Spark Assignment.

1) what is Spark? why do we require Apache spark
Apache Spark is an Oss open source fast and general purpose in memory processing engine designed for big data processing. It was

Initially developed at the University of California, Berkeley's AMPLab in 2009 and later became an Apache project in 2010. Spark is written in Scala and supports high level APIs in Java, Scala, Python and R. Spark is known for its in memory computing capabilities which significantly speed up data processing.

Why do we required Apache Spark?

1) Unified framework for different tasks: Spark can handle different tasks like batch processing & real-time processing that would have previously required separate engines. This makes it a versatile tool for various data processing needs.

2) Speed: Spark can process data 100 times faster than Hadoop mapreduce when data is processed in memory and 10 times faster when reading/writing from disk.

3) High level operators: Spark provides easy to use high level operators (like map, filter, etc.) to process data efficiently, which are not available in Hadoop mapreduce.

4) Interactive shell: Spark offers an interactive shell (for Scala and python) that allows users to explore and process data quickly, making it easier to learn and experiment with datasets.

5) Supports for multiple storage systems: Spark does not come with its own storage system, but it can work with a variety of storage systems like HDFS, S3 & Cassandra. This flexibility allows it to integrate with existing storage architectures.

2) Difference Btw Spark and Hadoop mapreduce

	Apache Spark	Hadoop MapReduce.
Input processing	In memory processing (data is processed in memory avoiding disk I/O)	Disk-based processing (reads from and writes to disk after each step).
Speed	upto 100 times faster than mapreduce for in memory operations & 10 times faster for disk based operations	slower due to frequent disk I/O especially for iterative tasks.
Best suited for	best suited for iterative processes, real time analytics, graph processing & ML	Best for linear batch processing of large datasets.
Data handling	can handle large data but may struggle with very large datasets when compared to mapreduce due to memory limitations	suitable for processing huge datasets where in memory operations are not feasible
Ease of use	provides high level APIs in scala, python, Java & R, making it easier to write complex operations with less code	Requires more verbose code in Java leading to more complex development.
Fault tolerance	fault tolerance via RDD lineage (can recompute lost data)	fault tolerance via replication & recovery from disk.
Real time processing	supports real time processing with spark streaming	primarily designed for batch processing with native real time support
ML	has built in machine learning library that operates in memory	Requires third party libraries (eg mahout) but may operate on disk
Support	more efficient for iterative computations & makes better use of cluster managers.	Requires more disk space & I/O can be more economical when immediate results are not required.
Programming	comes with Graph X for graph computations	needs external libraries for graph processing
Resilience	Resilient via RDDs (Resilient distributed datasets) when need data to be recomputed if lost.	Handles failure by writing intermediate results to disk which ensures resiliency but at a performance cost.

5) List the areas where Spark & Hadoop mapreduce are good.

→ Areas where Apache Spark is good:

→ Fast data processing: Spark's in memory processing makes it up to 100 times faster for operations that can be done in memory and 10 times faster for disk based operations compared to Hadoop.

→ Iterative processing: Spark excells in scenarios that require repeated processing of the same data, such as ML & graph processing. Its Resilient Distributed Datasets (RDDs) allow iterative operations in memory without writing intermediate results to disk.

→ Near Real time processing: Spark is ideal for use cases requiring near real time insights, such as live data streams, because of its Spark streaming component.

→ Graph processing: Spark's GraphX API is designed for graph computations which require iterative algorithms. Spark handles these efficiently both in memory processing.

→ Machine learning: Spark comes with a built-in MLlib library for machine learning offering fast iterative data processing.

→ Joining datasets: Spark is well suited for joining datasets, especially when combining smaller datasets quickly due to speed.

Areas where Hadoop mapreduce is good:

→ Linear processing of large datasets: Hadoop mapreduce is ideal for sequential processing of very large datasets that exceed available memory. It breaks data into chunks, processes them in parallel, and writes the results to disk.

→ Handling very large datasets: Hadoop mapreduce can process big datasets because it uses disk based storage, making it better suited for tasks where the data volume is too large to fit in memory.

→ Cost efficient processing: Hadoop mapreduce is an economical solution when speed is not critical such as batch processing jobs that can run overnight.

→ Data durability & fault tolerance: Hadoop MapReduce's reliance on writing intermediate data to disk provides stronger fault tolerance for certain scenarios where data loss is unacceptable. If a node fails, data recovery is straightforward due to the persistence of results on disk.

→ Joining very large datasets: For very large datasets that require a lot of shuffling and sorting, Hadoop MapReduce may outperform Spark as it disk based approach can handle greater data volumes without running into memory limits.

How spark manages the cluster & resources?

Spark is a distributed computing system that processes large datasets across multiple machines referred to as spark cluster. Resource management and task scheduling in Spark are handled through a resource manager such as YARN or Mesos. The main components that manage the cluster & resource cluster manager: The cluster manager is responsible for coordinating resources across the machines in the cluster. Spark relies on external resources managers like Apache YARN or Apache Mesos, or can use its own built in standalone resources manager.

Responsibilities of the cluster manager:

- Knows the location of workers in the cluster, as well as the available memory & CPU cores.
- Orchestrates the distribution of work by assigning tasks to worker based on the available resources.

Spark driver is the central coordinator of a spark application. It manages the execution of tasks and interacts with cluster managers.

- Responsibilities of the spark driver:
- Coordinates with the cluster manager to launch process called executors on the worker nodes.
- Distributes the tasks (pieces of the data processing logic) to the workers.

- Collects the results from executors, merges them and returns the final output to the user.
* provides an entry point for configuring and managing spark applications using the SparkSession class.

3) Spark executor is a JVM process that runs on worker nodes in the cluster. It performs the actual data processing tasks assigned by the spark driver. * Responsibilities of the spark executor:
* Executes the tasks (units of computation) and reports the results back to the spark driver.
* Caches data in memory or on disk when requested by the application logic to speed up.
* Reports the task progress and status back to the spark driver.
* The lifetime of an executor corresponding to the duration of a spark application.
4) Resource allocation and scheduling: spark uses the resources manager (YARN, Mesos or standalone) to allocate resources (CPU, memory) for both executors & tasks. Spark's ability to distribute workloads across multiple machines for a cluster allows it to scale for large data processing tasks.
* Spark parallelizes data processing across available executors by breaking large jobs into smaller units (tasks) which can run concurrently on multiple CPU cores within the cluster.
* Dynamic Resource allocation: spark can dynamically adjust the number of executors allocated to an application scaling up or down depending on the workload.
5) Fault tolerance and Recovery: Spark provides fault tolerance by allowing data to be recomputed in case of failure. Spark achieves this through RDD (lineage which tracks the transformations applied to the original dataset and can recover lost data by recomputing it from the source.

5) Write short note on spark driver and executors

The spark driver is the central coordinator of a spark application. It plays a key role in managing the overall execution of the data processing logic. The driver program communicates with the cluster manager to allocate resources & launch executors on the worker nodes.

main Responsibilities.
1) Submit the application to the cluster and initiate the execution of tasks.
2) Interact with the cluster manager to allocate resources.
3) Distribute tasks to the spark executors on worker nodes for parallel execution.
4) Collect and merge results from executors and provide the final output.
5) The entry point for a spark application is typically via a class called SparkSession which handles configurations and provides API's for data processing.

The spark executors are worker processes launched by the spark driver on the cluster's worker nodes. Each executor is responsible for performing the tasks assigned to it by the driver and for storing data in memory or on disk as needed.

main Responsibilities:
1) execute tasks on a separate CPU core on the worker node
2) Cache data in memory to speed up repetitive or iterative processing.
3) Report task status and results back to the spark driver.
4) Each executor is exclusively allocated to a specific spark application ensuring isolation b/w applications.
5) Executors live for the duration of the spark application and their resources are released once the application is complete.

6) Write the spark unified Stack and explain spark core

The spark unified stack is a versatile and powerful framework that consists of several components, each designed for specific data processing tasks. These components are built on top of spark core, which provides the foundational functionality for distributed data processing. The main components of the spark stack are:
1) Spark core: the foundation that manages distributed tasks & provides resilience fault tolerance and memory management.

2) Spark SQL: Handles structured data processing and supports both batch and interactive queries using SQL.

3) Spark streaming: Supports real time data processing, allowing the analysis of live data streams.

3) Spark MLlib: Spark's built in machine learning library offering various algorithms for data mining & ML tasks.

4) Spark GraphX: Designed for graph processing, it supports complex computations typically required for graph analytics.

5) Spark R: An API for running ML tasks using, integrating Spark's power with the R language for statistical computing.

Spark core is the central component and the backbone of Apache Spark. It provides the necessary functionalities for running and managing distributed applications on a cluster. Spark core enables fault tolerance, scheduling, task coordination, and data movement across distributed nodes. It also provides the programming abstractions and APIs for developer to build large scale data processing applications.

Key features of Spark core:

1) Distributed computing infrastructure: Manages the distribution of data & computation across a cluster of machines. Handles task scheduling, coordination and fault tolerance across nodes allowing efficient parallel processing.
* Supports data shuffling - the movement of data between nodes when necessary.

2) Resilient Distributed Datasets (RDDs) - RDDs are the fundamental data structure in Spark core. They are immutable, fault tolerant and distributed collections of objects that can be operated on in parallel.
* RDDs allows user to explicitly persist intermediate results in memory making iterative jobs (like machine learning) much faster.

3) In-memory computation: Spark core excels at in memory computation which allows for much faster processing than traditional disk based approaches. This is particularly useful for iterative algorithms and interactive queries.

4) Fault tolerance: Spark core provides fault tolerance by using RDD lineage. If any part of the computation fails, Spark can recompute lost data based on the sequence of transformations applied to the original dataset.

5) Rich set of operations: Spark core supports a wide variety of operations for data transformation and analysis, such as map, filter, reduce, join, groupBy and many others. These operations allows developers to easily express complex data processing workflows.

6) Language supports: Spark core provides APIs in multiple languages such as Scala, Java, Python, and R making it accessible to a wide range of developers.

What are RDDs? Explain the properties of RDD.

Resilient Distributed Datasets (RDDs) are the fundamental data structure in Apache Spark. They are immutable, distributed collections of objects that are processed in parallel across a Spark cluster. RDDs allow users to perform transformations and actions on large datasets efficiently and with fault tolerance. RDDs provide a high level abstraction for distributed data processing and support fault recovery through lineage. They also enable in-memory data processing, which significantly speeds up iterative computations like those used in machine learning & graph algorithms. properties of RDDs

* RDDs are immutable meaning they cannot be changed after they are created. Any transformation on an RDD results in a new RDD.
* RDDs automatically provide fault tolerance by keeping track of the transformations.
* parallel processing allows data to be distributed across multiple nodes in a cluster.
* In memory computation: One of the most significant advantages of RDD is their support for in memory computation.

* Lazy evaluation meaning the spark doesn't execute transformations immediately.
* Partitions, which are chunks of the data distributed across the nodes in a cluster.
* Persistence RDDs can be cached in memory across the cluster for faster access during iterative operations.
* Rich set of operations - Transformations, Actions
* Coarse grained operation meaning the same operation is applied to many data elements at once, rather than to a single element.

Q8) What are transformation and actions in spark? Explain with examples.

The spark Transformation & Actions are two types of operations that define how data is processed using RDDs.

* Transformations are lazy operations that describe how data in a RDD should be modified or processed to create a new RDD. They do not execute immediately, instead spark records the transformation in the form of a logical plan & execution occurs only when an action is triggered.

map(): Applies a function to each element of the RDD & returns a new RDD.
rdd = spark.sparkContext.parallelize([1,2,3,4])
result = rdd.map(lambda x: x*2) # each element is multiplied by 2
# Result RDD = [2,4,6,8].

filter(): Returns a new RDD containing only the elements that satisfy the given condition.
rdd = spark.sparkContext.parallelize([1,2,3,4])
result = rdd.filter(lambda x: x%2==0) # only even numbers
# Result = RDD: [2,4]

flatMap(): Applies a function to each element of the RDD and flattens the result into a new RDD.
rdd = spark.sparkContext.parallelize(["Hello world", "welcome to spark"])
result = rdd.flatMap(lambda line: line.split(" ")) # splits each string into words.
result = RDD: ["Hello", "world", "welcome", "to", "spark"]

groupByKey(): groups the values for each key in the original RDD.
rdd = spark.sparkContext.parallelize([("A",1), ("B",2), ("A",3)])
result = rdd.groupByKey()
# Result RDD: [("A", [1,3]), ("B", [2])]

Actions are operations that triggers the execution of transformations and returns a result or perform an operation, such as saving data to external storage. Actions force spark to evaluate the computation described by the transformations & return the output.

collect(): returns all the elements of RDD to the driver as a list.
rdd = spark.sparkContext.parallelize([1,2,3,4])
result = rdd.collect(). # collects all elements.
# Result = [1,2,3,4].

count(): Returns the number of elements in the RDD.
rdd = spark.sparkContext.parallelize([1,2,3,4])
result = rdd.count() # counts the elements
# result: 4.

first(): Returns the first element of the RDD.
rdd = spark.sparkContext.parallelize([1,2,3,4])
result = rdd.first() # retrieves the first element # result: 1

reduce(): Aggregates the elements of the RDD using a specified function.
rdd = spark.sparkContext.parallelize([1,2,3,4])
result = rdd.reduce(lambda x,y: x+y) # sums the elements
# result = 10.

saveAsTextFile(): Saves the RDD data to an external storage system.
rdd = spark.sparkContext.parallelize([1,2,3,4])
rdd.saveAsTextFile("/path/to/output") # saves the data to a file.

Key difference: Transformations are lazy & return new RDDs defining the operations but not executing them immediately.
Actions causes the execution of transformations and either return to the driver or write data to an external system.

Q9) what is lazy evaluation in spark? Explain with an example
Lazy evaluation in Spark means the transformations (like map, filter()) are not executed immediately when they are called. Instead spark builds a logical plan or execution DAG (Directed Acyclic Graph) of transformations, and waits until an action (like count(), collect()) is triggered to execute the plan.

```
rdd = spark.sparkContext.parallelize([1,2,3,4,5,6,7,8,9])
transformed_rdd = map(lambda x: x*2)
filtered_rdd = transformed_rdd.filter(lambda x: x>10)
result = filtered_rdd.collect()
print(result) # Output [12,14,16,18]
```

Q10) with logical & Physical plans explain how structured APIs are executed in Spark.
Structured APIs (such as Dataframes and Datasets) provides an abstraction for working with structured & semi-structured data. When executing queries on these APIs Spark uses lazy evaluation to optimize the process through logical and physical plans before performing the actual computation.
Spark's internal optimizer, called Catalyst, generates & optimizes the execution plans for structured APIs.
Logical planning: Spark first generates an unresolved logical plan based on the operations defined in the code.
Physical planning: Spark generates one or more physical execution plans based on the optimized logical plan. These plans specify how the data will be processed on the actual physical cluster.

Q11) Explain some use cases of stream processing.
Stream processing involves real time analysis of continuous data streams, allowing organizations to react quickly to changes in data.
1. Real time Analytics and monitoring: Stream processing enables real time monitoring and analytics, which is crucial for industries that need immediate insights into their operations.
- Website Activity Monitoring, IOT device monitoring.
2. Financial fraud Detection Real time stream processing can analyse transactional data to detect fraud as it happens. By processing data streams, such as credit card transactions, suspicious activities can be identified in real time and action can taken immediately. Fraud Detection in Baking.
3. Social media and Sentiment Analysis: social media platforms generates huge amounts of data every second, and stream processing allows real time analysis of this data for various purposes, such as sentiment analysis & trend detection.
- Real-Time Sentiment Analysis. Stream processing is essential in the advertising industry, where real time bidding (RTB) for ad space must be done in milliseconds.
- cybersecurity and threat detection, supply chain and logistics Healthcare monitoring and alerts, Telecommunications Network optimization. Real time gaming Analytics. Stock market and cryptocurrency trading.

Q12) list out the challenges of stream processing.
1) Handling High throughput and low latency
2) Fault tolerance and Reliability.

* State management * Event time vs processing time
* out of order and late → scalability
* complex event processing * Data consistency &
exactly once processing → windowing & aggregation
monitoring & Debugging & resource management
latency sensitivity * Data Integration, security.

Q(2) Explain continuous stream processing methods. what are its advantages & disadvantages?

continuous stream processing is a method in which data is processed as soon as it arrives, typically in real time or near real time. this done by continuously ingesting and analyzing a stream of data events with low latency processing.

Key characteristics: Real time processing, low latency event driven stateful processing.

Advantages of continuous stream processing.

1) Real time insights - low latency, Scalable & Distributed event driven Architecture, - efficient use of Resources. Stateful processing.

Disadvantages of continuous stream processing complexity in system design, data ordering & late data handling State management, fault tolerance, resource overhead, consistency & exactly once processing Debugging & monitoring

Q(3) Explain continuous stream Processing method. what are its advantages and disadvantages?

continous stream processing method that processes data as it occurs in real time, often in the form of continuous, unbounded data streams. This method enables immediate analysis and action on data as soon as it is produced, without for a complete dataset (as in batch processing). It is essential for applications that require low latency real time decision making and continuous data monitoring.

Key features of continuous stream processing:
event by event processing, low latency, Scalable.

Advantages of continuous stream processing:
Real time decision making low latency, scalability Event Driven processing, Efficiency with stateful operations

Disadvantages of continuous stream processing
complexity in system design out of order and late data State management complexity Higher Resources costs fault tolerance recovery, consistency and exactly once processing monitoring and debugging.

Q(4) Explain micro Batch stream processing method. what are its advantages and disadvantages?

Micro Batch stream processing is a method where data streams are divided into small, time based batches for processing. Instead of processing data event by event (as in continuous stream processing) the incoming data is collected for a short interval and then processed as a small batch.

key features of micro Batch stream processing: Time based batching near Real time Processing fault tolerance.

Advantages of Micro Batch stream processing.
1) simplified fault tolerance 2) Near Real time processing
3) simpler System Design 4) Efficient Resource usage
5) windowing and Aggregation 6) Scalable and Distributed
Disadvantages of micro Batch Stream processing.
1) Increased Latency 2) Limited Real time capabilities.
3) Handling late and out of order data 4) Complex Aggregation Across Batches 5) Higher Resource consumption
6) Resource contention During peaks.

Q15) write short notes on DStreaming and Structured Streaming.
Dstream (discretized stream) is the core Abstraction in Apache Spark Streaming, used to represent a continuous stream of data in micro batches. Dstreams are built on top of Resilient Distributed Datasets (RDDs) where incoming data is split into small batches and each batch is processed as an RDD. This method allows Spark Streaming to provide fault-tolerant, Scalable and real time stream processing.

```
dstream = ssc.socketTextStream("localhost", 9999)
words = dstream.flatMap(lambda line: line.split(" "))
wordCounts = words.map(lambda word: (word, 1)).reduceByKey
    (lambda x, y: x+y)
ssc.start()
```

Advantages of DStreams:
* easy integration with RDDs fault tolerance.
Disadvantages of DStreams * High Latency * Limited Real time processing
* Complexity with Advanced use cases.

Structured Streaming is the next generation stream processing engine introduced in Apache Spark 2.0 it builds on top of spark SQL's Dataframe and Dataset APIs providing a declarative approach to stream processing. Unlike DStream's Structured Streaming is designed for both real time and batch processing offering low latency and continuous processing capabilities with strong exactly once guarantees.
Key features of Structured Streaming: * Declarative API * Continuous processing * Event Time Handling * fault tolerance and exactly once Semantics.

```
lines = spark.readStream.format("socket").option("host","localhost")
    .option("port", 9999).
words = lines.select(explode(split(lines.value, " ")).alias(word))
wordCounts = words.groupBy("word").count().
query = wordCounts.writeStream.outputMode("complete")
    .format("console").start()
query.awaitTermination().
```

Advantages of structure streaming:
* Unified API for Batch and Streaming * Low latency
* Event time support * Exactly once semantics.
Disadvantages of Structured Streaming.
* Complexity for low level control * Maturity.

Q16) write short notes on Data sources and Data sinks of Structured Streaming in spark.
Data sources are the entry points through which streaming data is ingested into spark's structured streaming environment

Structured Streaming supports a variety of data sources allowing it to process real-time data from different systems and formats.

Common Data Sources:

- File source
- Socket source
- Kafka source
- Rate source

Data sinks in Structured Streaming are the endpoints to which processed streaming data is written. it is sent to a data sink for storage, display or further analysis. Structured Streaming supports a range of data sinks to suit various output requirements.

- File sink
- console sink
- Kafka sink
- Memory sink
- foreach sink.