**1d)**

When I tried to create the Maximal cluster, I triggered the quota error. Also, I had to change the region to ‘southamerica-east1’. The evidence for these is provided in Fig. 1, Fig. 2, and Fig. 3 in the screenshots section of Appendices.

**i)**

For the parallelization purpose, values 2, 4, 8, 16, 32, and 64 are being passed as the second parameter to the parallelize method. Below, Fig 1, 2, 3, 4, 5, and 6 show the resource utilization for parameter values 2, 4, 8, 16, 32, and 64 respectively. Fig 7 demonstrates the overall usage during the execution. The processing time can be found in the table.

**Simple Cluster**

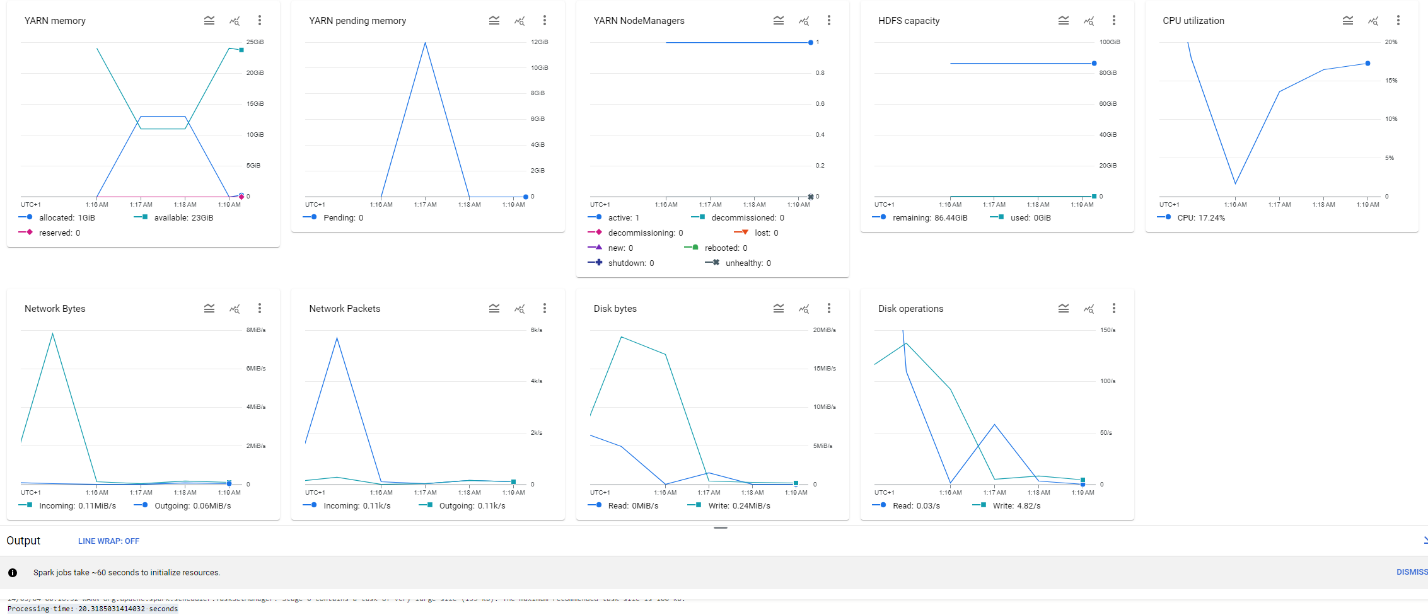
****

Fig. 1

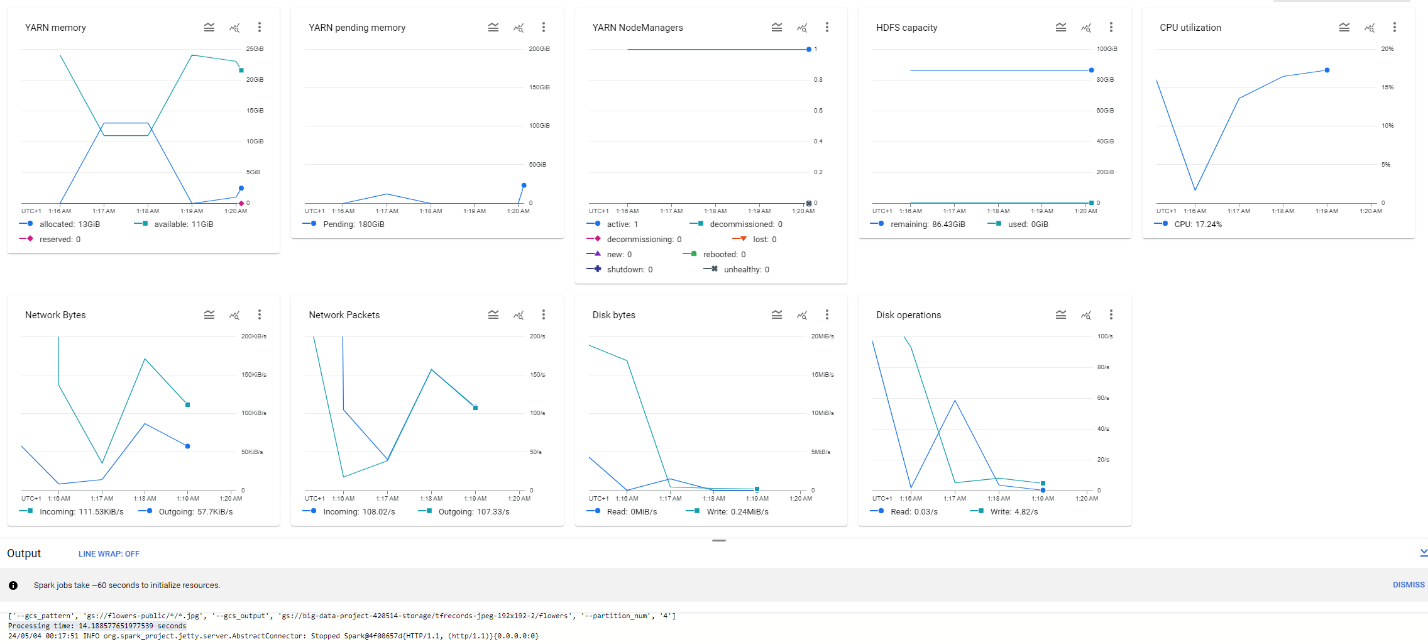
****

Fig. 2

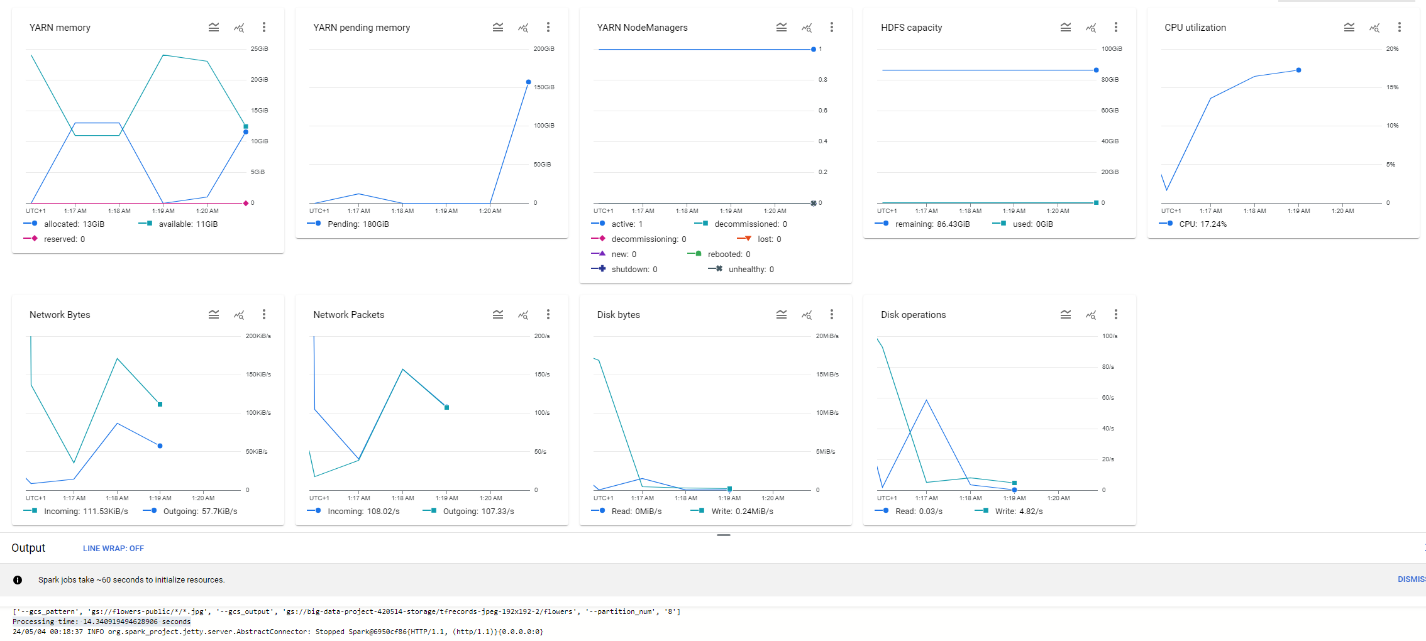
****

Fig. 3

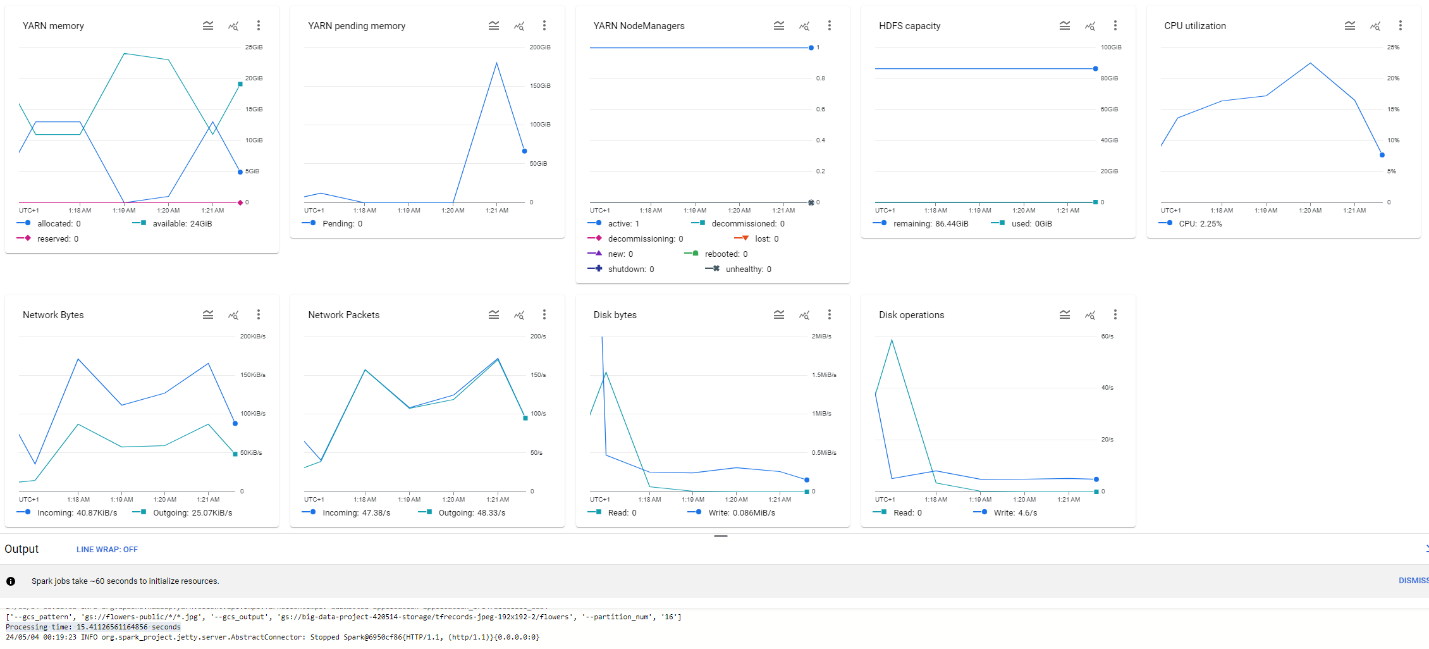
****

Fig. 4

****

Fig. 5

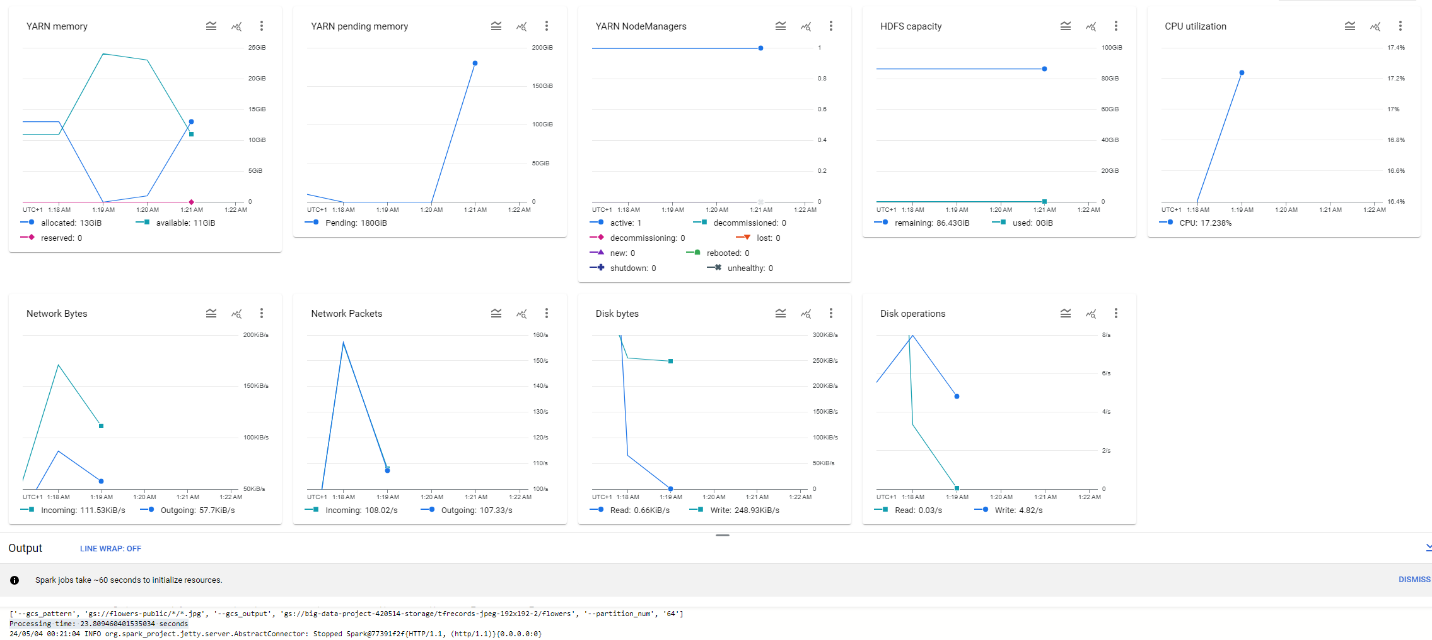
****

Fig. 6

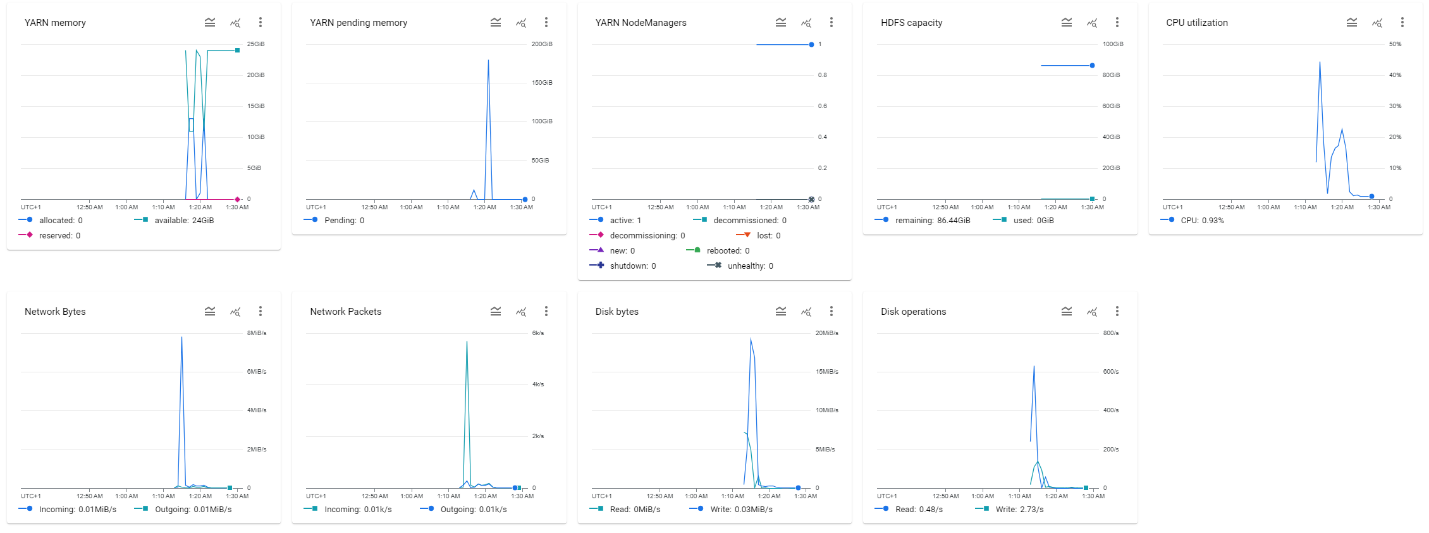


Fig. 7

**Maximal Cluster**



Fig. 1

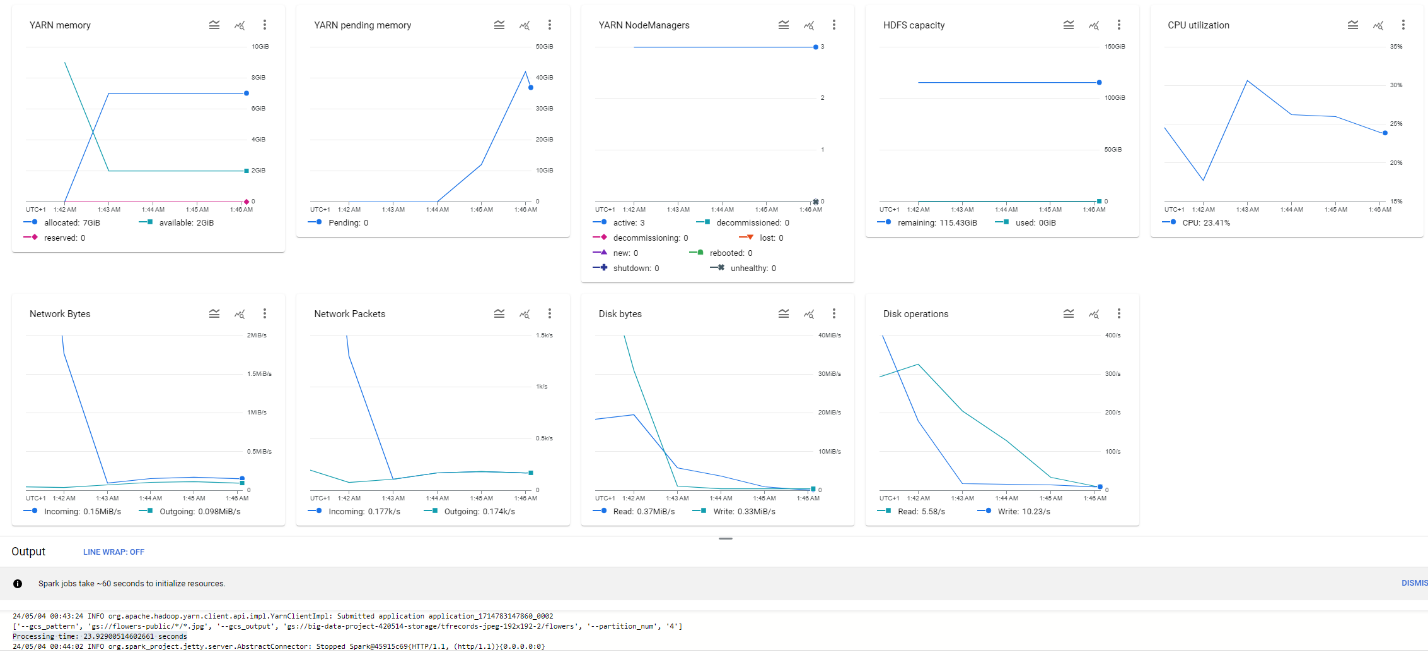


Fig. 2

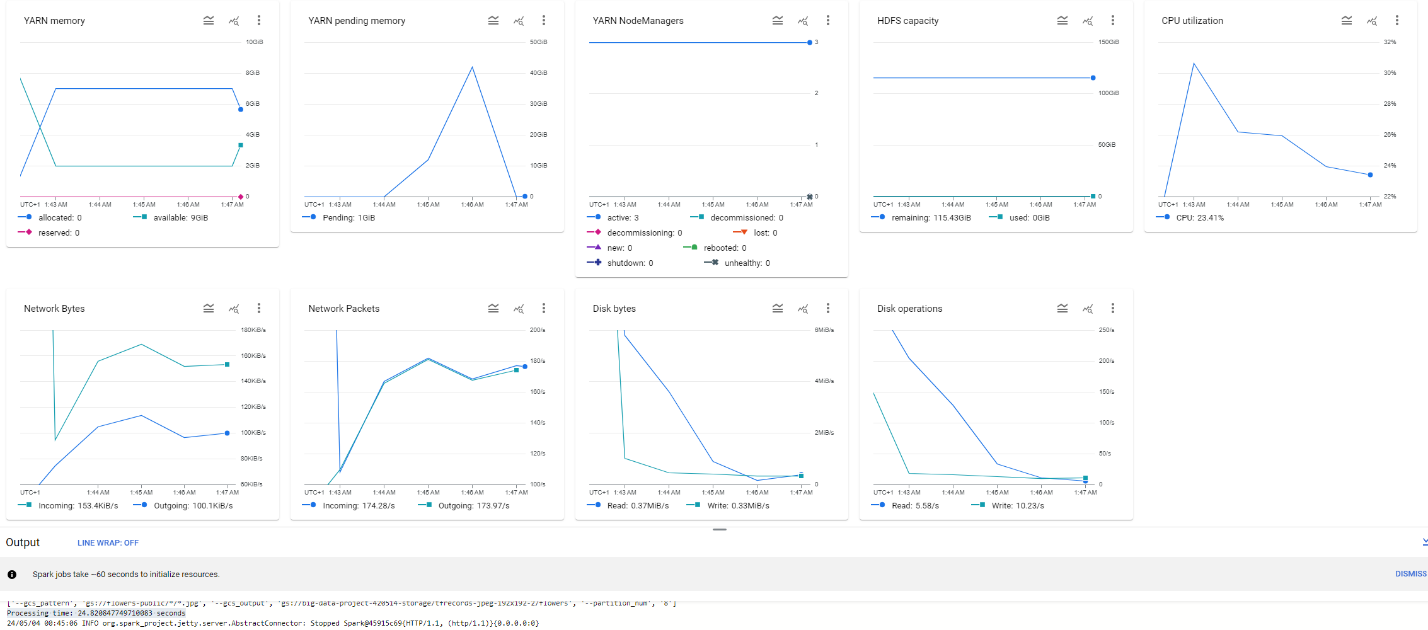


Fig. 3

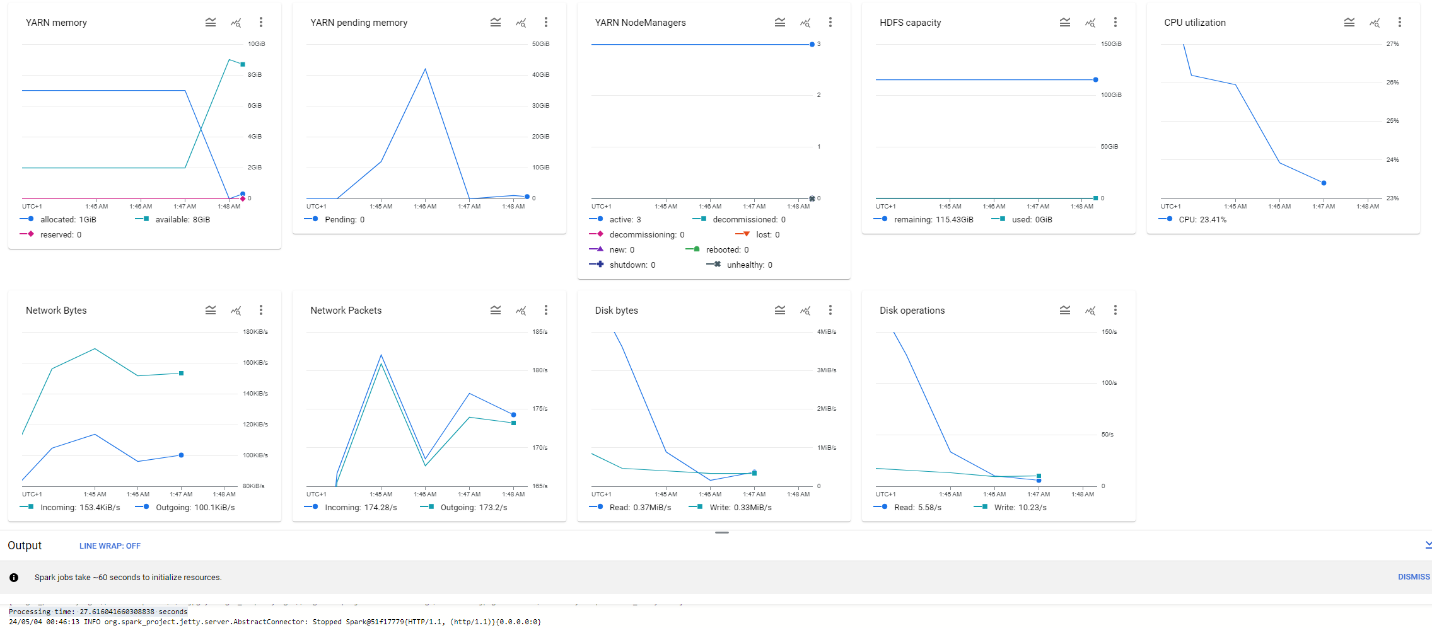


Fig. 4



Fig. 5

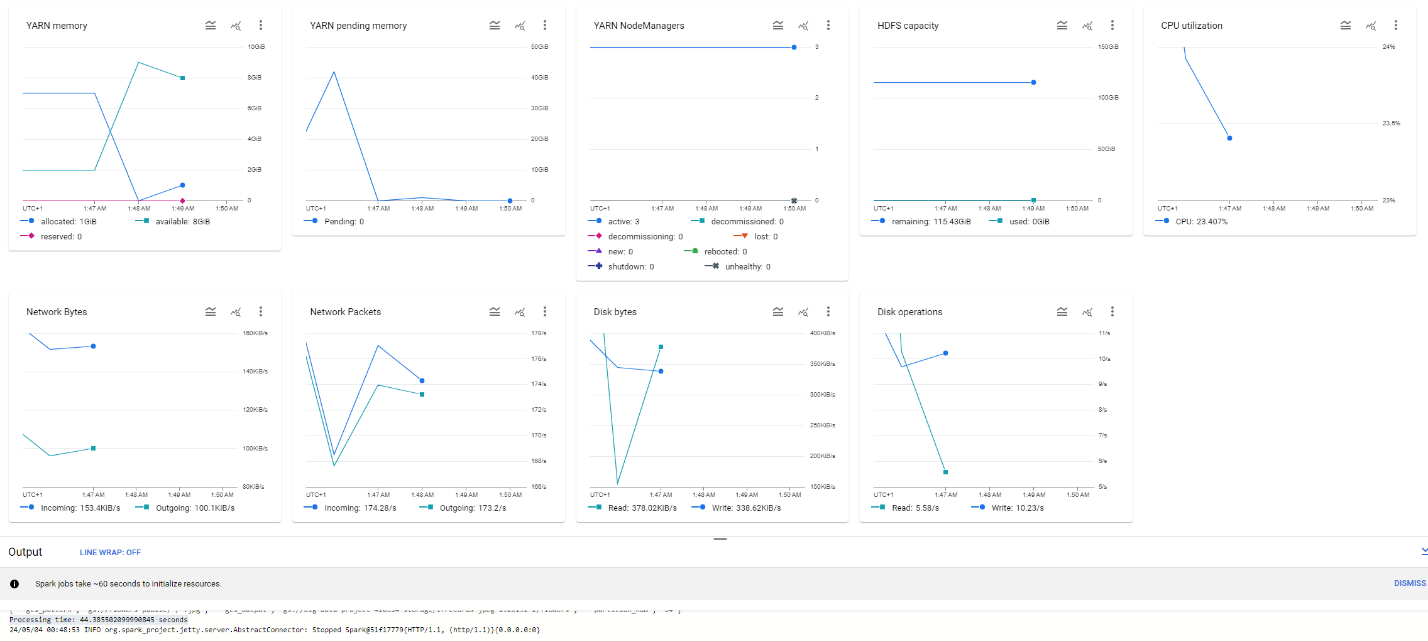


Fig. 6



Fig. 7

**Table 1 – Processing Times**

|  |  |  |
| --- | --- | --- |
| **Parameter Value** | **Simple Cluster Processing Time (s)** | **Maximal Cluster Processing Time (s)** |
| 2 | 20.3185031414032 | 25.092011213302612 |
| 4 | 14.188577651977539 | 23.92900514602661 |
| 8 | 14.340919494628906 | 24.820847749710083 |
| 16 | 15.41126561164856 | 27.616041660308838 |
| 32 | 19.65857768058777 | 33.52991580963135 |
| 64 | 23.809460401535034 | 44.385502099990845 |

**ii)**

Again, we have the figures and a table for 4-double Cluster and 1-eightfold Cluster.

**4-double Cluster**

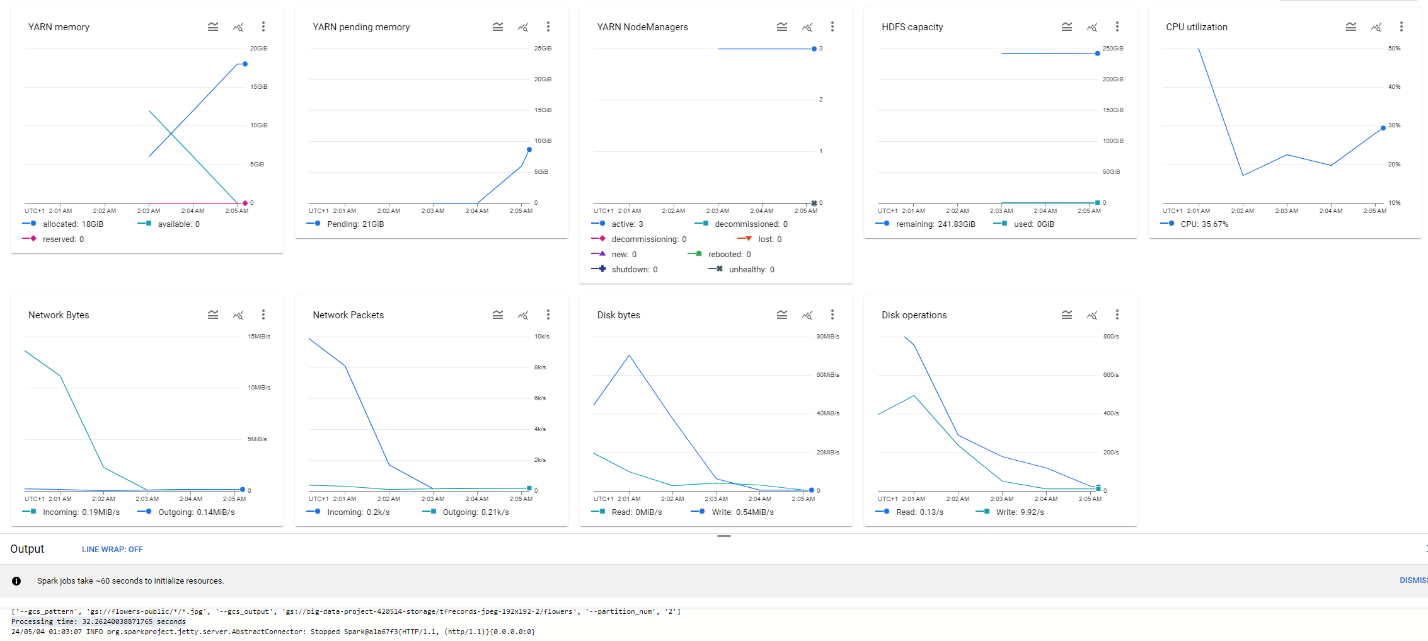
****

Fig. 1



Fig. 2

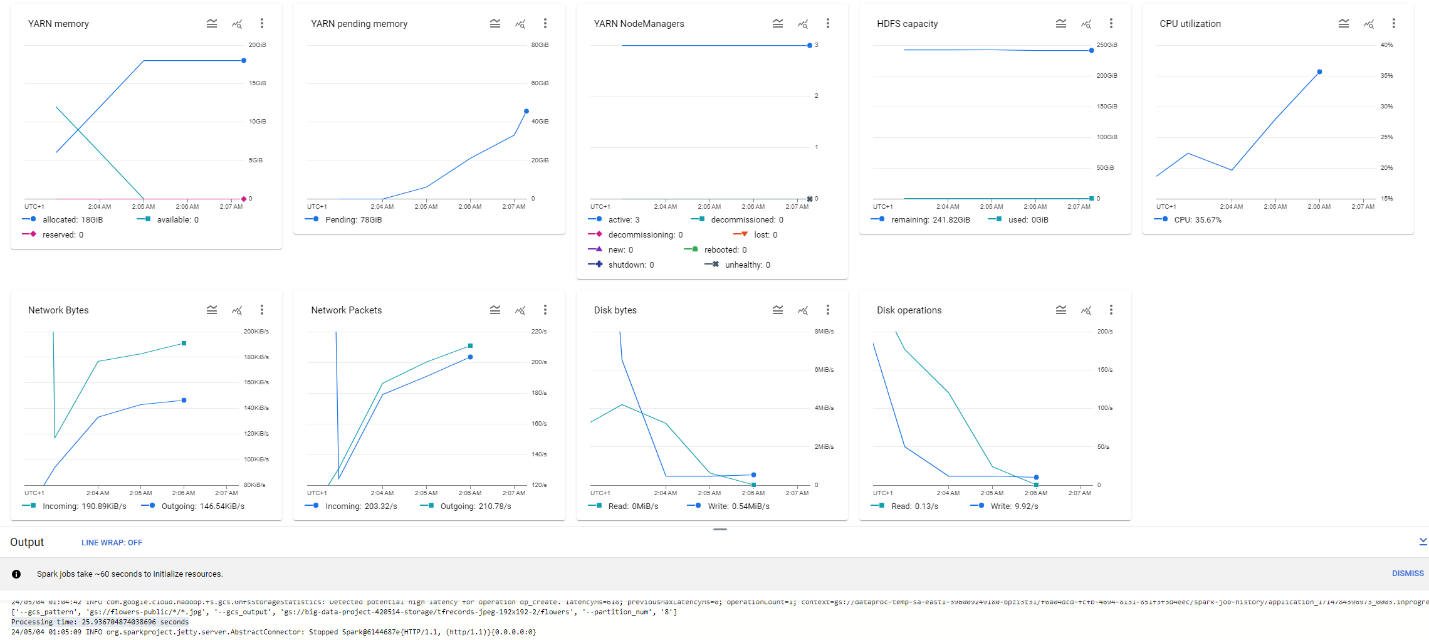


Fig. 3



Fig. 4

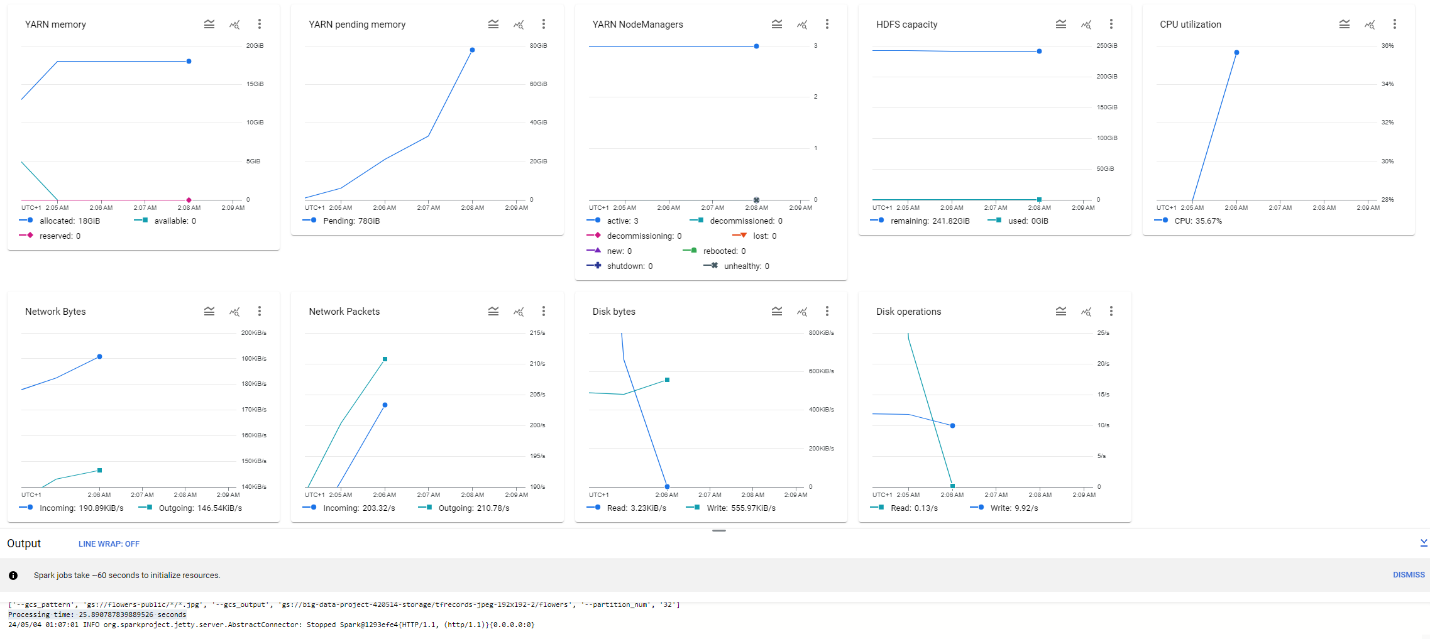


Fig. 5



Fig. 6

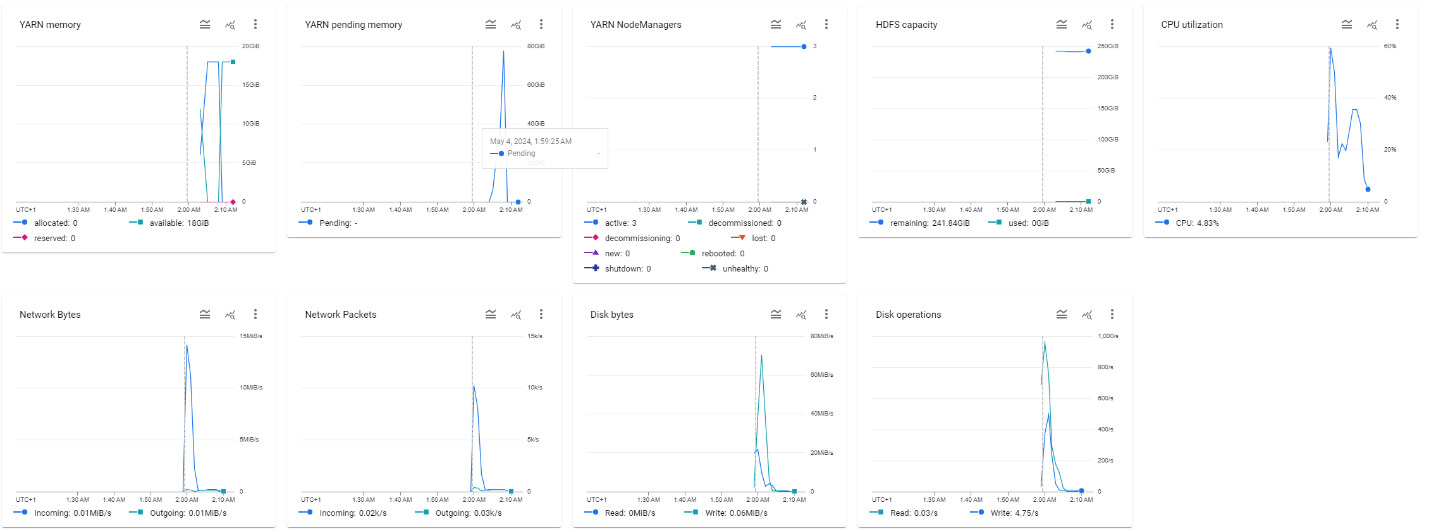


Fig. 7

**1-eightfold Cluster**

****

Fig. 1

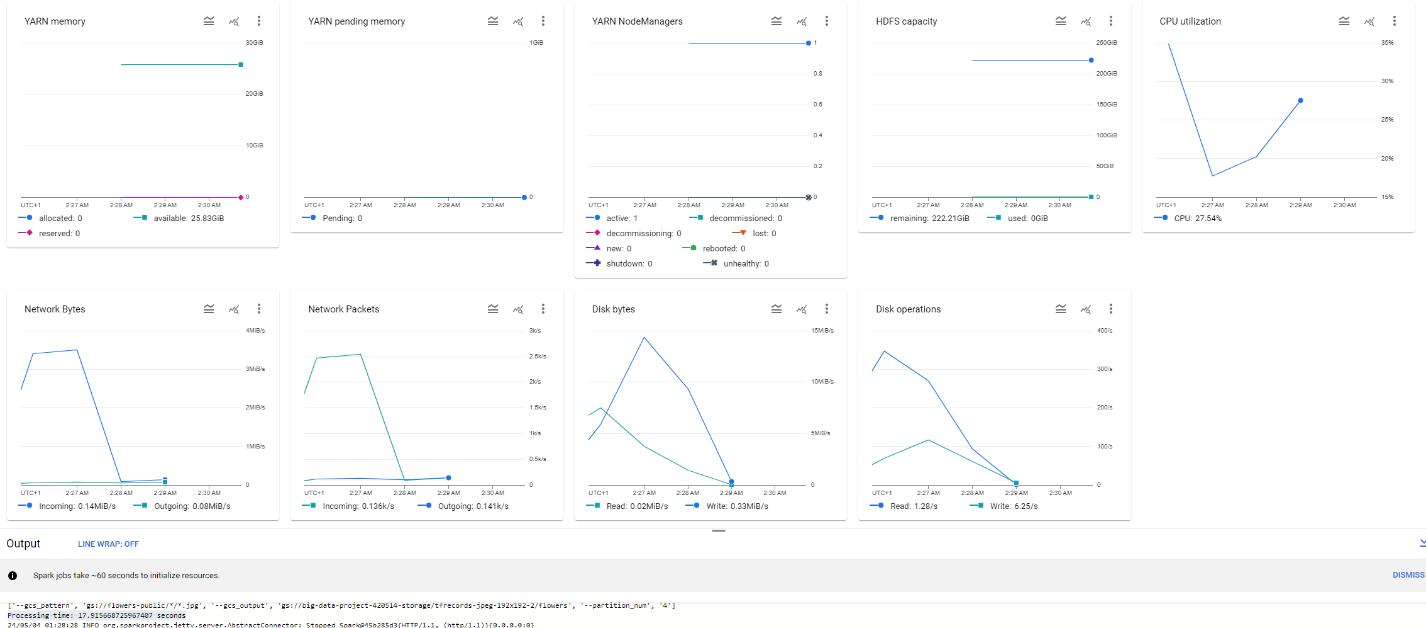


Fig. 2

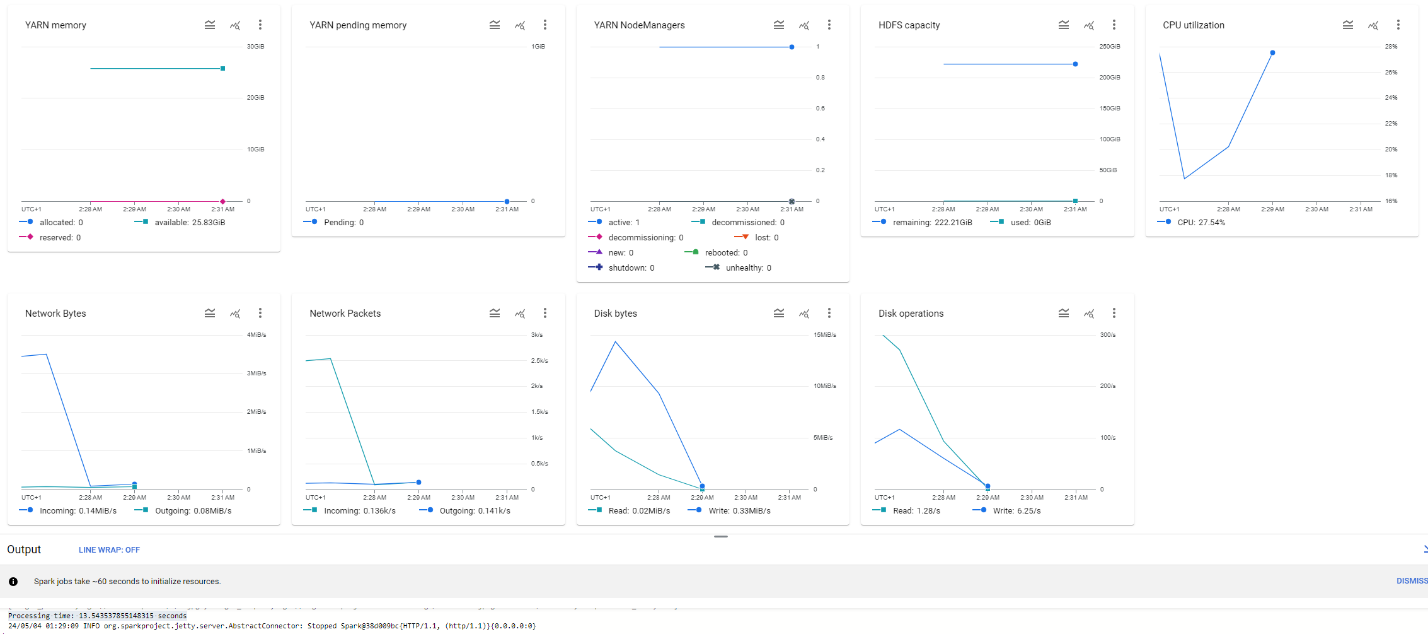


Fig. 3

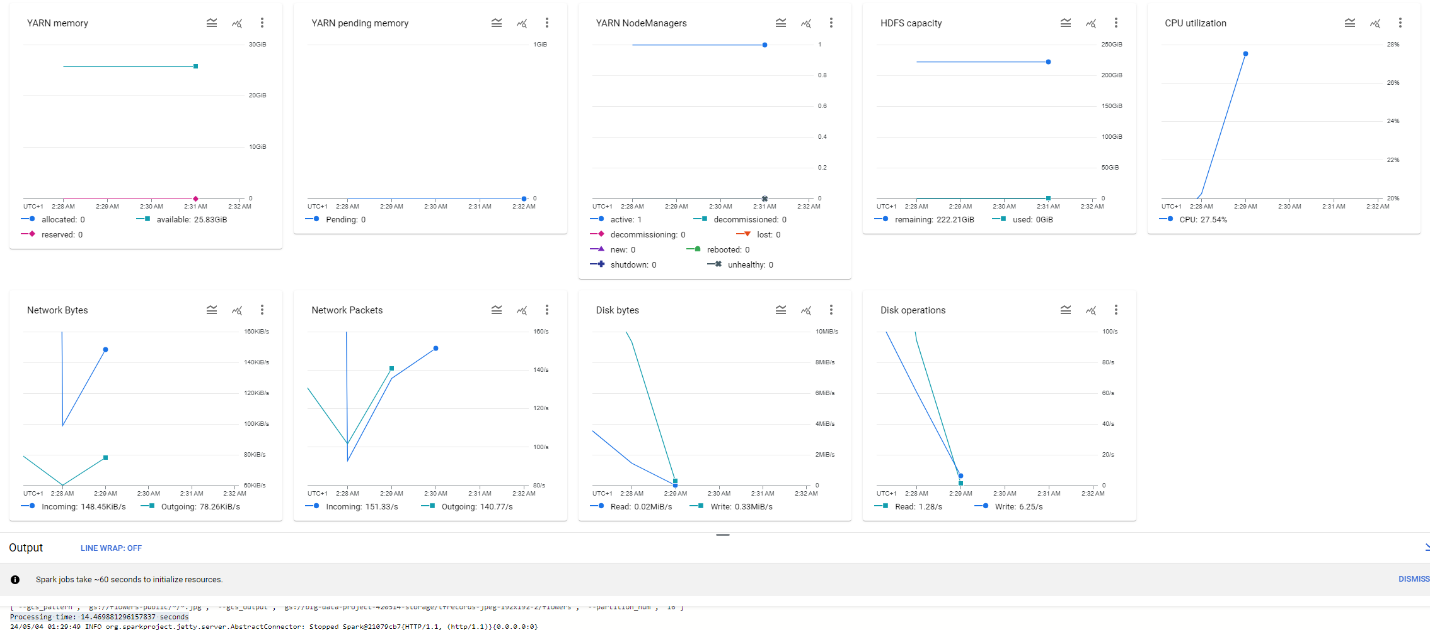


Fig. 4

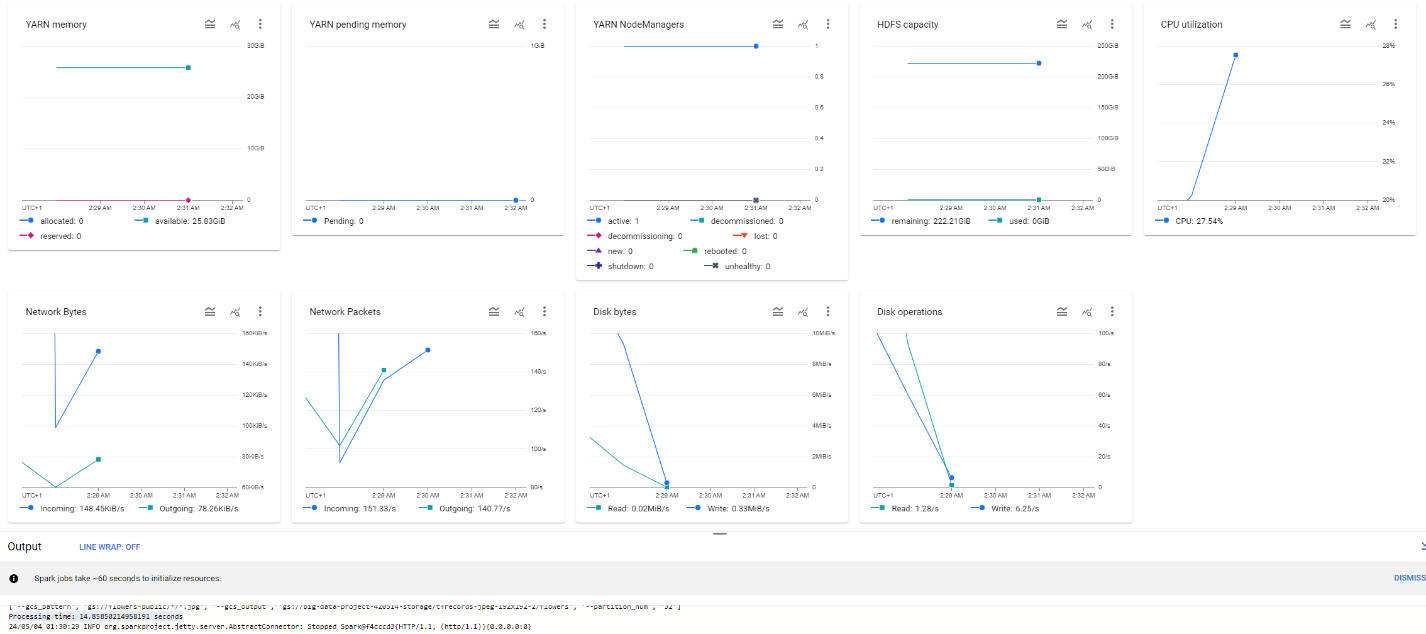


Fig. 5

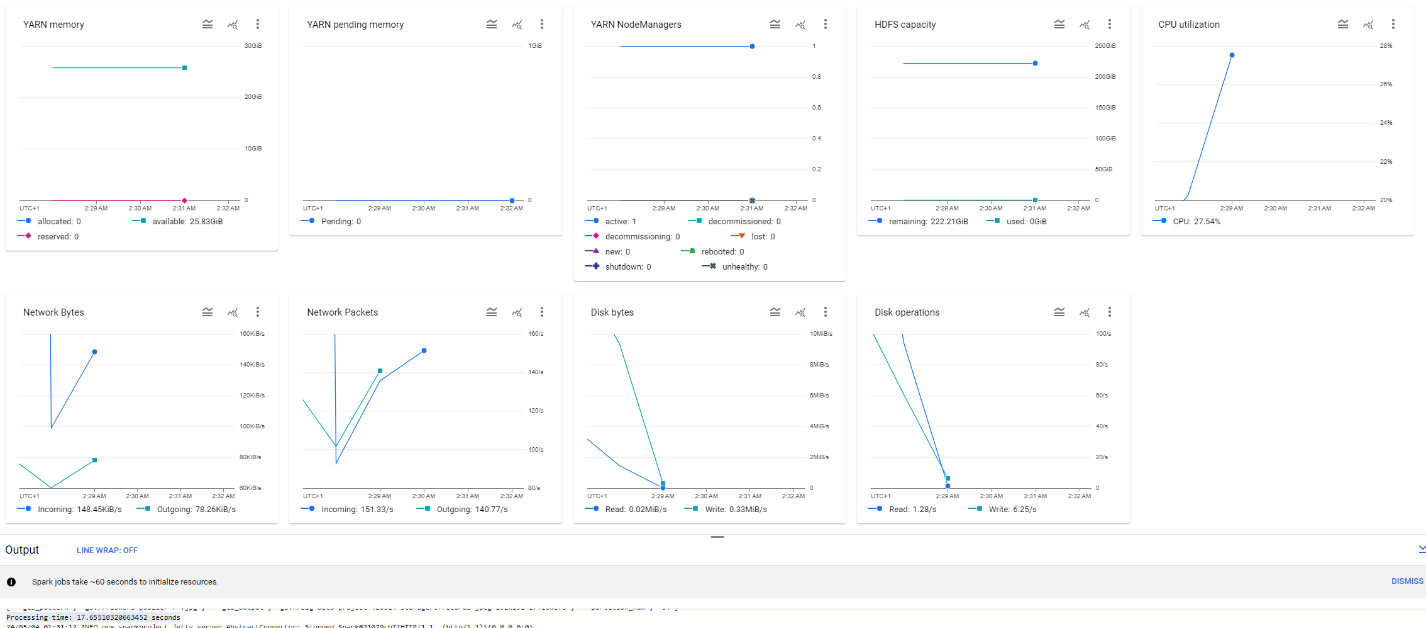


Fig. 6

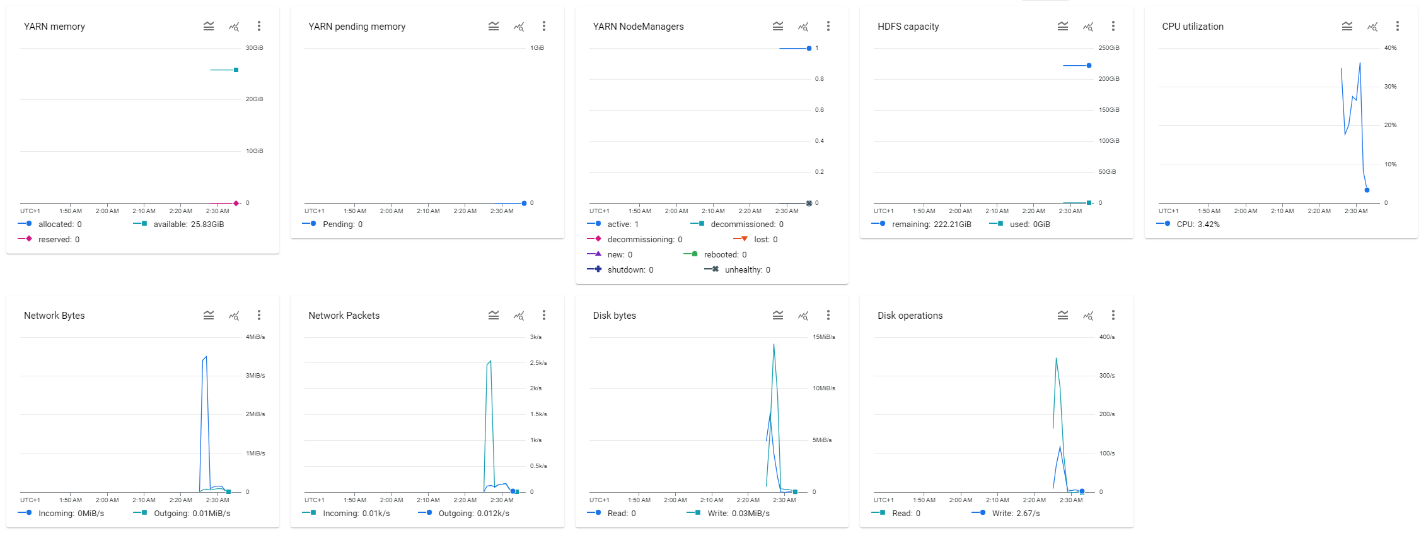


Fig. 7

**Table 1 – Processing Times**

|  |  |  |
| --- | --- | --- |
| **Parameter Value** | **4-double Cluster Processing Time (s)** | **1-eightfold Cluster Processing Time (s)** |
| 2 | 32.26240038871765 | 25.566425561904907 |
| 4 | 29.69374680519104 | 17.915668725967407 |
| 8 | 25.936704874038696 | 13.543537855148315 |
| 16 | 24.78178119659424 | 14.469881296157837 |
| 32 | 25.890787839889526 | 14.85850214958191 |
| 64 | 28.3889799118042 | 17.65510320663452 |

The processing time will generally increase by improving the penalization, and after a certain point, it might have diminishing returns.

**Disk I/O:**

Higher processing time might be an indication of increased I/O activity that includes frequent reads and writes to storage due to data shuffling or spillage while performing RDD operations. Overall, the disk I/O time has decreased by improving parallelization. It seems that the I/O time has decreased significantly within the 1-eightfold Cluster by improving the parallelization. In the Maximal Cluster and 4-double Cluster, the disk I/O time for lower parallelization value is relatively high, however, this significantly decreased by increasing the parallelization level.

**Network bandwidth Allocation:**

The Simple Cluster and 1-eightfold cluster seem to have less overall network bytes and network packets dissemination over time. This also occurs in a shorter time period when compared to the Maximal Cluster and 4-double Cluster that keep sending packages throughout the time. The Maximal Cluster is the slowest in terms of network communications followed by the 4-double Cluster. It can be said that larger clusters may have higher bandwidth capability. Also, they experience a longer duration of network activity, which is due to data transfer and/or resource contention. This behavior is because typically larger clusters possess more nodes and that means higher volumes of data exchange occur between the nodes, which will heighten the overall network activity.

**iii)**

In lab sessions, we mostly used Spark locally on a single machine (Google Collab VM instance). The data was obtained from local files or databases and processed locally with limited parallelization. However, in this coursework, the focus was mainly on the parallelization of tasks in Spark and the distribution of data. The data was also gathered through glob patterns. This allows multiple computations to be performed simultaneously through partitioning and chunking of data. It has also known benefits regarding processing speed, computational power, and scalability. Also, the type of data we worked with here was structured image data whereas in lab sessions we mostly utilized unstructured data such as text files.

**2c)**

Caching can improve the efficacy of Spark jobs via storing the intermediate RDDs in memory reduces the need for several recomputes. Since the same RDD is being accessed multiple times, both for images and tfrecrod, caching them is a reasonable choice. I tested three different scenarios:

1. No Cache: all the cache() calls are commented, thus, not applied.



Fig. 1

2. Caching ‘img\_results\_rdd’ and ‘tf\_results\_rdd’ (Cache Two)

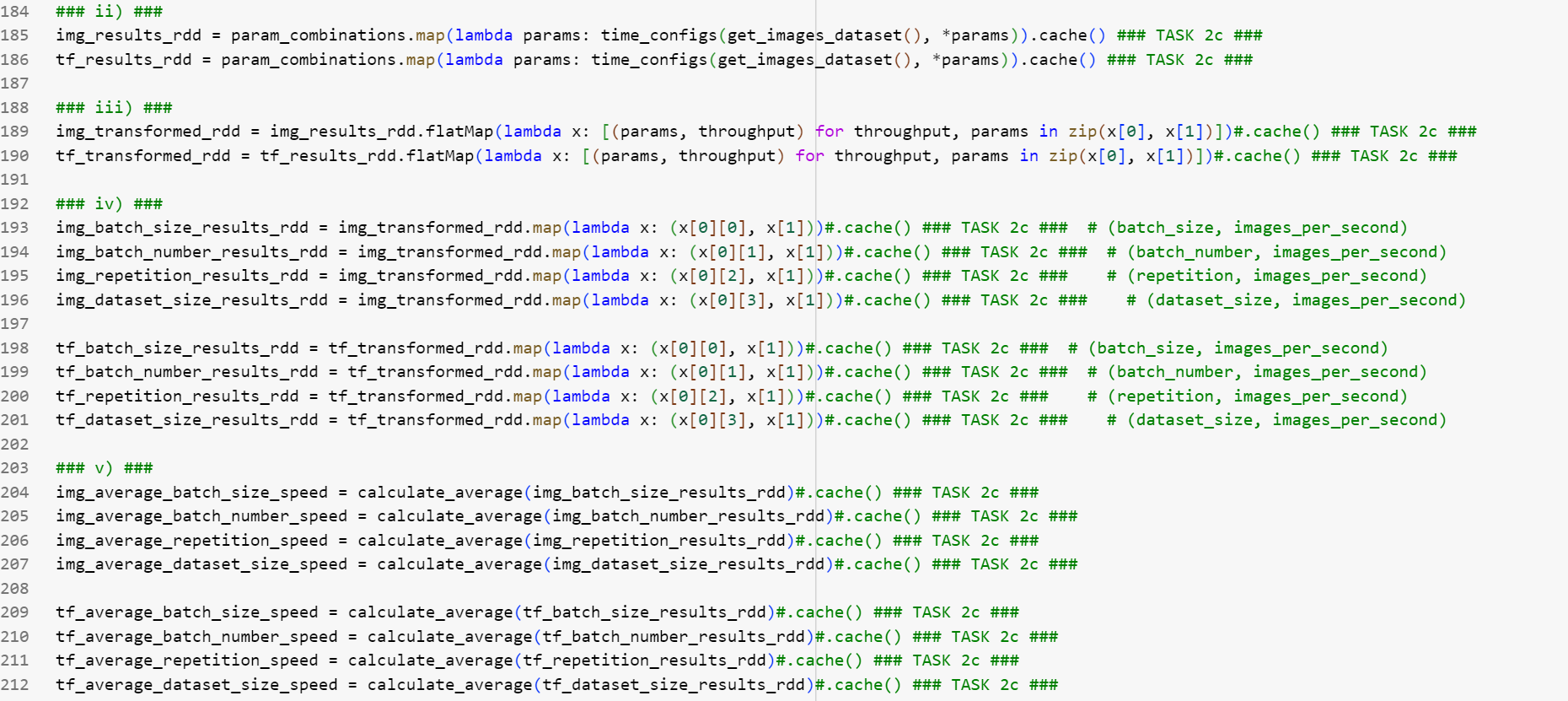


Fig. 2

3. Intensive Caching (Caching all RDDs)



Fig. 3

This test was conducted using the default parameters sets:

batch\_sizes = [2,4]

batch\_numbers = [3,6]

repetitions = [3]

It is also noteworthy that the ‘param\_combinations’, the code shown in Fig.4, is not cached in any scenarios. This is because the resulting RDD is not computationally heavy, given the fact that we have a limited number of parameter values.



Fig. 4

Then, I ran the code locally three times for each scenario. Here is the table of results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Scenario** | **Run #1 Speed (s)** | **Run #2 Speed (s)** | **Run #3 Speed (s)** | **Avg Speed (s)** |
| No Cache | 176.32 | 172.44 | 180.88 | 176.54 |
| Cache Two | 51.86 | 51.22 | 48.36 | 50.48 |
| Intensive Caching | 62.19 | 50.79 | 50.94 | 54.64 |

Table 1

I also repeated the same in the cloud (only once for each scenario due to the time limitations):

|  |  |
| --- | --- |
| **Scenario** | **Run Speed (s)** |
| No Cache | 695.22 |
| Cache Two | 108.38 |
| Intensive Caching | 95.99 |

Table 2

It is obvious that caching enhances the execution process and improves the speed. However, choosing between caching the two lines (scenario 2) and intensive caching (scenario 3) is a bit challenging. I have decided to choose the second scenario and only apply caching to ‘img\_results\_rdd’ and ‘tf\_results\_rdd’ RDDs. It is because the third scenario, where we cache all possible RDDs will become a bottleneck when executing for a larger number of parameters. It is also important to notice that caching requires space in the memory and might result in memory issues if applied intensively. This is despite the fact that the current intensive caching scenario shows a slightly better performance for our small parameter sets.

The cloud utilization graphs for the three scenarios are represented by Fig. 1, Fig. 2, and Fig. 3 in order.

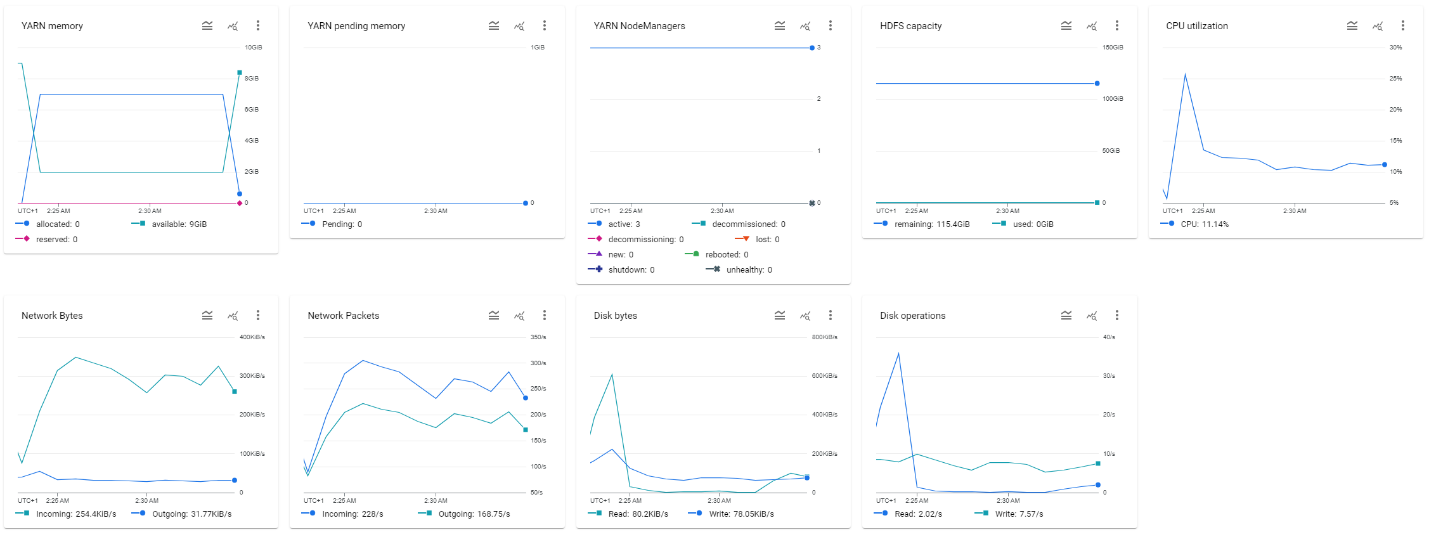


Fig. 1

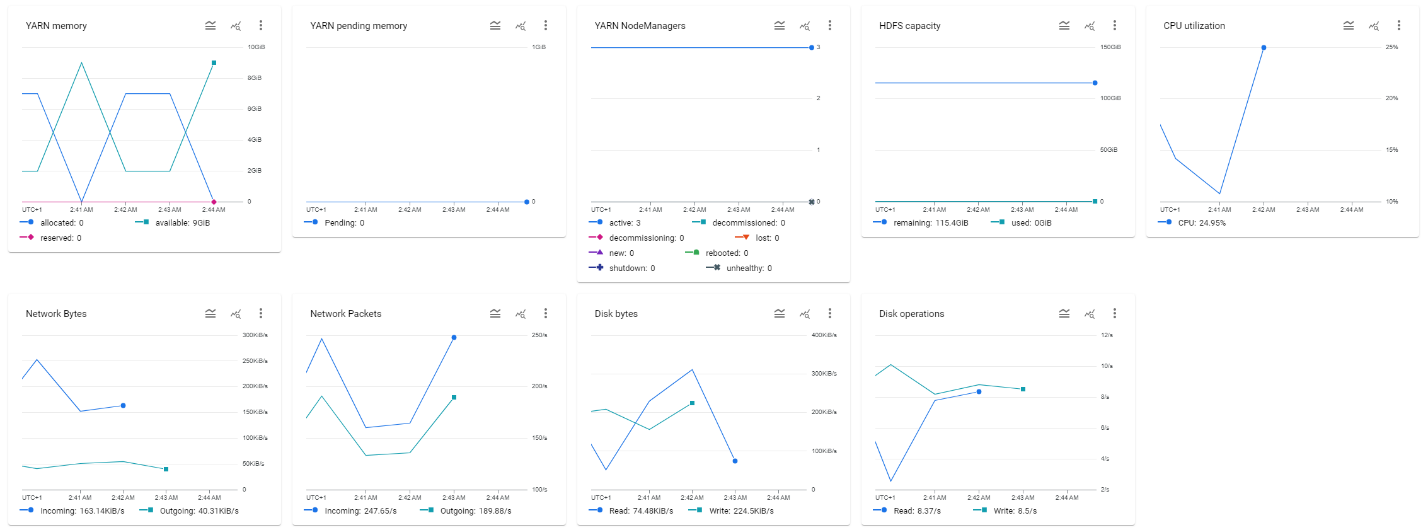


Fig. 2



Fig. 3

**2d)**

Table 1 summarizes the coefficient and intercept values for regression analysis of 16 sets of values (for both average value RDDs and Raw value RDDs). Also, the Fig. 1 to Fig. 16 displays their respective graph with the regression lines. From the figures, it’s clear that the average values result in higher accuracy than using the raw data. Also, the image files are more accurately represented by the regression lines than their respective tfrecord values.

Using average values results in higher accuracy which has significant implications for large-scale machine learning projects through reducing data transfer and processing time. This is crucial to minimize latency in accessing latent data.

The throughput linked to disk resources in the cloud is not similar to what we would expect on a single machine. Here, leveraging parallelism helps scalability and handling large data volumes effectively.

In a cloud environment, there are certain bottlenecks such as resource contention and network congestion that should be considered while optimizing the performance in distributed systems.

Linear modeling aids in understanding performance trends. However, the practical complexities like resource variability require to be validated based on another mechanism like the performance analysis graph provided by Google Cloud.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Coefficient** | **Intercept** |
| img\_average\_batch\_size\_speed | 0.020893520009728613 | 3.215122420556476 |
| img\_average\_batch\_number\_speed | -0.007791416331963853 | 3.378025643094847 |
| img\_average\_repetition\_speed | -0.04840501075392807 | 3.4406025474899384 |
| img\_average\_dataset\_size\_speed | 0.0007601438243554665 | 3.275423065788064 |
| raw\_img\_batch\_size\_results\_rdd | 0.020893520009728613 | 3.2151224205564755 |
| raw\_img\_batch\_number\_results\_rdd | -0.007791416331963808 | 3.3780256430948468 |
| raw\_img\_repetition\_results\_rdd | -0.048405010753928134 | 3.440602547489939 |
| raw\_img\_dataset\_size\_results\_rdd | 0.0003944708327434716 | 3.304797364377238 |
| tf\_average\_batch\_size\_speed | 0.00976629873269894 | 3.1649244638788714 |
| tf\_average\_batch\_number\_speed | -0.0032303202551130874 | 3.2379833594557144 |
| tf\_average\_repetition\_speed | -0.003812193415756138 | 3.2232864410817568 |
| tf\_average\_dataset\_size\_speed | -0.00020949462012029182 | 3.229572311245217 |
| raw\_tf\_batch\_size\_results\_rdd | 0.009766298732699012 | 3.164924463878871 |
| raw\_tf\_batch\_number\_results\_rdd | -0.0032303202551131113 | 3.2379833594557144 |
| raw\_tf\_repetition\_results\_rdd | -0.003812193415756144 | 3.2232864410817568 |
| raw\_tf\_dataset\_size\_results\_rdd | 0.0005231826412228078 | 3.194136608496511 |

Table 1

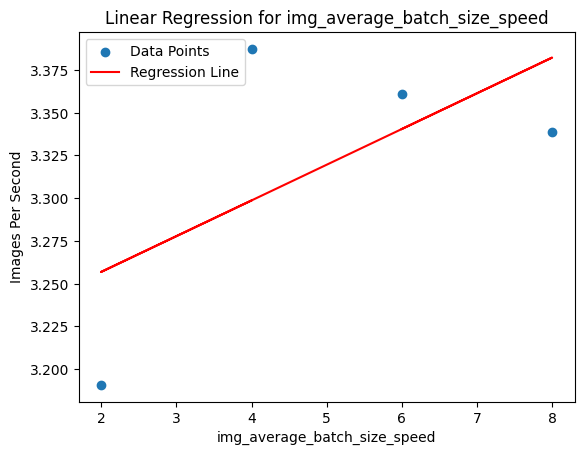
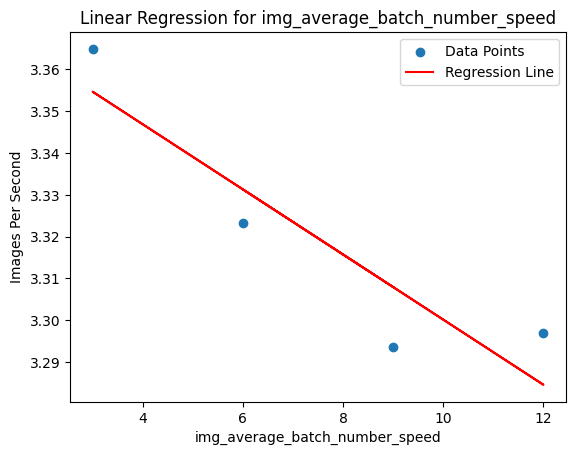
****

Fig. 1

Fig. 2****

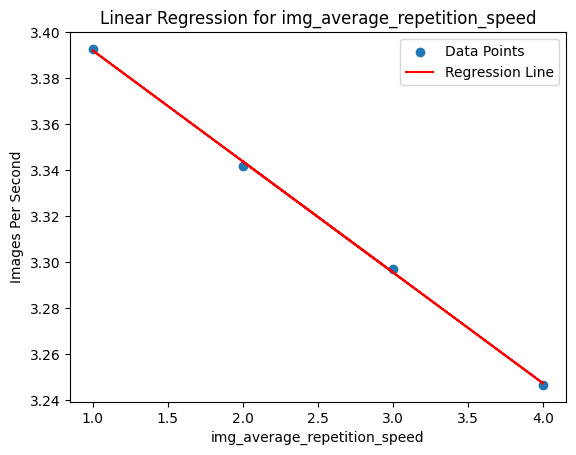
****

Fig. 3

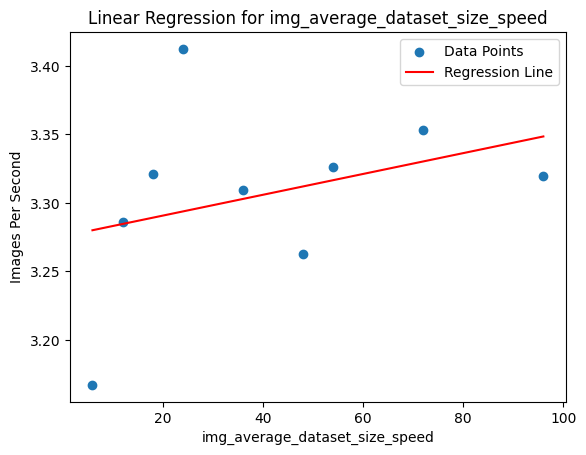
****

Fig. 4

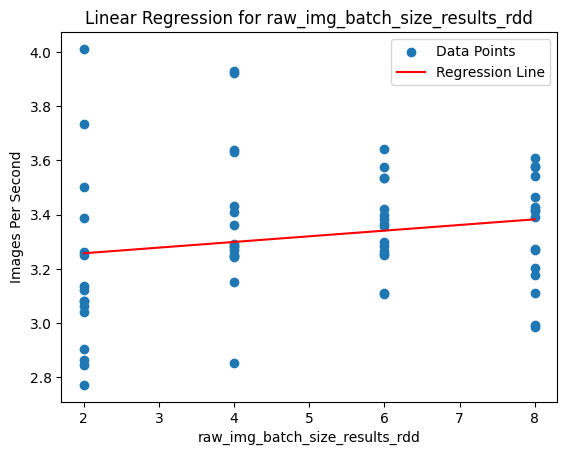
****

Fig. 5

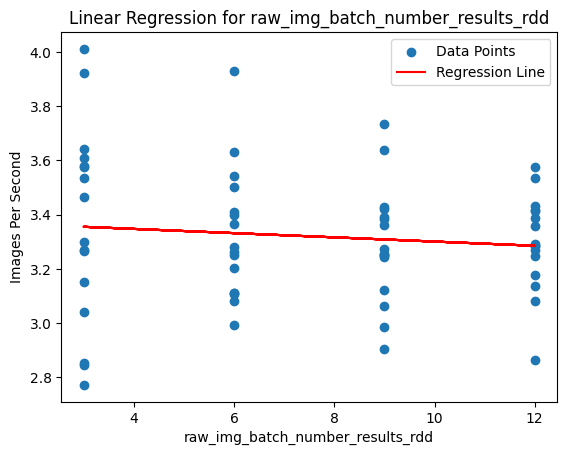
****

Fig. 6

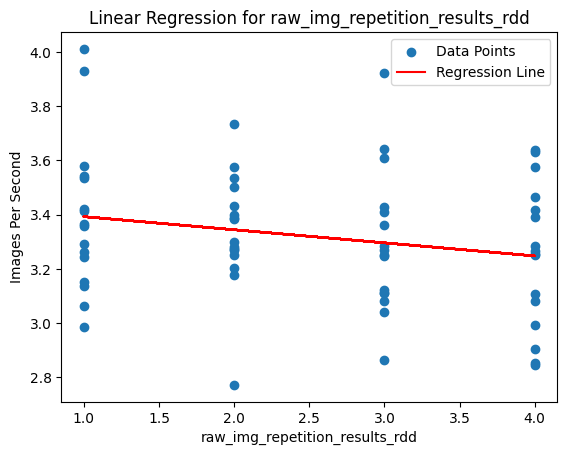
****

Fig. 7

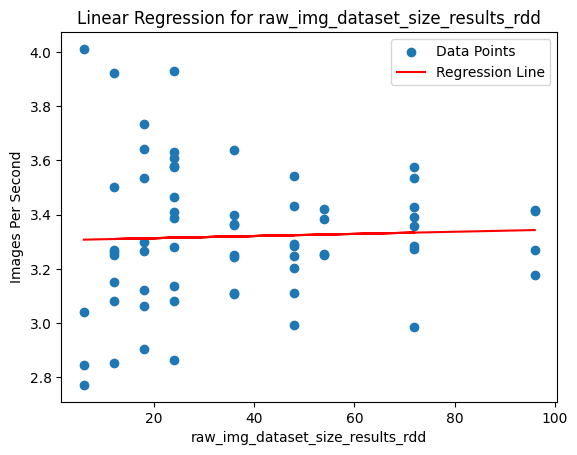
****

Fig. 8

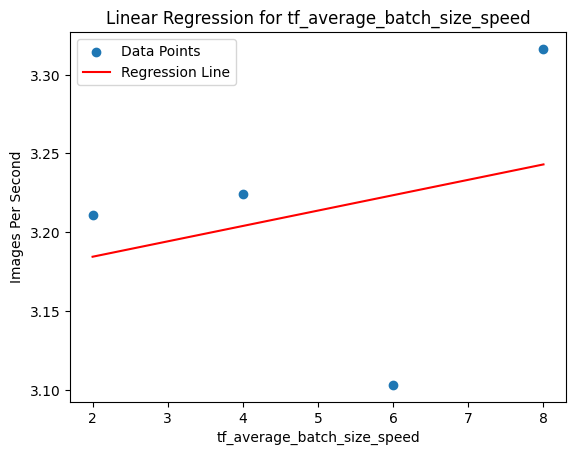
****

Fig. 9

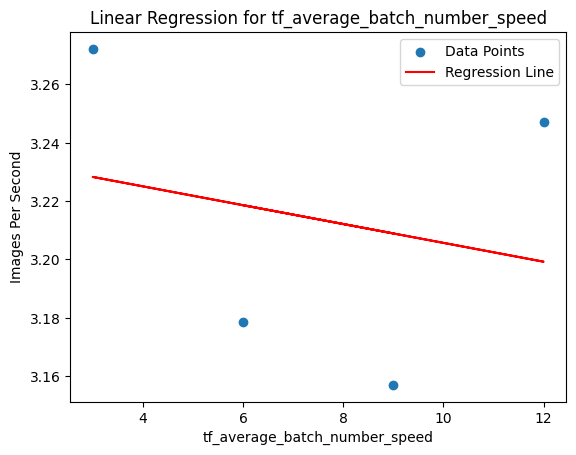
****

Fig. 10

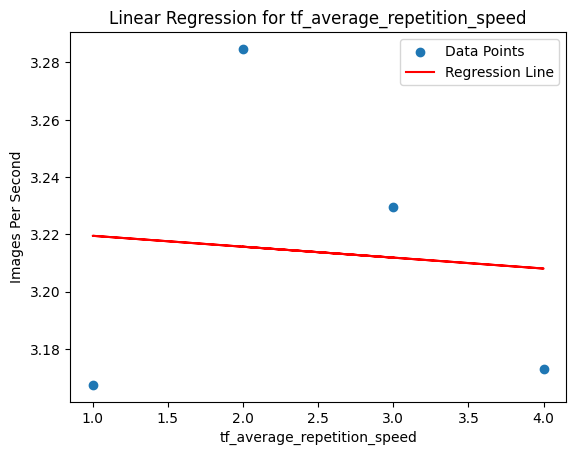
****

Fig. 11

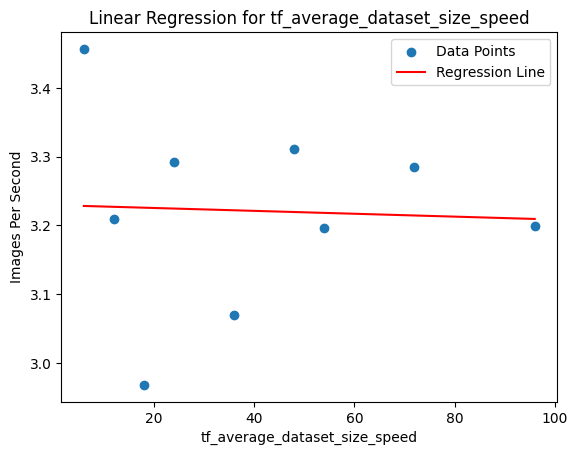
****

Fig. 12

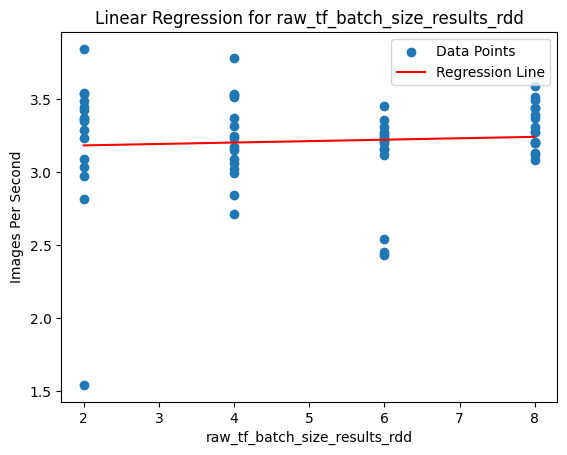
****

Fig. 13

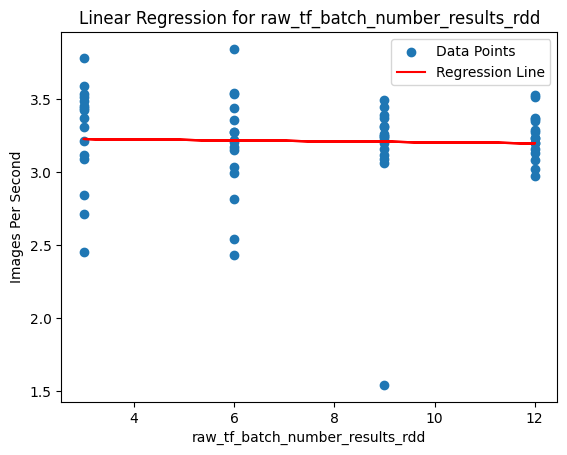
****

Fig. 14

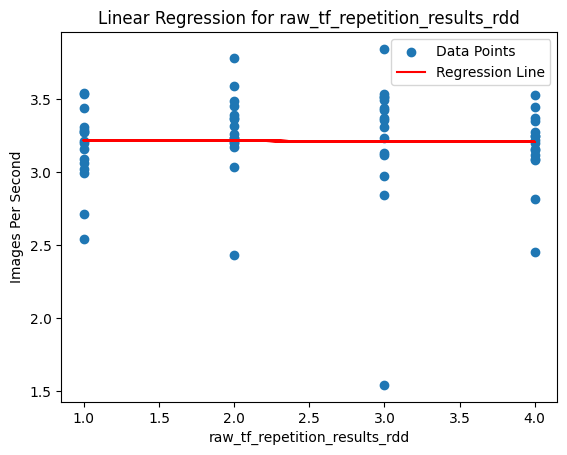
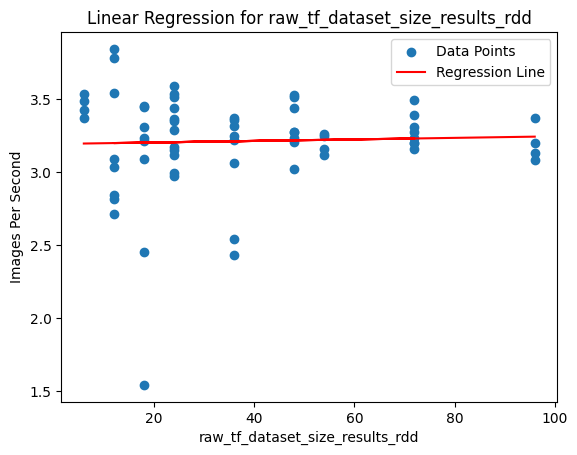
****

Fig. 15

Fig. 16****

**3a)**

CherryPick is the proposed solution, using Bayesian Optimization method, to help decide an optimal or near-optimal cloud configuration to minimize costs while maximize the performance. It uses a concept called confidence interval to define the boundaries and then cherry-picking the best set of features (cloud resources).

In the previous tasks, we used Spark RDD operations that involved distributed data processing and it was executed on Google Cloud Dataproc. Configuring the optimal cloud environment could further enhance the results of execution time in those tasks. Cherry-picking technique could help to find the most optimal setup for the virtual machine using Bayesian Optimization (BO).

This technique utilizes a custom version of BO for handling noise in the objective function to deal with the uncertainties inherent in the cloud environments. The Spark RDD operation can be affected by verity of factors like network latency, resource contention, and data skew. These can lead to noisy performance. By including this noise into the optimization process, CherryPick can handle these uncertainties to improve configuration selection process.

Moreover, its ability to transform the object function to handle multiplicative noise from cloud environment might be beneficial when using Spark RDD operation on Dataproc. This is because cloud environments know for their dynamic behavior such as fluctuating resource availability and performance characteristics. Cherrypick accounts for these dynamic and make better decision on configuration via the appropriate transformation of objective function.  
  
All in all, this approach can enhance the efficacy of configuration selection and ultimately improve the performance and cost-effectiveness of these tasks.

**3b)**

For instance, in batch processing, CherryPick starts by selecting key features and using BO to iteratively pick configurations while adapting to changes in performance and resources availability. Transforming object function then ensures informed decision that account for cloud dynamics. To confirm the efficacy of these selected configurations, their validity on the representative workloads should be observed.

In stream processing, real-time monitoring will guide dynamic feature selection and adaptive sampling strategies. Continuous updates to configuration selection are enabled through online learning. This is while incorporating performance constraints to ensure SLA compliance. Again, the validation with representative workload is necessary.

In both cases, the BO feature of CherryPick efficiently explores configurations, handle uncertainties, and adapt to dynamic environments. These strategies optimize Spark RDD operations when using Dataproc and results in improved performance and cost-effectiveness in batch and stream processing.

**Word Count: 1660**

**Appendices**

**1. Screenshots**

****

Fig. 1

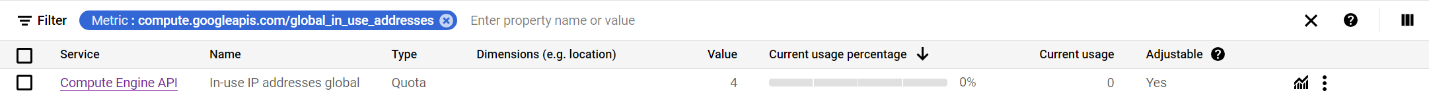
****

Fig. 2

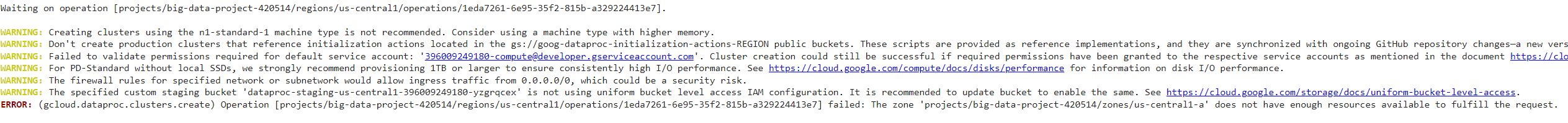


Fig. 3