**Basic-**

Topic: Common design pattern

**Q 1: RDD can be created in how many ways ?**

**A. one**

**B. two**

**C. three**

**d. four**

**Ans.: c. Three**

**Explanation: RDD can be created by parallelize method, by transformations and reading dataset**

Topic: Basic Syntax

**Q 2: which of following statement is correct for following code:**

**df.filter($"High">500).count()\*1.0/df.count())\*100**

**A. Extract What percentage of the time was the High greater than $500**

**B. Extract What was mean of the time was the High greater than $500**

**C. Extract What was the count of the time was the High greater than $500**

**d. Extract What std dev of the time was the High greater than $500**

**Ans.: A. Extract What percentage of the time was the High greater than $500**

**Explanation: This formula is used to calculate percentage in spark df**

Topic: Common design pattern

**Q 3: In which of following scenarios spark UDF can be used**

**A. To calculate pearson corelation**

**B. To add new column in an dataframe**

**C. To calculate standard deviation of an column values**

**d. To inferschema from dataset**

**Ans.: B. To add new column in an dataframe**

**Explanation: In order to add new column, spark UDF is used**

Topic: basic syntax

**Q 4: what would following code do:**

**df.na.drop(2)**

**A. drop initial 2 rows**

**B. drop rows with 2 null values for columns**

**C. drop last 2 rows**

**d. drop rows with minimum 2 column values equal to null**

**Ans.: d. drop rows with minimum 2 column values equal to null**

**Explanation: specification of drop function with na is to drop rows considerimng minimum int value**

Topic: Common

**Q 5: spark cannot be run in which of following mode ?**

**A. yarn**

**B. mesos**

**C. standalone**

**d. remote**

**Ans.: d. Remote**

**Explanation: spark can run in local, yarn and mesos mode**

Topic: Common

**Q 6: lit function is used for**

**A. To replace value with constant value**

**B. To lit or to negate value**

**C. For timestamp conversion in spark dataset**

**d. None of the above**

**Ans.: A. To replace value with constant value**

**Explanation: lit is used to replace with constant value**

Topic: Common

**Q 7: which of following is not an transformation**

**A. reduceByKey**

**B. aggregateByKey**

**C. sumByKey**

**d. countByKey**

**Ans.: c. SumByKey**

**Explanation: sumByKey is not an spark transformation**

Topic: Common

**Q 8: write operation on RDD is ?**

**A. coarse grained**

**B. fine grained**

**C. neither fine grained nor coarse grained**

**d. Both fine grained and coarse grained**

**Ans.: A. Coarse grained**

**Explanation:**it means that you can **write**you transformations to be applied to the whole dataset, but not individual elements on the dataset. **Operations**like map, filter, group reduce, but not get(index) or set(index).

Topic: Basic Syntax

**Q 9: which of the following is an action?**

**A. union(data-set)**

**B. intersection(data-set)**

**C. countByValue**

**d. distinct**

**Ans.: c. CountByValue**

**Explanation: countByValue is an action, other 3 are transformations**

Topic: Basic Syntax

**Q 10: which of following is true for wide transformation?**

**A. Data required to compute resides on multiple partitions**

**B. Data required to compute resides on single partition**

**C. There can only be single replica for the dataset with wide transformation**

**d. No caching can be done for wide transformations**

**Ans.: A. Data required to compute resides on multiple partitions**

**Explanation: Wide transformation means data resides on multiple partitions .**

**e.g. reduceByKey**

**Medium-**

Topic: Advanced implementation

**Q 1:  To maintain state while performing map operation i.e. in spark mapWithState which of the following is optional to be initialised?**

**A. checkpoint(“checkpoint\_dir”)**

**B. partitions()**

**C. initialState(initialStateDataFrame)**

**D. remember(60 seconds)**

**Ans.: D. Remember(60 seconds)**

**Explanation: Remember() function is optional and its not mandate to be used with mapWithState**

Topic: Advanced implementation

**Q 2:  In Spark, RDD storage level defined as MEMORY\_ONLY\_2 , what does \_2 means ?**

**A. replicas**

**B. partitions**

**C. shuffling level**

**d. buckets**

**Ans.: A. Replicas**

**Explanation: storage level MEMORY\_ONLY\_2 indicates spark transformations are being carried in memory and there are 2 replicas maintained for the same**

Topic: Advanced implementation

**Q 3:  When spark action is triggered, which of following configuration decides maximum number of output files in spark?**

**A. mapred.reduce.tasks**

**B. spark.sql.shuffle.partitions**

**C. spark.shuffle.tasks**

**d. spark.partitions**

**Ans.: B. spark.sql.shuffle.partitions**

**Explanation: spark.sql.shuffle.partitions is default set to 200, which means based on executor and number of cores, maximum 200 small files would get created**

Topic: common

**Q 4:  To adjust how long Spark will wait before it times out on each of the phases of data, spark.locality.\* config can contain which of the following phase is not present in configuration?**

**A. data local**

**B. process local**

**C. node local**

**D. stage local**

**Ans.: D. Stage local**

**Explanation: 4 stages of configuration are data, process, node and rack local.**

Topic: common found bug

**Q 5:  which of the following action would not cause performance issues on RDD or dataframe created by using large dataset ?**

**A. reduceByKey**

**B. countByKey**

**C. countByValue**

**D. collectAsMap**

**Ans.: A. ReduceByKey**

**Explanation: countByKey, countByValue and collectAsMap would give memory and performance issue as they would take all dataset into memory**

Topic: Advanced implementation

**Q 6:  which of the following would result in more shuffling?**

**A.** map or union()

**B.** aggregateByKey()

**C. cogroup()**

**D. filter()**

**Ans.: C. Cogroup()**

**Explanation: cogroup would form group and result in formation of new stage and hence would do more shuffling.**

Topic: common found bug or version limitation

**Q 7:  which of the following cannot be used for offset management in Spark version 1.6.0 with messaging queue kafka?**

**A. checkpoint(“checkpoint\_dir”)**

**B. zookeeper**

**C. hbase**

**D. kafka**

**Ans.: D. Kafka**

**Explanation: Spark 2.1.0 onwards we can do offset management using kafka itself. In spark 1.6.0 or previous only zookeeper, hbase or any db, checkpoint where the only options available**

Topic: Advanced implementation

**Q 8:  For 100 rules, spark.default.parallelism = 50, number of stages per task = 3, how much would be total number of tasks?**

**A. 15,000**

**B. 150**

**C. 5,000**

**D. 300**

**Ans.: A. 15,000**

**Explanation: number of jobs = 100**

**• number of stages per job = 3**

**• number of tasks per job = 3 \* 50 = 150**

**• total number of tasks = 100 \* 150 = 15,000**

Topic: Advanced implementation

**Q 9:  For an 8 core executor, with 15 second micro-batch duration and For 100 rules, spark.default.parallelism = 50, number of stages per task = 3, what would be average task time?**

**A. 324 ms**

**B. 394 ms**

**C. 424 ms**

**D. 494 ms**

**Ans.: B. 394 ms**

**Explanation:**

**number of jobs = 100**

**• number of stages per job = 3**

**• number of tasks per job = 3 \* 50 = 150**

**• total number of tasks = 100 \* 150 = 15,000**

**50 instances \* 8 cores = 400 task slots**

**Number of waves = 15,000/400 = 38 waves**

**15 seconds = 15000 ms micro-batch duration**

**15,000 ms / 38 waves = 394 ms per wave**

**average task time = 394 ms**

Topic: Usage of API

**Q 10:  spark model evaluation with MulticlassMetrics, which of the following statement is incorrect?**

**A. Need to import org.apache.spark.mllib.evaluation.MulticlassMetrics**

**B. pNeed to instantiate with label as below**

**// Need to convert to RDD to use this**

**val predictionAndLabels = results.select($"prediction",$"label").as[(Double, Double)].rdd**

**// Instantiate metrics object**

**val metrics = new MulticlassMetrics(predictionAndLabels)**

**C. can create confusion matrix as metrics.confusionMatrix**

**D. can create two dimentional metrics as metrics.ofDim**

**Ans.: D. can create two dimentional metrics as metrics.ofDim**

**Explanation: There is no function called as ofDim on MulticlassMetrics**

**Difficult-**

Topic: Architecture design

**Q 1:  which of the following is incorrect way to calculate standard deviation using spark transformations?**

**A. selectedData.groupBy($"user").agg(stdev($"duration"))**

**B. selectedData.groupBy($"user").agg((sqrt( avg($"duration" \* $"duration") - avg($"duration") \* avg($"duration"))).alias("duration\_sd"))**

**C. val stdevduration = duration.groupByKey().mapValues(value => org.apache.spark.util.StatCounter(value).stddev)**

**D. df.registerTempTable("df")**

**sqlContext.sql("""SELECT user, stddev(duration) FROM df GROUP BY user""")**

**Ans.: A. selectedData.groupBy($"user").agg(stdev($"duration"))**

**Explanation:**

**It would give compiler error on stddev as stddev is not valid function, right function is stddev\_pop. Hence answer is:**

**selectedData.groupBy($"user").agg(stdev\_pop($"duration"))**

Topic: Performance

**Q 2:  which of the following is incorrect way to perform concatenation of 2 columns of dataframe in spark?**

**A. val df = sc.parallelize(Seq(("foo", 1), ("bar", 2))).toDF("k", "v")**

df.registerTempTable("df")

sqlContext.sql("SELECT CONCAT(k, ' ', v) FROM df")

**B. df.select(concat($"k", lit(" "), $"v"))**

**C. val dfResults = dfSource.select(concat(",",dfSource.columns.map(c => col(c)): \_\*))**

**D. val dfResults = dfSource.select(concat\_ws(",",dfSource.columns.map(c => col(c)): \_\*))**

**Ans.: C. val dfResults = dfSource.select(concat(",",dfSource.columns.map(c => col(c)): \_\*))**

**Explanation:**

**It would give compiler error on concat function, the correct way to use syntax is mention in option D is to use concat\_ws function and use string for concatenation. Concat function doesnt require string parameter.**

Topic: Architecture design

**Q 3:  which of the following would not work for calculating aggregations for one hour timestamp using spark?**

**A.**

**val toSegment = udf((timestamp: String) => {**

val asLong = timestamp.toLong

asLong - asLong % 3600000 // period = 1 hour

})

**df.groupBy(toSegment($"timestamp")).count()**

**B.**

**val toSegmentGeneralized = udf((timestamp: String, period: Int) => {**

val asLong = timestamp.toLong

asLong - asLong % period

})

**df.groupBy(toSegment($"timestamp", $"3600000")).count()**

**C.**

**val toSegmentGeneralized = udf((timestamp: String, period: Int) => {**

val asLong = timestamp.toLong

asLong - asLong % period

**})**

**df.groupBy(toSegment($"timestamp", lit(3600000))).count()**

**D. None of the above**

**Ans.: B.**

**val toSegmentGeneralized = udf((timestamp: String, period: Int) => {**

val asLong = timestamp.toLong

asLong - asLong % period

})

**df.groupBy(toSegment($"timestamp", $"3600000")).count()**

**Explanation:**

**if we pass the arguments as in B to UDF, it would try to find the column named 3600000 and fail to get in dataframe.**

Topic: Best Practices

**Q 4:  which of the following does not take collection aggregateByKey or reduceByKey and throw error from following?**

**A. val resReduce = pairs.reduceByKey(\_ + \_)**

**B. val resAgg = pairs.aggregateByKey(0)(\_+\_,\_+\_)**

**C. val sets = pairs.aggregateByKey(new HashSet[Int])(\_+\_, \_++\_)**

**D. val sets = pairs.reduceByKey(new HashSet[Int])(\_ + \_)**

**Ans.: D. val sets = pairs.reduceByKey(new HashSet[Int])(\_ + \_)**

**Explanation:**

**reduceByKey doesnt works on hashset and it works only between partitions**

Topic: Advanced troubleshooting and best practices

**Q 5:  what would be output of following operations on spark dataframe:**

**val test = sqlContext.read.json(sc.parallelize(Seq("""{"a":1,"b":[2,3]}""")))**

**test.withColumn("b", explode($"b")).show**

**A.**

**+---+---+**

| a| b|

+---+---+

| 1| 2|

| 1| 3|

+---+---+

**B.**

**+---+---+**

| a| b|

+---+---+

| 1| 2|

+---+---+

**C.**

**+---+---+**

| a| b|

+---+---+

| 1| 3|

+---+---+

**D.**

**+---+**

| b|

+---+

| 2|

| 3|

+---+

**Ans.: A.**

**+---+---+**

| a| b|

+---+---+

| 1| 2|

| 1| 3|

**+---+---+**

**Explanation:**

**Explode function on dataframe column is used to flatten nested columns or array of columns.**

Topic: Best Practices

**Q 6:  If your RDD is so large that all of it's elements won't fit in memory on the drive machine then which of the following is not recommended and would cause performance and memory issues?**

**A. take()**

**B. takeSample()**

**C. collect()**

**D. show()**

**Ans.: C. collect()**

**Explanation:**

**if RDD size is too large, then collect() is not recommended as it would take entire RDD into memory and data might be too big to fit into memory, you can write out RDD to files or export the RDD to a databse that is large enough to hold all the data.**

Topic: Advanced troubleshooting

**Q 7:  what is not one of possible solution for resolving following exception in spark initialisation:**

**org.apache.spark.SparkException: Job aborted due to stage failure: Task not serializable: java.io.NotSerializableException: ...**

**A. Declare the instance only within the lambda function passed in map.**

**B. use flatmap operation instead of map operation**

**C. Make the NotSerializable object as a static and create it once per machine.**

**D. Call rdd.forEachPartition and create the NotSerializable object in there like this:**

rdd.forEachPartition(iter -> {

NotSerializable notSerializable = new NotSerializable();

// ...Now process iter

});

**Ans.: B. use flatmap operation instead of map operation**

**Explanation:**

**Even if flatmap is used, serializable exception would occur and to avoid exception, A, C or D option should be tried to serialie the class**

Topic: Architecture design

**Q 8:  which of the following would not be helpful to avoid exception from following code:**

JavaStreamingContext jssc = new JavaStreamingContext(sc, INTERVAL);

// This enables checkpointing.

jssc.checkpoint("/tmp/checkpoint\_test");

JavaDStream<String> dStream = jssc.socketTextStream("localhost", 9999);

NotSerializable notSerializable = new NotSerializable();

dStream.foreachRDD(rdd -> {

if (rdd.count() == 0) {

return null;

}

String first = rdd.first();

notSerializable.doSomething(first);

return null;

}

);

**A. Turning off checkpointing**

**B. Make the object being used Serializable.**

**C. Declare NotSerializable inside forEachRDD function**.

**D. Removing notSerializable object creation code**

**Ans.: D. Removing notSerializable object creation code**

**Explanation:**

code would cause"ERROR OneForOneStrategy: java.io.NotSerializableException

To avoid exception any of option A, B or C should be done

Topic: Performance

**Q 9:  which of the following is incorrect on spark joins?**

**A. if spark.sql.crossJoin.enabled = false then select \* from A inner join B doesn’t throw an error**

**B. if spark.sql.crossJoin.enabled = false then select \* from A cross join B doesn't throw an error**

**C. if spark.sql.crossJoin.enabled = true then select \* from A inner join B doesn't throw an error**

**D. if spark.sql.crossJoin.enabled = true then select \* from A cross join B doesn't throw an error**

**Ans.: A. if spark.sql.crossJoin.enabled = false then select \* from A inner join B doesn’t throw an error**

**Explanation:**

**if spark.sql.crossjoin.enabled is set to false then inner join throws an error**

Topic: Architecture design

**Q 10:  In spark to predict CPU/task resources, which of the following formulas or equations would be incorrect or not fit for calculation estimation of resources?**

**A. number of jobs = number of output ops**

**B. number of stages per job = number of shuffles + 1**

**C. number of tasks per job = number of stages \* number of partitions**

**D. number of tasks = number of jobs**

**Ans.: D. number of tasks = number of jobs**

**Explanation:**

**Each job can have multiple stages and each stage can have number of tasks associated with it so number of tasks would be greater than number of jobs.**