Hadoop Assignment-3

1. what is Spark? why do require Apache Spark?

→ Apache Spark is an open-source, distributed processing system designed for big data workloads. It processes large-scale data across clusters of computers using a high-level API with fault tolerance and in-memory computation.

* Require Apache Spark *

1. Speed: Spark is much faster than Hadoop's MapReduce for data processing tasks because of its in-memory computing capabilities.
2. Big Data Handling: with the explosion of big data, traditional systems can struggle to manage large datasets. Spark can process terabytes or petabytes of data efficiently across distributed clusters.
3. Versatility:- Spark provides libareies for machine learning (MLlib), SQL-like querying (Spark SQL), graph computation (Graphx), and stream processing making it a comprehensive solution for a wide range of data processing needs.
4. Real-Time Data Processing:- Spark's support for streaming data allows it to process and analyze data in real-time, which is crucial for applications like fraud detection,

5. Cost - Efficiency :- It optimizes resource utilization across clusters, which can reduce operational costs while handling big data processing.

2. Differentiate between Spark and Hadoop MapReduce

Spark	Hadoop MapReduce
i) Open Source big data processing framework	* Open Source framework to process structured and
* Faster and general-purpose data processing Engine	* Unstructured data that are stored in the Hadoop distributed file System (HDFS)
* Unified framework to process various tasks such as batch, interactive, iterative processing	* Hadoop MapReduce reads and writes from disk, which slows down the processing speed of Hadoop MapReduce.
* 100 times faster than Hadoop MapReduce when Spark runs in memory and 10 times faster when Spark runs on disk	* Developers need to hand code each operation in Hadoop MapReduce.
* Spark provides high-level operators such as map, filter & so on, which makes the developer's job easy	* Hadoop MapReduce supports only batch processing
* Spark supports both batch & real-time processing	* Hadoop MapReduce does not provide an interactive shell.

3. List the areas where Spark and Hadoop MapReduce are good.

-> Tasks Hadoop MapReduce is good for:

* Linear Processing of huge datasets. Hadoop MapReduce allows parallel processing of huge amount of data. It breaks a large chunk into smaller ones to be processed separately on ~~difficult~~ different data nodes and automatically gathers the results across the multiple nodes to return a single result. In case the resulting dataset is larger than available RAM, Hadoop MapReduce may outperform Spark.

-> Economical solution, if no immediate results are expected our Hadoop team considers MapReduce a good solution if the speed of processing is not critical. For instance, if data processing can be done during night hours, it makes sense to consider using Hadoop MapReduce.

* Tasks Spark is good for.

Fast data processing. In-memory processing makes Spark faster than Hadoop MapReduce - up to 100 times for data in RAM and up to 10 times for data in storage

Iterative Processing. If the task is to process data again and again - Spark defeats Hadoop MapReduce. Spark's Resilient Distributed Datasets (RDDs) Enable multiple

map operations in memory, while Hadoop MapReduce has to write interim results to disk.

→ Near real-time processing. If a business needs immediate insights, then they should opt for Spark and its in-memory processing.

→ Graph processing. Spark's computational model is good for iterative computations that are typical in graph processing.

→ Machine learning. Spark has MLlib - a built-in machine learning library, while Hadoop needs a third-party to provide it. MLlib has out-of-the box algorithms that also run in memory

→ Joining datasets :- Due to its speed, Spark can create all combinations faster, though Hadoop may be better if joining of very large data sets that requires a lot of shuffling and sorting is needed.

4. How Spark manages the cluster and Resources.

→ Apache Spark manages clusters and resources through a combination of its own architecture and External cluster managers

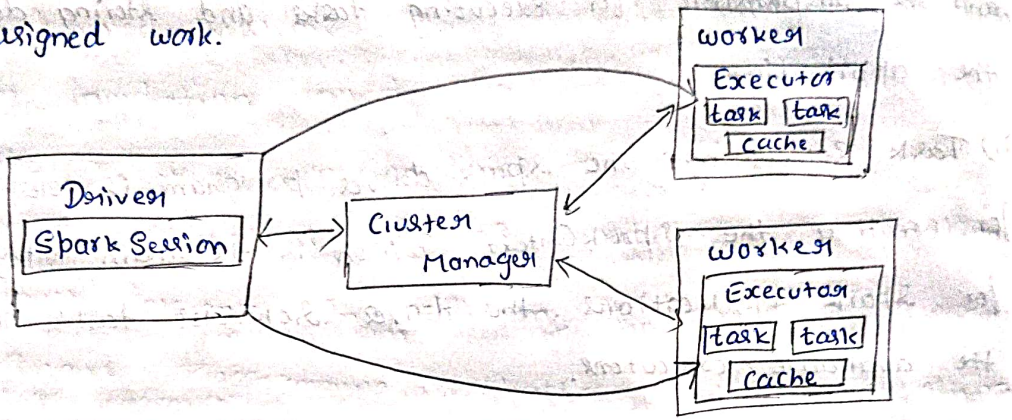
i) Cluster Managers :- Spark can work with different cluster managers, suitable for simple setups. It handles resource allocation & scheduling without External dependencies.

ii) Resource allocation:- when a Spark application is submitted, the cluster manager allocates resources in the form of Executor processes. Each Executor runs on a worker node and is responsible for executing tasks and storing data for the application.

iii) Task Scheduling :- The Spark driver program is responsible for creating the SparkContext, which is the main entry point for Spark applications. the driver schedules tasks across the available Executors.

iv) Spark clusters:- Spark is Essentially a distributed system that was designed to process a large volume of data Efficiently and quickly. this distributed system is typically deployed onto a collection of machines, which is known as a Spark cluster. A cluster size can be as small as a few machines or as large as thousands of machines. to Efficiently and intelligently manage a collection of machines, companies rely on a resource management system such as a few machines or as large as thousand such as Apache YARN or Apache Mesos. the two main components in a typical resource management system are the cluster manager and the workers. the cluster manager knows where the workers are located, how much memory they have, and the number of CPU cores Each one has. one of the main

responsibilities of the cluster manager, is to orchestrate the work by assigning it to each worker. Each worker offers resources to the clusters manager and performs the assigned work.

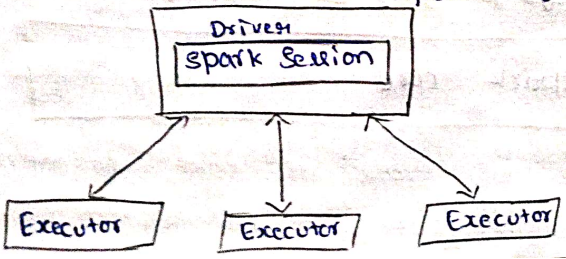


5. write short note on Spark Driver and Executors?

→ Spark Driver:- A Spark application consists of two parts. the first is the application data processing logic expressed using Spark API's, and the other is the Spark driver. the application data processing logic can be as simple as a few lines of code to perform a few data processing operations or can be as complex as training a large machine learning model that requires many iterations and could run for many hours to complete. the Spark driver is the central co-ordination of a Spark application, and it interacts with a cluster manager to figure out which machines to run the data processing logic on the entry point into a Spark application is through a class called SparkSession which provides

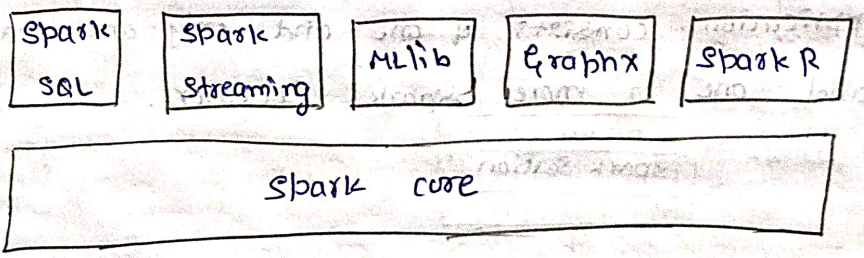
facilities for setting up configuration as well as APIs for expressing data processing logic.

Spark Executors:- Each Spark Executor is a JVM Process and is exclusively allocated to a specific Spark application. this was a conscious design decision to avoid sharing a Spark Executor between multiple Spark applications in order to isolate them from each other so one badly behaving Spark application wouldn't affect other Spark applications. the lifetime of a Spark Executor is the duration of a Spark application, which could run for a few minutes or for a few days. Since Spark applications are running in seperate Spark Executors, sharing data b/w them will require writing the data to an external storage like HDFS. Spark employs a master-slave architecture, where the Spark driver is the master and the Spark Executor is the slave. Each of these components runs as an independent process on a Spark cluster. A Spark application consists of one and only one Spark driver and one or more Spark Executors.



6. Write the Spark Verified Stack and Explain Spark Core

Spark Unified Stack:- Spark provides a unified data processing Engine known as the Spark Stack, similar to other well-designed systems, their stack is built on top of a strong foundation called Spark core, which provides all the necessary functionalities to manage and run distributed applications such as scheduling, co-ordination, and fault tolerance. In addition, it provides a powerful and generic programming abstraction of data processing called resilient distributed datasets (RDDs), on top of this strong foundation is a collection of components where Each one is designed for a specific data Processing workload Spark SQL is for-batch data Processing. Spark Streaming is for real-time stream data Processing. Spark Graph x is for graph processing. Spark MLlib for machine learning. Spark R is for-function machine learning tool is using R shell



Spark core:- Spark core provides all the necessary functionalities to manage and run distributed data processing Engine. It consists of two parts: the distributed computing infrastructure is responsible for the distribution, co-ordination and scheduling of computing tasks across many machines in the cluster. this Enables the ability to perform parallel data processing of a large volume of data Efficiently and quickly on a large cluster of machines, two important responsibilites of the distributed computing infrastructure are handling of computing task failures of Efficiently moving performing large-scale data processing without worrying where data resides on the cluster or dealing with machine failures. the RDD API's are exposed in multiple Programming languages and they allow uses to pass local functions to run to the clusters which is something that is powerful.

* what are RDDs ? Explain the properties of RDD

→ RDDs represent both the idea of how a large dataset is represented in Spark and the abstraction for working with it. RDD's are immutable, fault-tolerant, parallel data structures that let users Explicitly persist intermediate results in memory, control their

Partitioning to optimize data placement, and manipulate them using a rich set of operators.

Immutable :- RDDs are designed to be Immutable, which means you can't specifically modify a particular row in the dataset represented by that RDD. you can call one of the available RDD operations to manipulate the rows in the RDD into the way you want, but that operation will return a new RDD. the basic RDD will stay unchanged, and the new RDD will contain the data in the way that you want. the immutability of RDDs Essentially requires an RDD to carry its lineage information that Spark leverages to Efficiently provide the fault tolerance capability.

Fault Tolerant :- The ability to process multiple datasets in parallel usually requires a cluster of machines to host and Execute the computational logic. If one or more of those machines die or becomes Extremely slow because of unexpected circumstances, then how will that affect the overall data processing of those datasets? the good news is that Spark automatically takes care of handling the failure on behalf of its users by rebuilding the failed portion using the lineage information.

Parallel Data Structures :- Imagine the use case where someone gives you a large log file that is 1TB size & you are asked to find out how many log statements contain the word Exception in it. A slow solution would be to iterate through that log file from the beginning to the end & execute the logic of determining wheather a particular log statement contains the word Exception. A faster solution would be to divide that 1TB file into several chunks and execute the aforementioned logic on each chunk in a parallelized manner to speed up the overall processing time. Each chunk contains a collection of rows, the collection of rows is Essentially the data structure that holds a set of rows and provides the ability to iterate through each row. Each chunk contains a collection of rows, & as the chunks are being processed in parallel - this is where the phrase parallel data structures comes from.

In-memory computation :- RDD pushes the speed boundary by introducing a novel idea, which is the ability to do distributed in-memory computation. Machine learning algorithms are iterative in nature, meaning they need to go through many iterations to arrive at an optimal state. this is where distributed in-memory computation

can help in reducing the completion time from days to hours. Another use case that can hugely benefit from distributed in-memory computation is interactive data mining, where multiple adhoc queries are performed on the same subset of data. If that subset of data is persisted in memory, those queries will take seconds & not minutes to complete.

Rich Set of Operations :- RDD's provide a rich set of commonly needed data processing operations. they include the ability to perform data transform, filtering, grouping, joining, aggregation, sorting and counting. one thing to note about their operations is that they operate at the coarse-grained level, meaning the same operation is applied to many rows, not to any specific row. In summary, an RDD is represented as an abstraction & is defined by the following.

- A set of partitions, which are the chunks that make up the entire dataset
- A set of dependencies on parent RDDs
- A function for computing all the rows in the dataset
- Metadata about that partitioning scheme (optional),

⑧ what are transform and Action in Spark ? Give suitable Examples.

→ The RDD operations are classified into two types : Transformations and actions.

Transforms & Actions :- In Spark, the core data structure are immutable, meaning they cannot be changed after they're created. this might seem like a strange concept at first : If you cannot change it, how are you supposed to use it? To "change" a DataFrame, you need to instruct Spark how you would like to modify it to do what you want. these instructions are called transformations.

Transformation operations are lazily evaluated, meaning Spark will delay the evaluations of the invoked operations until an action is taken. In other words, the transforma-tion operations merely record the specified transformation logic and will apply them at a later point.

Examples :-

i) map() :- Applies a function to each element in the RDD and returns a new RDD

```
rdd = spark.sparkContext.parallelize([1,2,3,4])
mapped_rdd = rdd.map(lambda x: x * 2)
```

ii) filter() :- Returns a new RDD containing only the elements that satisfy a specified condition.

```
rdd = spark.sparkContext.parallelize([1,2,3,4])
filtered_rdd = rdd.filter(lambda x: x%2 == 0)
```

iii) flatMap() :- Similar to map(), but each input element can produce zero or more output elements

```
rdd = spark.sparkContext.parallelize(["hello world", "hello spark"])
flat_mapped_rdd = rdd.flatMap(lambda x: x.split(" "))
```

Actions :-

i) collect() :- Returns all the Elements of the RDD to the driver program

```
rdd = spark.sparkContext.parallelize([1,2,3,4])
result = rdd.collect()
```

ii) count() :- Returns the number of Elements in the RDD

```
rdd = spark.sparkContext.parallelize([1,2,3,4])
count = rdd.count()
```

9. What is Lazy Evaluation in Spark? Explain with an Example

→ Lazy Evaluation means that Spark will until the very last moment to Execute the graph of computation instructions. In Spark, instead of modifying the data immediately when you Express some operation, you build up an plan of transformations that you would like to apply to your source data. By waiting until the last minute to Execute the code, Spark compiles this plan from your raw DataFrame transformations to a streamlined physical plan that will run as Efficiently as possible across the clusters. This provides immense benefits because Spark can optimize the Entire data flow from End to End. An Example of this is something called predicate pushdown on DataFrames, If we build a large Spark job but Specify a filter at the End that only requires us to fetch one row from our source data,

Creating RDD's - There are three ways to create an RDD

One first way to create an RDD is to parallelize an object collection, meaning converting it to a distributed dataset that can be operated in parallel.

```
firstRdd = spark.sparkContext.parallelize (['Hello world',
welcome to the world of Spark programming'], 2)
```

The second way to create an RDD is to read a dataset from a storage system, which can be a local computer file system, HDFS, Cassandra, Amazon S3, and so on.

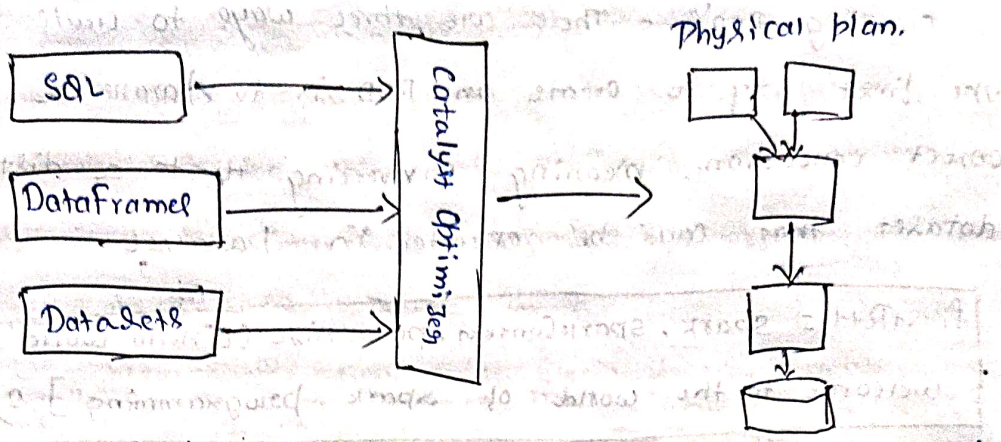
```
secondRdd = spark.SparkContext.textFile ('file')
```

The third way to create an RDD is by invoking one of the transformation operations on an existing RDD.

```
thirdRDD = firstRdd.flatMap (lambda word: word.split(' '))
```

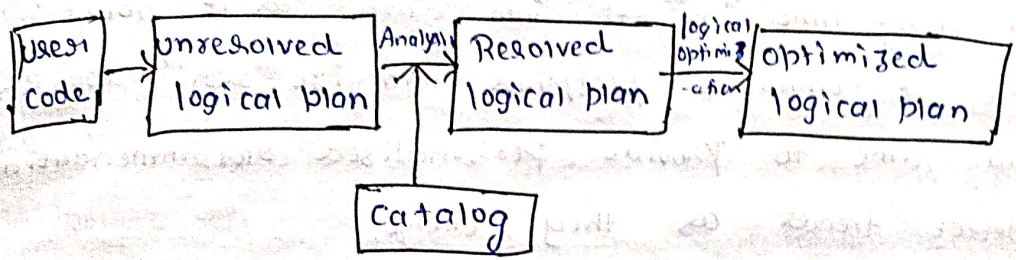
10) with logical and physical plans Explain how Structured APIs will Executed in Spark.

→ Overview of Structured API Execution :- To Execute code, we must write code. This code is then submitted to Spark either through the console or via a submitted job. this code then passed through the Catalyst Optimizer, which decides how the code should be Executed and lays out a plan for doing so before, finally, the code is run & the result is returned to the user.



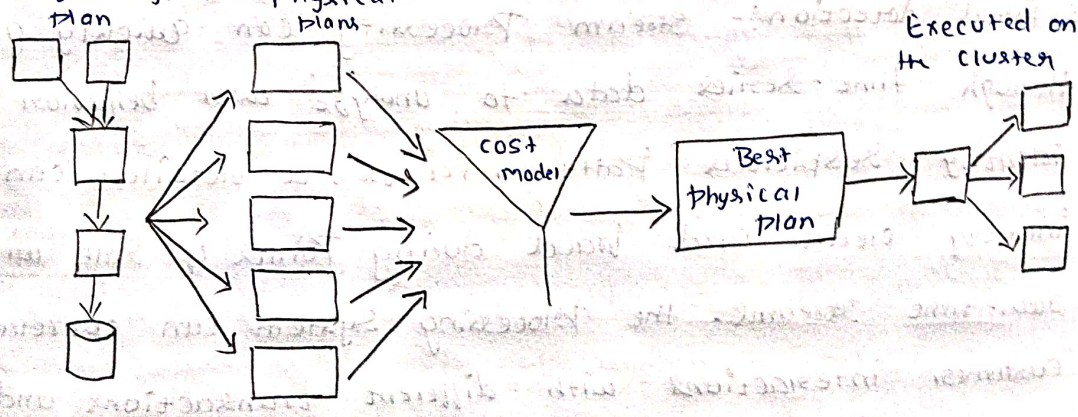
Logical planning:- The first phase of execution is meant to take user code & convert it into a logical plan. this logical plan only represents a set of abstract transformations that do not refer to executors or drivers, it's purely to convert the user's set of expressions into the most optimized version. It does this by converting user code into an unresolved logical plan, this plan is unresolved bcoz although your code might be valid, the tables or columns that it refers to might or might not exist. Spark uses the catalog, a repository of all table & Dataframe information, to resolve columns & tables in the analyzer. the analyzer might reject the unresolved logical plan if the required table or column name does not exist in the catalog. If the analyzer can resolve it, the result is passed through the catalyst Optimizer, a collection of rules that attempt to optimize the logical

plan by pushing down predicates or selections.



Physical Planning

After successfully creating an optimized logical plan, Spark then begins the physical planning process. The physical plan, often called a Spark plan, specifies how the logical plan will execute on the cluster by generating different physical execution strategies and comparing them through a cost model.



(11) Explain some use cases of Stream processing

→ Stream processing is used in various industries to handle and analyze real-time data. Some key use cases include:

1. Real-time Analytics:- Businesses use stream processing to gain real-time insights into data as it flows in for Ex:- e-commerce platforms analyize user interactions in real time to provide personalized recommendations or detect trends as they Emerge.

2. Log analysis:- It is a process that Engineering & IT teams use to identify bugs by reviewing computer-generated record called logs. Stream processing improves log analysis by collecting raw system logs structuring them, and converting them into a standardized format at lightning speed. Thus it is Easier for teams to detect bugs & fix them faster.

3. Fraud detection:- Stream processing can quickly go through time-series data to analyze user behavior & identify suspicious patterns. For Ex, a retailer can identify credit card fraud during points of sale with real-time streams. the processing system can correlate customer interactions with different transactions and channels, and review them instantly if the system discovers an unusual or inconsistent transaction.

4. IoT and Sensors data:- Sensors-powered IoT devices send and collect large amounts of data quickly, which is valuable to organizations. For instance, they can measure various data, such as temp, humidity, air, quality, air Pressure etc.

Once the data is collected, it can be transmitted to Servers for processing. With millions of records generated every second, you might also need to perform actions on the data, such as filtering, discarding irrelevant data, etc.

12. List out the challenges of Stream proce

→ Fault tolerance and Recovery:- Handling failures in a system stream processing system is difficult, Especially when the system needs to Ensure no data is lost and Processing is done Exactly once or at least once.

ii) Latency Management:- while stream processing aims for low latency response, achieving, consistently ~~when the system needs to~~ ~~Ensure~~ low latency can be difficult, Especially under high data loads.

iii) Data Consistency:- Ensuring data consistency in real-time system streams can be complex, Especially when working with distributed systems or Events that may arrive out of order. Stream Processing systems need mechanism to reorder events & Ensure consistency across the Entire stream.

iv) Scaling & load Management:- System processing systems need to handle potentially large and unpredictable volumes of incoming data. Scaling these systems to handle variable loads without degrading Performance requires careful planning & design

13. Explain Continuous Stream Processing method. what are its advantages and disadvantages?

→ Continuous Stream processing is a method where data is processed as soon as it arrives in real time, without waiting for the entire dataset to be collected.

Advantages:- i) low latency:- continuous stream processing provides instant or near instant results, this is critical in applications like fraud detection, stock trading or IoT device monitoring. where time-sensitive responses are required

ii) Scalability:- Stream processing systems can be highly scheduled, handling large volumes of data that would be impratical to batch-process. Systems like Apach kalka, Apache Flink

iii) Real-time Analytics:- continuous stream processing enables business to monitor key matrics & trends in real time, allowing for immediate decision-making.

Disadvantages:- i) complexity:- Developing, deploying and maintaining continuous stream processing systems is more complex compared to traditional batch processing.

ii) Data Quality & consistency:- Ensuring data accuracy and consistency in real-time can be challenging, particularly if Events arrive at of order or there are failures during processing.

iii) Limited Historical Analysis:- Stream processing is focused on real time or near-~~real time~~ term data. It's not ideal for complex historical analysis, which might require aggregating longer datasets. over longer time periods.

14. Explain Micro-Batch Stream Processing method. what are its advantages and disadvantages

→ Micro Batch stream processing:- It is a method that processes data in small, manageable batches rather than as

individual events. this approach sits b/w traditional batch processing and real-time stream processing.

Advantages:-

i) Lower Latency:- It reduces the time it takes to process data in small intervals. System can deliver results more quickly, making it suitable for near real-time applications.

ii) Simplified Architecture:- Micro batching can simplify system design. it allows for a unified approach to handling both batch & stream data reducing the complexity.

iii) Fault-tolerance:- Batch processing can offer better fault tolerance, as entire batches can be reprocessed if errors occur, rather than just individual events.

iv) Scalability:- Micro batch systems can be easier to scale compared to traditional real-time processing systems, as batch sizes can be adjusted according to system load.

Disadvantages:

i) complexity in Management:- Managing micro-batch jobs can be more complex as debugging & managing failures in a micro-batch system as it leads to errors & potential data loss on duplication

ii) Potential Data Loss:- In cases of failure before batch completion, there can be a risk of losing some data unless mechanisms are in place to Ensure data durability.

iii) Data consistency challenges:- If data is processed out of order due to the batching mechanism. it may lead to consistencies in the output. Also managing state across micro batches.

15. write a short note on DStreaming and Structured Streaming

-> Dstream (Discretized Stream) :- And Structured Streaming models provided by Apache Spark for real-time data processing

- Dstream is the original Streaming API in Spark Streaming. It divides the incoming data stream into small, discreate batches and process them periodically.
- Each Dstream is a sequence of Resilient Distributed Datasets (RDD) where each RDD contains data from a particular batch interval
- Fault tolerance is achieved through Spark's lineage-based approach, allowing recovery from failures by recomputing lost RDD's
- Limitations - Dstream is less user-friendly for complex analytics because it lacks a clear and consistent API for structured data, and user must manually handle event-time processing watermarks, and aggregation.

Structured Streaming :-

- Structured Streaming is a newer and more advanced API built on Spark SQL. It treats streaming data as a continuous unbound table that can be quired using SQL-like operations.
- It provides a declerative API, allowing user to write queries on Streaming data in the same way they would write SQL quiries & batch data
- It offers Exactly-once processing guarantees & better integration with the Spark Ecosystem (Data Frames & datasets).

16. write short notes on Data Sources and Data sinks of Structured Streaming in Spark.

→ Data Sources :- Spark SQL supports operating on a variety of data sources through the DataFrame interface. A DataFrame can be operated on using relational transformations and can also be used to create a temporary view. data source include kafka, Flume, Amazon Kinesis, files

→ Files are supported by reading & writing data streams to and from files in the same formats as the once supported in batch processing: plain text, csv, JSONet All files must be of the same format & same schema.

Apache kafka:- Structured Streaming has built in support for reading from & writing to Apache kafka which is a popular publish / subscribe system that is widely used for storage.

Socket :- Reads UTF8 text data from a socket connection the listening server socket is at the driver. this can be used only for testing as this does not provide end to end fault-tolerance gaurentees

Rate Source:- Generates data at the specified number of rows per second, each output row contains a timestamp & value. this source is intended for testing & benchmarking.

Data Sinks:- Spark provides a number of built in sink implementations for different output storage systems, including the ones for being used in the production Environment.

For Each Batch Sink:- It allows end user developers to specify a custom write function that is executed to write the output data of every micro-batch of a streaming query to the target storage system.

File System Sink:- It writes the results of a streaming query to parquet files in a target hadoop compatible storage system.

Delta Sink:- It is not built in sink but it is offered by delta lake package which supports to write the results of streaming query into Delta table

Kafka Sink:- It refers to mechanism that allows you to write data from a spark streaming application back to kafka topic which is useful for scenario where you want to process & transform data in real time.