* **What is the maximum size of worker node you have used?**

Storage 64GB Worker Nodes.

Worker node configuration should not exceed more than 64GB.If we increase size of worker nodes to more than 64GB, there is garbage collection overhead which reduces our performance. We will not need more than 64GB, if needed , we can increase worker nodes, we can increase number of cores we have.

* **How do you choose cluster configuration?**

Whenever we are trying to do any normal ETL/ELT work, then in that case we will go for memory optimized clusters. Whenever we have some development tasks that we need to do quickly and check it out , in that case always go for general purpose clusters. When we have a strong ETL task/ELT task which requires lot of shuffle, has lot of joins . in that case always go for storage optimized cluster . shuffle intensive work will be done by storage optimized because it has disk attached to it so it will speed up your process of any shuffle that is required. So that’s why we go for storage optimized one’s in case of shuffle intensive work.

* **What is difference between repartition and coalesce function?**

Repartition will increase/decrease number of partitions. Repartition always involve a shuffle. Repartition works by creating new partitions and doing full shuffle to move data around. Since full shuffle takes place, repartition is less performance than coalesce function.

Df2= df.repartition(2).write.mode(‘overwrite’).parquet(‘/dbfs/FileStore’)

Coalesce function is used to reduce number of partitions in data frame. Coalesce doesn’t involve full shuffle. If no of partitions is reduced from 5 to 2, coalesce will not move data in 2 executors and move the data from the remaining 3 executors to the 2 executors. There by avoiding a full shuffle. Since full shuffle is avoided, coalesce is more performance than repartition.

Df2= df.coalesce(1).write.mode(‘overwrite’).parquet(‘/dbfs/FileStore’)

Note: Even though coalesce seems to be useful in decreasing partitions.It also reduces degree of parallelism when we want to partition data. For example, if we do an extreme coalesce to 1 partition,then all computation would take place on single node which is not a good practice. In this case, repartition can be used.

* **What is the drawback of coalesce?**

Even though coalesce seems to be useful in decreasing partitions.It also reduces degree of parallelism when we want to partition data. For example, if we do an extreme coalesce to 1 partition,then all computation would take place on single node which is not a good practice. In this case, repartition can be used.

* **If we have .gz zip files , how are they distributed in spark?**

When we are working with zip file and when we are trying to read zip file, it will be single threaded only. All computation will go to one thread only.

* **What is difference between from pyspark.sql.types import \* and from pyspark.sql.types import xyz ?**

from pyspark.sql.types import \* 🡪 In production if we use this command, it will read everything , it will import all modules from library and there will be huge load on performance. So, we can import one particular module which is required so there will be performance optimization.

* **What is the default partition size?**

128 MB. Spark by default creates 1 partition for every 128MB of the file.

Spark.conf.get(“spark.sql.files.maxPartitionBytes”) output : 134217728b

* **What are the default number of partitions?**

Default 200 shuffle partitions are created. We can change it if required.

* **What are SQL end points?**

SQL end points don’t require to provide configurations of what virtual machines to use are called SQL end points. SQL end points are clusters only where we don’t need to provide configurations on how many cores, how many worker nodes, what driver configuration to be. In SQL end points, we only need to select cluster size small, medium large, x-small,2x small,3x small etc.

* **What is mean by Interactive and Job cluster?**

Interactive Cluster: Cluster created by us manually through cluster UI or “create cluster” API.

Job Cluster: Any cluster created from job(Jobs UI/Jobs API). Job cluster used for scheduling jobs. While scheduling jobs, we need to provide cluster configurations.

Interactive clusters are more expensive per DBU than Job/Automated clusters.

In Job clusters, when job is scheduled automatically cluster will get machines for us to do our job . It is not shared between jobs. In Interactive cluster, if we are going for high concurrency mode cluster ,in that case it can be shared as well. We can run multiple jobs on the same cluster.

* **What is Auto scaling in Data bricks cluster?**

Let’s say we have provided minimum number of worker nodes as 2. Now when our job will actually start,it will automatically fetch 2 workers first because we have specified minimum number of workers to be 2. Now let’s say our job runs and it needs more worker and we have specified a range up to 8 as maximum worker. Now our job needs 2 workers then what database will do automatically is it will go ahead and fetch machines for the 2 more workers and suddenly workload drops and then it doesnot need one worker then it will go ahead and drop that one worker. This is called autoscaling.

Based on the usage, databricks is trying to fetch machines/workers for us . This is called autoscaling. Autoscaling will help us to save money while processing our ETL/workload.

If we are keen on increasing performance,in that case autoscaling might not be good option because if we even go and fetch the machines .let’s say 2 extra machines fetched while doing job run it actually takes a little bit of time so time wise its not a good option but cost wise it’s a good option.

* **What is managed and unmanaged table in databricks?**

Whenever we create table in spark, what happens is it has metadata information(it stores schema and data).

Managed Table: Managed Table stores both metadata and data in DBFS in our account. So, by mistake if we delete table(drop table table\_name ), automatically metadata and data also gets deleted.

Unmanaged/External Table: In unmanaged table, spark will only manage metadata and data files in the data location will be controlled by us. If we drop table, spark will remove only metadata but not data itself because it knows where metadata will present but it has no control over data that we have stored. If we want to delete data file from the location path , we need to delete data file manually.

* **How do you configure number of cores in worker?**

Number of cores in worker should be number of partitions or multiples of partitions.

* **How do you handle Bad records in Databricks?**

There are 2 ways to handle bad records in databricks. MODE and BAD RECORD PATH

MODE:

**PERMISSIVE MODE**: will accept the corrupt record and it will mark corrupt record in a separate column and process will proceed further. There won’t be any impact to the entire process.

**DROP MALFORMED MODE :** In this method , when there is corrupt record in the input file, it will just ignore the corrupt records, or it will drop the corrupt records and entire process with proceed further. Here also there is no impact to the overall process.

**FAIL FAST MODE:** If there is corrupt record in the input file then entire process will be failed immediately, that is fail fast.

**df1= spark.read.format("csv").option("mode","PERMISSIVE").option("header","true").schema(schema).load("/FileStore/tables/PDC-3.txt")**

2nd option is we cam move bad records to log folder.

badrecord=spark.read.option(“badRecordPath”, “/FileStore/tables/badRecordPath”).option(“sep”,”,”).option(“header”,True).schema(schema).csv(‘dbfs:/FileStore/tables/abc.csv’)

* **What is difference between Accumulator and broadcast variable?**

Accumulator is a variable and we need to update that variable. That variable is called an accumulator variable.Accumulator variable is updated by associative or commutative operations only.This variable is write only. Reading of accumulator can be only done by driver program.

Broadcast variable : If we have small data frame, we can broadcast it in all executor i.e; we are giving broadcasted copy of our dataframes to all executors. So there by we do not need to read the data, they do not need to go and take the data from one another,there is no shuffle happening because they have data with themselves and they can directly use that particular data from data frame. So, there is no shuffle involved between the executors. What happens is it reduces shuffle and there is no network io involved.