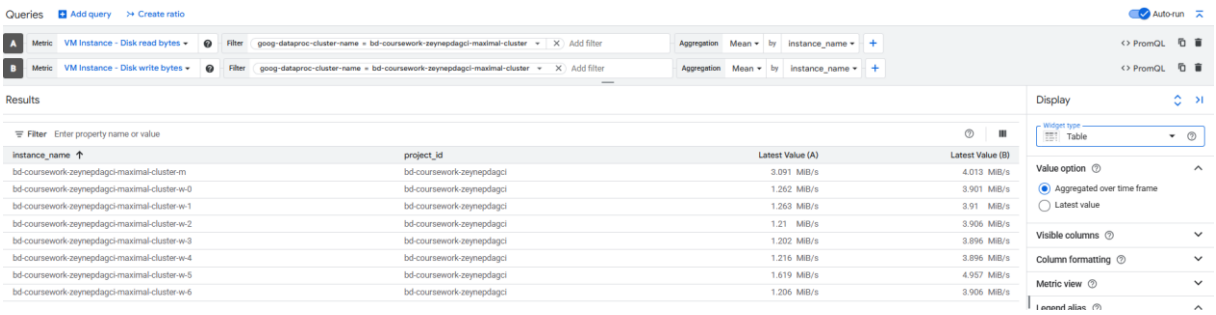
### 1. 8x1

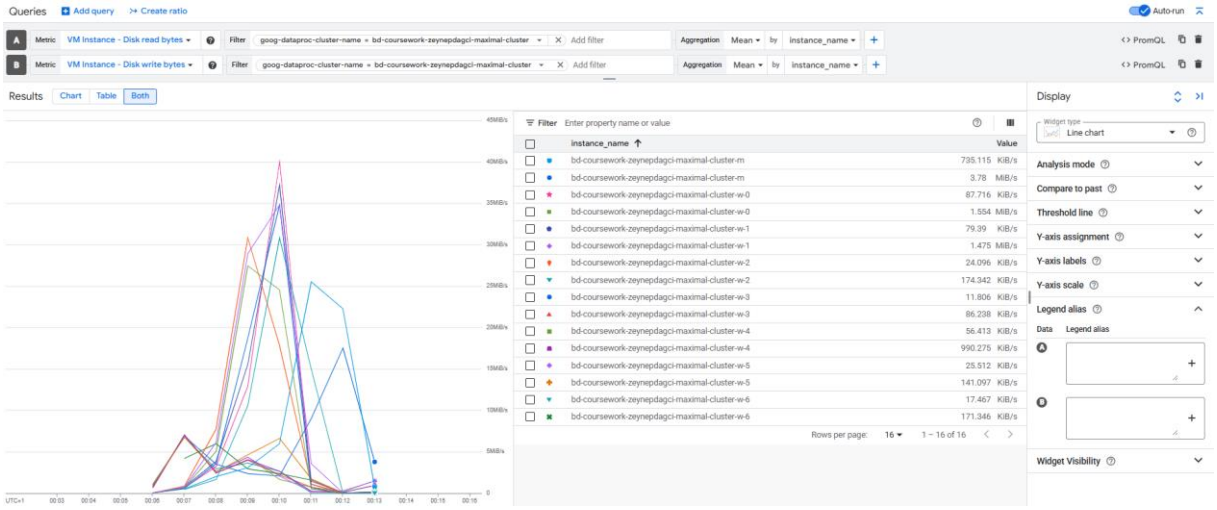
CPU:

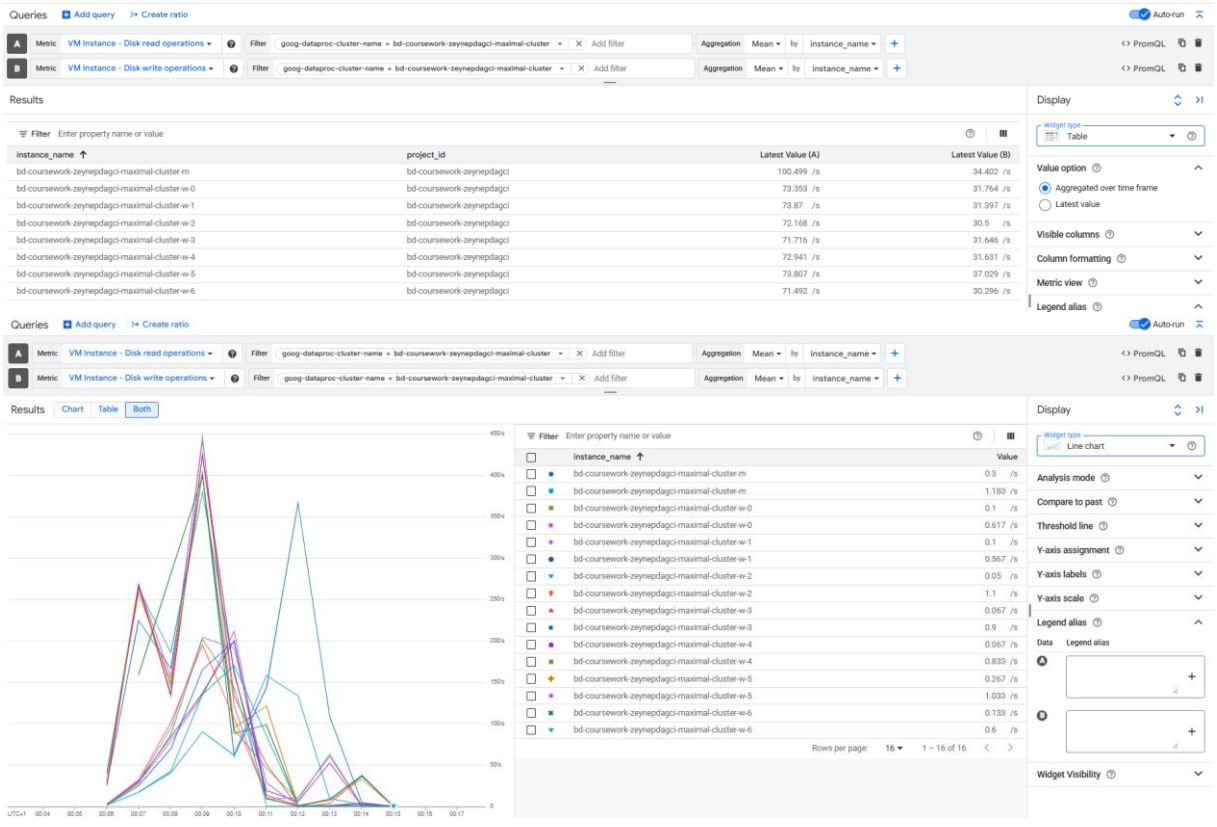



Disk I/O:



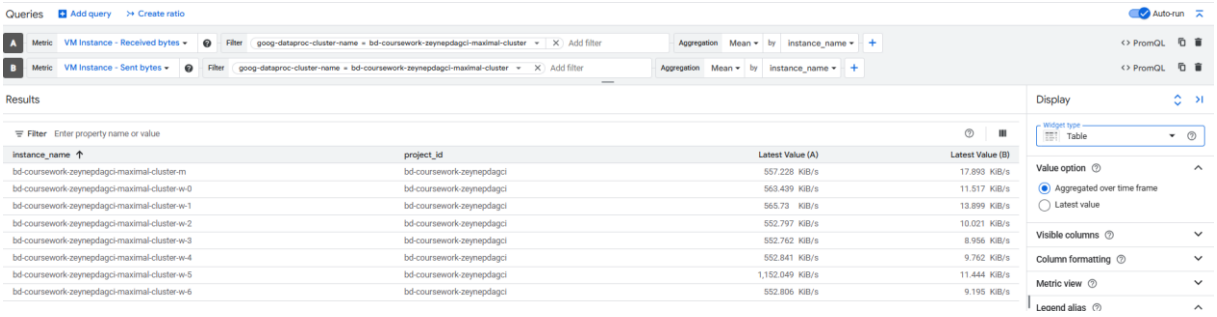


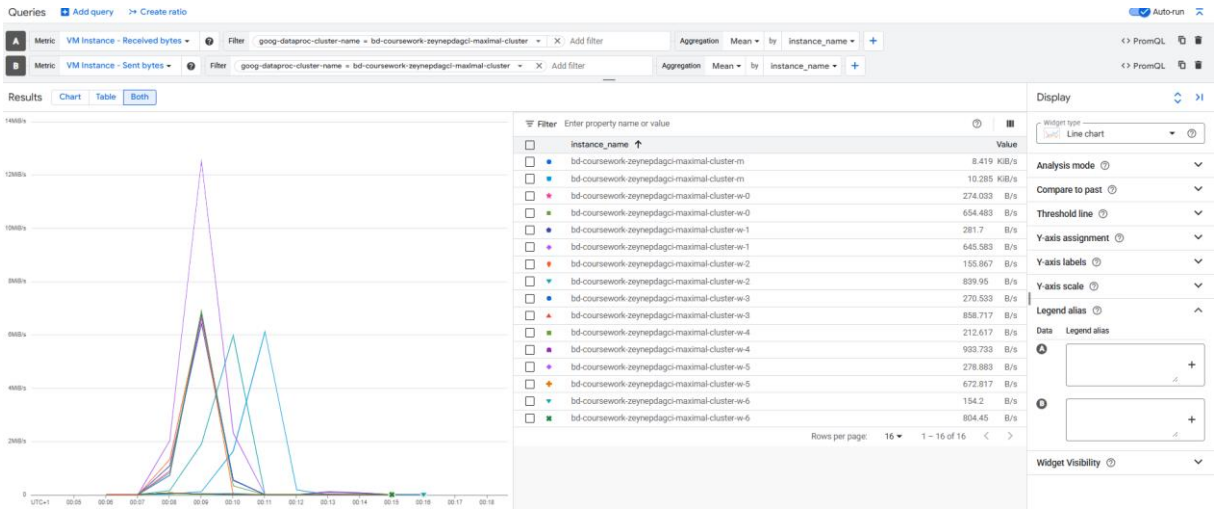



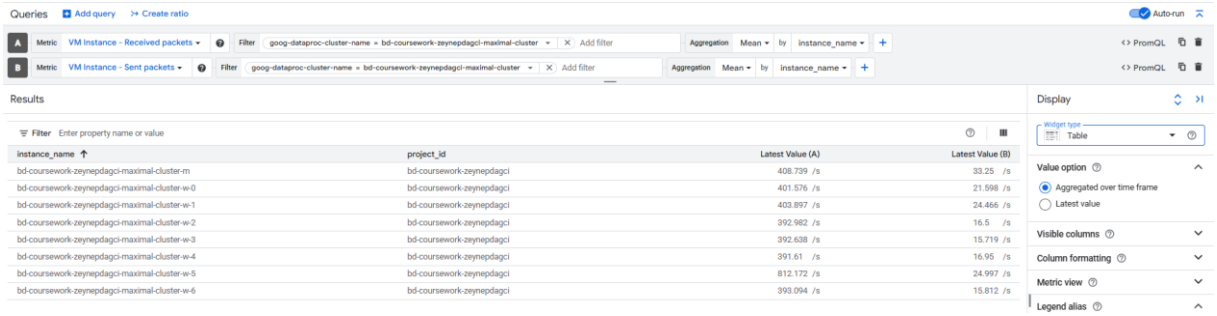


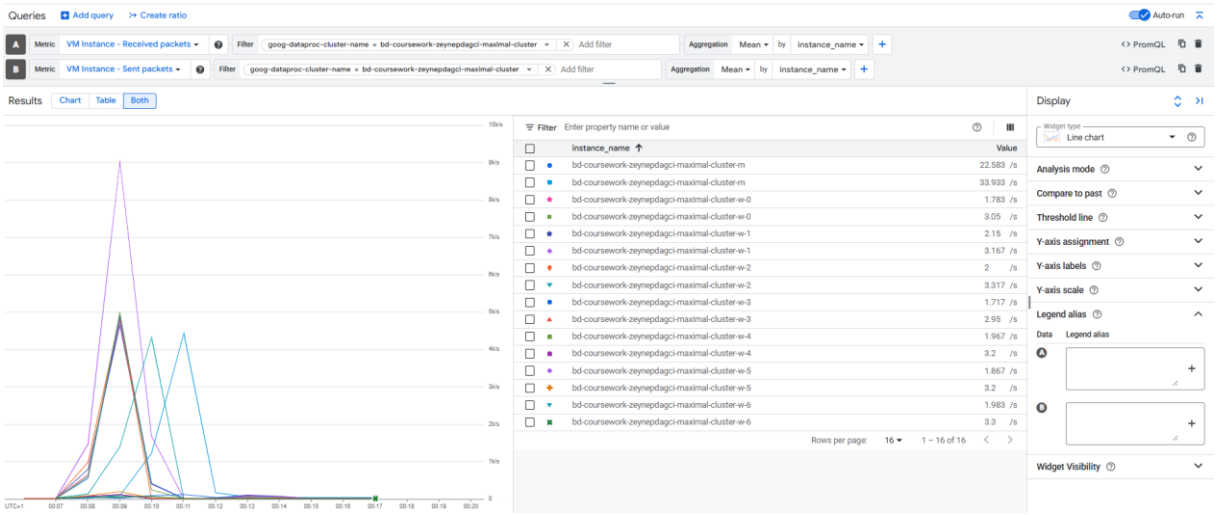


## Network Bandwith:



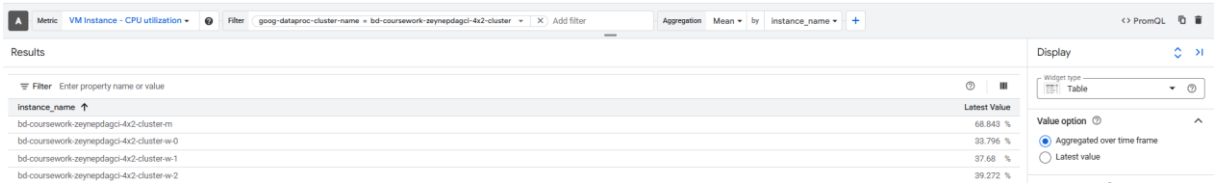

Queries
Add query
Create ratio
Auto-run
Metric
VM Instance - Received bytes
Filter
goog-dataproc-cluster-name = bd-coursework-zeynepdagci-maximal-cluster
Add filter
Aggregation
Mean
by
instance_name
PromQL
VM Instance - Sent bytes
Results
Chart
Table
Both
Display
Widget type
Line chart
Analysis mode
Compare to past
Threshold line
Y-axis assignment
Y-axis labels
Y-axis scale
Legend alias
Data
Legend alias
Widget Visibility
Enter property name or value
instance_name
Value
bd-coursework-zeynepdagci-maximal-cluster-m 8.419 KiB/s
bd-coursework-zeynepdagci-maximal-cluster-m 10.285 KiB/s
bd-coursework-zeynepdagci-maximal-cluster-w-0 274.033 B/s
bd-coursework-zeynepdagci-maximal-cluster-w-0 654.483 B/s
bd-coursework-zeynepdagci-maximal-cluster-w-1 281.7 B/s
bd-coursework-zeynepdagci-maximal-cluster-w-1 645.583 B/s
bd-coursework-zeynepdagci-maximal-cluster-w-2 155.867 B/s
bd-coursework-zeynepdagci-maximal-cluster-w-2 839.95 B/s
bd-coursework-zeynepdagci-maximal-cluster-w-3 270.533 B/s
bd-coursework-zeynepdagci-maximal-cluster-w-3 858.717 B/s
bd-coursework-zeynepdagci-maximal-cluster-w-4 212.617 B/s
bd-coursework-zeynepdagci-maximal-cluster-w-4 933.733 B/s
bd-coursework-zeynepdagci-maximal-cluster-w-5 278.883 B/s
bd-coursework-zeynepdagci-maximal-cluster-w-5 672.817 B/s
bd-coursework-zeynepdagci-maximal-cluster-w-6 154.2 B/s
bd-coursework-zeynepdagci-maximal-cluster-w-6 804.45 B/s
Rows per page: 16
1 – 16 of 16

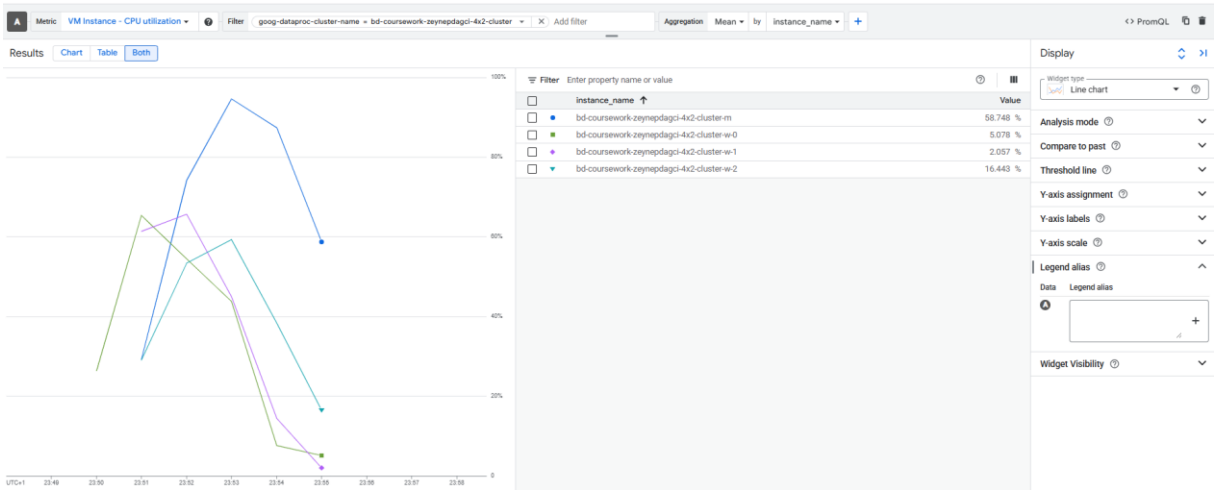

Queries
Add query
Create ratio
Auto-run
Metric
VM Instance - Received packets
Filter
goog-dataproc-cluster-name = bd-coursework-zeynepdagci-maximal-cluster
Add filter
Aggregation
Mean
by
instance_name
PromQL
VM Instance - Sent packets
Results
Display
Widget type
Table
Value option
Aggregated over time frame
Latest value
Visible columns
Column formatting
Metric view
Legend alias
Enter property name or value
instance_name
project_id
Latest Value (A)
Latest Value (B)
bd-coursework-zeynepdagci-maximal-cluster-m bd-coursework-zeynepdagci 408.739 /s 33.25 /s
bd-coursework-zeynepdagci-maximal-cluster-w-0 bd-coursework-zeynepdagci 401.576 /s 21.598 /s
bd-coursework-zeynepdagci-maximal-cluster-w-1 bd-coursework-zeynepdagci 403.897 /s 24.466 /s
bd-coursework-zeynepdagci-maximal-cluster-w-2 bd-coursework-zeynepdagci 392.982 /s 16.5 /s
bd-coursework-zeynepdagci-maximal-cluster-w-3 bd-coursework-zeynepdagci 392.638 /s 15.719 /s
bd-coursework-zeynepdagci-maximal-cluster-w-4 bd-coursework-zeynepdagci 391.61 /s 16.95 /s
bd-coursework-zeynepdagci-maximal-cluster-w-5 bd-coursework-zeynepdagci 812.172 /s 24.997 /s
bd-coursework-zeynepdagci-maximal-cluster-w-6 bd-coursework-zeynepdagci 393.094 /s 15.812 /s


Queries
Add query
Create ratio
Auto-run
Metric
VM Instance - Received packets
Filter
goog-dataproc-cluster-name = bd-coursework-zeynepdagci-maximal-cluster
Add filter
Aggregation
Mean
by
instance_name
PromQL
VM Instance - Sent packets
Results
Chart
Table
Both
Display
Widget type
Line chart
Analysis mode
Compare to past
Threshold line
Y-axis assignment
Y-axis labels
Y-axis scale
Legend alias
Data
Legend alias
Widget Visibility
Enter property name or value
instance_name
Value
bd-coursework-zeynepdagci-maximal-cluster-m 22.583 /s
bd-coursework-zeynepdagci-maximal-cluster-m 33.933 /s
bd-coursework-zeynepdagci-maximal-cluster-w-0 1.783 /s
bd-coursework-zeynepdagci-maximal-cluster-w-0 3.05 /s
bd-coursework-zeynepdagci-maximal-cluster-w-1 2.15 /s
bd-coursework-zeynepdagci-maximal-cluster-w-1 3.167 /s
bd-coursework-zeynepdagci-maximal-cluster-w-2 2 /s
bd-coursework-zeynepdagci-maximal-cluster-w-2 3.317 /s
bd-coursework-zeynepdagci-maximal-cluster-w-3 1.717 /s
bd-coursework-zeynepdagci-maximal-cluster-w-3 2.95 /s
bd-coursework-zeynepdagci-maximal-cluster-w-4 1.967 /s
bd-coursework-zeynepdagci-maximal-cluster-w-4 3.2 /s
bd-coursework-zeynepdagci-maximal-cluster-w-5 1.867 /s
bd-coursework-zeynepdagci-maximal-cluster-w-5 3.2 /s
bd-coursework-zeynepdagci-maximal-cluster-w-6 1.983 /s
bd-coursework-zeynepdagci-maximal-cluster-w-6 3.3 /s
Rows per page: 16
1 – 16 of 16

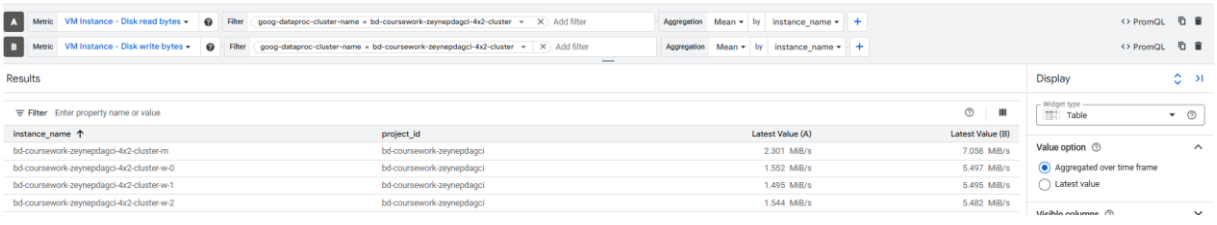
## 2. 4x2

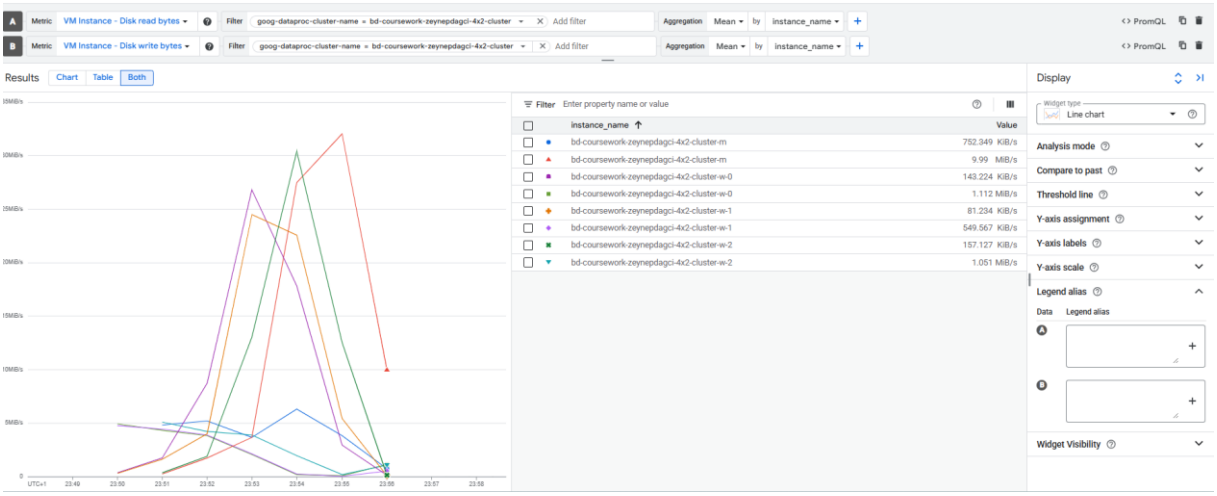
CPU:

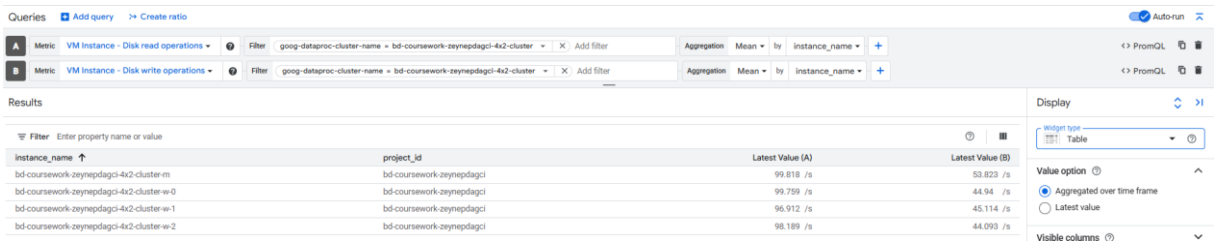

Metric
VM Instance - CPU utilization
Filter
goog-dataproc-cluster-name = bd-coursework-zeynepdagci-4x2-cluster
Add filter
Aggregation
Mean
by
instance_name
PromQL
Results
Display
Widget type
Table
Value option
Aggregated over time frame
Latest value
Enter property name or value
instance_name
Latest Value
bd-coursework-zeynepdagci-4x2-cluster-m 68.843 %
bd-coursework-zeynepdagci-4x2-cluster-w-0 33.796 %
bd-coursework-zeynepdagci-4x2-cluster-w-1 37.68 %
bd-coursework-zeynepdagci-4x2-cluster-w-2 39.272 %

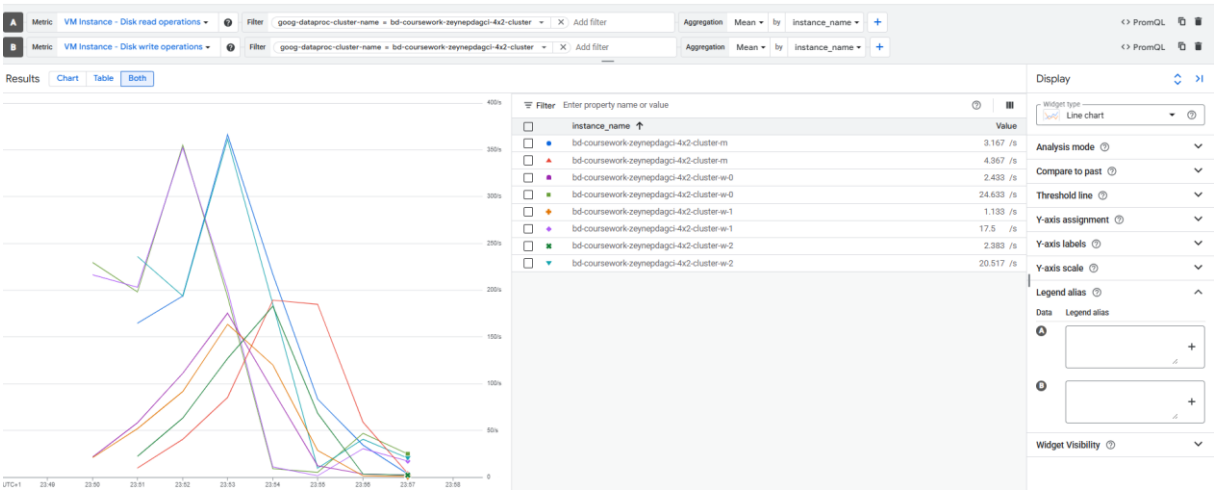

Metric
VM Instance - CPU utilization
Filter
goog-dataproc-cluster-name = bd-coursework-zeynepdagci-4x2-cluster
Add filter
Aggregation
Mean
by
instance_name
PromQL
Results
Chart
Table
Both
Filter
Enter property name or value
instance_name
Value
bd-coursework-zeynepdagci-4x2-cluster-m
58.748 %
bd-coursework-zeynepdagci-4x2-cluster-w-0
5.078 %
bd-coursework-zeynepdagci-4x2-cluster-w-1
2.057 %
bd-coursework-zeynepdagci-4x2-cluster-w-2
16.443 %
Display
Widget type
Line chart
Analysis mode
Compare to past
Threshold line
Y-axis assignment
Y-axis labels
Y-axis scale
Legend alias
Data
Legend alias
Widget Visibility

## Disk I/O:

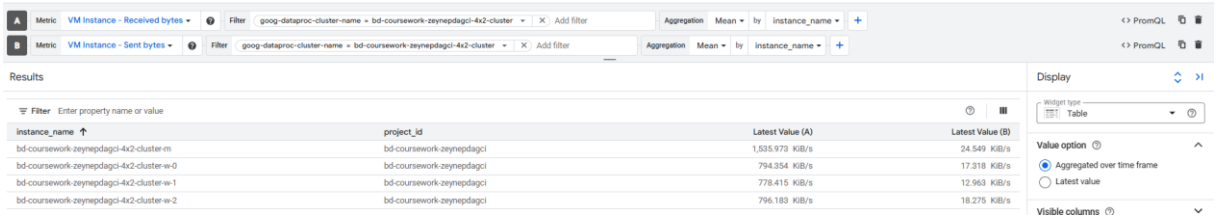

Metric
VM Instance - Disk read bytes
VM Instance - Disk write bytes
Filter
goog-dataproc-cluster-name = bd-coursework-zeynepdagci-4x2-cluster
Add filter
Aggregation
Mean
by
instance_name
PromQL
Results
Filter
Enter property name or value
instance_name
project_id
Latest Value (A)
Latest Value (B)
bd-coursework-zeynepdagci-4x2-cluster-m
bd-coursework-zeynepdagci
2.301 MiB/s
7.058 MiB/s
bd-coursework-zeynepdagci-4x2-cluster-w-0
bd-coursework-zeynepdagci
1.552 MiB/s
5.497 MiB/s
bd-coursework-zeynepdagci-4x2-cluster-w-1
bd-coursework-zeynepdagci
1.495 MiB/s
5.495 MiB/s
bd-coursework-zeynepdagci-4x2-cluster-w-2
bd-coursework-zeynepdagci
1.544 MiB/s
5.482 MiB/s
Display
Widget type
Table
Value option
Aggregated over time frame
Latest value

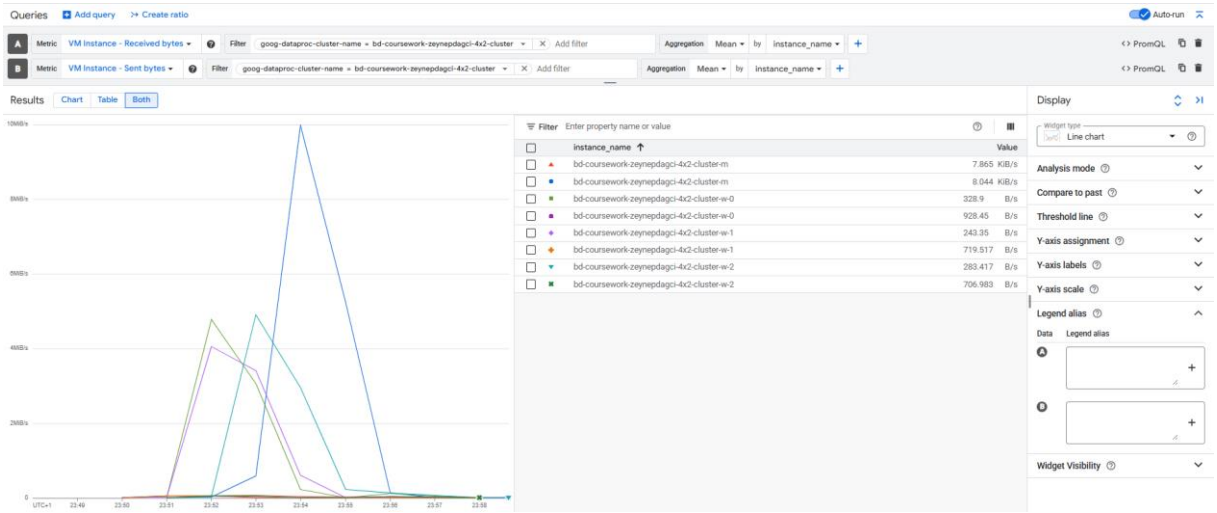

Metric
VM Instance - Disk read bytes
VM Instance - Disk write bytes
Filter
goog-dataproc-cluster-name = bd-coursework-zeynepdagci-4x2-cluster
Add filter
Aggregation
Mean
by
instance_name
PromQL
Results
Chart
Table
Both
Filter
Enter property name or value
instance_name
Value
bd-coursework-zeynepdagci-4x2-cluster-m
752.349 KiB/s
bd-coursework-zeynepdagci-4x2-cluster-m
9.99 MiB/s
bd-coursework-zeynepdagci-4x2-cluster-w-0
143.224 KiB/s
bd-coursework-zeynepdagci-4x2-cluster-w-0
1.112 MiB/s
bd-coursework-zeynepdagci-4x2-cluster-w-1
81.234 KiB/s
bd-coursework-zeynepdagci-4x2-cluster-w-1
549.567 KiB/s
bd-coursework-zeynepdagci-4x2-cluster-w-2
157.127 KiB/s
bd-coursework-zeynepdagci-4x2-cluster-w-2
1.051 MiB/s
Display
Widget type
Line chart
Analysis mode
Compare to past
Threshold line
Y-axis assignment
Y-axis labels
Y-axis scale
Legend alias
Data
Legend alias
Widget Visibility

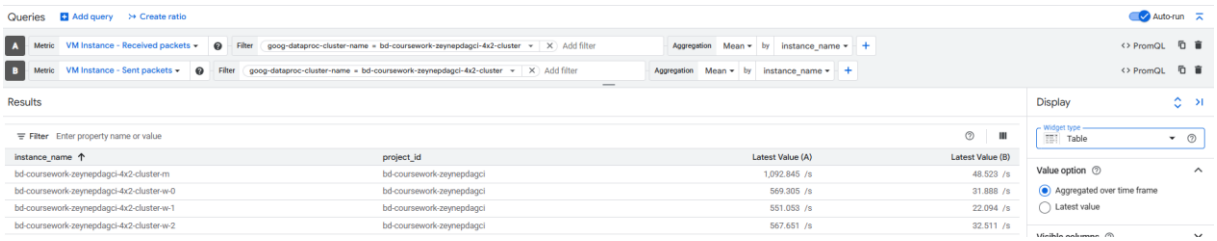

Queries
Add query
Create ratio
Auto-run
Metric
VM Instance - Disk read operations
VM Instance - Disk write operations
Filter
goog-dataproc-cluster-name = bd-coursework-zeynepdagci-4x2-cluster
Add filter
Aggregation
Mean
by
instance_name
PromQL
Results
Filter
Enter property name or value
instance_name
project_id
Latest Value (A)
Latest Value (B)
bd-coursework-zeynepdagci-4x2-cluster-m
bd-coursework-zeynepdagci
99.818 /s
53.823 /s
bd-coursework-zeynepdagci-4x2-cluster-w-0
bd-coursework-zeynepdagci
99.759 /s
44.94 /s
bd-coursework-zeynepdagci-4x2-cluster-w-1
bd-coursework-zeynepdagci
96.912 /s
45.114 /s
bd-coursework-zeynepdagci-4x2-cluster-w-2
bd-coursework-zeynepdagci
98.189 /s
44.093 /s
Display
Widget type
Table
Value option
Aggregated over time frame
Latest value
Visible columns

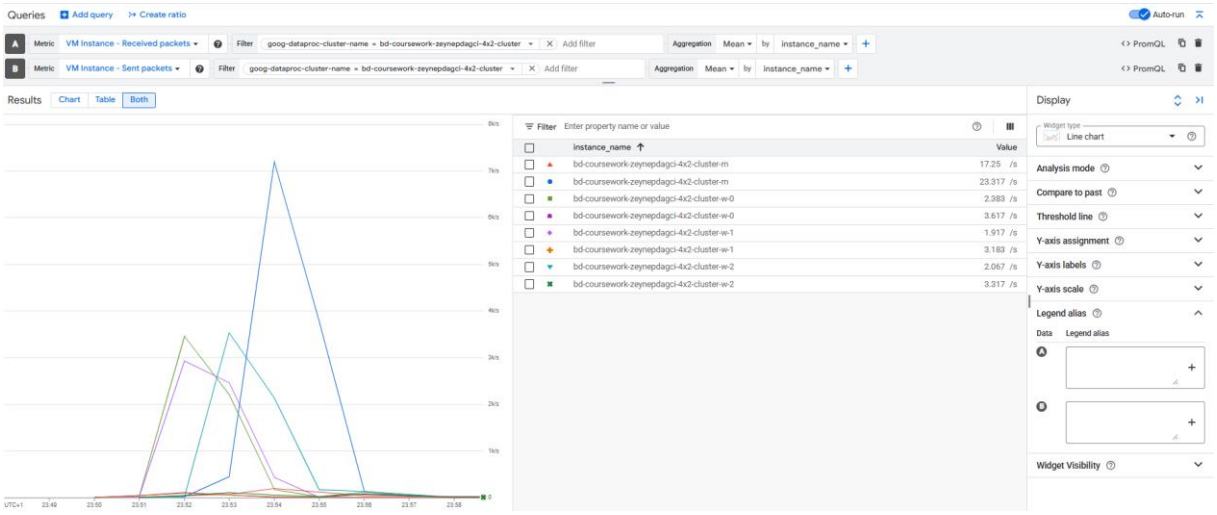



Network Bandwith:



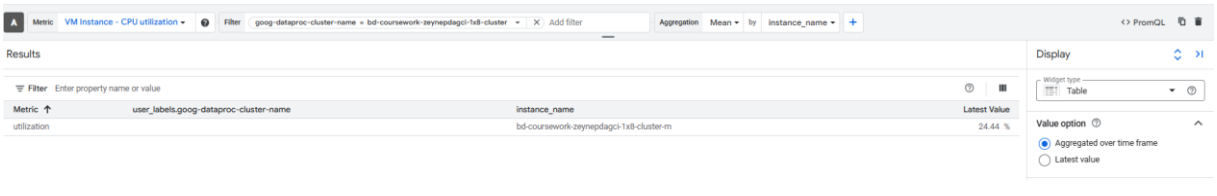



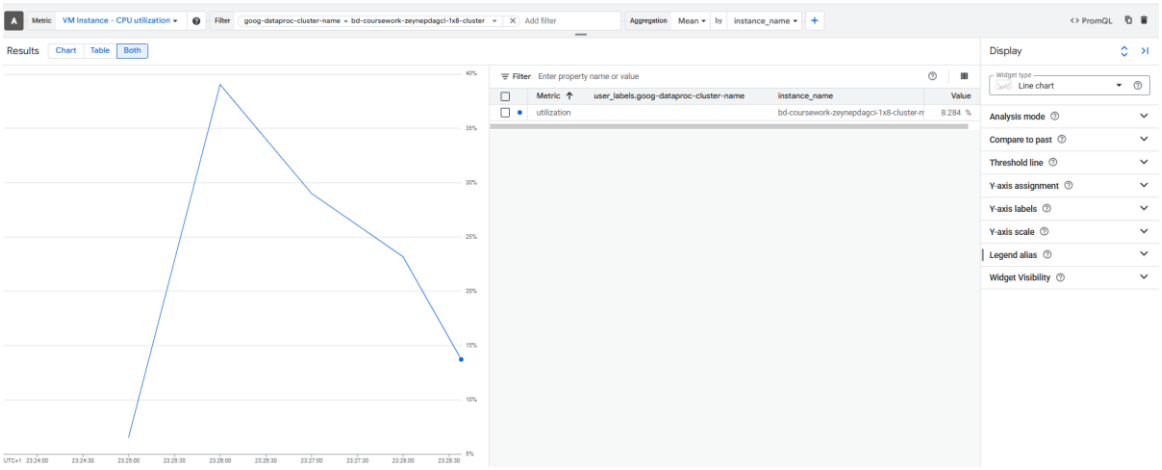





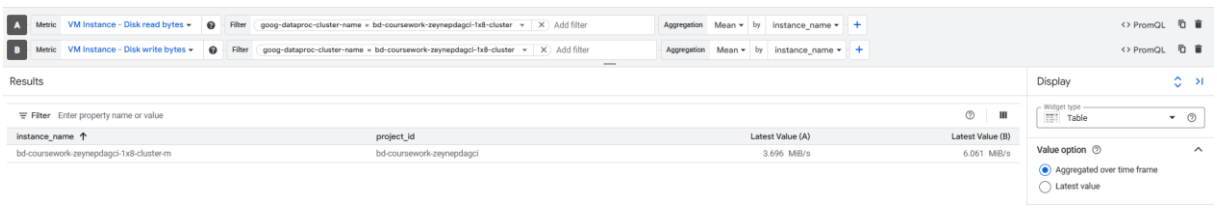

## 3. 1x8

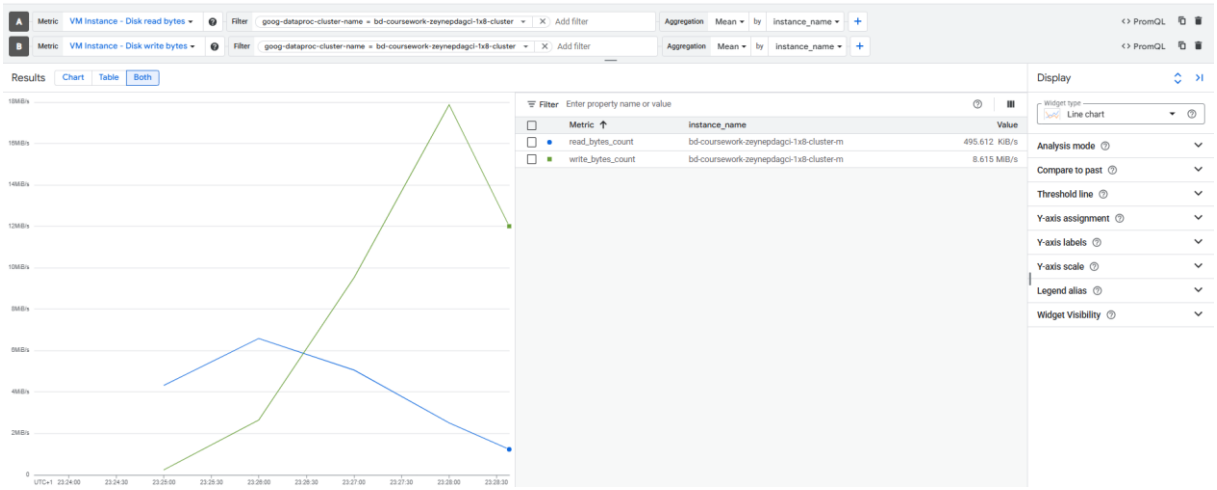
CPU:

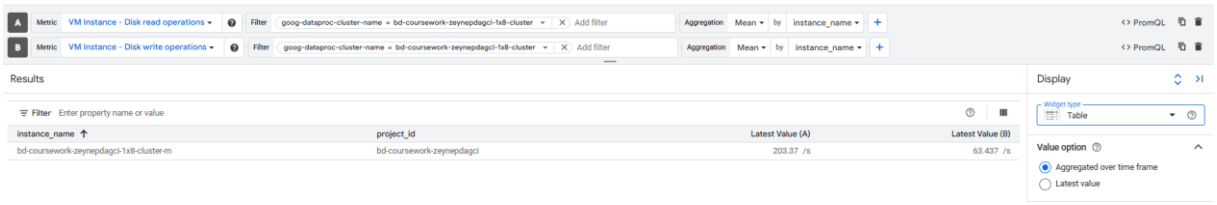


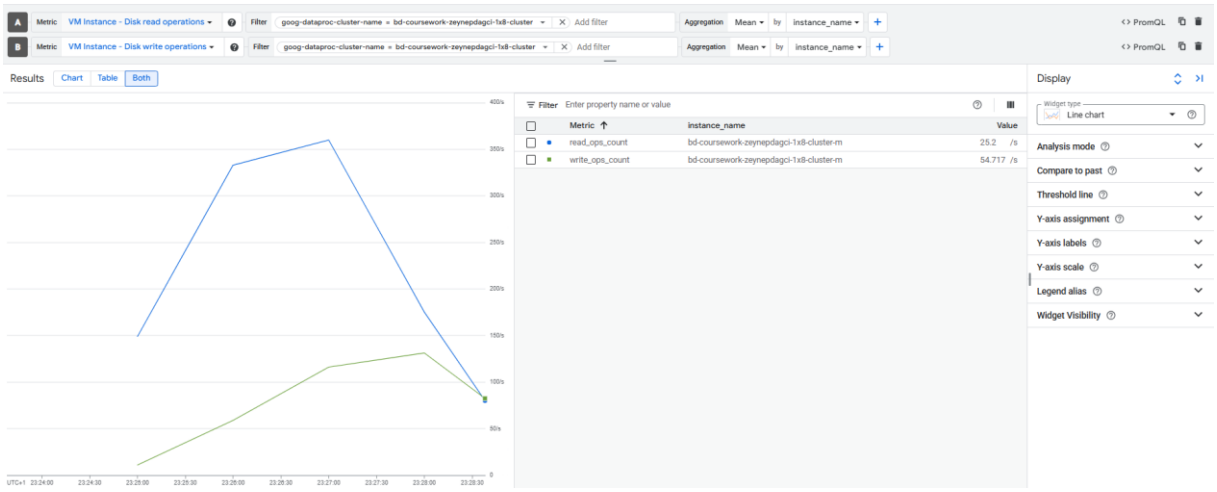




Disk I/O:

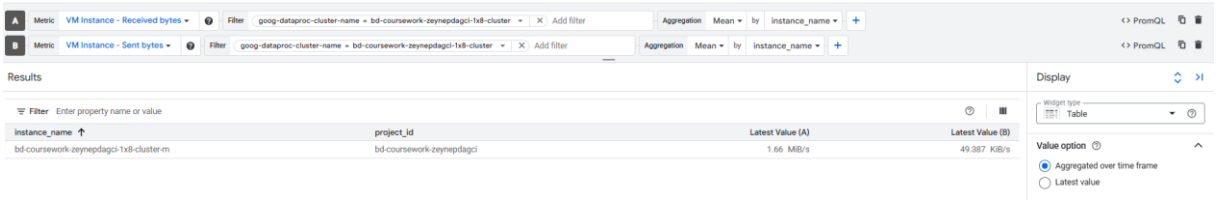


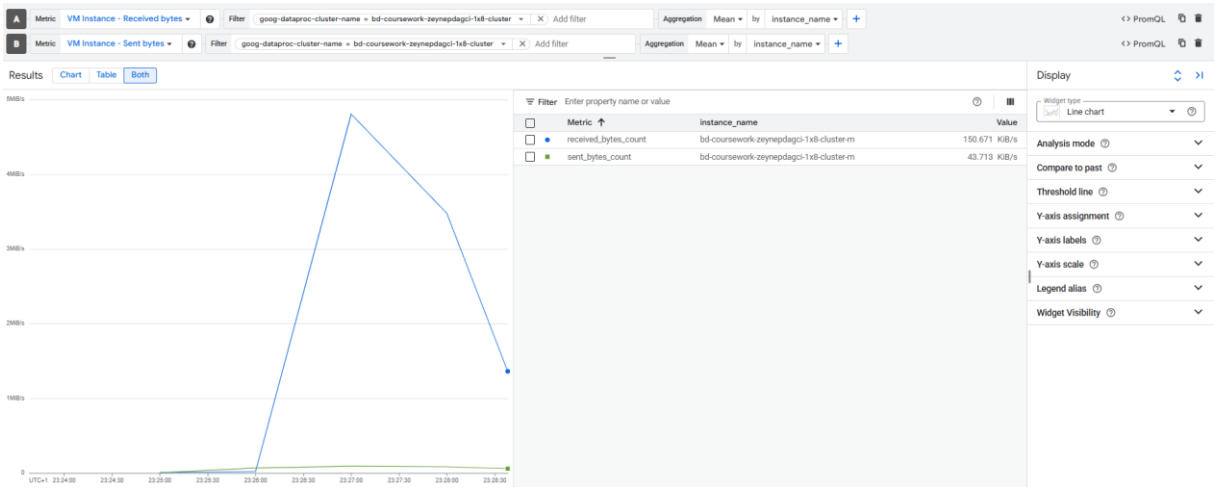


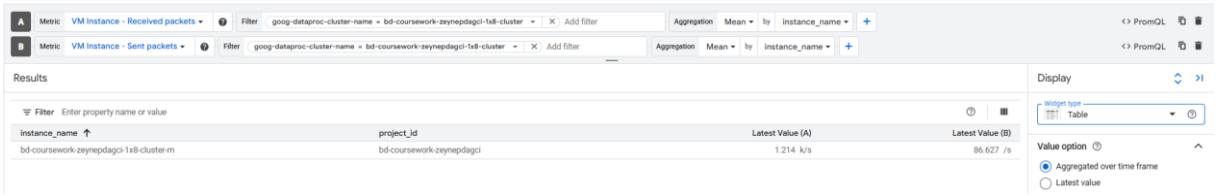



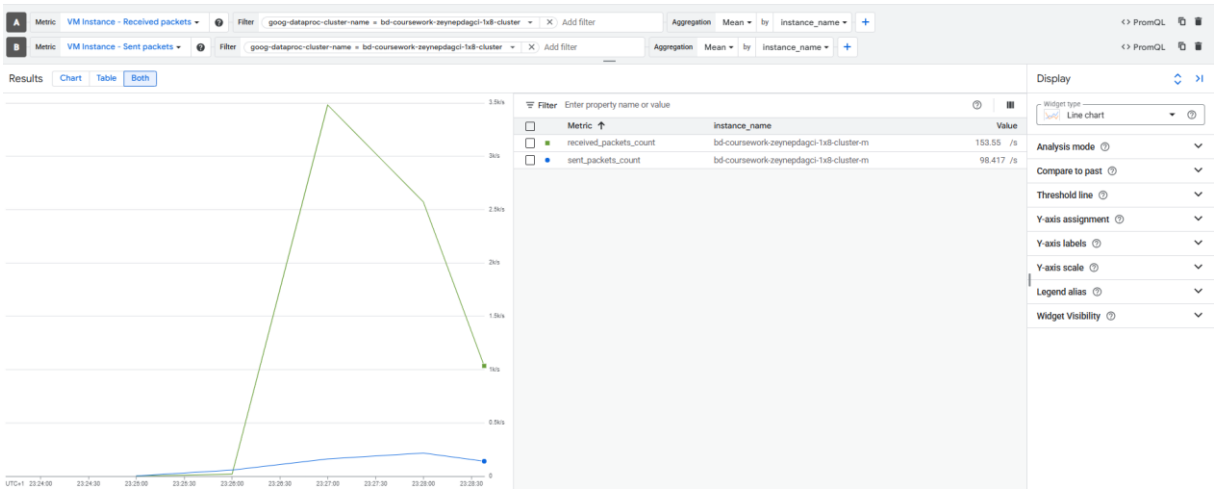




## Network Bandwith:






**Summary of the Figures**

8x1 has delivered the highest disk I/O of all three, while 1x8 had the lowest disk I/O because all 8 vCPUs are packed into one VM which limits disk bandwidth, whereas the 8x1 has 8 separate VMs with each of them having its own disk channel.4x2 is in the middle while offering I/O parallelism with fewer nodes to manage. 8x1 also delivered the highest network throughput and packet rates, while 1x8 had the lowest network capacity since all traffic is channelled through one single VM, and 4x2 is in between them in terms of network performance by using 4 VMs, each with its own network interface. In conclusion, for I/O heavy workloads, 8x1 is the better choice for maximum throughput. 4x2 is a good choice for balancing I/O with manageable cluster overhead. 1x8 can be considered if there is a workload that is heavily CPU bound with minimal dependency on disk I/O.

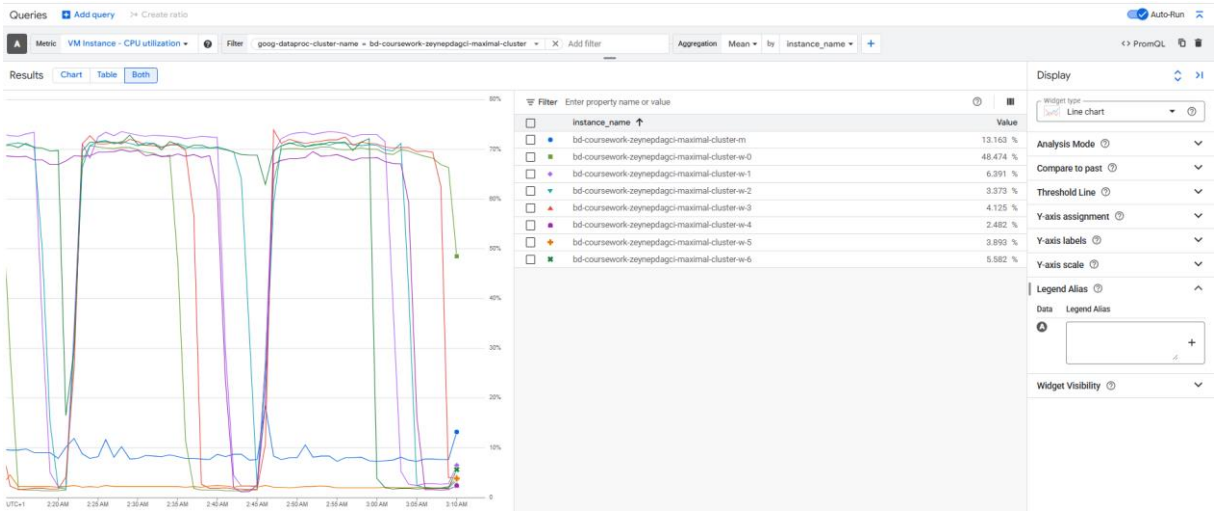
**Task 1d – part iii**

In the labs, data is usually mounted from the Google Drive, while in the coursework data is stored in remote locations and pulled from Google Cloud Storage. gs:// paths are used to read and write objects in GCS. Spark uses the data that is obtained with gs://paths to build RDD instead of getting the data locally.

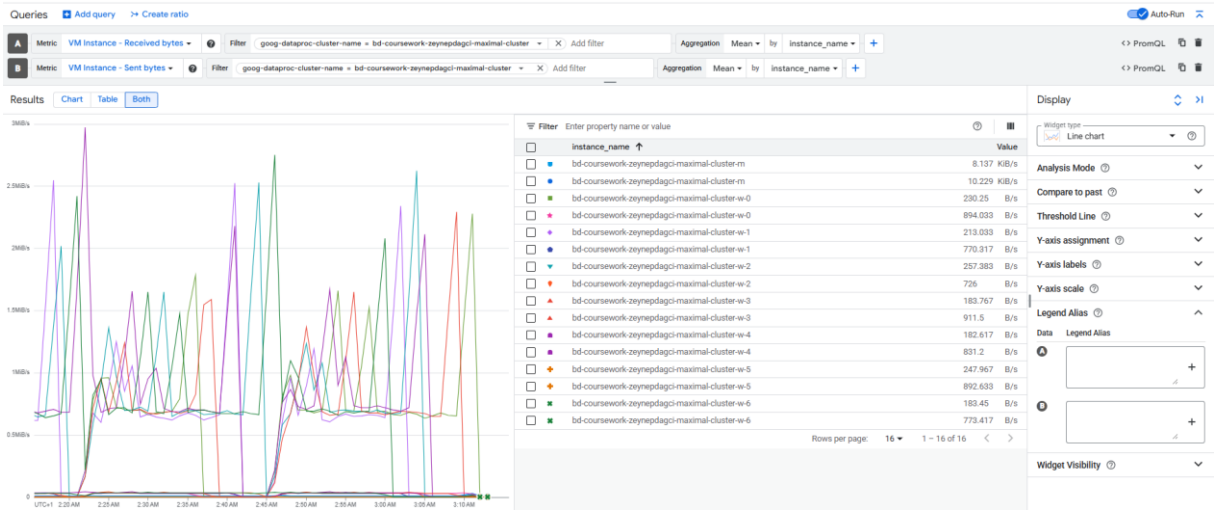
Prallelization is also used in the labs; however, compared the coursework, it uses a small dataset. In the labs, data is usually reshuffled using reduceByKey, while in the coursework, large file paths are parallelized as in sc.parallelize(filePaths, N). In one of the tasks, N=16, which means each of the 16 tasks is executed independently in parallel to process images.

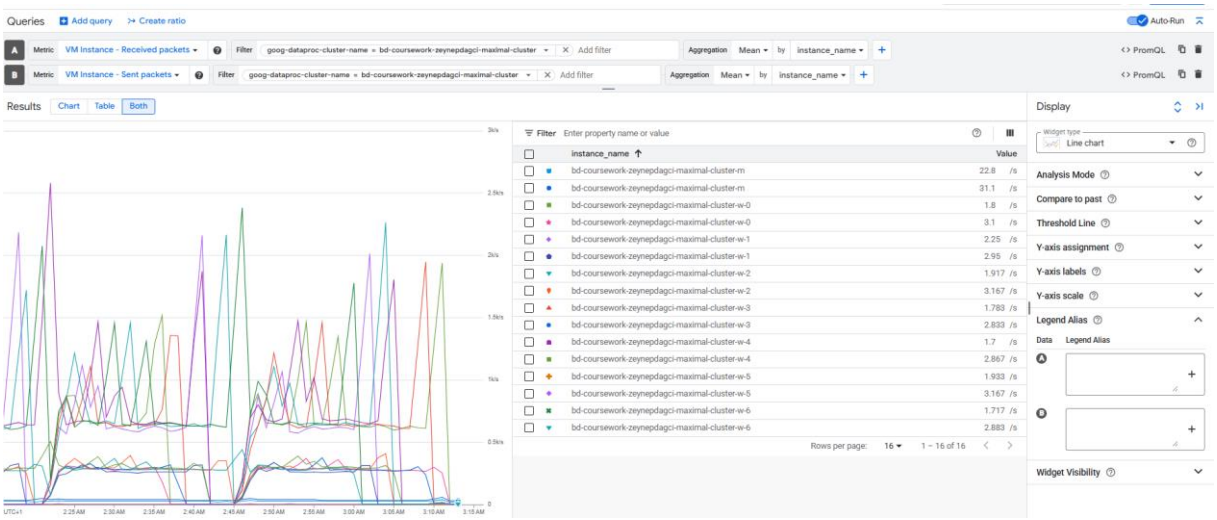
**Task 2b – part ii**

CPU Utilization:




Network Load:






**Task 2c**

After rdd = params_rdd.flatMap(run_speed_test), cache is implemented because this is the expensive computation which is better to be run once. The result is kept in the memory and then, when there is a second action, it extracts from the memory without running it again. Without caching, run_speed_test is rebuilt on every partition, however with caching it is rebuilt once and every action re-uses from the memory. Therefore, there is a 1.1 speedup (No-cache run: 6002.7s, cache run: 5488.1s).

**Task 2d**

**Interpreting the results:**

TFRecord files deliver higher throughput compared to JPEG by processing more images per second. Combination of batch size and batch number has a significant effect on the speed. Best option to achieve high processing speed is using TFRecord files and increasing batch size and batch number. For TFRecord, the batch size has a strong positive effect (~2.36 coef) which means throughput noticeably increased as batch size grows. r2 has low value when bs and ds are looked at individually, meaning they don't explain throughput on their own, however, bs x ds has around 0.86 for both JPEG and TFRecord, showing a strong combined effect. For intercept, when bs x ds is looked at, for TFRecord, throughput starts at higher (~90 images/s) increasing rapidly compared to JPEG (~4.5 images/s). As for p-value, the values are low which means all parameters are significantly important (p-values < 0.001), while bs x ds is the most important predictor among them for both JPEG and TFRecord files.

1. According to the average results in Colab, JPEG reads peaks around 10 images/s while TFRecord reads peaks around 265 images/s. "Latency numbers" gist discusses the following:
    - SSD random read = 150 µs
    - 1 MB sequentially read from SSD = 1 ms
    - Round trip within same datacenter = 0.5 ms
    - Packet round trip CA to Netherlands = 150 ms

Around 0.5 ms of network latency happens to fetch each image file and that is bigger than SSD random read 150 µs (0.15 ms) which is the actual cost of reading the image. In a large-scale machine learning application, it is better not to read images one by one over the network since a lot of computing power is used. Batching images while making the request makes the cost of the network spread out and it is faster.

2. On a single machine, data is read with high throughput and minimal latency from local storage, while reading from remote storage in the cloud comes with network latency. According to my findings, when batch sizes are big, throughput also increases. Cloud providers tie throughput to capacity of disk resources to make the resource use fair therefore, larger buckets have higher capacity of disk resources.

3. When tests are run in parallel on the cloud, Google Cloud bucket can handle multiple read requests at the same time, however there is also the possibility of bottlenecks (e.g., network bandwidth, disk I/O), latency, provider throttling. When compared my results of throughput, TFRecord showed higher speed because of the larger and sequential reads which avoids those bottlenecks, while JPEG's throughput slightly improved in parallel.

4. By looking at linear regression, it is possible to understand the speed effect of batch size and batch numbers. It shows pattern such as bigger batches results in faster processing as in TFRecord files. On the other hand, cloud systems are more complex than linear regressions, therefore, even though they can predict trends, real-world limitations can occur (e.g., bottleneck, capacity limits).

**Task 3a**

CherryPick's goal is to build a lightweight model using a small dataset to test cluster configurations in cloud without the heavy workload of testing every configuration manually since there can be many possible VM instance types or cluster sizes. A small number of tests is run, and a performance model is built using those results, and for the next set of configurations, that model is used without the need for testing everything.

For instance, in task 1a and 1d, the generated data is measured in time by running TFRecord files on 2 partitions, and on 16 partitions. Elapsed time is measured for both cases and compared. Therefore, for the relevance, the time measurements can be used in CherryPick as a small set of configurations of 2 and 16 partitions. The time measurements can be used for the performance model to perform the following tasks: fitting a simple model such as time measurements vs partition numbers, predicting the speed of using those partition numbers, testing only predicted ones instead of using brute-force search. Also, for the creation of cluster in task 1c and 1d, every configuration is created and time is measured for them manually to see the effects of write time with the machine size and number of nodes, however CherryPick would do this automatically and give the best possible configuration.

In task 2, comparison of reading from JPEG and TFRecord files are performed using different parameters such as number of batch sizes and batch numbers. CherryPick can take the time measurements such as images/sec at batch_size, 8 or 16 and its model is trained on the small

number of tests, which results in the prediction of optimal batch configurations without testing every possible combination for the best results.

**Task 3b**

For batch processing, following examples will be discussed: monthly report generation, data conversion, payroll processing. All the scenarios, use large datasets and CherryPick can be used to find the fastest or the cheapest cluster configuration to finish the tasks.

1. Monthly report generation:
    a. Financial information, work plans, operations can be compiled into reports automatically.
2. Data conversion:
    a. Choosing the best possible cluster configuration is important since big datasets may be in the need of conversion from one format to another (e.g., CSV to JSON)

Online processing processes data input in real-time without significant delays while managing the workload. Low-latency and high-throughput configurations for handling continuous and fast data flows are important.

1. Fraud detection:
    a. Financial transactions are monitored since suspicious activities can occur. Real-time processing is needed with high accuracy, and usually machine learning models are used to detect it faster and accurately.
2. Real-time stock recommendation
    a. User is suggested products based on their clicks on the products

For stream processing, new incoming data is grouped and processed periodically (e.g., per second)

1. Social media stream:
    a. Social media platforms can be collected for sentiment analysis or trends in small time windows such as per 20 seconds, and it can be processed together.
2. Price monitoring:
    a. Product prices are tracked from time to time (e.g., every few minutes or seconds) and in the case of change or another condition, an alert or an automated action is triggered. The system should process data and respond quickly.

**General Relationship to CherryPick and the Coursework**

Leveraging Bayesian Optimization, CherryPick builds performance models for several applications. It is built to find best cloud configuration (e.g., CPU, VM type, memory size).

**Batch processing (monthly report generation, data conversion):**

Samples: A small dataset can be selected, 8% of this month's sales for reporting and 2 GB of CSV data for conversion. Using 3 Dataproc clusters (e.g., 8x1 vCPU, 4x2 vCPU, 1x8 vCPU as in task 1c and task 1d), they can be tested. Two spark partitions number can be tested within each cluster, 2 and 16 partitions as in task 1a task 1d, and wall-clock time and vCPU hour are measured.
Models: A linear regression is performed to predict time or cost per GB. As in task 1, 2 vs 16 partitions can be compared, and linear regression can be done as in task 2.

**Online processing (fraud detection and real-time stock recommendation):**

Samples: 5% of the live traffic can be taken from financial transactions and product clicks to avoid overloading the system which aligns with the low overhead evaluation approach of CherryPick. That small dataset is duplicated and processed in parallel while each of them running with different configurations as it is done in task 2a and task 2b to compare JPEG and TFRecord file reads.

Models: A Bayesian model is used for the estimation of the expected per-event latency in different configurations settings as it is used by CherryPick, and this way it can predict which configuration is best using small number of samples.

**Stream processing (social media stream, price monitoring):**
Samples: Incoming data from social media and price monitoring system can be grouped together in small time windows such as 20 seconds. Using 3 different configurations such as 2, 4, 8 executors, processing time, cost and latency can be measured.
Models: A Bayesian Optimization model can be used to predict processing time of the windows based on number of partitions and cluster size. Another approach would be to use linear regression as in task 2d, then extend it to the Bayesian model to capture non-linear effects.

**Conclusion**
For all three scenarios, same pattern is followed: selecting a small dataset, building a lightweight performance model, using the model for optimization to avoid heavy and expensive workload of exhaustive search. Task 1 and task 2 have the data (e.g., TFRecord file write times, number of partititions, cluster configurations, batch sizes and batch numbers) that CherryPick can automate.

**Word Count:** 1926