Hello Dr. Amira Soliman,

I've tried two cluster configurations with 2 worker nodes and 5 workers nodes in a cluster. In my project, the action which is going to be impacted by the cluster configurations is the join operation. Spark provides us to use various optimization options as per the need. In my project, I use two kinds of joins

1. Broadcast Hash Join
2. Shuffle Hash join.

These join configuration settings can be obtained from using the property variable ***spark.sql.autoBroadcastJoinThreshold***. In this mail, I am attaching the code files and results.
I did calculate the time taken by String Indexer and model ALS building operations. These time requirements are dependent on the Join configurations. By default, 10 MB is the maximum size in bytes for a table that will be broadcast to all worker nodes when performing a join, I changed this default value to 100MB and found a significant improvement in all operations. I used 100MB as the limit because my table size *triplets* are around 64MB in size.
Please find my answer below:
Q1. Can you discuss changing the number of partitions with respect to the network configuration you created using the google cloud platform?
Answer: I am not explicitly changing the number of partitions but is taken by Spark internals which is dependent on the configurations we set.

	Shuffle Hash Join			**Broadcast Hash Join**		
Worker Nodes in a cluster	**Partitions**	**Time Taken in String Indexer**	**Time Taken in Model building**	**Partitions**	**Time Taken in String Indexer**	**Time Taken in Model building**
2	100	206	199	5	67	171
5	100	83	88	7	39	64

We can conclude from the above table if we increase the number of worker nodes and use the right partition strategy, performance gets improved significantly.

Q2. Can you also discuss if you used random or range partitioner in your project? What is the difference between random and range partitioners?
Answer: As we know, we are controlling the partitioning using configuration parameter using setting variables, we are not explicitly using random or range partitioners. By default, it uses a random partitioner. It seems, skewness condition arises when we use soft merge join which is another kind of join that we are not using in our project, skewness can be taken care of by setting the configuration parameter ***spark.sql.adaptive.skewJoin.enabled***. I displayed the number of records present in each partition and it is evenly distributed (not skewed) across all the partitions.

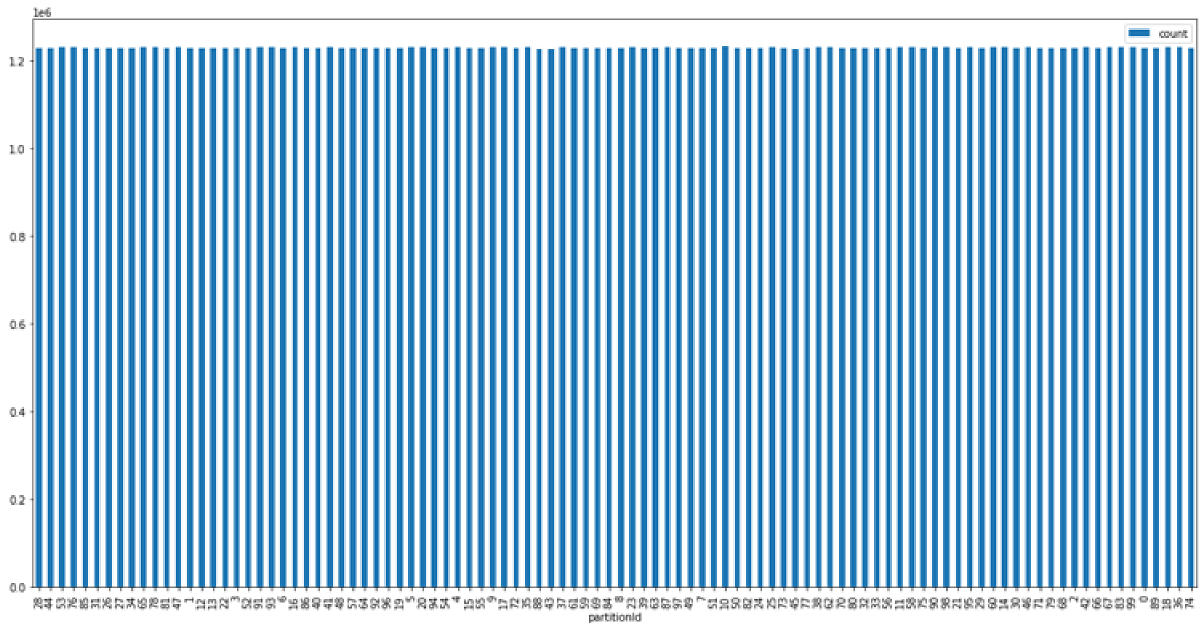

Figure: Partitions in 2 worker nodes cluster for Shuffle Hash Join

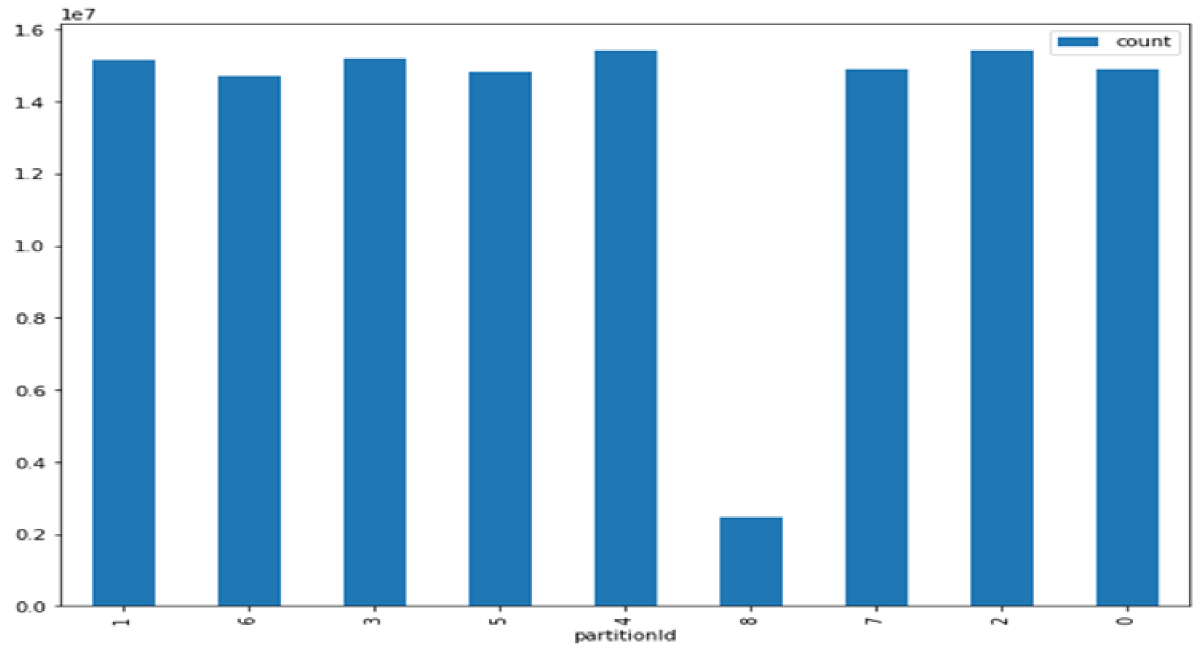

Figure: Partitions in 2 worker nodes cluster for Broadcast Hash Join

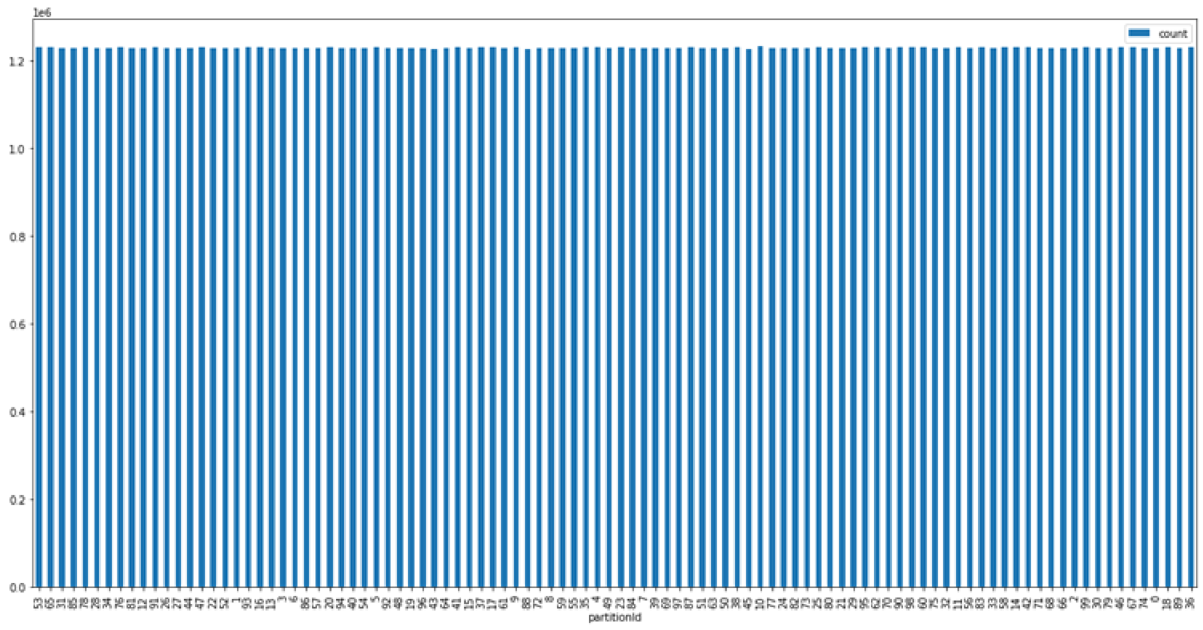

Figure: Partitions in 5 worker nodes cluster for Shuffle Hash Join

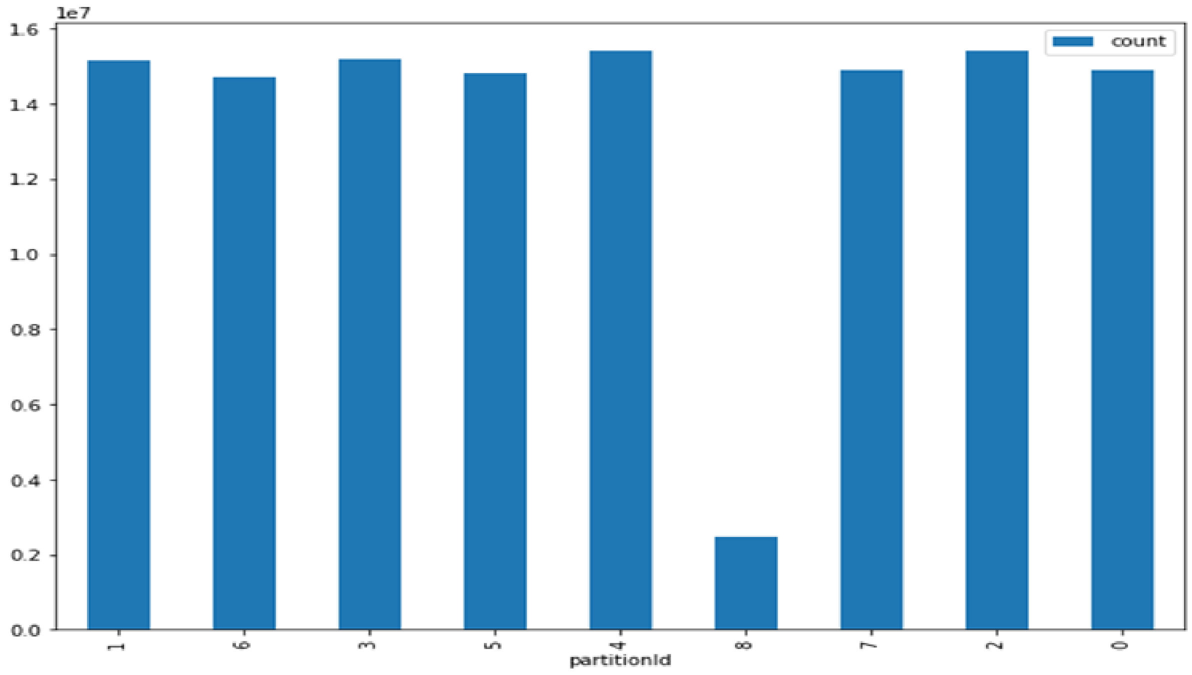

Figure: Partitions in 5 worker nodes cluster for Broadcast Hash Join

About Random and Range Partitioner: Partitioning helps us process data parallelly. Random partitioner uses hash function whereas Range Partitioner divides data based on a sorted key. We need to take care of the skewness among partitions and also need to optimize the performance. The right strategy is decided based on the above two considerations.

Kindly find the code implementations as attached files.

Please let me know if you want me to work further on this project.