Streaming with Spark

The streaming data model of Apache Spark deals with multiple streams of varying types, and can handle these stream processing tasks simultaneously. The streams come at different rates and are not synchronized.

For this, Spark uses a Stream processor to classify streams into the archival storage and limited working storage, based on the stream. Queries sent to this streamed data can be Standing queries, that continuously monitor the stream, or ad-hoc queries, that are issued asynchronously at any point in time.

Issues with stream processing include

1. Velocity
    a. Streams can have high data rates
    b. Need to be processed very fast
2. Volume
    a. Large number of streams
3. Need to store in memory
    a. Memory is limited
    b. Can lead to approximations

The main requirements for a framework to mitigate this are

1. Scalability to large clusters
2. Second-scale latencies
3. Simple programming model
4. Integrated with batch and interactive processing
5. Efficient fault tolerance in stateful computations

Stateful stream processing leverages nodes having mutable states and an event-driven record-at-a-time model. A state assigned can be lost if the node dies, making stateful stream processing challenging.

Spark streaming helps mitigate these problems. It uses discretized streaming, where it runs a streaming computation on a series of very small deterministic batch jobs.

Spark treats each batch of data as RDDs and processes them using RDD operations. Finally, the processed results are returned in batches.

In this batch sizes are really small and can be adjusted, to provide both batch and stream processing in the same framework.

A DStream is a sequence of RDDs representing a stream of data. DStreams are stored in memory as RDDs, which are immutable and distributed.

Every DStream is associated with a receiver. A receiver reads data from a source and stores it into spark memory for processing. They can be basic (Filesystems or sockets) or advanced (Kafka, Flume).

Transformations in Spark can be stateless or stateful.

Stateless transformations are applied to every batch of data independently, and no information is carried over to the next batch. Examples are

1. Map
2. FlatMap
3. Filter
4. Repartition
5. ReduceByKey
6. GroupByKey

Stateful transformation maintains state information across windows. For this, we use pair RDDs where the key stores the state of the RDD. Examples of this are

1. updateStateByKey
2. reduceByKeyAndWindow
3. Window

Spark provides two options for stateful processing

1. Window operator : short period maintenance of state
2. Session based : Maintain state for longer

Window based countByValue() is used to make counting in streaming more efficient. In this, the previous window's first Dstream is neglected and the new Dstream is added. This is an inverse reduce, and reduceByKeyAndWindow can do the same.

All window based operations need two parameters,

1. Window duration - How many previous batches are considered
2. Sliding duration - How frequently the new DStream computes results

Both must be a multiple of the batch interval.

UpdateStateByKey is used to maintain arbitrary state per window.

Fault Tolerance

1. Stateless
    a. Previous history is not required and processing can just be recomputed
2. Stateful
    a. Retain a finite time interval for the state

Some output transformation are

1. Print
2. foreachRDD
3. saveAs

RDDs remember the sequence of operations that got them to that state, so can just be recomputed using the logs.

Checkpointing is done for more efficient storage and a checkpointed RDD forgets the lineage and stores the consistent state at that time step.

The limitation of spark streaming is that it is near real time stream processing and not completely realtime.