**Checkpoint:**

In Spark streaming, we have streaming data coming 24/7 into the system, we check the data from a period of time and process these as events like some kind of computation or aggregations on top of these events. Now, if our application fails due to some error, then to recover we conceptually need to re-process all the events that are already processed in the time which is a waste of resources, time-consuming and it is unnecessary.

To overcome this, we use the concept of Checkpoint. Checkpoint is a mechanism where every so often Spark streaming application stores data and metadata in a fault-tolerant file system, so that it would be easy to restart from the failure point.

1. Data Checkpoint:

Data Checkpoint stores the generated RDD in a file system. This is used when any computation or aggregations on the current micro-batch are dependent on the prior micro-batches.

1. Metadata Checkpoint:

Metadata Checkpoint stores the information related to the streaming computation, configuration, and queues in fault-tolerant storage. It is used to recover from failures unrelated to the application logic. Metadata mainly store below:

* Incomplete Batches:

Spark Streaming application chucks the incoming events and groups for a period of time and processes them as batches. This stores information about the batches that are not processed/ Queued.

* Configuration:

It stores the configuration that was set up for the Spark streaming application like maxFilesPerTrigger, Processing time interval, e.t.c…

* DStream Operations:

It stores all information about the operations that are performed on the continuous streaming data. Spark streaming application uses this information to recover from failures and re-start from the failure position instead of starting from the beginning.

Metadata checkpointing is primarily needed for recovery from driver failures, whereas data checkpointing is necessary even for basic functioning if stateful transformations are used. To set the Spark checkpoint directory, we can pass the checkpoint location as an option to writeStream of a streaming dataframe.

========================================== Spark Streaming provides two types of stream processing: stateful and stateless ======================================

Stateless Streaming:

Stateless streaming treats each batch of data independently and processes it without any reference to previous batches.

This means that the output of a batch is only based on the data in that batch.

Examples of stateless operations include filtering, mapping, and aggregations that do not require state.

Stateless streaming is well-suited for operations that do not require any memory of previous batches.

Stateless operations are typically faster and consume less memory than stateful operations.

Stateless streaming is also easy to scale horizontally since there is no need to keep track of state across multiple nodes.

Stateful Streaming:

Stateful streaming maintains state across multiple batches of data.

This means that the output of a batch depends not only on the data in that batch, but also on the state of the system accumulated from previous batches.

Examples of stateful operations include windowed operations, which compute aggregates over a sliding window of data, and updating state based on new input data.

Stateful streaming is well-suited for operations that require a memory of previous batches.

Stateful operations are typically slower and consume more memory than stateless operations.

Stateful streaming is also harder to scale horizontally since there is a need to keep track of state across multiple nodes.

Choosing the Right Processing Mode:

When choosing between stateful and stateless processing modes in Spark Streaming, you should consider the following factors:

Memory Requirements:

Stateful operations require more memory than stateless operations since they need to maintain state across batches.

If you have limited memory resources, stateless processing may be the better option.

Processing Speed:

Stateless operations are typically faster than stateful operations since they do not require any memory of previous batches.

If you need real-time processing, stateless processing may be the better option.

Data Complexity:

Stateful operations are better suited for applications that require more complex processing, such as windowed computations or stateful aggregations.

Stateless operations are better suited for simpler operations that do not require state.

Scalability:

Stateless operations are easier to scale horizontally since there is no need to keep track of state across multiple nodes.

Stateful operations require more complex coordination across nodes, which can make scaling more challenging.

Fault Tolerance:

Stateful operations are more prone to failures since they require maintaining state across multiple batches.

Spark Streaming provides fault-tolerance mechanisms to recover from node failures, but the recovery process can be slower and more complex for stateful operations.

==================================================== Watermarking & windowing ==================================================

Refer 4. Watermarking.docx

==================================================== Tumbling vs Sliding windows ================================================

In tumbling windows, the time windows are non-overlapping.

Useful in scenarios like what is the max trade\_value in the time period 10:45 to 11:00.

In sliding windows, the time windows are overlapping.

Useful in scenarios like what is the max trade\_value in the last 15 minutes.