

SPARK BASED TOPICS KEYWORDS:

Spark Intro:

1. Spark : In-memory processing engine
2. Why spark is fast: Due to less I/O disc reads and writes
3. RDD: It is a data structure to store data in spark
4. When RDD fails: Using lineage graph we track which RDD failed and reprocess it
5. Why RDD immutable : As it has to be recovered after its failure and to track which RDD failed
6. Operations in spark: Transformation and Action
7. Transformation: Change data from one form to another, are lazy.
8. Action: Operations which processes the transformations, not lazy. creates DAG to remember sequence of steps.
9. Broadcast Variables: Data which is distributed to all the systems. Similar to map side join in hive
10. Accumulators: Shared copy in driver, executors can update but not read. Similar to counters in MR
11. MR before Yarn: Job tracker (scheduling & monitoring), task manager (manages tasks in its node)
12. Limitations of MR: Unable to add new clusters(scalable), resource under-utilization, only MR jobs handled
13. YARN: Resource manager(scheduling), application master(monitors & resource negotiation), node manager (manages tasks in its node)
14. Uberization: Tasks run on AM itself if they are very small

15. Spark components: Driver (gives location of executors) and executors (process data in memory)
16. Client Mode: Driver is at client side
17. Cluster Mode: Driver is inside AM in the cluster
18. Types of transformation: Narrow and Wide
19. Narrow: Data shuffling doesn't happen (map, flatMap, filter)
20. Wide: Data shuffling happens (reduceByKey, groupByKey)
21. reduceByKey() is a transformation and reduce() is an action
22. reduceByKey(): Data is processed at each partition, groupByKey() : Data is grouped at each partition and complete processing is done at reducer.
23. Repartition: used to increase/decrease partitions. Use it for INCREASE
Coalesce: used to decrease partitions and optimized as data shuffling is less

SPARK DATAFRAMES:

1. Cache() : It is used to cache the data only in memory.
Rdd.cache()
2. Persist() : it is used to cache the data in different storage levels (memory, disc, memory & disc, off heap).
Rdd.persist(StorageLevel.____)
3. Serialization: Process of converting data in object form into bytes, occupies less space

4. De-Serialization: Process of converting data in bytes back to objects, occupies more space.
5. DAG : Created when an action is called, represents tasks, stages of a job
6. Map : performs one-to-one mapping on each line of input
7. mapPartitions: performs map function only once on each partition
8. Driver: converts high level programming constructs to low level to be fed to executors (dataframe to rdd)
9. Executors: Present in memory to process the rdd
10. Spark context: creates entry point into spark cluster for spark appl
11. Spark session: creates unified entry point into spark cluster
12. Data frame: it is a dataset[row] where type error caught only at run time
13. Data set: it is a dataset[object] where type error caught at compile time
14. Modes of dealing with corrupted record: permissive, malformed, fail fast
15. Schema types: implicit, infer, explicit (case class, StructType, DDL string)

SPARK OPTIMIZATIONS

1. Spark optimization:
 - a. Cluster Configuration : To configure resources to the cluster so that spark jobs can process well.

- b. Code configuration: To apply optimization techniques at code level so that processing will be fast.
- 2. Thin executor: More no. of executors with less no. of resources. Multithreading not possible, too many broadcast variables required. Ex. 1 executor with each 2 cpu cores, 1 GB ram.
- 3. Fat executor: Less no. of executors with more amount of resources. System performance drops down, garbage collection takes time. Ex 1 executor 16 cpu cores, 32 GB ram.
- 4. Garbage collection: To remove unused objects from memory.
- 5. Off heap memory: Memory stored outside of executors/ jvm. It takes less time to clean objects than garbage collector, used for java overheads (extra memory which directly doesn't add to performance but required by system to carry out its operation)
- 6. Static allocation: Resources are fixed at first and will remain the same till the job ends.
- 7. Dynamic Allocation: Resources are allocated dynamically based on the job requirement and released during job stages if they are no longer required.
- 8. Edge node: It is also called as gateway node which is can be accessed by client to enter into hadoop cluster and access name node.
- 9. How to increase parallelism :
 - a. Salting : To increase no. of distinct keys so that work can be distributed across many tasks which in turn increase parallelism.
 - b. Increase no. of shuffle partitions
 - c. Increase the resources of the cluster (more cpu cores)
- 10. Execution memory : To perform computations like shuffle, sort, join
- 11. Storage memory : To store the cache
- 12. User memory : To store user's data structures, meta data etc.
- 13. Reserved memory : To run the executors

14. Kyro Serializer: Used to store the data in disk in serialized manner which occupies less space.
15. Broadcast join: Used to send the copies of data to all executors. Used when we have only 1 big table.
16. Optimization on using coalesce() rather than repartition while reducing no. of partitions
17. Join optimizations:
 - a. To avoid or minimize shuffling of data
 - b. To increase parallelism
1. How to avoid/minimize shuffling?
 - a. Filter and aggregate data before shuffling
 - b. Use optimization methods which require less shuffling (coalesce())
18. How to increase parallelism ?
 - a. **Min (total cpu cores, total shuffle partitions, total distinct keys)**
 - b. Use salting to increase no. of distinct keys
 - c. Increase default no. of shuffle partitions
 - d. Increase resources to inc total cpu cores
19. Skew partitions : Partitions in which data is unevenly distributed. Bucketing, partitioning, salting can be used to handle it.
20. Sort aggregate: Data is sorted based on keys and then aggregated. More processing time
21. Hash aggregate: Hash table is created and similar keys are added to the same hash value. Less processing time.
22. Stages of execution plan :
 - a. Parsed logical plan (unresolved logical plan) : To find out syntax errors

- b. Analytical logical plan (Resolved logical plan) : Checks for column and table names from the catalog.
- c. Optimized logical plan (Catalyst optimization) : Optimization done based on built in rules.
- d. Physical plan : Actual execution plan is selected based on cost effective model.
- e. Conversion into Rdd : Converted into rdd and sent to executors for processing.

****Note:**

1 hdfs block = 1 rdd partition = 128mb

1 hdfs block in local=1 rdd partition in local spark cluster= 32mb

1 rdd ~ can have n partitions in it

1 cluster = 1 machine

N cores = N blocks can run in parallel in each cluster/machine

N stages = N - 1 wide transformations

N tasks in each stage= N partitions in each stage for that rdd/data frame