Spark Streaming

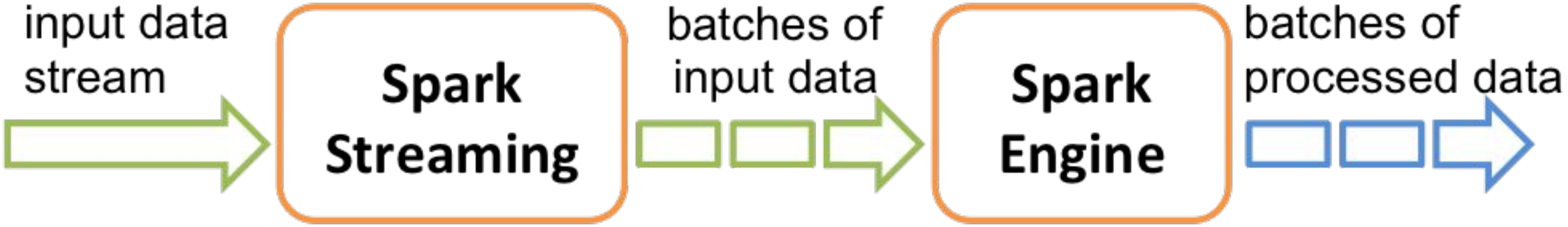
cerenode

cerenode.io

Overview

- Used for processing real-time streaming data
- Enables high-throughput and fault-tolerant stream processing of live data streams
- Fundamental unit is DStreams, which is basically a series of RDDs to process the real-time data

Steps


input data stream
Spark Streaming
batches of input data
Spark Engine
batches of processed data

Streaming Context

- Consumes a stream of data in Spark
- Registers an InputDStream to produce a Receiver object
- Main entry for Spark Streaming functionality
- Provides number of default implementations of sources like Twitter, Akka Actor and ZeroMQ that are accessible from the context

Streaming Context Initialization

- Can be created from a SparkContext object

```
import org.apache.spark._
import org.apache.spark.streaming._
new StreamingContext(sc, Seconds(1))
```

Streaming Context Initialization

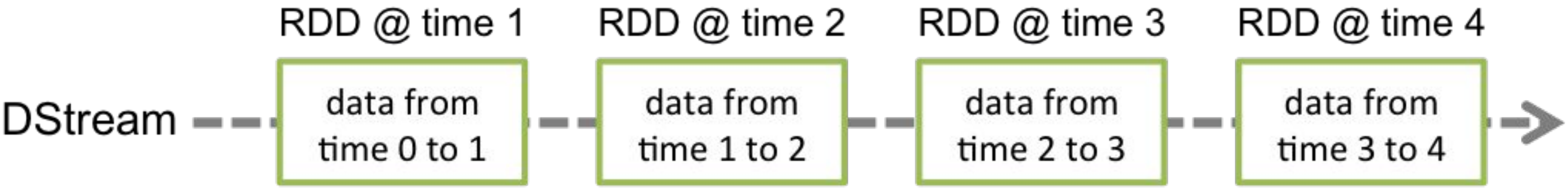
- Can be created from a SparkConf object

```
   import org.apache.spark._
   import org.apache.spark.streaming._
   val conf = new
SparkConf().setAppName(appName).setMaster(master)
   val ssc = new StreamingContext(conf, Seconds(1))
```

DStream

- Discretized Stream (DStream) is the basic abstraction provided by Spark Streaming
- It is a continuous stream of data
- It is received from source or a processed data stream generated by transforming the input stream
- Internally, a DStream is represented by a continuous series of RDDs and each RDD contains data from a certain interval

DStream


RDD @ time 1
RDD @ time 2
RDD @ time 3
RDD @ time 4
DStream
data from time 0 to 1
data from time 1 to 2
data from time 2 to 3
data from time 3 to 4

Input DStreams and Receivers

- Input DStreams represents stream of input data received from streaming sources
- Two categories sources
  - Basic: Sources directly available in the StreamingContext
  - Advanced: Sources like Kafka, Flume, Kinesis, etc.
- Every input DStream (except file stream) is associated with a Receiver object which receives the data from a source and stores it in Spark's memory for processing.

Basic Sources

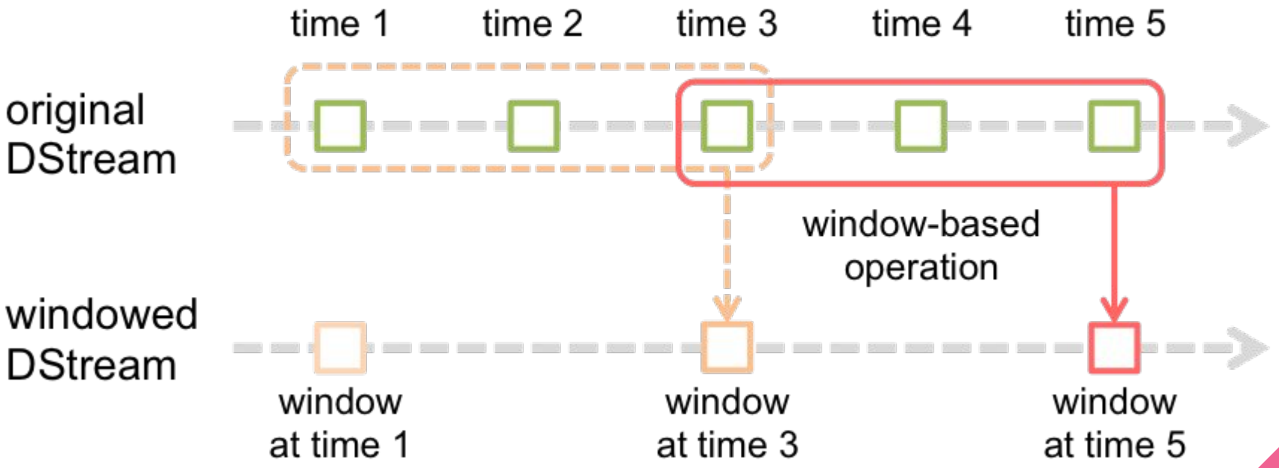
- SocketTextStream
  - `StreamingContext.socketTextStream("<hostname>", <port>)`
- File Streams
  - `streamingContext.fileStream[KeyClass, ValueClass, InputFormatClass](dataDirectory)`
  - `streamingContext.textFileStream(dataDirectory)`

Transformations on DStream

- Allows the data from input DStream to be modified similar to RDDs
- Supports many of the normal transformations available for normal RDDs
- Common transformations
  - `map, flatMap, filter, reduce, groupBy, foreachRDD, updateStateByKey`

DStream Window

- Spark Streaming provides windowed computations which allow us to apply transformations over a sliding window of data



DStream Window

Some of the common window operations are as follows..

- window(*windowLength*, *slideInterval*)
- countByWindow(*windowLength*, *slideInterval*)
- reduceByWindow(*func*, *windowLength*, *slideInterval*)
- reduceByKeyAndWindow(*func*, *windowLength*, *slideInterval*, [*numTasks*])

Output Operations on DStream

- Output operations allow DStream’s data to be pushed out to external systems like databases or file systems
- Output operations trigger the actual execution of all the DStream transformations
- Common output operations are
    - `print()`
    - `saveAsTextFiles(prefix, [suffix])`
    - `saveAsObjectFiles(prefix, [suffix])`
    - `foreachRDD(func)`

updateStateByKey

The updateStateByKey operation allows you to maintain arbitrary state while continuously updating it with new information. To use this, you will have to do two steps.

1. Define the state - The state can be an arbitrary data type.
2. Define the state update function - Specify with a function how to update the state using the previous state and the new values from an input stream.

Checkpointing

Spark Streaming needs to *checkpoint* enough information to a fault- tolerant storage system such that it can recover from failures.

There are two types of data that are checkpointed.

- *Metadata checkpointing* - Saving of the information defining the streaming computation to fault-tolerant storage like HDFS. This is used to recover from failure of the node running the driver of the streaming application (discussed in detail later). Metadata includes:
  - *Configuration* - The configuration that was used to create the streaming application.
  - *DStream operations* - The set of DStream operations that define the streaming application.
  - *Incomplete batches* - Batches whose jobs are queued but have not completed yet.
- *Data checkpointing* - Saving of the generated RDDs to reliable storage.