Inspire…Educate…Transform.

Spark Streaming

Ref: Spark Streaming Reference

Spark Streaming

- Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.
- Data can be ingested from many sources like Kafka, Flume, Kinesis, or TCP sockets, and can be processed using complex algorithms expressed with high-level functions like map reduce, joi

Contd..

- Finally, processed data can be pushed out to file systems, databases, and live dashboards. In fact, you can apply Spark's machine learning and graph processing algorithms on data streams.

Contd..

- Spark Streaming receives live input data streams and divides the data into batches, which are then processed by the Spark engine to generate the final stream of results in batches.
-

Contd..

- Spark Streaming provides a high-level abstraction called *discretized stream* or *DStream*, which represents a continuous stream of data.
- DStreams can be created either from input data streams from sources such as Kafka, Flume, and Kinesis, or by applying high-level operations on other DStreams. Internally, a DStream is represented as a sequence of RDDs.

Initialize Streaming Context

- A **StreamingContext** object has to be created which is the main entry point of all Spark Streaming functionality.
-

```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
sc = SparkContext(master, appName)
ssc = StreamingContext(sc, 1)
```

Batch Interval

After defining Streaming Context

- Define the input sources by creating input DStreams.
- Define the streaming computations by applying transformation and output operations to DStreams.
- Start receiving data and processing it using streamingContext.start().
- Wait for the processing to be stopped (manually or due to any error) using streamingContext.awaitTermination().
- The processing can be manually stopped using streamingContext.stop().

DStream

- DStream is represented by a continuous series of RDDs, which is Spark's abstraction of an immutable, distributed database.

- Any operation applied on a DStream translates to operations on the underlying RDDs.It applies on every RDD that is splitted by time.

Input DStreams and Receivers

- Every input DStream is associated with a **Receiver** (Scala doc, Java doc) object which receives the data from a source and stores it in Spark's memory for processing.
- Spark Streaming provides two categories of built-in streaming sources.
  - *Basic sources*: Sources directly available in the StreamingContext API. Examples: file systems, and socket connections.
  - *Advanced sources*: Sources like Kafka, Flume, etc. are available through extra utility classes. These require linking against extra dependencies as discussed in the linking section.

Transformations

- Return a new DStream after the transformation.

Types:

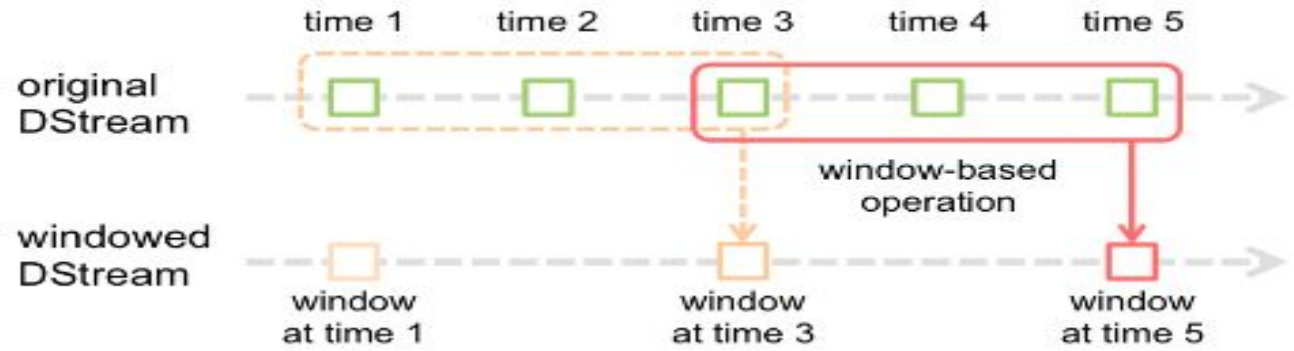
Stateless Transformation:

- Do not consider the previous state of the batch
- Applied on to each RDD that Dstreams is composed of.
- Ex: map,flatmap,reduceByKey,

Stateful Transformation:

- Results of the previous batch are used to produce current batch.
- Ex : upDateStateByKey

Window Operations



The source RDDs that fall within the window are combined and operated upon to produce the RDDs of the windowed DStream.

This shows that any window operation needs to specify two parameters.

- *window length* - The duration of the window
- *sliding interval* - The interval at which the window operation is performed

Output Operations

- Operations that help in pushing the data on to a external System.
- This will trigger the actual execution of the RDD

  Ex: pprint,saveAsTextFile etc

Cache & Persist

- DStream will automatically persist every RDD of that DStream in memory.
- This is useful if the data in the DStream will be computed multiple times (e.g., multiple operations on the same data).
- For window-based operations like `reduceByWindow` and `reduceByKeyAndWindow` and state-based operations like `updateStateByKey`, this is implicitly true. Hence, DStreams generated by window-based operations are automatically persisted in memory, without the developer calling `persist()`.

Check Pointing

- Streaming application must operate 24/7 and hence must be resilient to failures.
  - *Metadata checkpointing*
  - *Data checkpointing*