Basics:

**Apache Spark** - A fast, general-purpose cluster-computing framework for large-scale data processing. It competes with MapReduce but can use Hadoop's HDFS for distributed storage.

**Spark Core** - The main component that provides basic functionalities such as task scheduling, memory management, fault recovery, and interaction with storage systems.

**RDD (Resilient Distributed Dataset)** - A fault-tolerant collection of elements that can be operated on in parallel. They are immutable and can be created from Hadoop InputFormats or by transforming other RDDs.

**Partitions** - Logical divisions of data within RDDs that allow for parallel processing. Each partition can be processed independently.

**Transformations** - Functions applied to RDDs to create new RDDs, such as map() and filter(). They are lazily evaluated.

**Actions** - Operations that trigger execution and return results, such as reduce() and collect(). They bring data back to the local machine from RDDs.

**DAG (Directed Acyclic Graph)** - A scheduling layer in Spark that represents RDD transformations to optimize execution. It ensures no cyclic dependencies in the sequence of computations.

**Spark Driver** - The program that runs on the master node, coordinates the execution of tasks on RDDs, and manages the SparkContext.

**SparkContext** - The entry point for a Spark application, responsible for connecting to the cluster and creating RDDs.

**Shuffling** - The process of redistributing data across partitions, which can involve moving data across nodes.

**Lazy Evaluation** - Spark's optimization technique where transformations on RDDs are not executed until an action is called, optimizing the data processing workflow.

**YARN (Yet Another Resource Negotiator)** - A cluster management technology from Hadoop used for resource management and job scheduling, allowing Spark to run on top of it for consistent operations, security, and data governance.

**MapReduce** - A programming model for processing large datasets with a parallel, distributed algorithm on a cluster.

**Spark Engine** - Responsible for scheduling, distributing, and monitoring applications across a cluster.

**Spark Session** - The entry point for using Spark functionalities, replacing SparkContext in versions of Spark 2.0 and later.

**RDD Lineage** - A process that reconstructs lost data partitions by keeping track of the transformations that created an RDD.

**Deploy Modes** - Two main deploy modes in Spark are Client Mode (driver runs on the submitting machine) and Cluster Mode (driver runs on the cluster's worker nodes).

**Receivers** - Components in Spark Streaming that consume data from different data sources and move it into Spark for processing.

**Repartition vs. Coalesce** - Repartition increases/decreases partitions with a full shuffle, while coalesce reduces partitions without a full shuffle.

**Data Formats** - Spark supports formats such as Parquet, JSON, XML, CSV, RC, Avro, and TSV.

**Spark Job** - A sequence of transformations and actions performed on data in a Spark application.

**Adaptive Query Execution (AQE)** - Dynamically optimizes query plans at runtime based on actual data characteristics to improve performance.

**HashJoin vs. Broadcast Join** - HashJoin uses hash-based partitioning for large datasets, while Broadcast Join distributes a small dataset across all nodes for efficient joining.

**Accumulator Variables** - Used for counting and summing values across multiple tasks, with updates sent from executors to the driver.

**Broadcast Variables** - Distribute read-only data efficiently to all executors, avoiding repeated transfer from the driver.

**Catalyst Optimizer** - An optimization framework in Spark SQL that transforms logical query plans into optimized physical execution plans.

**Tungsten Project** - Enhances Spark's performance through memory management, code generation, and optimizing CPU and memory usage.

\*\*What is Apache Spark and how does it differ from Hadoop MapReduce?\*\*

Apache Spark is an open-source, distributed computing system that performs in-memory data processing, enabling faster execution compared to Hadoop MapReduce, which relies on disk-based storage. Spark supports various data processing tasks, including batch processing, real-time processing, and iterative algorithms, through its advanced DAG execution engine.

\*\*Describe the architecture and main components of Spark.\*\*

Spark's architecture includes the Spark Driver, which runs the main program and coordinates tasks, and Spark Executors, which run on worker nodes to execute tasks and store data. The Cluster Manager (e.g., YARN, Mesos, Kubernetes) manages resources across the cluster.

\*\*What are RDDs and how do they function in Spark?\*\*

RDDs (Resilient Distributed Datasets) are the fundamental data structures in Spark, representing immutable, distributed collections of objects that can be processed in parallel. RDDs support two types of operations: transformations (e.g., `map`, `filter`) that define a new RDD and actions (e.g., `count`, `collect`) that return a result to the driver.

\*\*Explain the Spark ETL process.\*\*

The ETL (Extract, Transform, Load) process in Spark involves extracting data from various sources, transforming it using Spark operations (e.g., filtering, aggregation), and loading it into target systems like data warehouses or storage solutions. Spark's distributed computing capabilities make ETL processes efficient and scalable.

\*\*What are Datasets and DataFrames in Spark, and how do they differ from RDDs?\*\*

DataFrames are distributed collections of data organized into named columns, similar to a table in a relational database. Datasets are type-safe, object-oriented collections of data that provide the benefits of both RDDs and DataFrames. Unlike RDDs, DataFrames and Datasets offer optimized execution through the Catalyst optimizer.

\*\*How does Spark's deployment and WebUI work?\*\*

Spark can be deployed on various cluster managers like YARN, Mesos, and Kubernetes. The Spark WebUI provides a graphical interface for monitoring and managing Spark applications, displaying information about the DAG, stages, tasks, and job execution status.

\*\*What is the Catalyst optimizer and how does it improve Spark performance?\*\*

The Catalyst optimizer is a rule-based optimization engine in Spark SQL that automatically transforms logical query plans into optimized physical plans. It improves performance by applying optimizations such as predicate pushdown, column pruning, and join reordering.

\*\*What is the Tungsten Project and how does it enhance Spark's performance?\*\*

The Tungsten Project is a Spark initiative aimed at improving execution efficiency through memory management, code generation, and whole-stage code generation. It enhances performance by optimizing CPU and memory usage, reducing garbage collection overhead, and generating optimized bytecode for execution.

\*\*What is the role of a Spark Job and Spark Session in Spark workflows?\*\*

A Spark job consists of transformations and actions performed on data, while a Spark session is the entry point for using Spark functionalities. Spark jobs and sessions are integral to data processing workflows, facilitating data preparation, execution, and monitoring through distributed processing.

\*\*How do narrow transformations affect datasets in Spark?\*\*

Narrow transformations (e.g., `map`, `filter`) do not cause data shuffling and operate on a single partition. They result in more efficient execution since data movement between nodes is minimized, making them suitable for tasks that can be completed within the same partition.

\*\*Describe how Avro and Parquet file formats are used in Spark.\*\*

Avro and Parquet are columnar storage formats optimized for efficient storage and retrieval. They reduce I/O operations by compressing and organizing data efficiently, making them suitable for big data applications where read/write performance is critical.

\*\*What is data skew, and how can it be identified and fixed in Spark?\*\*

Data skew occurs when partitions are unevenly distributed, leading to imbalanced processing where some nodes are overloaded while others are idle. It can be identified by analyzing task execution times and partition sizes. To fix data skew, techniques like repartitioning to redistribute data evenly, salting keys to spread data more uniformly, or using custom partitioners can be employed.

\*\*How does adaptive query execution (AQE) enhance Spark's performance?\*\*

Adaptive Query Execution (AQE) dynamically optimizes query plans at runtime based on actual data characteristics. It adjusts join strategies, reduces the number of shuffle partitions, and reoptimizes query stages, improving performance and efficiency.

\*\*What are HashJoin and Broadcast Join in Spark, and when are they used?\*\*

HashJoin uses hash-based partitioning to join large datasets efficiently, while Broadcast Join distributes a small dataset across all nodes to join with a larger dataset without shuffling. Broadcast joins are more efficient for small lookup tables, reducing data transfer and execution time.

\*\*What are accumulator variables in Spark and how do they work on Executors and Driver?\*\*

Accumulators are used for counting and summing values across multiple tasks. Executors update the accumulator values locally during task execution and send these updates to the driver. The driver aggregates the values to maintain a global state.

\*\*What are broadcast variables in Spark and how do they work on Executors and Driver?\*\*

Broadcast variables distribute read-only data efficiently to all executors, avoiding repeated transfer of large data sets from the driver to executors. The driver broadcasts the data once, and executors cache the broadcasted data locally, making it available for tasks.

**Can you explain the key features of Apache Spark?** Apache Spark boasts several key features, including support for multiple programming languages (Java, Python, R, Scala), lazy evaluation, machine learning capabilities through MLib, compatibility with various data sources, real-time computation, exceptional speed (up to 100 times faster than Hadoop MapReduce), and seamless integration with Hadoop.

**What advantages does Spark offer over Hadoop MapReduce?** Spark excels over Hadoop MapReduce in terms of enhanced speed due to in-memory processing, multitasking support, reduced disk dependency, and the ability to perform iterative computations.

**Please explain the concept of RDD (Resilient Distributed Dataset) and how you can create RDDs in Apache Spark.** RDD is a fault-tolerant collection of operational elements distributed across nodes, created by parallelizing a collection in the driver program or loading an external dataset from storage systems like HBase, HDFS, etc.

**What are the various functions of Spark Core?** Spark Core acts as the base engine for large-scale parallel and distributed data processing, handling job distribution, monitoring, scheduling on clusters, interacting with storage systems, and managing memory and fault recovery.

**Enumerate the various components of the Spark Ecosystem.** The Spark Ecosystem includes components like GraphX for graph processing, MLib for machine learning, Spark Core as the base engine, Spark Streaming for real-time data processing, and Spark SQL for integrating functional programming API with relational processing.

**Is there any API available for implementing graphs in Spark?** GraphX is the API used for implementing graphs and graph-parallel computing in Apache Spark, extending Spark RDD with a Resilient Distributed Property Graph.

**What do you mean by the worker node?** A worker node is a node in a cluster capable of running code, assigned tasks by the master node, processing data, and reporting resources to the master node for task scheduling.

**Please explain the sparse vector in Spark.** A sparse vector in Spark efficiently stores non-zero entries using two parallel arrays for indices and values.

**How will you connect Apache Spark with Apache Mesos?** Connecting Apache Spark with Apache Mesos involves configuring the Spark driver program, placing the Spark binary package where Mesos can access it, installing Apache Spark in the same location as Mesos, and configuring the spark.mesos.executor.home property.

**Can you explain how to minimize data transfers while working with Spark?** Minimizing data transfers in Spark involves avoiding operations that trigger shuffles, using accumulators for parallel variable updates, and employing broadcast variables to enhance efficiency in joins.

**Does Apache Spark provide checkpoints?** Yes, Apache Spark provides checkpoints, enabling programs to run continuously and recover from failures unrelated to application logic using lineage graphs.

**What are the different levels of persistence in Spark?** Spark offers various persistence levels, including DISK\_ONLY, MEMORY\_AND\_DISK, MEMORY\_ONLY\_SER, MEMORY\_AND\_DISK\_SER, MEMORY\_ONLY, and OFF\_HEAP, each with distinct characteristics for storing RDDs.

**Can you list down the limitations of using Apache Spark?** Limitations of Apache Spark include the lack of a built-in file management system, higher latency with lower throughput, micro-batch processing instead of true real-time data stream processing, fewer available algorithms, the need for distribution over multiple clusters, and challenges with cost-efficient in-memory processing.

**What are the Key Features of the Spark Ecosystem?** The Spark Ecosystem includes speed (up to 100 times faster than Hadoop in memory), ease of use (APIs in Python, Java, Scala, and R), modular design (including Spark SQL, MLlib, GraphX, Spark Streaming), integration with Hadoop, fault tolerance via RDDs, and support for advanced analytics.

**Explain what RDD is?** RDD stands for Resilient Distributed Dataset, Spark's fundamental data structure representing an immutable, distributed collection of objects that can be processed in parallel. RDDs support two types of operations: transformations (creating new RDDs) and actions (returning values to the driver program).

**What does DAG refer to in Apache Spark?** DAG stands for Directed Acyclic Graph, representing a sequence of computations performed on data. When Spark runs an application, it creates a DAG of tasks to be executed, allowing optimization by rearranging computations and minimizing data shuffling.

**List the types of Deploy Modes in Spark.** Spark supports two main deploy modes: Cluster Mode (driver runs inside the cluster) and Client Mode (driver runs on the machine that initiated the Spark job).

**What is YARN in Spark?** YARN (Yet Another Resource Negotiator) is a cluster management technology from Hadoop used for resource management and job scheduling, allowing Spark to run on top of it for consistent operations, security, and data governance.

**Explain how Spark runs applications with the help of its architecture.** Spark applications run as independent processes coordinated by the SparkSession object in the driver program. The resource manager assigns tasks to worker nodes, which process data and send results back to the driver or save them to disk.

**What is a lazy evaluation in Spark?** Lazy evaluation means that Spark does not immediately execute the transformations on an RDD but instead builds a logical plan until an action is called, optimizing the data processing workflow.

**How can you trigger automatic clean-ups in Spark to handle accumulated metadata?** Automatic clean-ups in Spark can be triggered by setting the parameter spark.cleaner.ttlx.

**What does Spark do with Akka?** Akka is used by Spark for scheduling, facilitating communication between the master and worker nodes.

**What is Executor Memory in a Spark application?** Executor memory is the memory allocated to each executor in a Spark application, controlled by the spark.executor.memory property. It determines how much memory each executor uses on the worker nodes.

**Define Partitions in Apache Spark.** Partitions are smaller, logical divisions of data in Spark that enable parallel processing. They help optimize data transformations and minimize network traffic.

**What operations does RDD support?** RDDs support transformations (e.g., map, filter) and actions (e.g., reduce, collect). Transformations create new RDDs, while actions return values or write data to storage.

**What do you understand by Transformations in Spark?** Transformations are operations on RDDs that produce a new RDD. They are lazy, meaning they do not execute until an action is called.

**Define Actions in Spark.** Actions are operations that trigger the execution of transformations to return results or save data to an external storage system.

**What do you understand by Pair RDD?** Pair RDDs are key-value pair RDDs that allow operations on each key in parallel. They support operations like reduceByKey and join.

**What file systems does Spark support?** Spark supports Hadoop Distributed File System (HDFS), local file systems, and Amazon S3.

**Explain how Spark uses Akka.** Spark uses Akka for scheduling and communication between the master and worker nodes.

**What are the various persistence levels in Apache Spark?** Persistence levels in Spark include MEMORY\_ONLY, MEMORY\_AND\_DISK, MEMORY\_ONLY\_SER, MEMORY\_AND\_DISK\_SER, DISK\_ONLY, and OFF\_HEAP.

**What does DAG refer to in Apache Spark?**

* DAG (Directed Acyclic Graph) in Apache Spark represents a sequence of computations performed on data, ensuring no cyclic dependencies and forming a directed graph of operations.

**List the types of Deploy Modes in Spark.**

* **Client Mode:** The driver runs on the submitting machine.
* **Cluster Mode:** The driver runs on the cluster's worker nodes.

**What are receivers in Apache Spark Streaming?**

* Receivers consume data from sources and move it into Spark Streaming, running as long-term tasks and can be either reliable or unreliable.

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

* **Repartition:** Increases or decreases partitions with a full shuffle, suitable for increasing partitions.
* **Coalesce:** Reduces partitions without a full shuffle, suitable for decreasing partitions.

**What are the data formats supported by Spark?**

* Spark supports Parquet, JSON, XML, CSV, RC, Avro, and TSV formats.

**What do you understand by Shuffling in Spark?**

* Shuffling redistributes data across partitions, causing potential data movement between executors or JVM processes, essential for operations like joins and aggregations.

**Compare MapReduce with Spark.** Spark is faster due to in-memory computation and offers better APIs for iterative algorithms and interactive data analysis, unlike MapReduce which relies on disk storage.

**What does a Spark Engine do?** The Spark engine is responsible for scheduling, distributing, and monitoring applications across a cluster.

**Define Partitions.** Partitions are smaller, logical divisions of data in Spark, which allow for distributed processing.

**What operations does an RDD support?** RDD supports transformations (e.g., map, filter) and actions (e.g., reduce, collect).

**What do you understand about Transformations in Spark?** Transformations create new RDDs from existing ones and are evaluated lazily.

**Define Actions in Spark.** Actions trigger the execution of transformations to return results or write data.

**What is RDD Lineage?** RDD lineage is the process through which lost data is recomputed using the transformations that created it, ensuring fault tolerance.

**What is Spark Driver?** The Spark Driver runs on the master node and is responsible for converting a user program into tasks and scheduling them on executors.

SQL:

**Spark SQL** - A Spark module for structured data processing, providing DataFrame APIs and SQL-like query capabilities.

**Delta Lake** - A storage layer designed to work with Apache Spark, enabling ACID transactions, schema enforcement, and scalable metadata handling.

**Delta Table** - A data collection stored using Delta Lake technology, including delta files, a transaction log, and a table registered in the metastore.

**Atomicity** - An ACID property ensuring that each transaction is all-or-nothing, either fully completed or fully failed.

**Consistency** - An ACID property ensuring that transactions move the database from one valid state to another, maintaining database rules.

**Isolation** - An ACID property ensuring that concurrent transactions result in the same state as if the transactions were executed sequentially.

**Durability** - An ACID property guaranteeing that once a transaction is committed, it remains so even in the event of a system failure.

**Delta Engine** - A high-performance, Spark-compatible query engine for efficient data lake processing, especially with Delta Lake.

**Delta Lake Transaction Log (DeltaLog)** - An ordered record of all transactions performed on a Delta Lake table, ensuring data consistency and integrity.

**Schema Enforcement** - Also known as schema validation, it ensures data quality by rejecting writes that do not match the table's schema.

**Schema Evolution** - A feature that allows for the modification of a table's schema to accommodate changing data structures over time.

**DataFrame** - A distributed collection of data organized into named columns, similar to a table in a relational database.

**Dataset** - A type-safe, object-oriented collection of data that combines the benefits of RDDs and DataFrames, optimized through the Catalyst optimizer.

**Catalyst Optimizer** - A rule-based optimization engine in Spark SQL that transforms logical query plans into optimized physical plans for improved performance.

**Data Source API** - An API allowing Spark SQL to connect to various data sources, enabling read and write operations for structured data.

**Spark Session** - The entry point for using Spark SQL, allowing for the creation of DataFrames, execution of SQL queries, and configuration of Spark settings.

**Managed Table** - A table where Spark SQL manages both metadata and data, storing them in the default warehouse location.

**Unmanaged Table** - A table where Spark SQL manages only the metadata, with data stored externally.

**User-Defined Functions (UDFs)** - Custom functions defined by users to extend the functionality of Spark SQL, enabling complex operations on DataFrame columns.

**Thrift JDBC/ODBC Server** - A server that enables connectivity for BI tools to execute SQL queries on Spark using standard JDBC/ODBC protocols.

**Temporary View** - A session-scoped view that exists only within the Spark session.

**Global Temporary View** - A system-wide view that is visible to all Spark sessions and persists until Spark is stopped.

**JSON Data Handling** - Spark SQL's capability to read and write JSON data using its built-in JSON data source, enabling seamless integration with JSON files.

**Explain Method** - A method in Spark SQL used to display the physical and logical execution plans of a query, aiding in performance optimization.

A diagram of data architecture

Description automatically generated

\*\*What is SparkSQL and how does it integrate with Spark?\*\*

SparkSQL is a Spark module for working with structured data using SQL queries and the DataFrame API. It allows seamless integration with Spark applications, enabling users to perform complex queries and analytics on structured data using familiar SQL syntax.

\*\*How does the Catalyst optimizer work within SparkSQL?\*\*

The Catalyst optimizer transforms logical query plans into optimized physical plans using a set of rules and strategies. It enhances query performance by applying various optimization techniques such as predicate pushdown, constant folding, and join reordering.

\*\*What are the differences between bronze, silver, and gold data stages in Delta Lake?\*\*

In Delta Lake, the bronze stage contains raw, ingested data. The silver stage includes cleaned and enriched data, and the gold stage holds aggregated and business-level data ready for analysis. These stages represent different levels of data refinement.

\*\*When a Delta Table is created, what files are generated along with it?\*\*

When a Delta Table is created, it generates Parquet files for storing the actual data and transaction log files for maintaining ACID transactions and versioning information.

\*\*What are the benefits of using Delta Table in Spark?\*\*

Delta Tables provide ACID transactions, scalable metadata handling, efficient storage with Parquet, and time travel capabilities. These features ensure data integrity, support for schema evolution, and enhanced query performance.

\*\*How does schema evolution and enforcement work in Delta Lake?\*\*

Schema evolution allows changes to the schema over time, such as adding or modifying columns, without affecting existing data. Schema enforcement ensures that all data written to a Delta table adheres to the defined schema, preventing invalid data insertion and maintaining data quality.

\*\*What kind of operations can you perform on Delta Tables?\*\*

Operations on Delta Tables include update, delete, merge (upsert), and time travel. These operations enable efficient data management and querying, ensuring consistency and enabling historical data analysis.

\*\*How can you time travel with Delta Tables?\*\*

Time travel in Delta Tables allows querying historical data using version numbers or timestamps. This feature enables users to access previous states of the data for audits, backfill data, or recover from accidental deletions.

\*\*What is the purpose of the Unity Catalog in Spark?\*\*

Unity Catalog is a unified governance solution for data and AI that provides centralized metadata management, access control, and data lineage tracking. It simplifies data management, ensures consistent access policies, and improves data discoverability and compliance.

\*\*How do you identify and traverse different table versions in Delta Lake?\*\*

Different versions of Delta Tables can be identified and traversed using version numbers or timestamps. This allows querying historical states of the data, making it easy to access previous versions for auditing or recovery purposes.

\*\*What is the role of logical and physical plans in SparkSQL?\*\*

The logical plan represents high-level operations on data, constructed from the user's SQL queries or DataFrame operations. The physical plan details the actual execution strategy, including tasks and their dependencies. The Catalyst optimizer converts the logical plan into an optimized physical plan, selecting the most efficient execution strategy.

\*\*What are the main features and benefits of the DataFrame API in SparkSQL?\*\*

The DataFrame API provides a high-level abstraction for working with structured data, offering benefits such as optimization through Catalyst, ease of use with SQL-like operations, and integration with various data sources. It supports a wide range of transformations and actions, enabling efficient data manipulation and analysis.

\*\*How does SparkSQL handle different data sources?\*\*

SparkSQL can connect to and query data from various sources, including JSON, Parquet, Avro, ORC, JDBC, and Hive. It provides a unified interface for reading and writing data, allowing seamless integration and interoperability with different data formats and storage systems.

\*\*What is the significance of the DataFrame and Dataset APIs in SparkSQL?\*\*

The DataFrame and Dataset APIs provide a higher level of abstraction over RDDs, enabling more efficient and optimized data processing. DataFrames offer an SQL-like interface, while Datasets provide type safety and object-oriented programming capabilities, combining the benefits of RDDs and DataFrames.

\*\*Explain the concept of schema inference in SparkSQL.\*\*

Schema inference is the process by which SparkSQL automatically determines the schema of a dataset based on its structure and contents. This feature simplifies data loading and processing, as users do not need to manually define schemas for well-known formats like JSON and Parquet.

\*\*How does SparkSQL's support for Hive improve its functionality?\*\*

SparkSQL integrates with Apache Hive, allowing users to query and manage Hive tables using Spark's execution engine. This integration provides compatibility with existing Hive data warehouses, supports HiveQL, and leverages Hive's metastore for schema management and data cataloging.

\*\*What is the role of the SQLContext and SparkSession in SparkSQL?\*\*

SQLContext and SparkSession are entry points for working with structured data in SparkSQL. SparkSession is a unified entry point that supersedes SQLContext, providing a consistent interface for DataFrame and SQL operations, managing configuration, and interacting with underlying Spark functionality.

\*\*How does SparkSQL optimize join operations?\*\*

SparkSQL optimizes join operations through the Catalyst optimizer, which applies various techniques such as join reordering, broadcast joins, and predicate pushdown. These optimizations reduce data shuffling, minimize I/O operations, and improve overall query performance.

\*\*What are the different join types supported by SparkSQL?\*\*

SparkSQL supports various join types, including inner join, outer join (left, right, full), semi join, and anti join. These join types enable flexible data merging and filtering based on specific conditions and relationships between datasets.

\*\*How does SparkSQL handle partitioning and bucketing?\*\*

SparkSQL uses partitioning to divide data into smaller chunks based on specific columns, improving query performance and parallelism. Bucketing further subdivides partitions into buckets, distributing data evenly and enabling efficient joins and aggregations on bucketed columns.

\*\*What are the advantages of using SparkSQL for data warehousing and BI?\*\*

SparkSQL offers several advantages for data warehousing and business intelligence (BI), including support for SQL queries, integration with BI tools, scalability, performance optimizations, and compatibility with various data sources. It enables complex analytics and reporting on large datasets with ease.

**How will you implement SQL in Spark?** Spark SQL modules facilitate the integration of relational processing with Spark’s functional programming API. It supports querying data via SQL or HiveQL and provides libraries like DataFrame API, Data Source API, Interpreter & Optimizer, and SQL Service.

**What do you understand by the Parquet file?** Parquet is a columnar format supported by Spark SQL for read and write operations. It offers advantages such as fetching specific columns, space efficiency, type-specific encoding, limited I/O operations, and better-summarized data.

**How does SparkSQL handle different data sources?** SparkSQL can connect to and query data from various sources, including JSON, Parquet, Avro, ORC, JDBC, and Hive. It provides a unified interface for reading and writing data, allowing seamless integration and interoperability with different data formats and storage systems.

**Explain the concept of schema inference in SparkSQL.** Schema inference is the process by which SparkSQL automatically determines the schema of a dataset based on its structure and contents. This feature simplifies data loading and processing, as users do not need to manually define schemas for well-known formats like JSON and Parquet.

**How does SparkSQL's support for Hive improve its functionality?** SparkSQL integrates with Apache Hive, allowing users to query and manage Hive tables using Spark's execution engine. This integration provides compatibility with existing Hive data warehouses, supports HiveQL, and leverages Hive's metastore for schema management and data cataloging.

**What is the role of the SQLContext and SparkSession in SparkSQL?** SQLContext and SparkSession are entry points for working with structured data in SparkSQL. SparkSession is a unified entry point that supersedes SQLContext, providing a consistent interface for DataFrame and SQL operations, managing configuration, and interacting with underlying Spark functionality.

**How does SparkSQL optimize join operations?** SparkSQL optimizes join operations through the Catalyst optimizer, which applies various techniques such as join reordering, broadcast joins, and predicate pushdown. These optimizations reduce data shuffling, minimize I/O operations, and improve overall query performance.

**What are the different join types supported by SparkSQL?** SparkSQL supports various join types, including inner join, outer join (left, right, full), semi join, and anti join. These join types enable flexible data merging and filtering based on specific conditions and relationships between datasets.

**How does SparkSQL handle partitioning and bucketing?** SparkSQL uses partitioning to divide data into smaller chunks based on specific columns, improving query performance and parallelism. Bucketing further subdivides partitions into buckets, distributing data evenly and enabling efficient joins and aggregations on bucketed columns.

**How can you manipulate structured data using domain-specific language in Spark SQL?** Structured data in Spark SQL can be manipulated using DataFrame operations such as select, filter, groupBy, and agg to query and transform data.

**How can you connect Hive to Spark SQL?** To connect Hive to Spark SQL, place the hive-site.xml file in the Spark conf directory and use the SparkSession object to construct a DataFrame.

**What is the role of Catalyst Optimizer in Spark SQL?** The Catalyst optimizer leverages advanced programming language features to build an extensible query optimizer, optimizing query plans for improved performance.

**What are the functions of Spark SQL?** Spark SQL loads data from various structured data sources, queries data using SQL statements, integrates SQL with Python/Java/Scala code, and provides functionalities to join RDDs and SQL tables.

**How can Apache Spark be used alongside Hadoop?** Spark can leverage Hadoop's HDFS for storage, run on YARN for resource management, and complement Hadoop's MapReduce for batch and real-time processing.

**How can you connect Hive to Spark SQL?** Place the hive-site.xml file in Spark's configuration directory and use the Spark Session object to query Hive tables.

**What are the various data sources available in Spark SQL?** Data sources in Spark SQL include Parquet files, JSON datasets, and Hive tables.

**Explain the concept of DataFrames in Spark SQL.** DataFrames are distributed collections of data organized into named columns, similar to tables in a relational database, providing an easy-to-use API for data manipulation.

**What is the DataFrame API?** The DataFrame API allows for the manipulation of structured data, providing high-level methods for querying and transforming data efficiently.

**What module is used for implementing SQL in Apache Spark?**

* SparkSQL is the module for relational data processing, supporting SQL and Hive Query Language (HQL) and integrating with various data sources.

**What are the different persistence levels in Apache Spark?**

* **MEMORY\_ONLY:** Default, stores RDDs as deserialized Java objects in JVM.
* **MEMORY\_AND\_DISK:** Stores RDDs as deserialized Java objects in JVM, spills to disk if necessary.
* **MEMORY\_ONLY\_SER:** Stores RDDs as serialized Java objects.
* **MEMORY\_AND\_DISK\_SER:** Similar to MEMORY\_ONLY\_SER, but spills to disk if necessary.
* **DISK\_ONLY:** Stores RDD partitions only on disk.
* **OFF\_HEAP:** Stores data in off-heap memory.

**How is Apache Spark different from MapReduce?**

* Spark processes data in real-time and batches, runs 100 times faster, stores data in memory for low latency, and has its job scheduler, unlike MapReduce.

**Explain the working of Spark with the help of its architecture.**

* Spark applications run as independent processes coordinated by the SparkSession object. The cluster manager assigns tasks to worker nodes. Each task processes data within its partition, and results are returned to the driver program.
* **What is Spark SQL?** Spark SQL is a module for structured data processing that allows running SQL queries alongside Spark programs.
* **List the functions of Spark SQL.** Functions include loading data from structured sources, running SQL queries, integrating with existing RDDs, and providing rich data handling capabilities.
* **What is a Parquet file?** A Parquet file is a columnar storage file format optimized for use with Spark SQL.
* **What file systems does Apache Spark support?** Spark supports HDFS, local file systems, Amazon S3, HBase, and Cassandra.
* **What is Hive on Spark?** Hive on Spark allows Hive queries to execute using the Spark execution engine for better performance compared to traditional MapReduce.
* **What is Directed Acyclic Graph in Spark?** A DAG is a series of vertices and edges where vertices represent RDDs and edges represent the operations applied on RDDs, forming an acyclic graph.

ML:

**MLlib** - A scalable machine learning library in Spark, providing common learning algorithms and utilities.

**Supervised Learning** - A type of machine learning where the algorithm is trained on labeled data to predict outcomes.

**Unsupervised Learning** - A type of machine learning where the algorithm analyzes and clusters unlabeled data.

**Semi-Supervised Learning** - Combines a small amount of labeled data with a large amount of unlabeled data during training.

**Reinforcement Learning** - The machine learning method where an agent learns to make decisions by receiving rewards or penalties.

**Feature Extraction** - The process of deriving useful information from raw data to form features for model building.

**Feature Selection** - Identifying the most relevant features for use in model construction.

**Correlation** - Measures the linear relationship between two or more variables.

**Regression Algorithms** - Techniques for predicting continuous values, including linear, ridge, logistic, and lasso regression, and SVM.

**Confusion Matrix** - A table used to describe the performance of a classification model.

**Accuracy** - The ratio of correctly predicted observations to the total observations.

**Precision** - The ratio of true positive predictions to the total positive predictions.

**Recall** - The ratio of true positive predictions to the total actual positives.

**F1 Score** - The harmonic mean of precision and recall.

**Bias** - The difference between the model's predicted values and actual values due to simplistic assumptions.

**Variance** - The model's sensitivity to fluctuations in the training data.

**Underfitting** - A model that is too simple to capture the underlying patterns of the data.

**Overfitting** - A model that is too complex, capturing noise along with the underlying pattern.

**Bias–Variance Tradeoff** - Balancing the model complexity to minimize both bias and variance.

**Training Data** - Data used to train the model.

**Validation Data** - Data used to tune the hyperparameters and feature selection.

**Test Data** - Data used to evaluate the performance of a trained model.

**Cross-Validation** - A method to evaluate the generalizability of a model by partitioning the data into subsets.

**Parameters** - Properties that an algorithm learns from the training data.

**Hyperparameters** - Properties set before training that guide the learning process.

**Hyperparameter Optimization** - The process of finding the best hyperparameter values to improve model performance.

**Grid Search** - Exhaustively searching through a manually specified subset of hyperparameters.

**Random Search** - Randomly sampling hyperparameter values from a specified distribution.

**Coarse-to-Fine** - Exploring a broad range of variables first, then narrowing the focus.

**Bayesian Optimization** - Using a probabilistic model to find the best hyperparameters.

**DataFrame** - ML API in Spark that uses DataFrame as an ML dataset.

**Transformer** - An algorithm that transforms one DataFrame into another.

**Estimator** - An algorithm that can be fit on a DataFrame to produce a transformer.

**Pipeline** - A sequence of stages, each either a transformer or an estimator, to streamline the workflow.

**Param** - A named parameter with documentation.

**MLflow** - An open-source platform for managing the machine learning lifecycle.

**MLflow Tracking** - An API and UI for logging and visualizing parameters, code versions, metrics, and output files.

**MLflow Model** - A standard format for packaging machine learning models.

**Flavors** - Conventions that deployment tools use to understand models, enabling compatibility across various ML libraries.

**Sparse Vector** - A data structure that efficiently stores non-zero elements using parallel arrays for indices and values.

**Dense Vector** - A data structure that stores all elements, including zeros, in a contiguous block of memory.

**One-Hot Encoding** - A process of converting categorical data into a binary matrix representation.

**StandardScaler** - A transformer that standardizes features by removing the mean and scaling to unit variance.

**PCA (Principal Component Analysis)** - A dimensionality reduction technique that transforms data into a set of orthogonal components.

**K-Means Clustering** - An unsupervised algorithm that partitions data into k distinct clusters based on feature similarity.

**Decision Trees** - A supervised learning method used for classification and regression tasks that splits data into branches to build a model.

**Random Forest** - An ensemble learning method that builds multiple decision trees and merges them to improve accuracy and prevent overfitting.

\*\*What are the different model types supported in Spark MLlib?\*\*

Model types in Spark MLlib include clustering (e.g., k-means), regression (e.g., linear regression), classification (e.g., logistic regression), and collaborative filtering (e.g., ALS for recommendations). These models support various machine learning tasks such as grouping similar items, predicting continuous values, and categorizing data points.

\*\*How do you measure the performance of a machine learning model in Spark?\*\*

Model performance is measured using metrics like accuracy (percentage of correct predictions), precision (positive predictive value), F1 score (harmonic mean of precision and recall), and sensitivity (true positive rate). These metrics evaluate different aspects of a model's performance, helping to determine its effectiveness in various scenarios.

\*\*Explain the difference between ETL and ELT in the context of Spark.\*\*

ETL (Extract, Transform, Load) transforms data before loading it into the target system, making it ready for analysis upon arrival. ELT (Extract, Load, Transform) loads raw data into the target system first and then transforms it as needed, leveraging the processing power of the target system for transformation tasks.

\*\*What is the role of a Spark Job and Spark Session in machine learning workflows?\*\*

A Spark job consists of transformations and actions performed on data, while a Spark session is the entry point for using Spark functionalities. Spark jobs and sessions are integral to machine learning workflows, facilitating data preparation, model training, and evaluation through distributed processing.

\*\*How do narrow transformations affect datasets in Spark?\*\*

Narrow transformations (e.g., `map`, `filter`) do not cause data shuffling and operate on a single partition. They result in more efficient execution since data movement between nodes is minimized, making them suitable for tasks that can be completed within the same partition.

\*\*Describe how Avro and Parquet file formats are used in Spark.\*\*

Avro and Parquet are columnar storage formats optimized for efficient storage and retrieval. They reduce I/O operations by compressing and organizing data efficiently, making them suitable for big data applications where read/write performance is critical.

\*\*What is data skew, and how can it be identified and fixed in Spark?\*\*

Data skew occurs when partitions are unevenly distributed, leading to imbalanced processing where some nodes are overloaded while others are idle. It can be identified by analyzing task execution times and partition sizes. To fix data skew, techniques like repartitioning to redistribute data evenly, salting keys to spread data more uniformly, or using custom partitioners can be employed.

\*\*How does adaptive query execution (AQE) enhance Spark's performance?\*\*

Adaptive Query Execution (AQE) dynamically optimizes query plans at runtime based on actual data characteristics. It adjusts join strategies, reduces the number of shuffle partitions, and reoptimizes query stages, improving performance and efficiency.

\*\*What are HashJoin and Broadcast Join in Spark, and when are they used?\*\*

HashJoin uses hash-based partitioning to join large datasets efficiently, while Broadcast Join distributes a small dataset across all nodes to join with a larger dataset without shuffling. Broadcast joins are more efficient for small lookup tables, reducing data transfer and execution time.

\*\*How does Spark support model training and evaluation?\*\*

Spark supports model training and evaluation through MLlib, which provides a range of algorithms for classification, regression, clustering, and collaborative filtering. MLlib also includes utilities for feature extraction, transformation, and selection, as well as tools for model evaluation and validation.

\*\*What are the key components of Spark MLlib?\*\*

Spark MLlib consists of three main components: algorithms for machine learning tasks (e.g., classification, regression), utilities for feature extraction and transformation (e.g., tokenization, scaling), and tools for pipeline construction and evaluation. These components work together to support scalable and efficient machine learning workflows.

\*\*How does Spark handle distributed model training?\*\*

Spark handles distributed model training by parallelizing the computation across multiple nodes in a cluster. It divides the training data into partitions and processes them in parallel, aggregating the results to build the final model. This approach leverages the power of distributed computing to handle large datasets and complex models efficiently.

\*\*What are the benefits of using Spark for machine learning?\*\*

Using Spark for machine learning offers several benefits, including scalability, fault tolerance, and integration with other Spark components. Spark's distributed architecture allows it to handle large datasets and complex computations efficiently, while its support for in-memory processing accelerates model training and evaluation.

\*\*How does Spark integrate with other machine learning libraries?\*\*

Spark integrates with other machine learning libraries through APIs and connectors. For example, it can interface with TensorFlow and Keras for deep learning, allowing users to leverage Spark's distributed computing capabilities to train and deploy deep learning models on large datasets.

**How does Spark support model training and evaluation?** Spark supports model training and evaluation through MLlib, which provides a range of algorithms for classification, regression, clustering, and collaborative filtering. MLlib also includes utilities for feature extraction, transformation, and selection, as well as tools for model evaluation and validation.

**What are the key components of Spark MLlib?** Spark MLlib consists of three main components: algorithms for machine learning tasks (e.g., classification, regression), utilities for feature extraction and transformation (e.g., tokenization, scaling), and tools for pipeline construction and evaluation. These components work together to support scalable and efficient machine learning workflows.

**How does Spark handle distributed model training?** Spark handles distributed model training by parallelizing the computation across multiple nodes in a cluster. It divides the training data into partitions and processes them in parallel, aggregating the results to build the final model. This approach leverages the power of distributed computing to handle large datasets and complex models efficiently.

**What are the benefits of using Spark for machine learning?** Using Spark for machine learning offers several benefits, including scalability, fault tolerance, and integration with other Spark components. Spark's distributed architecture allows it to handle large datasets and complex computations efficiently, while its support for in-memory processing accelerates model training and evaluation.

**What are the different MLlib tools available in Spark?** MLlib provides tools for ML algorithms (classification, regression, clustering, collaborative filtering), featurization (extraction, transformation, dimensionality reduction), pipelines (construction, evaluation, tuning), persistence (saving/loading models), and utilities (linear algebra, statistics).

**Describe how model creation works with MLlib and how the model is applied.** MLlib has two components: Transformer (reads a DataFrame and returns a transformed DataFrame) and Estimator (a machine learning algorithm that trains a model and returns it as a Transformer). Models are applied using a pipeline to live data.

**What are the different data types supported by Spark MLlib?** Spark MLlib supports local vectors (dense and sparse), labeled points (local vectors with a label), local matrices, and distributed matrices (RowMatrix, IndexedRowMatrix, CoordinatedMatrix).

**What is a Sparse Vector?** A Sparse Vector is a type of local vector represented by an index array and a value array, efficiently storing non-zero entries.

**What are the different data types supported by Spark MLlib?** Spark MLlib supports local vectors and matrices (dense and sparse), labeled points, and distributed matrices (RowMatrix, IndexedRowMatrix, CoordinatedMatrix).

**What is a Sparse Vector?** A sparse vector is a type of local vector that efficiently stores non-zero entries using parallel arrays for indices and values.

**Describe how model creation works with MLlib and how the model is applied.** MLlib models are created using Transformers and Estimators. Transformers apply transformations to data, while Estimators train models from data, producing Transformers.

**Explain the role of Spark Streaming in machine learning.** Spark Streaming enables real-time data processing, allowing for the continuous training and updating of machine learning models with streaming data.

**What are the different MLlib tools available in Spark?** MLlib tools include ML algorithms (classification, regression, clustering), feature extraction, transformation, dimensionality reduction, and selection.

**Explain how Spark can access and analyze data stored in Cassandra databases.** Using the Spark Cassandra Connector, Spark can access and analyze data stored in Cassandra databases. The connector allows Spark executors to query local Cassandra nodes for data, minimizing network usage and speeding up queries.

**How can you achieve machine learning in Spark?**

* Spark provides MLlib, a scalable machine learning library offering algorithms and utilities for classification, regression, clustering, collaborative filtering, and more.

**What is the difference between Sparse Vectors and Dense Vectors?**

* **Sparse Vectors:** Use two parallel arrays for indices and values, storing non-zero entries to save space.
* **Dense Vectors:** Store all entries, suitable when most values are non-zero.

**What API is used for Graph Implementation in Spark?**

* GraphX is the API for graph and graph-parallel computations, extending Spark RDDs to support graph operations and providing a set of operators and algorithms for graph analytics.

**What can you say about Spark Datasets?**

* Spark Datasets combine features of RDDs and DataFrames, offering type safety, object-oriented programming interface, and optimized query execution through Catalyst optimizer and Tungsten execution engine.
* **What is Spark Streaming?** Spark Streaming is an extension of Spark for processing real-time data streams.
* **What is GraphX?** GraphX is the Spark API for graph and graph-parallel computation, allowing for graph analytics.
* **What does MLlib do?** MLlib is Spark's scalable machine learning library, providing algorithms for classification, regression, clustering, and more.
* **What are Spark Datasets?** Datasets are a distributed collection of data with the benefits of RDDs and optimizations from Spark SQL.
* **What is PageRank in GraphX?** PageRank measures the importance of each vertex in a graph, based on the structure of incoming edges.
* **What is lazy evaluation?** Lazy evaluation in Spark delays the execution of transformations until an action is triggered, optimizing the processing pipeline.

**CAP theorem**, formulated by Eric Brewer, states that in a distributed data store, it is impossible to achieve all three of the following properties simultaneously:

1. \*\*Consistency\*\*: Every read receives the most recent write or an error.

2. \*\*Availability\*\*: Every request receives a response, without guarantee that it contains the most recent write.

3. \*\*Partition Tolerance\*\*: The system continues to operate despite network partitions.

In essence, a distributed system can only guarantee at most two out of these three properties at any given time.

- \*\*CP Systems (Consistency and Partition Tolerance)\*\*: These systems ensure data consistency and can handle network partitions but may not always be available (e.g., HBase, MongoDB with specific configurations).

- \*\*AP Systems (Availability and Partition Tolerance)\*\*: These systems remain available even during partitions but might serve stale data (e.g., Cassandra, DynamoDB).

- \*\*CA Systems (Consistency and Availability)\*\*: These systems provide consistent and available data but cannot handle partitions (though true CA systems are impractical for large-scale systems due to the necessity of partition tolerance).

Understanding CAP helps in making trade-offs when designing distributed systems based on application requirements and network conditions.

\*\*Stack Memory\*\*:

- Used for static memory allocation.

- Stores local variables and function call information.

- Memory is managed in a Last In, First Out (LIFO) manner.

- Automatically managed by the compiler.

\*\*Heap Memory\*\*:

- Used for dynamic memory allocation.

- Stores objects and data that need to persist beyond the scope of a single function.

- Memory is managed manually or via garbage collection.

- Allows for efficient use of memory but requires more management.

In summary, the stack is for short-term, temporary storage with automatic management, while the heap is for longer-term, dynamic storage with manual or automatic management.

Salting modifies keys by adding additional values (salts) to distribute data more evenly across partitions. Without salting, records with the same key cluster are in one partition, causing imbalances and inefficiencies. Salting spreads these records across multiple partitions, ensuring balanced workloads and faster processing times.

### Performance Tuning in Spark

\*\*Executors and Cores:\*\*

Executors run individual tasks in a Spark job, and cores are the CPU threads assigned to each executor.

\*\*Memory Management:\*\*

- \*\*Executor Memory:\*\* Memory allocated to each executor.

- \*\*Off-Heap Memory:\*\* Additional memory outside JVM management to reduce garbage collection overhead.

\*\*Shuffling:\*\*

Data movement between partitions, causing performance bottlenecks during joins, aggregations, and repartitions.

\*\*Caching and Persistence:\*\*

Storing intermediate data in memory or disk for faster iterative algorithms.

\*\*Broadcast Variables:\*\*

Small read-only data distributed to all nodes to avoid repeated large data transfers.

\*\*Data Serialization:\*\*

Efficient formats like Kryo reduce data size for network transmission.

\*\*Catalyst Optimizer:\*\*

Optimizes SQL queries by transforming logical plans into efficient physical plans.

\*\*Tungsten Project:\*\*

Enhances CPU and memory efficiency through whole-stage code generation and improved memory management.

\*\*Task Parallelism:\*\*

Balancing tasks across the cluster to avoid skew and maximize parallel processing.

\*\*Partitioning:\*\*

Proper data partitioning ensures balanced workloads and efficient processing.

### Why Performance Tuning is Important

Performance tuning ensures efficient resource utilization, cost savings, and faster data processing times in big data applications. By focusing on key aspects like memory management, shuffling, caching, and optimizers, Spark can efficiently handle large-scale data tasks, improving execution speed and resource use.

Kryo serializer is an optimized serialization framework for Java, used in Apache Spark to enhance performance during data processing. It is faster than Java's default serializer because it employs more compact encoding techniques, reduces the size of serialized objects, registers classes to avoid writing class names, and uses direct memory access. These features result in improved speed and memory efficiency, making Kryo especially beneficial for handling large datasets in distributed computing environments.

200 partitions

Enable statistics

Difference between ACID in delta and RDBMS

ACID in delta supported only in scope of one table

Delta lake warehouse vs delta table vs delta

Data warehouse - system supporting, only schema on write

Delta lake - supports schema on read

Delta warehosuse - schema on read and acid

Data factory pattern

Joiny с дупликатами