Big-Data-Assessment-Questions

Table of Contents

1. What is the default storage of cache()?

- The default storage level of `cache()` is `MEMORY_ONLY`.
- When you call `cache()` on an RDD or DataFrame in Spark, it stores the data **in memory only** as deserialized Java objects. This means Spark tries to keep the dataset in RAM to speed up future computations.
- If there isn't enough memory to store the entire dataset, Spark **does not spill the data to disk** by default. Instead, the parts that don't fit in memory will be **recomputed** when

needed.

2. What is the result of List(1, 2, 3).zip(List("a", "b", "c"))?

```
List((1, "a"), (2, "b"), (3, "c"))
```

3. What is the return type of val ds = Seq(1, 2, 3).toDS()?

- The return type is a **`Dataset[Int]`**.
- `toDS()` converts a **Scala sequence (`Seq`)** into a **Spark Dataset**.
- Since the sequence contains integers, the resulting Dataset holds `Int` values, so its type is `Dataset[Int]`.
- A **Dataset** is a distributed collection of typed objects that provides the benefits of both RDDs (strong typing, functional programming) and DataFrames (optimized execution).
- Terms
  - **Dataset**: A distributed collection of strongly-typed objects in Spark, combining the benefits of RDDs and DataFrames.
  - **DataFrame**: A Dataset organized into named columns (essentially `Dataset[Row]`).
  - **`toDS()`**: A method to convert Scala collections or DataFrames to Datasets.

4. What is Zookeeper in Kafka?

- Zookeeper is a **coordination service** used by Kafka.
- Zookeeper helps manage and coordinate Kafka brokers in the cluster.
- It keeps track of **which brokers are alive**, **which topics and partitions exist**, and **who is the leader for each partition**.
- This coordination helps Kafka maintain **high availability** and **fault tolerance**.
- Zookeeper also helps in **managing configuration**, **cluster metadata**, and **distributed synchronization**
- Terms

- **Kafka Broker**: A server that stores and serves Kafka topics.
- **Partition Leader**: A broker responsible for handling read/write for a partition.
- **High Availability**: Ensuring the system remains available even if some nodes fail.
- **Distributed Coordination**: Managing multiple distributed nodes to work together smoothly.

5. Why is Apache Spark considered an integrated solution for all Lambda architecture layers?

- Because Apache Spark can **handle batch processing, real-time streaming, and serving layers all within one unified platform**.
- Lambda architecture divides data processing into three layers:
  - **Batch Layer:** Processes large volumes of historical data.
  - **Speed (Real-time) Layer:** Processes data streams in real-time for quick insights.
  - **Serving Layer:** Combines results from batch and speed layers to answer queries
- Apache Spark supports all these layers by:
  - Using **Spark Core** and **Spark SQL** for batch processing.
  - Using **Spark Streaming** or **Structured Streaming** for real-time data processing.
  - Using **Spark SQL or MLlib** for serving processed data and analytics.
- Terms
  - **Lambda Architecture:** A design pattern for handling big data by combining batch and streaming processing.
  - **Spark Streaming:** Spark's real-time processing engine.
  - **Structured Streaming:** A newer, higher-level streaming API in Spark.
  - **Serving Layer:** The layer that answers user queries by combining batch and streaming results.

6. What does the following code print: val numbers = List(4, 99, 2, 348, 99, 1).takeOrdered(2)?

```
List(1, 2)
```

- `takeOrdered(n)` returns the **first n smallest elements** from the list in ascending order.
- Here, `takeOrdered(2)` takes the **2 smallest numbers** from the list `[4, 99, 2, 348, 99, 1]`.
- The two smallest numbers are **1** and **2**, so it returns `List(1, 2)`.
- Terms
  - **takeOrdered(n):** Returns the smallest `n` elements sorted in ascending order.
  - **List:** An immutable sequence in Scala.

7. What does the following code print

```
val mammals = List("Lion", "Dolphin", "Whale")
val lengths = mammals.map(_.length)
println(lengths)
```

- `map(_.length)` takes each string in the list and returns its length (number of characters).
- For `"Lion"` → 4 characters
- For `"Dolphin"` → 7 characters
- For `"Whale"` → 5 characters
- So the output is a list of these lengths: `List(4, 7, 5)`.

8. When does Spark evaluate an RDD?

- **Upon Action**
- Spark follows **lazy evaluation**, which means it **doesn't execute operations immediately**.
- When you define transformations like `map`, `filter`, etc., Spark just **builds a plan**.
- The actual computation (evaluation) **only happens when an *action*** like `collect()`, `count()`, or `saveAsTextFile()` is called.
- Terms
  - **Transformation**: A step like `map`, `filter`, `flatMap` that defines how to process data, but **does not run immediately**.
  - **Action**: A step like `count`, `collect`, `reduce`, which **triggers Spark to compute and return results**.

- **Lazy Evaluation**: Spark waits until it knows what final output is needed before doing any work — this saves resources and optimizes the plan.

9. Spark Core's fast scheduling for streaming is leveraged by?

- **Ans: Spark Streaming**
- **Spark Streaming** is a component of Apache Spark that lets you process **real-time data streams** (like logs, sensor data, social feeds).
- It works by **breaking streaming data into small batches** (called micro-batches).
- These batches are processed using the **same Spark Core engine** that handles batch jobs.
- Because Spark Core is designed for **fast, in-memory computation and task scheduling**, Spark Streaming can **process data with very low latency**.
- Terms
  - **Spark Core**: The foundation of Apache Spark; provides basic functionality like task scheduling, memory management, fault recovery.
  - **Spark Streaming**: A high-level API for processing real-time data streams using Spark's batch processing model (DStream or Structured Streaming).
  - **Micro-batching**: Dividing continuous data into small time-based chunks to process them as mini-batches.

10. Columns in HBase are organized into?

- **Ans: Column families**
- In **HBase** (a NoSQL database built on top of Hadoop), data is stored in **tables**, but not like traditional RDBMS.
- Instead of rows and columns, HBase stores data as **key-value pairs** inside **column families**.
- A **column family** is a **logical group of columns** stored together on disk.
- Each table in HBase:
  - Has **one or more column families**
  - Columns are grouped under these families

    - Each **cell** in HBase is uniquely identified by:
        - `Row Key + Column Family + Column + Timestamp`
- Terms
    - **Column Family**: A group of related columns stored together (e.g., `personal:name`, `personal:age` → `personal` is the column family).
    - **HBase**: A distributed, scalable, big data store that supports real-time read/write.
    - **Row Key**: Unique ID for each row.
    - **Schema-less Columns**: You don't have to pre-define individual columns—just the families.

11. Each version of data in a cell adds versioning through?

- **Ans: KeyValue**
- In **HBase**, each cell (i.e., a combination of row key, column family, and column) can **store multiple versions** of the same data. These versions are managed using **timestamps**, and each version is stored as a **KeyValue** pair.

A KeyValue in HBase includes

```
(Key) → RowKey + ColumnFamily + Column + Timestamp
(Value) → Actual data (cell content)
```

- So if the same cell is updated multiple times, HBase keeps the old versions as well, distinguished by their **timestamp** inside the **KeyValue** structure.
- Terms
    - **KeyValue**: A data structure in HBase that stores both the key parts (row key, column family, column, timestamp) and the value (actual data).
    - **Versioning**: HBase can keep multiple versions of data in the same cell, usually controlled by timestamp and configured version limits.
    - **Cell**: The intersection of a row and a column.

Example

Suppose you insert these two values into the same cell:

```
Put("user1", "info:name") = "Alice"  (timestamp 1)
Put("user1", "info:name") = "Alicia" (timestamp 2)
```

HBase stores both as different **KeyValue** entries, and you can choose how many versions to keep.

RowKey	Column Family	Column (Qualifier)	Timestamp	Value
user1	info	name	1	Alice
user1	info	name	2	Alicia

12. Which command fetches contents of a row or cell in HBase?

- **Ans: get**
- The `get` command in HBase is used to **retrieve data** from a **specific row**, and optionally a specific **column family or column**.

13. An HBase table can be?

- **Ans: Dropped after disabling**
- In HBase, before you can **drop (delete)** a table, you must first **disable** it.
- Disabling a table makes it **inactive and unavailable** for read/write operations.
- Once disabled, you can safely **drop (delete) the table and all its data**.
- This two-step process prevents accidental data loss.

14. What is the general-purpose computing model for distributed data analytics?

- **Ans: Mapreduce**

- **MapReduce** is a programming model used to process and generate large datasets across distributed clusters.
- It breaks down a task into two main steps:
  - **Map**: Processes input data and produces key-value pairs.
  - **Reduce**: Aggregates or processes those key-value pairs to produce the final output.
- This model allows parallel processing of data on multiple machines, enabling big data analytics.

15. Which scenario is not a good fit for HDFS?

- HDFS is not suitable for scenarios requiring multiple/simultaneous writes to the same file
- HDFS (Hadoop Distributed File System) is designed primarily for **high-throughput, batch processing of large files**.
- It supports **write-once, read-many** access model — meaning once a file is written, it cannot be modified or appended by multiple writers simultaneously.
- This makes HDFS **unsuitable for applications that need many users or processes to write to the same file at the same time** (like transactional systems or databases).
- Terms
  - **Write-once, read-many (WORM)**: Files can be written only once but read many times.
  - **Batch Processing**: Processing large amounts of data in bulk, typically offline.
  - **Transactional Systems**: Systems requiring frequent, concurrent updates and immediate consistency.

16. HDFS breaks large files into?

- **Blocks**
- In HDFS (Hadoop Distributed File System), **large files are split into fixed-size pieces called blocks**.
- Each block is typically **128 MB** (default size, can be configured).
- These blocks are stored across different nodes in the cluster, enabling **distributed storage and parallel processing**.
- Breaking files into blocks allows HDFS to handle very large files efficiently.

- Terms
  - **Block**: The smallest unit of data storage in HDFS.
  - **DataNode**: The node in HDFS cluster that stores data blocks.
  - **NameNode**: The master node that keeps metadata of files and block locations.

17. What is the default block size in Hadoop 2.0?

- **128MB**

18. The Hadoop Tool interface is used for?

- **Any Java MapReduce application driver**
- The **Hadoop Tool interface** is a Java interface that helps developers run their **MapReduce programs** easily.
- It provides a standard way to **configure, parse command-line arguments, and run** MapReduce jobs.
- Implementing the `Tool` interface allows your application to integrate well with the Hadoop ecosystem and use Hadoop's configuration features.
- Terms
  - **MapReduce Application Driver**: The main class that sets up and runs a MapReduce job.
  - **Configuration**: Hadoop settings that control job behavior, resources, etc.
  - **ToolRunner**: A utility that helps run `Tool` implementations with command-line parsing.

19. When is a Cartesian Product Join needed?

- When we do not specify the key on which we want to make the join

20. Hive is mainly designed for?

- OLAP
- **Hive** is a data warehouse system built on top of Hadoop.
- It is designed to support **OLAP workloads**, which means it's optimized for **complex queries, reporting, and analysis of large datasets**.
- Hive uses a SQL-like language called **HiveQL** to query data stored in Hadoop's HDFS or other storage systems.
- It is **not designed for OLTP** (transactional workloads), which require fast, frequent inserts, updates, and deletes.

21. Why is data denormalized in Hive?

- **Avoid multiple disk seeks and Improve the performance**
- In **Hive**, data is often **denormalized**, which means **combining related data into a single table** instead of spreading it across many smaller related tables.
- This is done because Hive is built on top of **HDFS**, which is optimized for **sequential reads** and **batch processing**, not for fast lookups or joins.
- If data were highly normalized (like in traditional RDBMS), Hive would need to do **lots of joins** → this leads to **multiple disk reads (disk seeks)** and **slower performance**.
- So by **denormalizing**:
  - We **reduce the number of joins**,
  - Make **queries faster**, and
  - Take advantage of Hive's **read-heavy, write-once** nature.

22. Partitioning in Hive creates more?

- **subdirectories under the table name**

23. What is the default database when logging into Hive?

- **default**

24. How does Hadoop make the system more resilient?

- **It keeps multiple copies of data**
- **Hadoop** uses **HDFS (Hadoop Distributed File System)** to store data.
- To prevent **data loss in case of hardware failure**, HDFS **stores multiple copies (replicas)** of each data block across different machines (nodes) in the cluster.
- By default
  - Each block is **replicated 3 times** (can be configured).
  - So even if one or two machines fail, the data is still safe and available from other copies.

25. Which are the three major parallel computing platforms?

- **Clusters or grids, MPP, HPC**

1. Clusters or Grids

- A collection of **independent machines** working together.
- Often used in **big data platforms** like Hadoop or Spark.
- Focused on **scalability and fault tolerance**.

2. MPP (Massively Parallel Processing)

- A system where **many processors** work in **parallel**, each with its own memory and copy of the OS.
- Common in **data warehouses** and systems like **Amazon Redshift**, **Teradata**, etc.
- Best for **large-scale database queries**.

3. HPC (High Performance Computing)

- Focuses on solving **complex calculations** like scientific simulations, weather prediction, etc.

- Involves **supercomputers or specialized hardware**.
- Optimized for **speed and precision** over fault tolerance.

26. How is data from GPS and Web classified?

- **Both structured and unstructured data**

27. Which lists all types of Big Data?

- **Structured Data, Unstructured Data and Semi Structured Data**

28. What format does the Kafka property file use?

- **key-value pairs**

29. Kafka can serve as an?

- **external commit-log**

30. True or False: Combiner increases reducer work by reducing network traffic

- **False**
- A **combiner** is a **mini-reducer** that runs **on the map side**, **before** the data is sent across the network to reducers.
- Its **main purpose** is to **reduce the amount of data transferred** to the reducers — **not increase their work**.
- So it actually
    - **Reduces network traffic**

    - **Reduces reducer workload**
    - **Improves performance**
- Terms
    - **Combiner**: Optional function in MapReduce that aggregates intermediate data after the map phase.
    - **Reducer**: The phase that processes the final grouped data.
    - **Network Shuffle**: The process of transferring map output to reducers — combiner helps minimize this.

31. A generalization of output collection in MapReduce is?

- **OutputCollector**
- In **early versions of Hadoop MapReduce**, the `OutputCollector` class was used to **collect key-value pairs** emitted by the **Mapper** and **Reducer**.
- It's a **general-purpose interface** for output collection during MapReduce processing.

32. What do Mapper and Reducer use to report progress?

Reporter

- In older versions of Hadoop `Reporter` is used by **Mapper** and **Reducer** tasks to:
    - Report **progress**
    - Set **status messages**
    - **Update counters** (for custom metrics)

33. Input to the Reducer is?

- **The sorted output of the mappers**
- In **MapReduce**, after the **map phase**, the output key-value pairs are:
    - **Shuffled** across the cluster to the appropriate reducers.
    - **Sorted by key** before being fed into the reducer.

- So, each **Reducer** receives a **sorted list of key-value pairs**, grouped by key.
- This sorting and grouping allow the reducer to process all values for a given key together.

34. What does '1@ a'.isprintable() return?

- **True**
- The Python string method `.isprintable()` returns **True** if **all characters in the string are printable** (letters, digits, punctuation, spaces).
- In `'1@ a'`:
  - `1` is a digit (printable)
  - `@` is a symbol (printable)
  - Space is printable
  - `a` is a letter (printable)
- Since there are **no non-printable characters** (like `\n`, `\t`, or control characters), the method returns **True**.

35. SBT command to create a JAR:

- **package**
- **SBT (Simple Build Tool)** is a popular build tool for Scala projects.
- The command `package` compiles the project and **creates a JAR file** in the `target/` directory.
- This JAR contains your compiled classes and resources, which you can run or distribute.

36. List(2,4,6,7).flatMap(x => List(x,5)) gives?

- **List(2,5,4,5,6,5,7,5)**
- The `flatMap` function applies the given function to each element of the list and **flattens** the results into a single list.
- For each element `x`, the function `x => List(x, 5)` creates a **list with two elements: the original element and 5**.

- Applying this to each element in `List(2,4,6,7)`:
  - `2` → `List(2, 5)`
  - `4` → `List(4, 5)`
  - `6` → `List(6, 5)`
  - `7` → `List(7, 5)`
- Flattening all these together gives:
  - `List(2, 5, 4, 5, 6, 5, 7, 5)`

Related Terms:

- **flatMap**: Combines map and flatten operations.
- **map**: Applies a function to each element, returning a list of lists if function returns lists.
- **flatten**: Converts list of lists into a single list.

37. Where is a Sqoop-imported table stored?

- **In HDFS inside a directory with the same name of the table**
- When you import data from a relational database into Hadoop using **Sqoop**, it stores the imported data as files in **HDFS**.
- Sqoop creates a directory in HDFS with the **same name as the table** from the source database.
- Inside this directory, the table data is stored in one or more files (usually in text or other formats).

38. Can multiple clients write into an HDFS file concurrently?

- **Answer: False**
- **HDFS allows only a single writer to a file at a time.**
- This means multiple clients cannot write to the same file simultaneously.
- HDFS uses a **write-once, read-many** model — once a file is being written, other writes to that file are blocked until the write finishes.
- This design helps maintain data consistency and simplifies system architecture.

39. Once compilation and optimization complete, the executor executes the tasks

- **Answer: True**
- In systems like **Apache Spark**, when you submit a job:
  - The job is **compiled** into tasks.
  - The tasks are **optimized** (e.g., task scheduling, pipelining).
  - Then, **executors** on worker nodes **run these tasks** in parallel.
- Executors are the **processes that actually perform the computations** on the data.

40. Map operator trees are executed on mapper

- **Answer: True**
- In **Hadoop MapReduce**, the **map operator tree** means all the map-side operations (like filtering, transforming, mapping) done on input data
- These operations happen **on the mapper nodes** — the machines where the map tasks run.
- This local processing helps **reduce data size and prepare it** before sending to reducers

41. Checkpointing is a feature for any non-stateful transformation.

- **False**
- **Checkpointing** is mainly used for **stateful transformations** in systems like Apache Spark Streaming.
- It **saves the state and progress of the streaming job** to a reliable storage (like HDFS) so that the job can recover from failures.
- For **non-stateful transformations** (which don't maintain state across batches), checkpointing is usually **not required**.
- Checkpointing helps with **fault tolerance and recovery** when maintaining state over time.
- Terms

- **Stateful Transformation**:
  - Operations that remember information across batches
  - It **remembers past data** and updates it with new data.
- **Non-stateful Transformation**:
  - You **only look at the current batch** of data without remembering anything from before.
  - Operations that process each batch independently.
  - No memory of past data is kept.
- **Checkpointing**: Saving progress and state to durable storage to recover from failures.

42. Hive Shell can run in Non-Interactive mode and Interactive mode

- **True**
- Interactive
  - You run queries **directly in the Hive shell** (`hive` command in terminal).
- Non-Interactive
  - You run Hive commands **from a file or script**.
  - Useful for **automated jobs** or when you want to run a batch of queries.

43. Which of the following is a platform for constructing data flows for extract, transform, and load (ETL) processing and analysis of large datasets.

- **Pig Latin**
- Pig Latin is the language used by Apache Pig, a platform for ETL and data flow construction.

44. Which of the following jobs are optimized for scalability but not latency?

- **Hive**
- **Hive** is built for **batch processing** of large datasets using **MapReduce** under the hood.
- It is **optimized for scalability**, meaning it can handle **huge volumes of data** across many machines.
- But it’s **not designed for low latency** — queries can take **seconds to minutes** to run, depending on size.