**Big Data & Apache Spark Summary**

**1. Hadoop & HDFS (Hadoop Distributed File System)**

Hadoop is a framework for distributed data storage and processing across clusters of computers.

**1.1 HDFS Architecture**

* **NameNode**:
  + Stores metadata (file paths, block locations, replication details).
  + Keeps track of the overall file system structure.
* **DataNode**:
  + Stores actual data blocks.
  + Sends heartbeat signals to the NameNode to indicate availability.
* **Replication Factor**:
  + Ensures fault tolerance by keeping multiple copies of data (default = 3).
  + If a node fails, data can still be retrieved from other copies.

**1.2 File Storage in HDFS**

* Files are split into **blocks (default 128MB or 256MB)** and distributed across nodes.
* Hadoop follows a **Write Once, Read Many (WORM)** model.
* Data is stored in **a distributed manner across multiple nodes** for fault tolerance and parallel processing.

**2. Apache Spark Overview**

Apache Spark is a distributed computing framework that provides fast, scalable, in-memory data processing.

**2.1 Spark vs. Hadoop MapReduce**

| **Feature** | **Spark** | **Hadoop MapReduce** |
| --- | --- | --- |
| Processing | In-memory | Disk-based |
| Speed | Faster | Slower |
| Iterative Jobs | Optimized | Inefficient |
| Ease of Use | Python, Scala, Java | Java-based |

**2.2 Spark Components**

1. **Driver**:
   * The central process that **orchestrates execution** of Spark applications.
   * Builds **DAG (Directed Acyclic Graph)** of transformations.
2. **Executors**:
   * Perform **actual computations** on data.
   * Store intermediate results for fast access.
3. **Cluster Manager (YARN, Standalone, Mesos, Kubernetes)**:
   * Allocates **resources** (CPU & memory) for Spark jobs.

**3. Spark Execution Workflow**

1. **Spark reads data from a source** (HDFS, GCS, S3, JDBC, etc.).
2. **Driver program** creates a DAG (Directed Acyclic Graph) of transformations.
3. **Job is divided into Stages**, and stages into Tasks.
4. **Executors process tasks** in parallel across worker nodes.
5. **Results are collected** or written back to storage.

**3.1 DAG (Directed Acyclic Graph)**

* Represents the **logical execution plan** of Spark operations.
* Divides the **job into stages**, which contain tasks that run in parallel.

**4. Spark RDDs (Resilient Distributed Datasets)**

**4.1 RDD Basics**

* Immutable, distributed collections of data.
* Automatically **partitioned across worker nodes**.
* Supports **fault tolerance** via lineage tracking.

**4.2 RDD Transformations & Actions**

| **Transformations (Lazy)** | **Actions (Eager)** |
| --- | --- |
| map() | collect() |
| filter() | count() |
| flatMap() | reduce() |
| groupByKey() | take() |
| reduceByKey() | foreach() |

* **Transformations** are **lazy** (not executed immediately).
* **Actions** trigger execution.

**5. Spark DataFrame API**

* A **distributed table-like structure** optimized for performance.
* Uses **Catalyst Optimizer** and **Tungsten Execution Engine** for efficiency.
* Supports SQL queries (df.select("column").show()).
* Better than RDDs for **structured data processing**.

**6. Spark Storage (Memory & Disk Management)**

**6.1 In-memory Computation**

* Stores intermediate results in RAM.
* Reduces disk I/O overhead.

**6.2 Caching & Persistence**

* df.cache() → Stores data in memory.
* df.persist(StorageLevel.DISK\_ONLY) → Stores on disk.

**7. Partitioning & Parallel Processing**

* Spark **divides data into partitions** to allow parallel execution.
* **HDFS-based processing** benefits from data locality.
* **GCS (Google Cloud Storage) lacks data locality**, but parallelism is achieved via distributed execution.

**8. Spark Execution Modes**

| **Mode** | **Description** |
| --- | --- |
| Local Mode | Runs on a single machine. |
| Standalone | Spark's built-in cluster manager. |
| YARN | Runs Spark on Hadoop clusters. |
| Kubernetes | Containerized Spark deployment. |

**9. Data Formats (Parquet vs. Avro vs. CSV)**

| **Format** | **Type** | **Best For** |
| --- | --- | --- |
| Parquet | Columnar | Big Data Analytics |
| Avro | Row-based | Data Exchange |
| CSV | Row-based | Simple structured data |

**10. Spark & Cloud (Dataproc vs. Hadoop)**

| **Feature** | **Hadoop (On-Prem)** | **Dataproc (Cloud)** |
| --- | --- | --- |
| Data Storage | HDFS | Google Cloud Storage (GCS) |
| Cluster Usage | Always running | Ephemeral (temporary) |
| Cost | Higher maintenance | Pay-as-you-go |
| Data Locality | Yes | No (uses network transfer) |

**11. Spark Memory Management**

* **Driver Memory**: Stores **DAG, metadata, and small collected results**.
* **Executor Memory**: Allocated across:
  + Task Execution
  + Caching
  + Shuffle Memory

**11.1 Handling Out-of-Memory Issues**

* Increase spark.executor.memory and spark.driver.memory.
* Use **broadcast variables** instead of large joins.
* Optimize **shuffle operations** using reduceByKey() instead of groupByKey().

**12. Recovering RDDs & Fault Tolerance**

* **RDD Lineage** ensures lost partitions can be recomputed.
* **Checkpointing** saves RDDs to storage to avoid recomputation.

**13. Git & GitHub for PySpark Development**

**13.1 Common Git Commands**

| **Command** | **Description** |
| --- | --- |
| git clone <repo> | Clone a repository |
| git branch | List branches |
| git checkout <branch> | Switch branch |
| git merge <branch> | Merge changes |
| git push origin <branch> | Push changes |

**14. Apache Airflow & DAG Scheduling**

* **Airflow DAGs** automate PySpark job execution.
* Use PythonOperator and BashOperator to trigger Spark jobs.
* Cloud Composer runs Airflow in GCP.

**15. PySpark Code Deployment & CI/CD**

* **Jenkins Pipeline** automates Spark script deployment.
* Uses **GitHub → Jenkins → GCS → Dataproc** workflow.
* Ensures code quality with **SonarQube & CyberFlow Scans**.