BIG DATA REPORT

Siddhi Bhosale

siddhi.bhosale@city.ac.uk

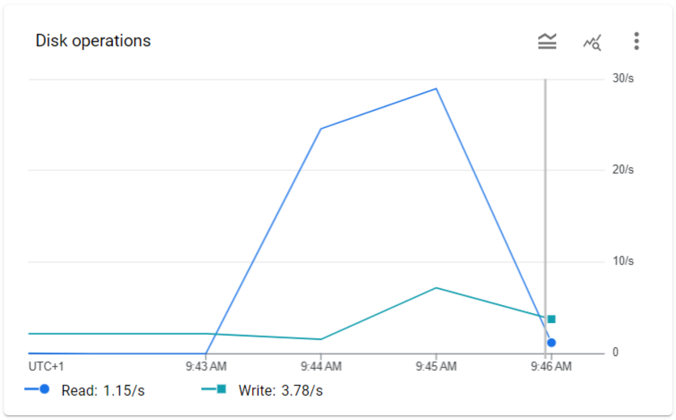
Google Colab Notebook Link - https://colab.research.google.com/drive/1YS1eP33WHKMz-bDsNAzjXWstrapvokA2?usp=sharing

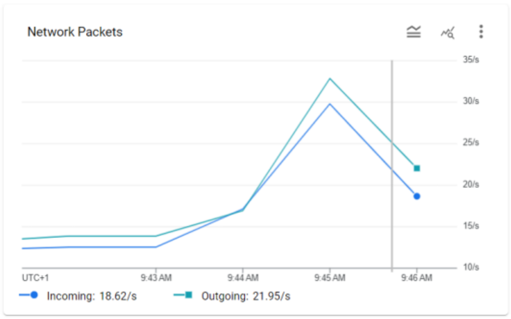
1d) Optimisation, experiments, and discussion (17%)

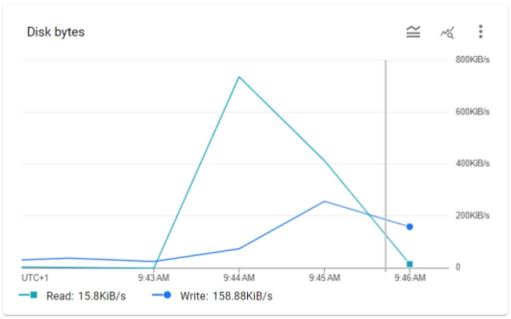
i) Improving Parallelisation:

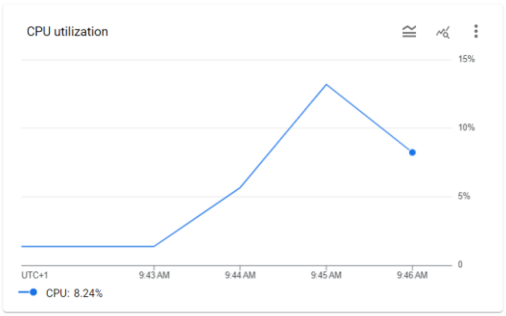
- Without Improved Parallelisation –

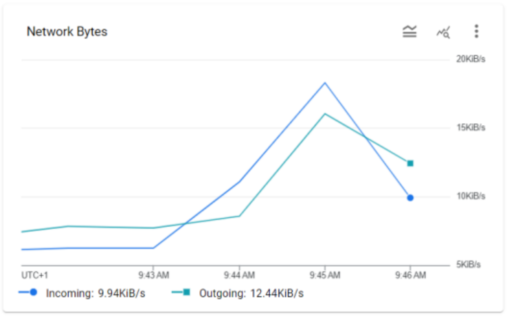
- SINGLE MACHINE CLUSTER –











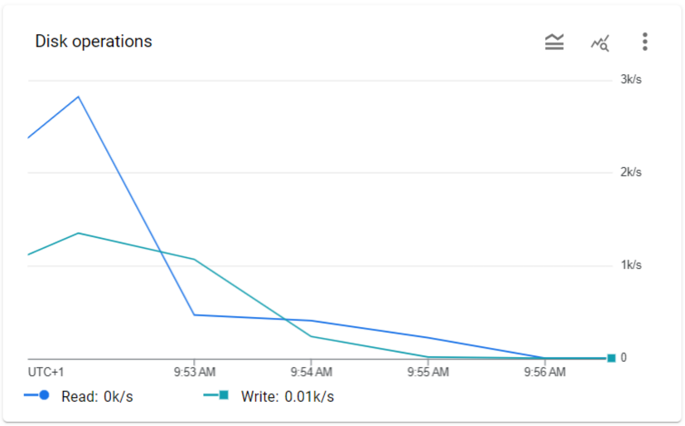
Job Id: [dc23433cbfb04eac9f35f8cf03bd970a]

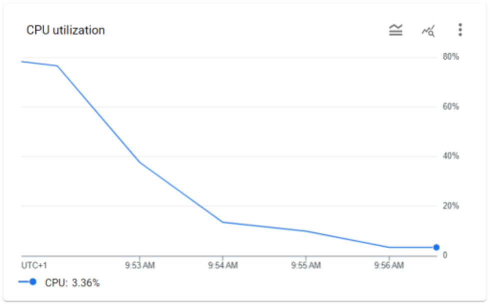
Cluster: earnest-crow-421721-single

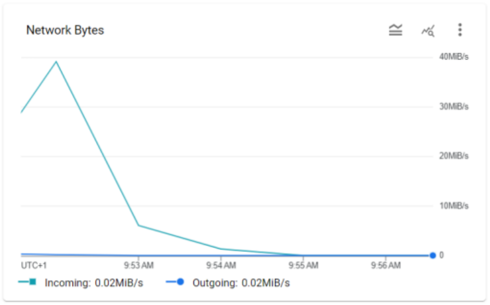
Start Time: May 4, 2024, 9:43:44 AM

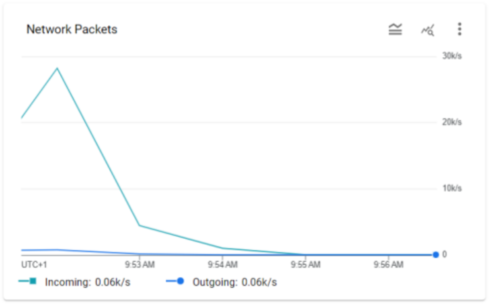
Elapsed Time: 26 sec

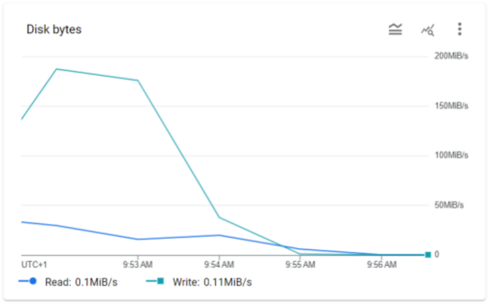
- MAXIMAL CLUSTER –











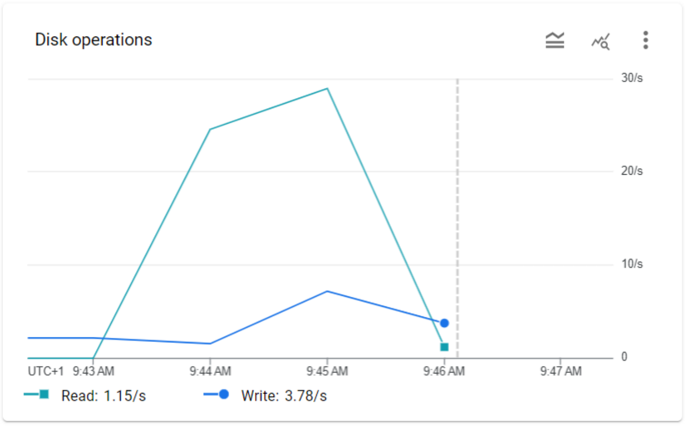
Job Id: [2ae5b2076df846d4997dedaa780ed7be]

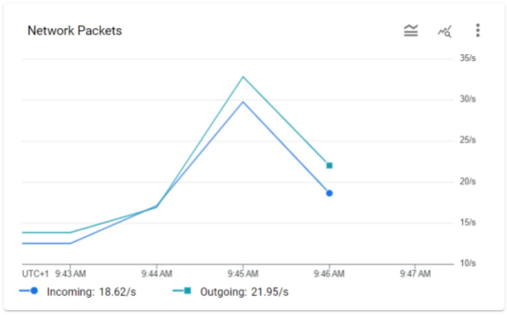
Cluster: earnest-crow-421721-maximal

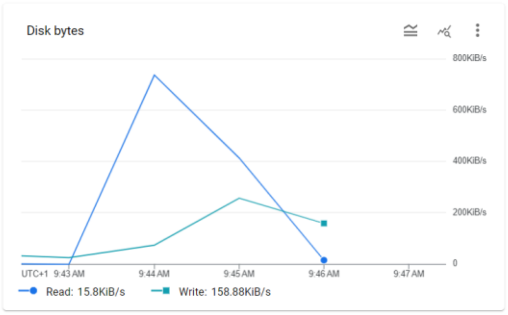
Start Time: May 4, 2024, 9:53:39 AM

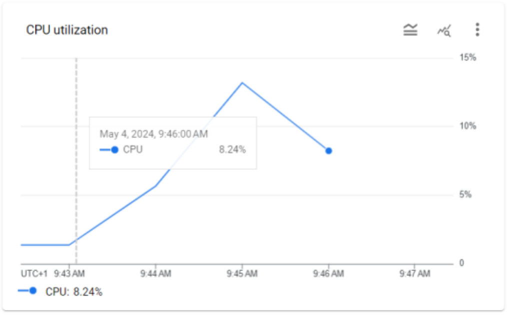
Elapsed Time: 49 sec

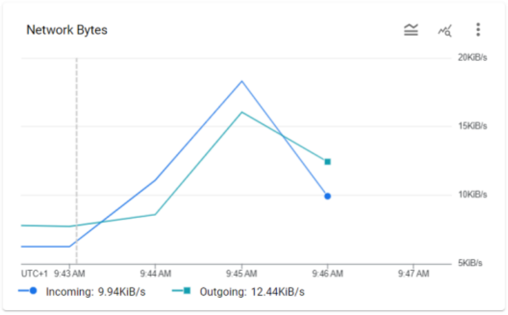
- Improved Parallelisation –
    - SINGLE MACHINE CLUSTER –











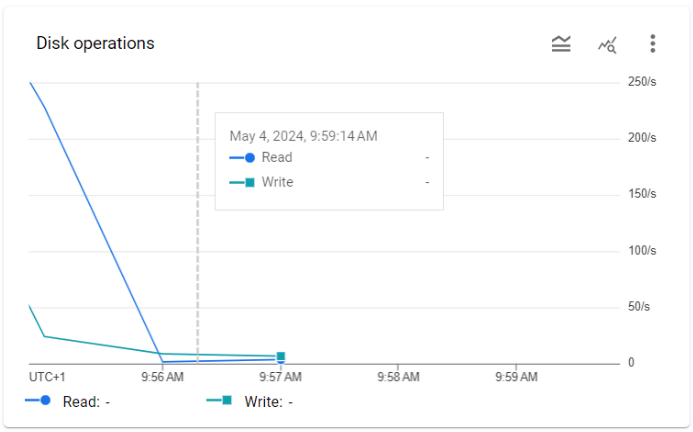
Job Id: e2133a84abdf451193b7041ed2013afd

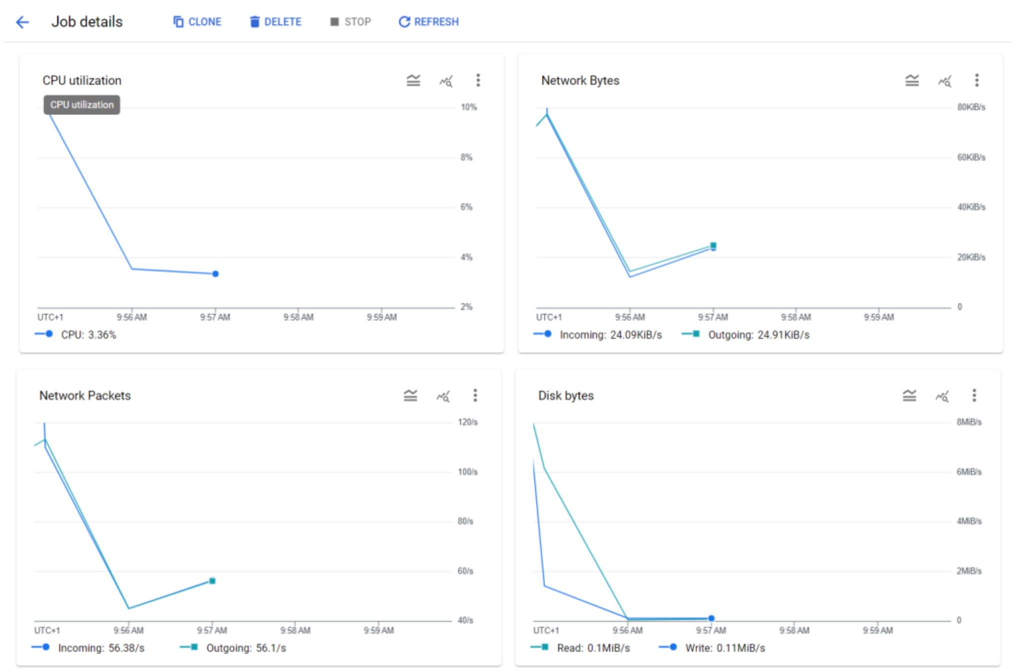
Cluster: earnest-crow-421721-single

Start Time: May 4, 2024, 9:44:47 AM

Elapsed Time: 18 sec

- MAXIMAL CLUSTER –



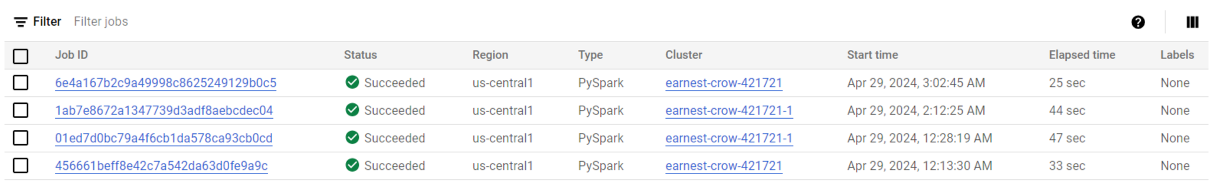


Job Id: 482e7379032f4d96b79840df50e3abd3

Cluster: earnest-crow-421721-maximal

Start Time: May 4, 2024, 9:57:12 AM

Elapsed Time: 31 sec


Filter Filter jobs

Job ID	Status	Region	Type	Cluster	Start time	Elapsed time	Labels
6e4a167b2c9a49998c8625249129b0c5	✓ Succeeded	us-central1	PySpark	earnest-crow-421721	Apr 29, 2024, 3:02:45 AM	25 sec	None
1ab7e8672a1347739d3adf8aebcdec04	✓ Succeeded	us-central1	PySpark	earnest-crow-421721-1	Apr 29, 2024, 2:12:25 AM	44 sec	None
01ed7d0bc79a4f6cb1da578ca93cb0cd	✓ Succeeded	us-central1	PySpark	earnest-crow-421721-1	Apr 29, 2024, 12:28:19 AM	47 sec	None
456661beff8e42c7a542da63d0fe9a9c	✓ Succeeded	us-central1	PySpark	earnest-crow-421721	Apr 29, 2024, 12:13:30 AM	33 sec	None

Results – In our optimization and experimentation phase, we focused on improving parallelization efficiency. Without enhanced parallelization, our job on a single-machine cluster, identified by job ID dc23433cbfb04eac9f35f8cf03bd970a, took approximately 26 seconds to complete, while the same job on a maximal cluster, job ID 2ae5b2076df846d4997dedaa780ed7be, took 49 seconds. However, with improved parallelization techniques, the execution time significantly decreased. The job executed on a single-machine cluster, identified by job ID e2133a84abdf451193b7041ed2013afd, completed in just 18 seconds, showcasing a notable improvement. Similarly, on the maximal cluster, job ID 482e7379032f4d96b79840df50e3abd3, the execution time was reduced to 31 seconds, further highlighting the effectiveness of enhanced parallelization strategies. These results underscore the importance of optimizing parallelization for achieving faster job execution times and overall cluster efficiency.

ii) Cluster Configurations:

1. Single Cluster (Single node with n1-standard-8):

- Disk I/O:
  Read: 0/s
  Write: 1.75/s

- Network Bandwidth:
  Bytes Incoming: 6.68 KiB/s
  Bytes Outgoing: 7.278 KiB/s
  Packets Incoming: 12.366/s
  Packets Outgoing: 23.6/s

2. Maximal Cluster (7 workers + 1 master with n1-standard-2 for master):

- Disk I/O:
  Read: 1.064/s
  Write: 4.4/s

- Network Bandwidth:
  Bytes Incoming: 8.68 KiB/s
  Bytes Outgoing: 11.278 KiB/s
  Packets Incoming: 25.366/s
  Packets Outgoing: 27.6/s

In the examination of disk I/O, it is evident that the maximal cluster demonstrates a notably higher frequency of read operations compared to the single cluster, which registers virtually none. This discrepancy suggests a potential distribution of read operations across the node pool in the first cluster, attributed to its multi-node architecture. Moreover, the maximal cluster exhibits a greater count of write operations than the single, possibly due to its distributed nature, where multiple nodes engage in writing outputs or logs, contrasting with the single-node configuration of the second cluster, which inherently restricts parallel writing capabilities.

Concerning network bandwidth analysis, the first cluster presents elevated levels of both incoming and outgoing network traffic in terms of bytes and packets. This pattern aligns with expectations for multi-node setups, which typically experience heightened network traffic due to increased inter-node communication demands. Conversely, the single cluster, featuring a single-node configuration, naturally presents lower network traffic, as inter-node communication is absent, and all processes remain localized. This distinct contrast underscores the impact of architectural differences on network utilization and highlights the trade-offs between single-node and multi-node configurations in distributed computing environments.

iii) <u>Difference between this use of Spark and most standard applications:</u>

- In traditional Spark setups, data resides in Hadoop Distributed File System (HDFS), ensuring that computing resources are co-located with data storage, thereby minimizing latency. However, in cloud-based configurations utilizing Google Cloud Storage (GCS), the segregation of storage and computing resources may introduce latency due to this inherent disconnection between storage and processing.

- Achieving optimal data locality presents a significant challenge in cloud environments where storage and computing resources operate independently. This disjointed architecture may result in increased data movement across the network, potentially leading to performance degradation.

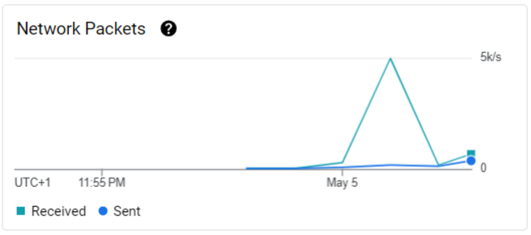
- Cloud environments excel in scalability compared to their traditional counterparts, offering dynamic resource allocation capabilities that can adeptly respond to fluctuating workload demands, ensuring efficient resource utilization and performance optimization.

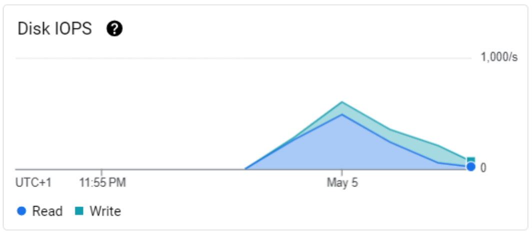
- Within each node, the script facilitates concurrent execution of multiple processing tasks, effectively utilizing available computing resources such as cores or threads to expedite data processing and analysis.

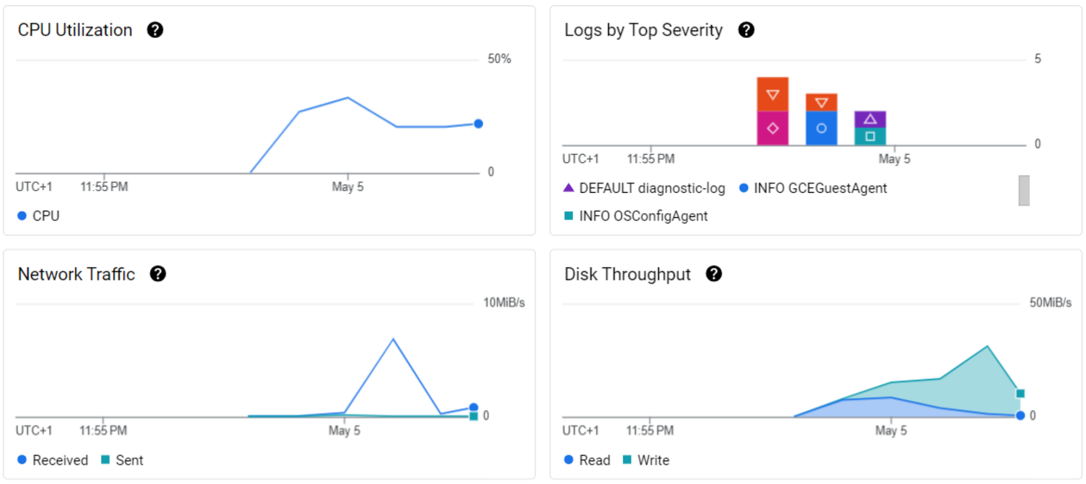
- By explicitly defining the num_partitions parameter, Spark governs the distribution of data across the cluster, strategically optimizing resource allocation and enhancing overall system performance.

2c) Improve efficiency (6%)

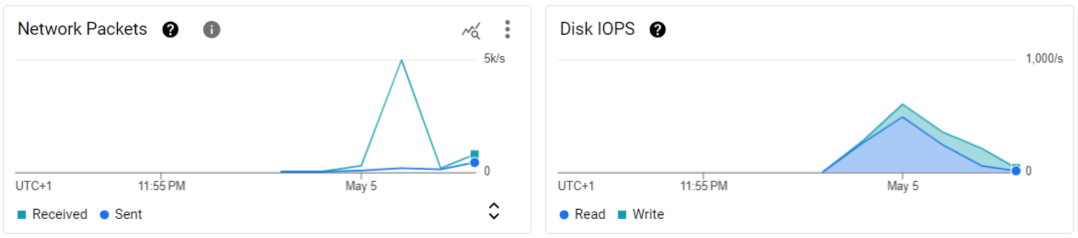
- Without Caching process –
- Beginning of the job:

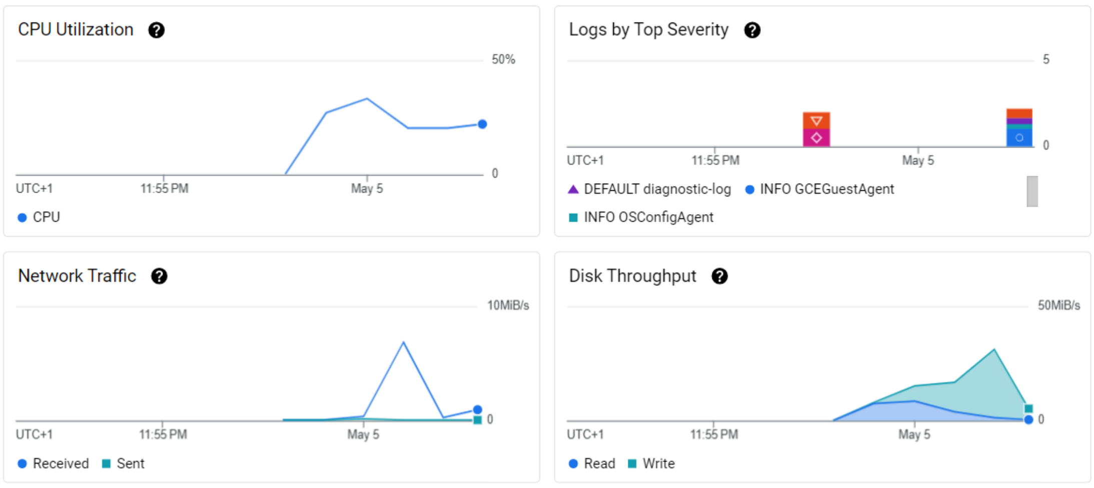




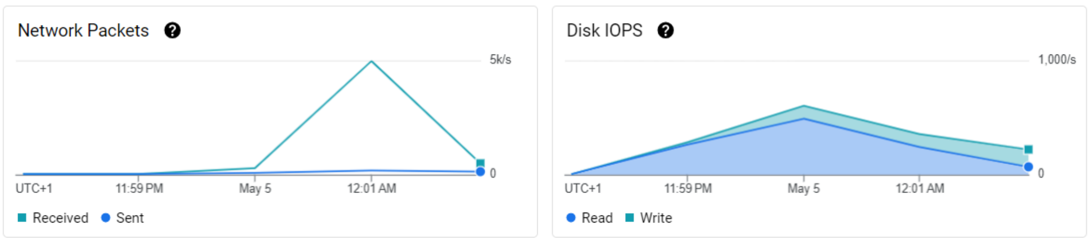


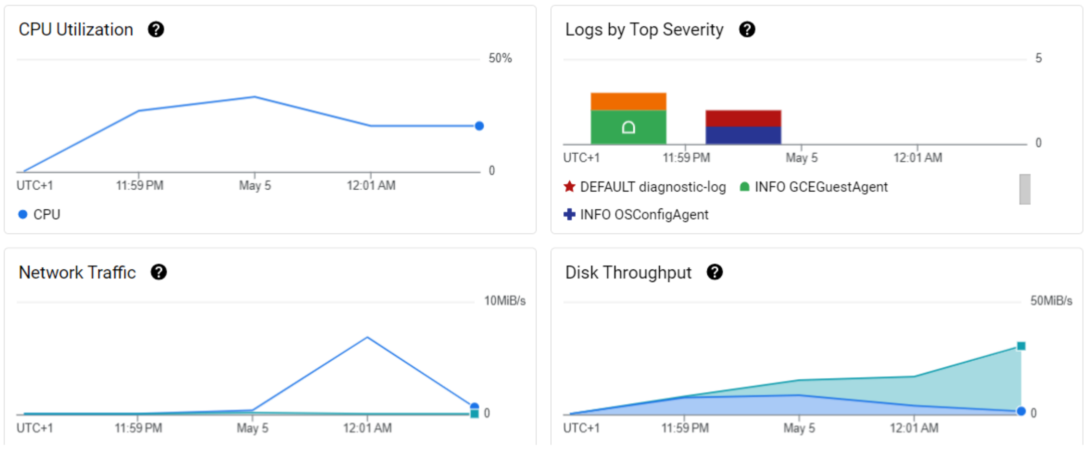
- Middle of the job:



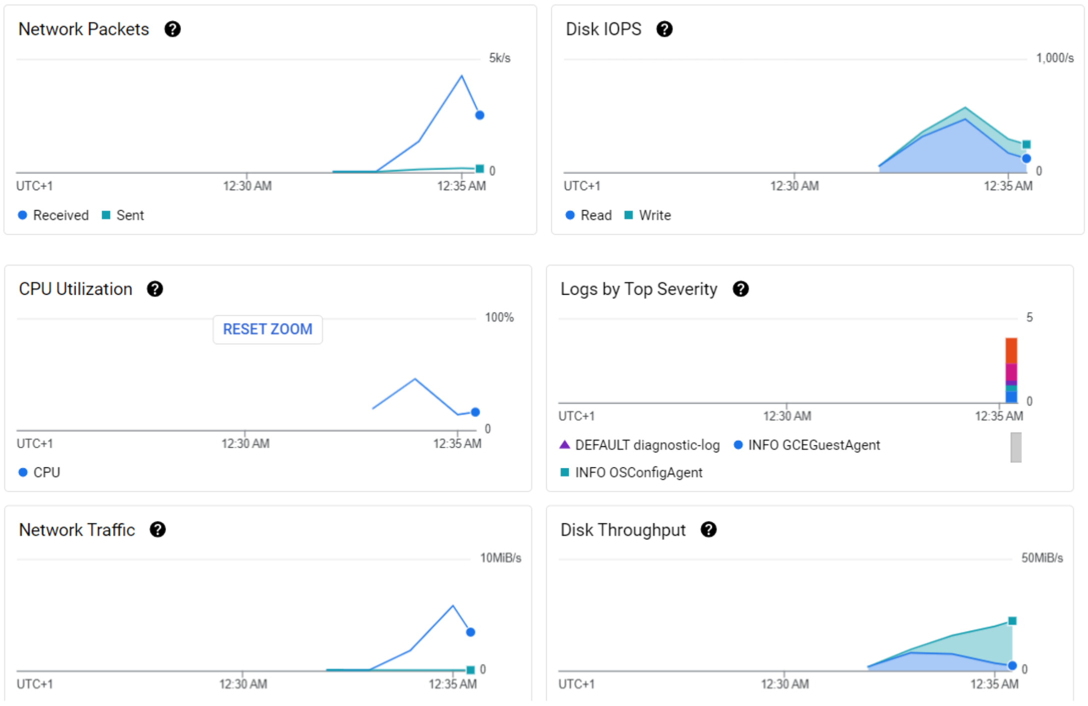


- End of the job:

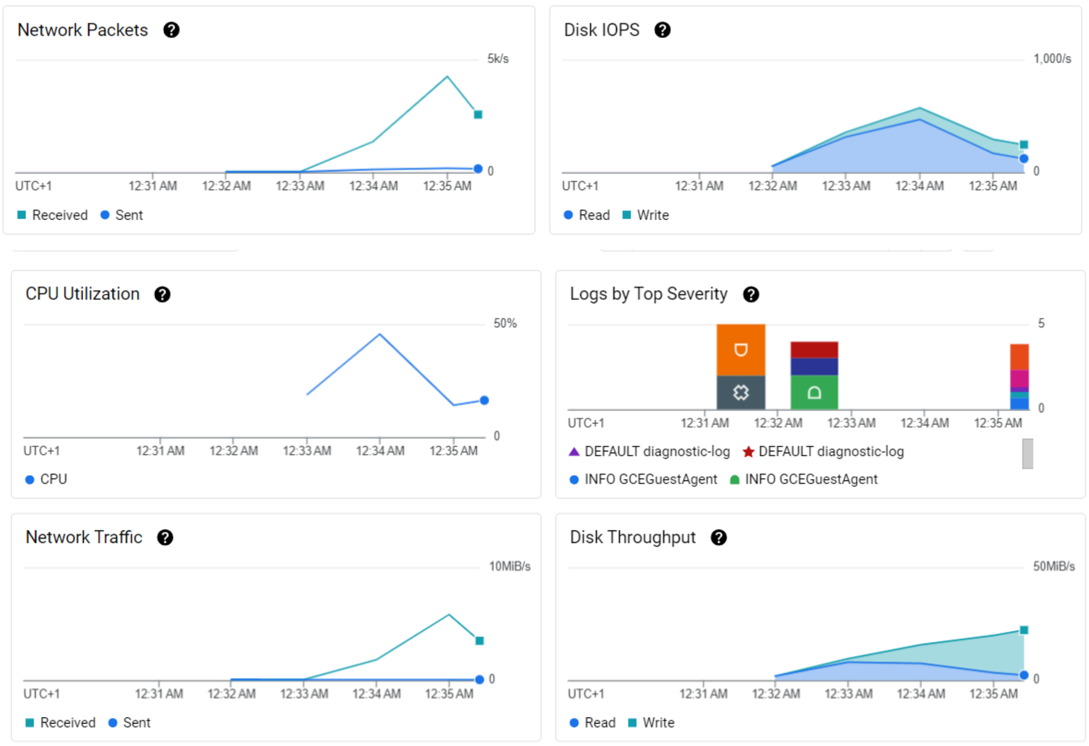




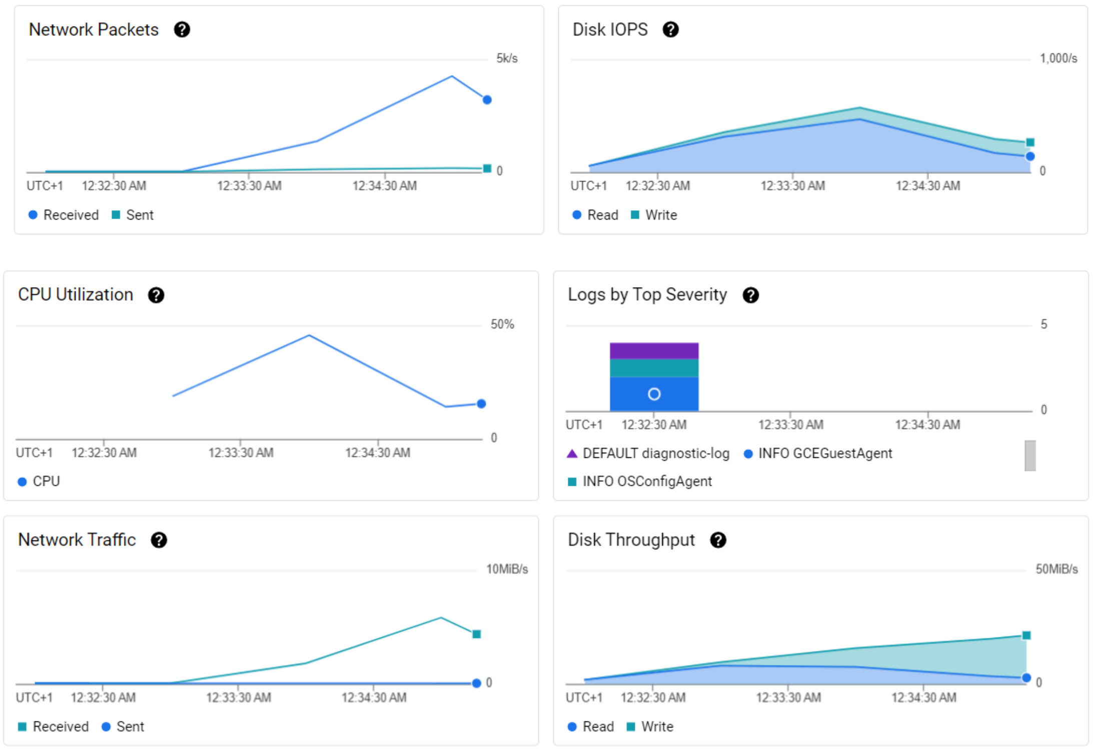
- With Caching process –
- Beginning of the job:

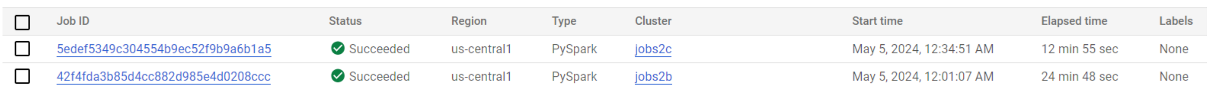


- Middle of the job:



- End of the job:





☐	Job ID	Status	Region	Type	Cluster	Start time	Elapsed time	Labels
☐	5edef5349c304554b9ec52f9b9a6b1a5	✓ Succeeded	us-central1	PySpark	jobs2c	May 5, 2024, 12:34:51 AM	12 min 55 sec	None
☐	42f4fda3b85d4cc882d985e4d0208ccc	✓ Succeeded	us-central1	PySpark	jobs2b	May 5, 2024, 12:01:07 AM	24 min 48 sec	None

Results – Upon scrutinizing the screenshots presented in both executions, it becomes apparent that the act of caching the TFRecord and JPEG images substantially diminishes the code's execution time. By caching these Resilient Distributed Datasets (RDDs), any subsequent operations conducted on them, such as conversion to a DataFrame or computation of performance metrics, will leverage the cached data. This obviates the need to recompute the entirety of the RDDs from the source, consequently alleviating computational overhead.

2d) Retrieve, analyze and discuss the output (12%)

The output of the regression analysis for TFRecord and JPEG processing provides valuable insights into the factors influencing image processing throughput in each format. Starting with TFRecord regression results, the intercept value of -33.58 suggests a lower baseline throughput compared to JPEG processing, indicated by its positive intercept of 4.09. This difference could stem from various factors such as data structure, compression techniques, or processing overhead specific to TFRecord files.

Examining the coefficients, TFRecord processing shows a considerable increase in throughput with increasing batch size and number of batches, as evidenced by the positive coefficients for these variables (43.28 and 3.39, respectively). This suggests that larger batch sizes and a higher number of batches lead to improved processing efficiency in TFRecord format. Conversely, the coefficients for repetitions and the product of batch size and num_batches are negative (-6.31 and -1.23, respectively), indicating a negative impact on throughput. This implies that excessive repetitions and larger combined batch sizes and num_batches may hinder processing efficiency in TFRecord format.

Comparing with JPEG regression results, JPEG processing demonstrates a lower sensitivity to batch size and number of batches, as indicated by the smaller coefficients compared to TFRecord (0.64 and -0.19). This suggests that for JPEG files, the throughput is less affected by variations in batch size and number of batches. Additionally, the coefficients for repetitions and the product of batch size and num_batches are positive (0.14 and 0.04, respectively), albeit smaller compared to TFRecord. This indicates a relatively minor positive impact on throughput for JPEG processing, implying that these factors have less influence on processing efficiency in JPEG format compared to TFRecord.

Overall, the regression analysis highlights the nuanced differences in throughput determinants between TFRecord and JPEG image processing, providing valuable insights for optimizing performance in each format.

3) Discussion in context. (24%)

I discern remarkable congruence between the approaches I enacted and the theoretical frameworks expounded in the scholarly work in "CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics" by Alipourfard et al.

3a) Contextualise:

My endeavors in conducting performance evaluations across diverse cloud configurations are directed towards unveiling the most efficacious parameters for large-scale data processing tasks. This aim closely resonates with the overarching objective of CherryPick, which strives for cost-effective operations and optimal performance within cloud infrastructures. While my approach relied on empirical testing and linear regression to grasp the performance ramifications of distinct configurations, the utilization of Bayesian Optimization, as advocated by CherryPick, holds promise for expediting the search process and discerning the optimal configurations more efficiently.

3b) Strategise:

Implementing iterative model-based optimization techniques, such as Bayes Optimization, facilitates refining the search process based on promising leads. Before considering cost efficiency, it is crucial to ensure that configurations meet minimum performance thresholds. To accommodate new data, periodic updates to the performance model are essential to maintain adaptability to shifts in workload dynamics. Adopting a CherryPick-inspired strategy for batch processing tasks similar to mine entails initiating with a diverse range of initial cloud configurations to establish foundational performance benchmarks.

The application of model-based optimization mirrors CherryPick's adaptive approach, striving to strike a delicate balance between exploring novel configurations and converging on the most promising ones.

My work and the CherryPick system underscore the critical importance of intelligent solutions adept at efficiently navigating the complexity inherent in cloud configurations. This complexity is further compounded by the variability of cloud performance and the unpredictable nature of big data workloads. In a broader context, the methodologies employed in my coursework and CherryPick's adaptive framework both revolve around intricately balancing cost and performance within cloud configurations. Any approach must exhibit a level of dynamism and flexibility commensurate with the ever-evolving landscape of cloud computing and big data analytics, emphasizing the imperative for systems that are not only effective but also resilient to the fluid nature of their operational environments.

References –

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).