Question:

How does Apache Spark compare to OpenMPI for data focused tasks?

1. Summation of columns in the data
2. Counting number of each countries

Reference papers : The two reference papers I read are added to the bottom on the page

Experimental Setup:

I have set up three nodes for this experiment. These nodes consist of one Primary Node and two Worker Nodes. The Primary Node also acts as one of the worker Nodes when the tasks are divided.

Each node is a VM set up on Google Cloud, and they are placed in us-central1-a. The nodes are set up using the machine type n1, which utilises Intel Skylake processors. I chose the n1-standard-1 type to ensure that the cores are not shared. Each node has one core and 3.75GB of RAM. I have installed Ubuntu 22.04 as the operating system, and each node has a standard persistent disk with a capacity of 10GB. Every node is connected to another node through internal SSH. I ran Ansible on the master node, which installs all the dependencies on both the master and the secondary nodes.

To start the Spark master, use the following command:

/opt/spark/sbin/start-master.sh

And to start the Spark worker, use the following command on each worker node:

/opt/spark/sbin/start-worker.sh spark://10.128.0.2:7077

Side effects/ Artifacts

To test my script on Spark, I need to run my code in Java, whereas for MPI, I need to execute my code in Python. I believe this difference might impact the results, considering Python is an interpreted language while Java is a low-level language.

During the setup, I discussed my findings with Shaun regarding how we were executing the Java scripts for Spark. In case we encountered any issues, we discussed things promptly.

Testing Procedure

To test the nodes, run the code on the primary node, which triggers the secondary nodes. I performed the test three times for both the codes, namely "estimate summation" and

"country count", to ensure accuracy. Subsequently, I calculated the average of these tests. Additionally, I monitored the traffic on the individual nodes to confirm that the data was indeed distributed among them. The following commands were used to build the files and run the task. Note that for Java files, I had to build the files before I could execute them on the Spark server. The data [1] used for testing is replicated so that there is more data to process. The data shows International Immigration for December 2023.

MPI:

mpirun -hostfile ../hostfile -np 3 python3 EstimateSummation.py

Spark:

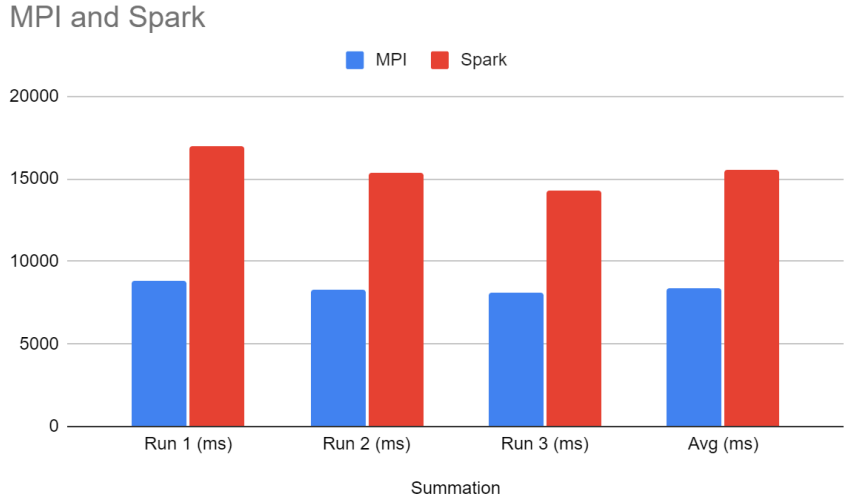
- javac -cp "/opt/spark/jars/*" EstimateSummation.java
- jar cf EstimateSummation.jar -C . .
- spark-submit --class EstimateSummation--master spark://10.128.0.2:7077 --conf spark.default.parallelism=3 --conf spark.default.hostFile="../hostfile" EstimateSummation.jar

Test Accuracy:

Estimate Summation:

Summation	Run 1 (ms)	Run 2 (ms)	Run 3 (ms)	Avg (ms)
MPI	8780	8323	8096	8399
Spark	17023	15358	14277	15552

T
he below graph shows the comparison between MPI and Spark for the different runs for Estimate Count



MPI:

Below are the three test runs of Estimate Summation on the nodes using MPI. We can see the Total time taken to run on the nodes.

Run - 1:

```
Process 1 received 504761 rows of data
Process 2 received 504761 rows of data
Process 0 received 504761 rows of data
Sum of 'estimate' column: 151165761
Time taken for sum calculation:  8780.164241790771  milliseconds
svhenriques@csci698-1:/mnt/mpi_shared/CodeExamplesF19/MPI/Python$
```

Run - 2:

```
Process 1 received 504761 rows of data
Process 2 received 504761 rows of data
Process 0 received 504761 rows of data
Sum of 'estimate' column: 151165761
Time taken for sum calculation:  8323.330163955688  milliseconds
```

Run - 3:

```
Process 1 received 504761 rows of data
Process 2 received 504761 rows of data
Process 0 received 504761 rows of data
Sum of 'estimate' column: 151165761
Time taken for sum calculation:  8096.592664718628  milliseconds
```

Spark:

Below are the three test runs of Estimate Summation on the nodes using Spark. We can see the Total time taken to run on the nodes.

Run - 1:

```
+-------------+
|sum(estimate)|
+-------------+
| 1.51165761E8|
+-------------+

Time taken for sum calculation: 17023 milliseconds
24/03/05 03:14:22 INFO SparkContext: SparkContext is stopping with exitCode 0.
```

Run - 2:

```
+-------------+
|sum(estimate)|
+-------------+
| 1.51165761E8|
+-------------+

Time taken for sum calculation: 15358 milliseconds
24/03/05 03:23:53 INFO SparkContext: SparkContext is stopping with exitCode 0
```

Run - 3:

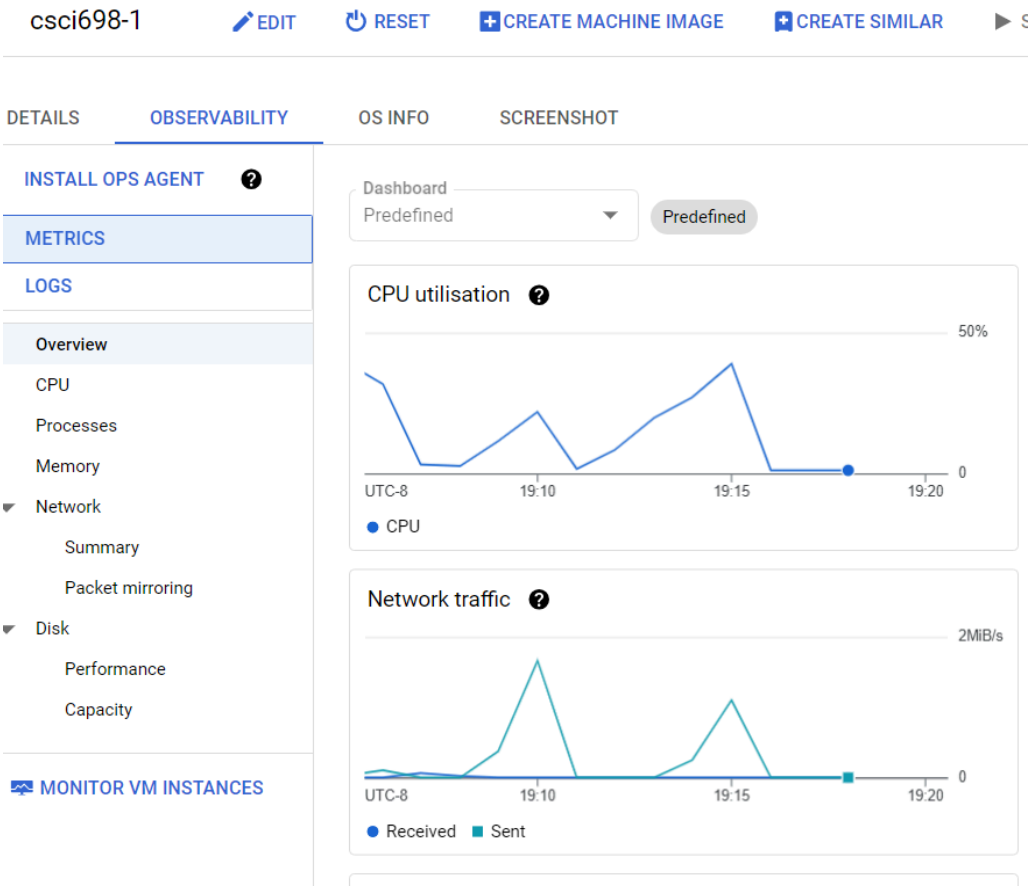
```
+-------------+
|sum(estimate)|
+-------------+
| 1.51165761E8|
+-------------+

Time taken for sum calculation: 14277 milliseconds
```

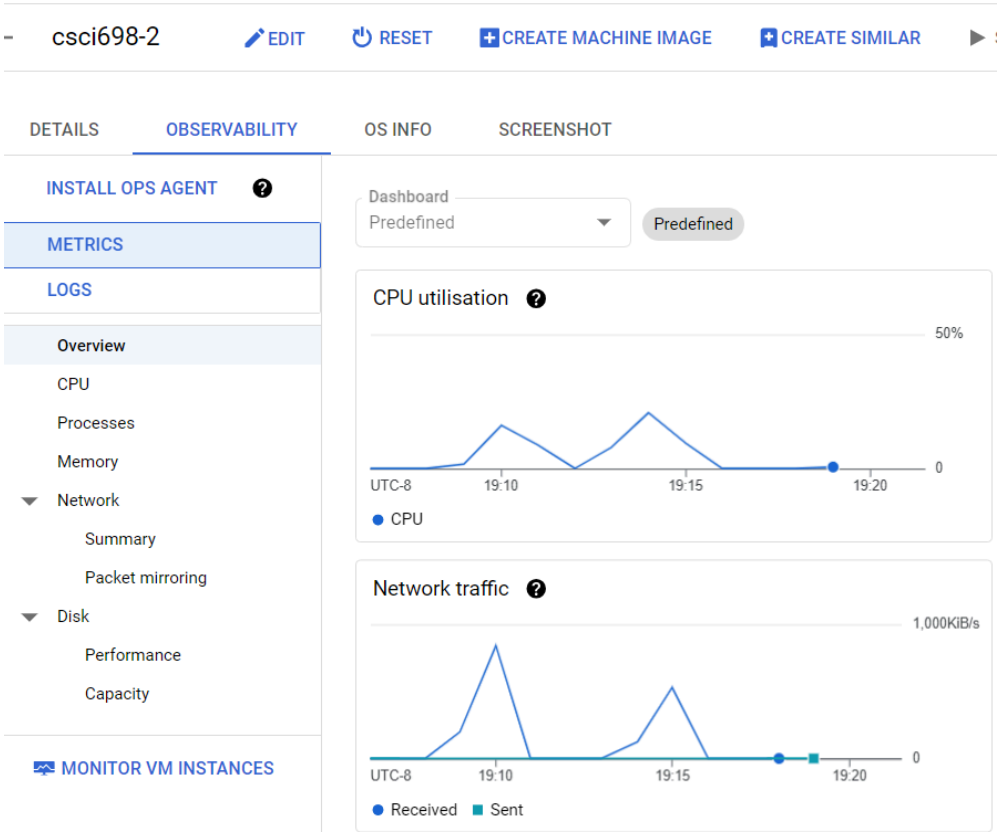
Resource Consumption:

Below are the screenshots of the CPU and network utilization during the testing of the node. The first screenshot displays the utilization of resources when the above code was executed on MPI. As indicated by the graph, there was a spike on all three nodes, and the network graph suggests that data was sent out from the primary node, and the secondary nodes received data during the same time.

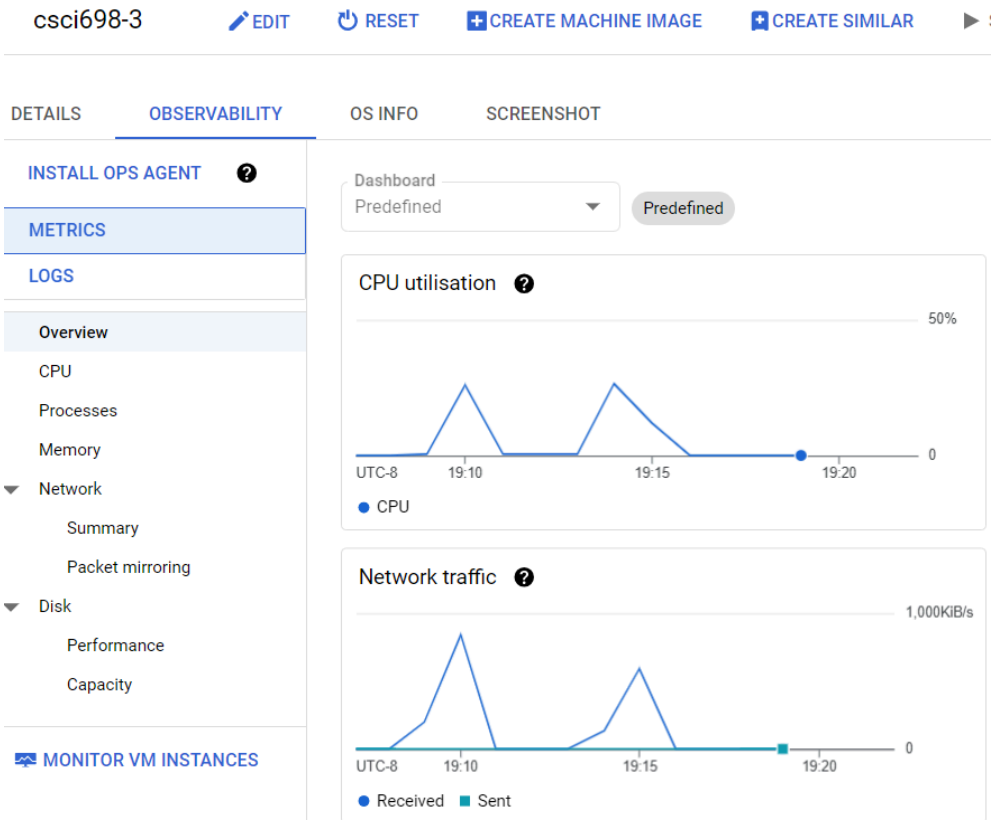
Primary Node:



Secondary Node 1:

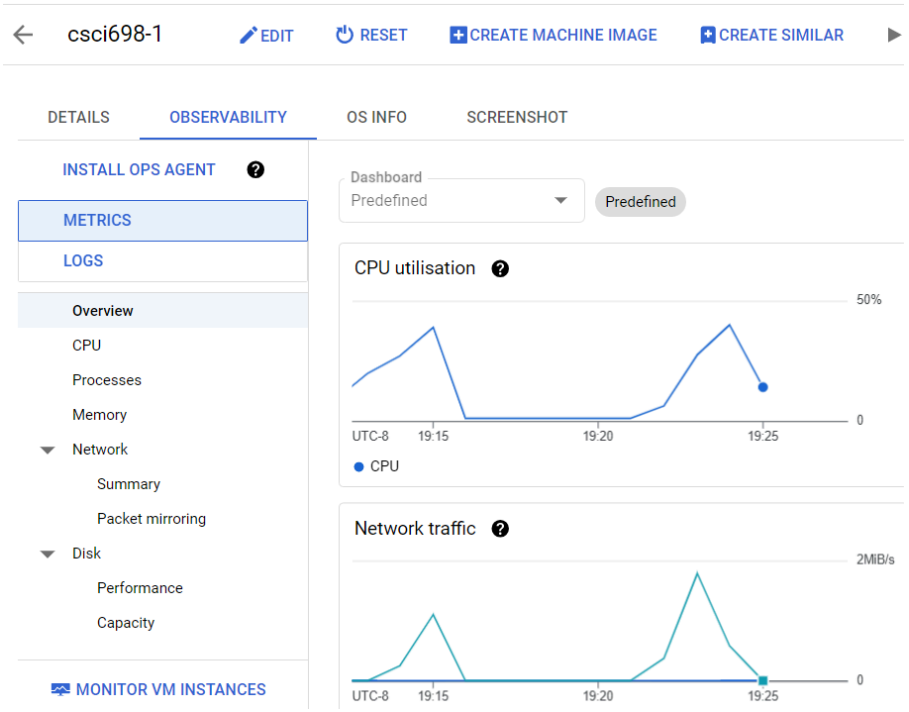


Secondary Node 2:

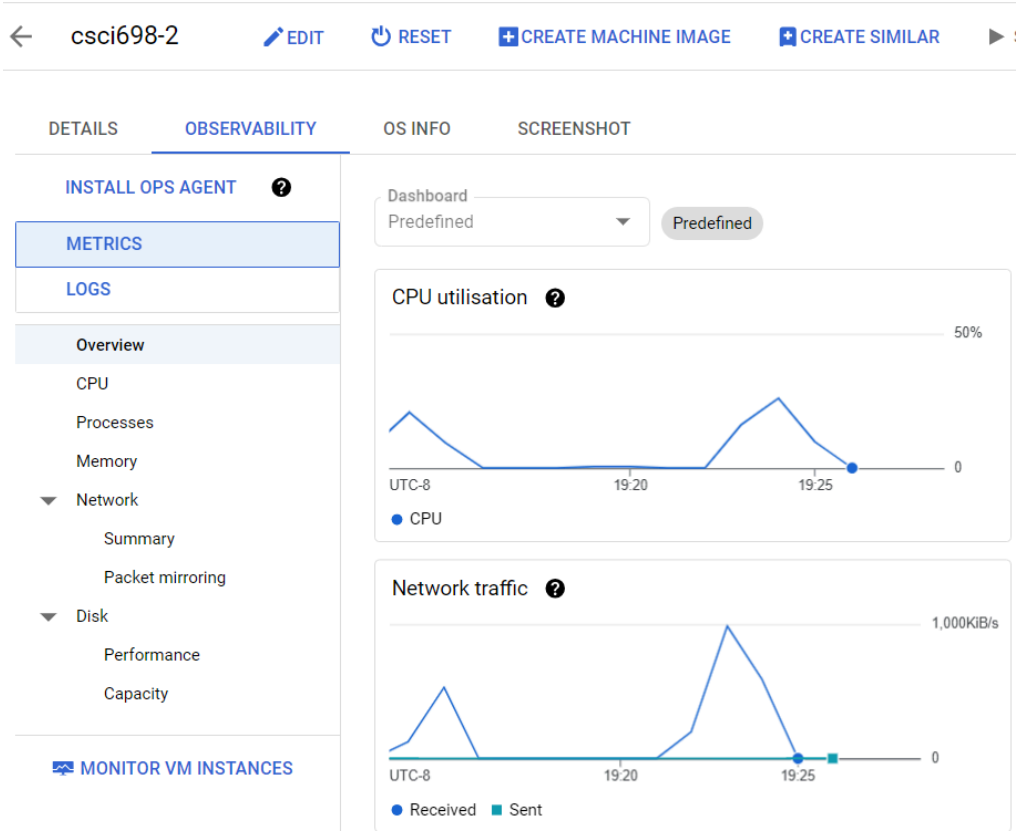


Resource utilization when the above program was executed on Spark server.

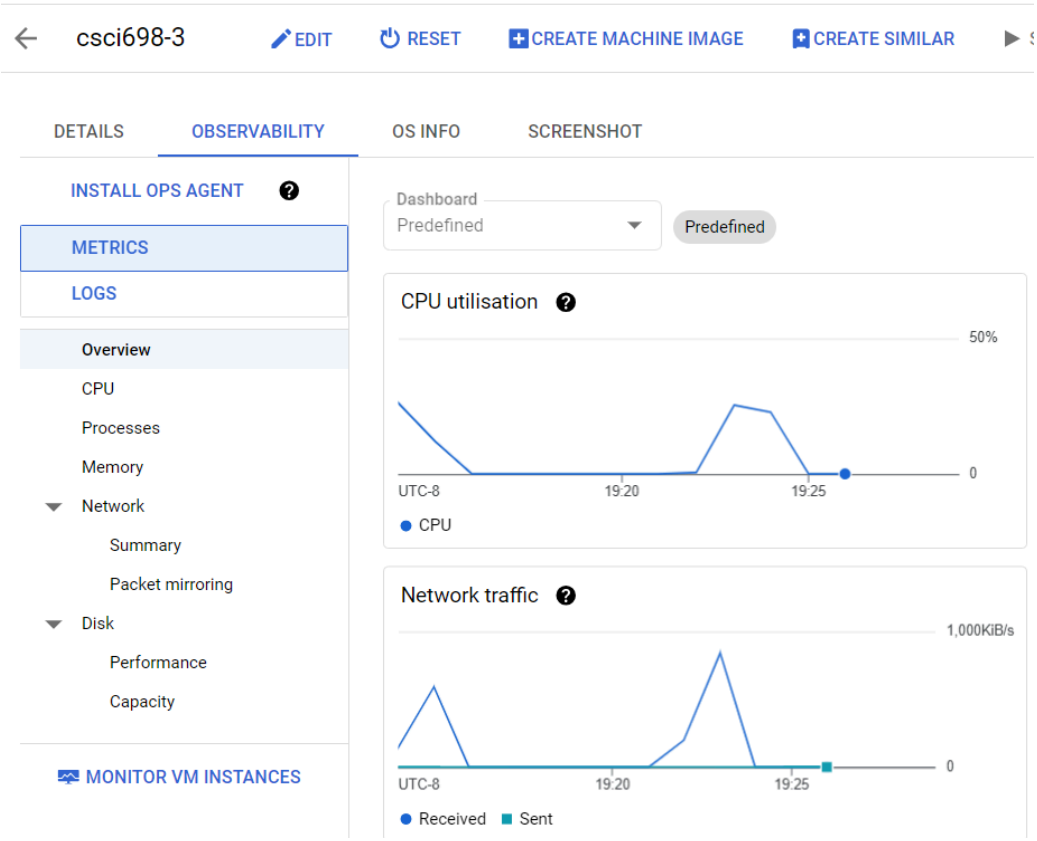
Primary Node:



Secondary Node 1:



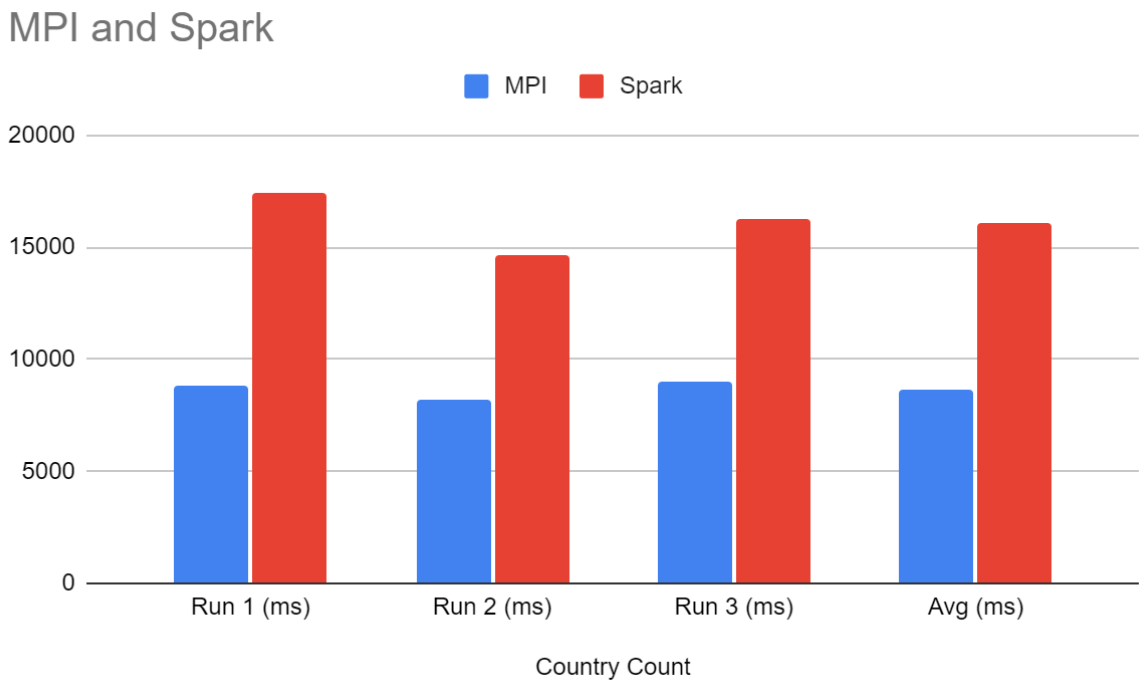
Secondary Node 2:



Country Count:

Country Count	Run 1 (ms)	Run 2 (ms)	Run 3 (ms)	Avg (ms)
MPI	8799	8177	9031	8669
Spark	17469	14626	16217	16104

The below graph shows the comparison between MPI and Spark for the different tests



MPI:

Below are the three test runs of Country Count on the nodes using MPI. We can see the Total time taken to run on the nodes.

Run - 1:

```
French Guiana : 233
Liechtenstein : 282
Vatican City State : 76
Mayotte : 36
San Marino : 171
British Indian Ocean Territory : 54
Central African Republic : 54
Sao Tome and Principe : 36
Guinea-Bissau : 18
Time taken for counting:  8799.588203430176  milliseconds
```

Run - 2:

```
Vatican City State : 76
Mayotte : 36
San Marino : 171
British Indian Ocean Territory : 54
Central African Republic : 54
Sao Tome and Principe : 36
Guinea-Bissau : 18
Time taken for counting:  8177.945137023926  milliseconds
```

Run - 3:

```
Vatican City State : 76
Mayotte : 36
San Marino : 171
British Indian Ocean Territory : 54
Central African Republic : 54
Sao Tome and Principe : 36
Guinea-Bissau : 18
Time taken for counting:  9031.538009643555  milliseconds
```

Spark:

The below image shows the task being distributed over the nodes.

```
24/03/05 03:54:30 INFO TaskSchedulerImpl: Adding task set 3.0 with 3 tasks resource profile 0
24/03/05 03:54:30 INFO TaskSetManager: Starting task 0.0 in stage 3.0 (TID 6) (10.128.0.3, executor 1, partition 0, NODE_LOCAL
24/03/05 03:54:30 INFO TaskSetManager: Starting task 1.0 in stage 3.0 (TID 7) (10.128.0.4, executor 0, partition 1, NODE_LOCAL
24/03/05 03:54:30 INFO TaskSetManager: Starting task 2.0 in stage 3.0 (TID 8) (10.128.0.2, executor 2, partition 2, NODE_LOCAL
24/03/05 03:54:30 INFO BlockManagerInfo: Added broadcast_3_piece0 in memory on 10.128.0.4:34495 (size: 20.2 KiB, free: 413.9 M
24/03/05 03:54:30 INFO BlockManagerInfo: Added broadcast_3_piece0 in memory on 10.128.0.3:42219 (size: 20.2 KiB, free: 413.9 M
24/03/05 03:54:30 INFO BlockManagerInfo: Added broadcast_3_piece0 in memory on 10.128.0.2:41107 (size: 20.2 KiB, free: 413.9 M
24/03/05 03:54:30 INFO MapOutputTrackerMasterEndpoint: Asked to send map output locations for shuffle 0 to 10.128.0.2:43526
24/03/05 03:54:31 INFO MapOutputTrackerMasterEndpoint: Asked to send map output locations for shuffle 0 to 10.128.0.4:57010
24/03/05 03:54:31 INFO MapOutputTrackerMasterEndpoint: Asked to send map output locations for shuffle 0 to 10.128.0.3:53004
24/03/05 03:54:32 INFO TaskSetManager: Finished task 1.0 in stage 3.0 (TID 7) in 2268 ms on 10.128.0.4 (executor 0) (1/3)
24/03/05 03:54:32 INFO TaskSetManager: Finished task 2.0 in stage 3.0 (TID 8) in 2320 ms on 10.128.0.2 (executor 2) (2/3)
24/03/05 03:54:33 INFO TaskSetManager: Finished task 0.0 in stage 3.0 (TID 6) in 2822 ms on 10.128.0.3 (executor 1) (3/3)
```

Run - 1:

```
|            Kiribati|10499|
|              Guyana| 1359|
|         Philippines|17894|
|             Eritrea| 1149|
|  St Kitts and Nevis|  600|
|               Tonga|17519|
|      Norfolk Island| 3861|
|            Djibouti|  427|
+--------------------+-----+
only showing top 20 rows

Time taken for counting: 17469 milliseconds
```

Run - 2:

```
|            Kiribati|10499|
|              Guyana| 1359|
|         Philippines|17894|
|             Eritrea| 1149|
|  St Kitts and Nevis|  600|
|               Tonga|17519|
|      Norfolk Island| 3861|
|            Djibouti|  427|
+--------------------+-----+
only showing top 20 rows

Time taken for counting: 14626 milliseconds
24/03/05 04:16:09 INFO SparkContext: SparkContext is stopping with exitCode 0
```

Run - 3:

```
|            Kiribati|10499|
|              Guyana| 1359|
|         Philippines|17894|
|             Eritrea| 1149|
|  St Kitts and Nevis|  600|
|               Tonga|17519|
|      Norfolk Island| 3861|
|            Djibouti|  427|
+--------------------+-----+
only showing top 20 rows

Time taken for counting: 16217 milliseconds
```

Conclusion:

In both tasks, Apache Spark consistently exhibits higher runtimes (in milliseconds) compared to OpenMPI. For the summation task, the average runtime for Spark is approximately 15552 ms, while for OpenMPI, it is notably lower at 8399 ms. Similarly, for the country count task, Spark's average runtime stands at 16104 ms, whereas OpenMPI's average is 8669 ms. This indicates that OpenMPI performs more efficiently in terms of runtime for both tasks. This was also noted in the paper "Performance Evaluation of Apache Spark Vs MPI: A Practical Case Study on Twitter Sentiment Analysis," [2] where they observed that MPI outperforms in parallel and distributed cluster computing environments, making it a better choice for big data applications. Through this project, I learned that while Apache Spark is known for its ease of use through its resilient distributed datasets (RDDs) and DataFrame APIs, OpenMPI offers more fine-grained control over the distributed computing process.

Learning:

Through this project, I gained hands-on experience with Apache Spark and OpenMPI. I bettered my understanding of the underlying principles and characteristics of distributed computing frameworks. I learned how to set up the required environment to simulate the Spark and MPI frameworks and how nodes communicate with each other. Additionally, I learned how to write programs that leverage distributed computing systems to process data faster. I conducted several tests, both on single nodes and multiple nodes, and observed a noticeable difference between them, clearly demonstrating the advantage of such a setup, especially in the context of data-focused problems for real-world tasks.

From this project, I concluded that OpenMPI performs better for data-focused tasks, as evidenced by the test runs. It's important to note that all reported timings are only for the part where the data is distributed between the nodes and processed. The time required to divide the data into chunks and any other prerequisites have not been captured in the timings. Another point to note is that MPI encountered the issue of maximum CPU utilization more often than Spark. While CPU utilization is automatically handled in Spark, it needs to be manually managed for MPI [2] to avoid such cases.

References:

CSV Data:

[1] https://www.stats.govt.nz/assets/Uploads/International-trade/International-trade-December-2023-quarter/Download-data/international-trade-december-2023-quarter-csv.zip

[2] Performance Evaluation of Apache Spark Vs MPI: A Practical Case Study on Twitter Sentiment Analysis https://thescipub.com/pdf/jcssp.2017.781.794.pdf

[3] Big Data in metagenomics: Apache Spark vs MPI
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7537910/

[4]
https://stackoverflow.com/questions/44185405/efficient-aggregation-sum-on-a-single-column-data-frame-in-spark-scala