Introduction to Spark Streaming

Real time processing on Apache Spark

Agenda

- Real time analytics in Big data
- Unification
- Spark streaming
- DStream
- DStream and RDD
- Stream processing
- DStream transformation
- Hands on

3 V’s of Big data

- **Volume**
  - TB’s and PB’s of files
  - Driving need for batch processing systems
- **Velocity**
  - TB’s of stream data
  - Driving need for stream processing systems
- **Variety**
  - Structured, semi structured and unstructured
  - Driving need for sql, graph processing systems

Velocity

- Speed at which
  - Collect the data
  - Process to get insights
- More and more big data analytics becoming real time
- Primary drivers
  - Social media
  - IoT
  - Mobile applications

Use cases

- Twitter needs to crunch few billion tweets/s to publish trending topics
- Credit card companies needs to crunch millions of transactions/s for identifying fraud
- Mobile applications like whatsapp needs to constantly crunch logs for service availability and performance

Real Time analytics

- Ability to collect and process TB’s of streaming data to get insights
- Data will be consumed from one or more streams
- Need for combining historical data with real time data
- Ability to stream data for downstream application

Stream processing using M/R

- Map/Reduce is inherently batch processing system which is not suitable for streaming
- Need for data source as disk put latencies in the processing
- Stream needs multiple transformation which cannot be expressed effectively on M/R
- Overhead in launch of a new M/R job is too high

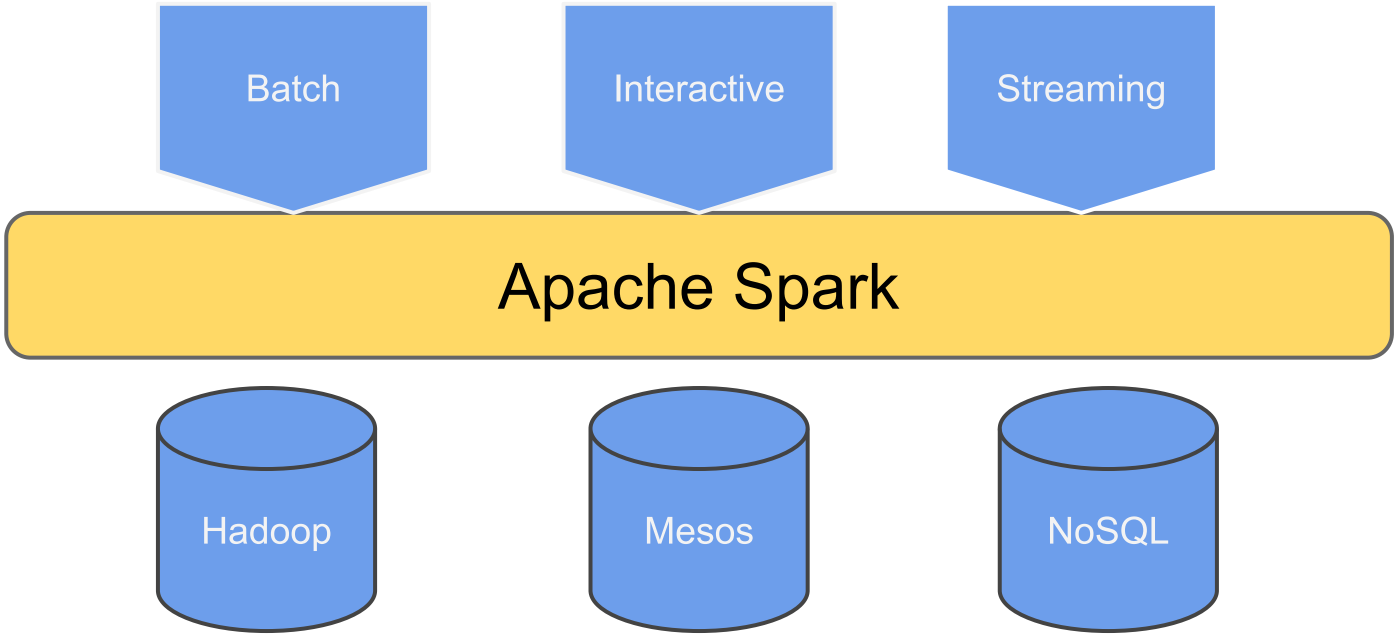
Apache Storm

- Apache storm is a stream processing system build on top of HDFS
- Apache storm has it’s on API’s and do not use Map/Reduce
- It’s a one message at time in core and micro batch is built on top of it(trident)
- Built by twitter

Limitations of Streaming on Hadoop

- M/R is not suitable for streaming
- Apache storm needs learning new API’s and new paradigm
- No way to combine batch result from M/R with Apache storm streams
- Maintaining two runtimes are always hard

Unified Platform for Big Data Apps



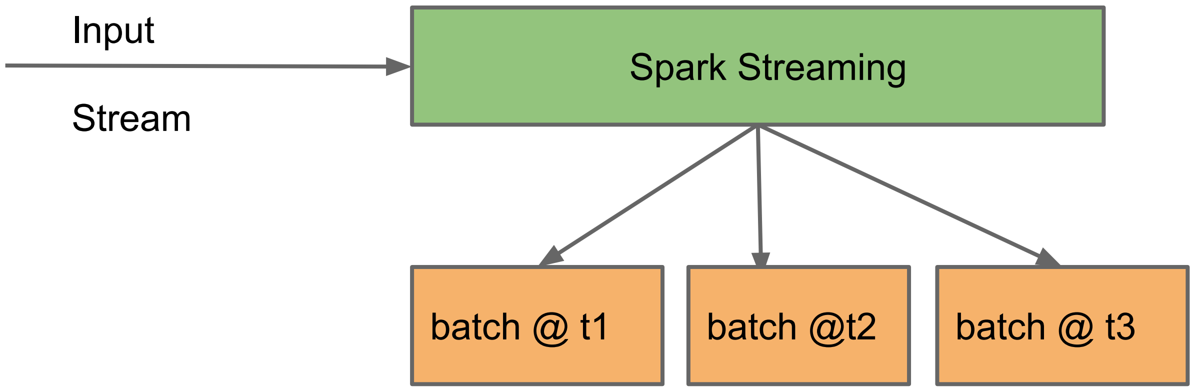
Spark streaming

Spark Streaming is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams

Micro batch

- Spark streaming is a fast batch processing system
- Spark streaming collects stream data into small batch and runs batch processing on it
- Batch can be as small as 1s to as big as multiple hours
- Spark job creation and execution overhead is so low it can do all that under a sec
- These batches are called as DStreams

Discretized streams (DStream)

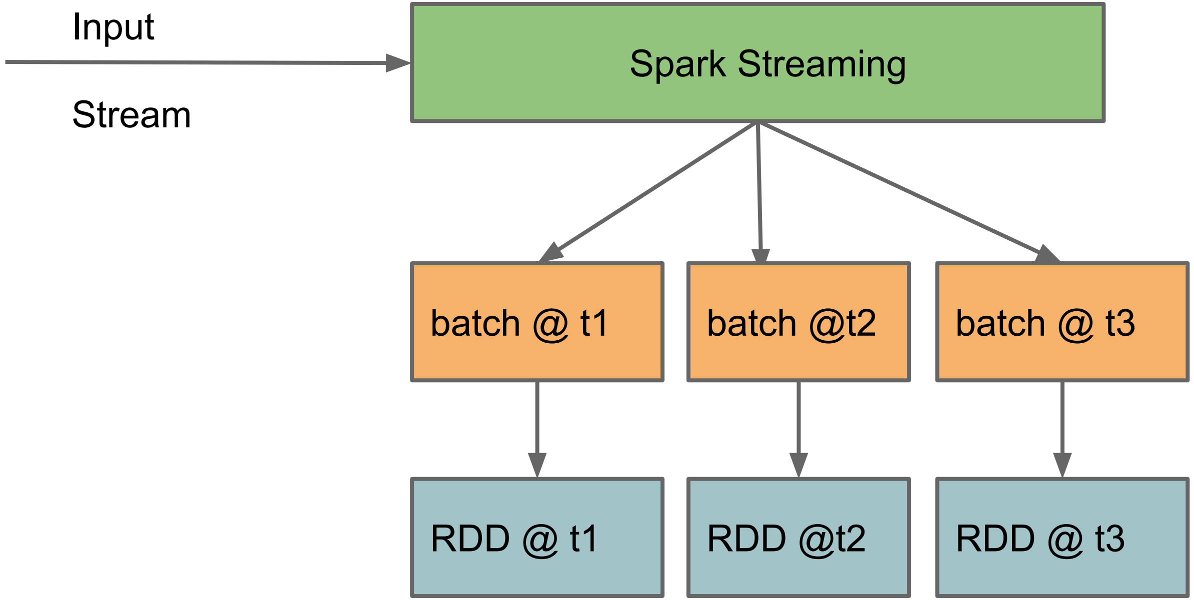


Input stream is divided into multiple discrete batches. Batch is configurable.

DStream

- Discretized streams
- Each batch of data is converted to small discrete batches
- Batch size can be from 1s - multiple mins
- DStream can be constructed from
    - Sockets
    - Kafka
    - HDFS
    - Custom receivers

DStream to RDD



Dstream to RDD

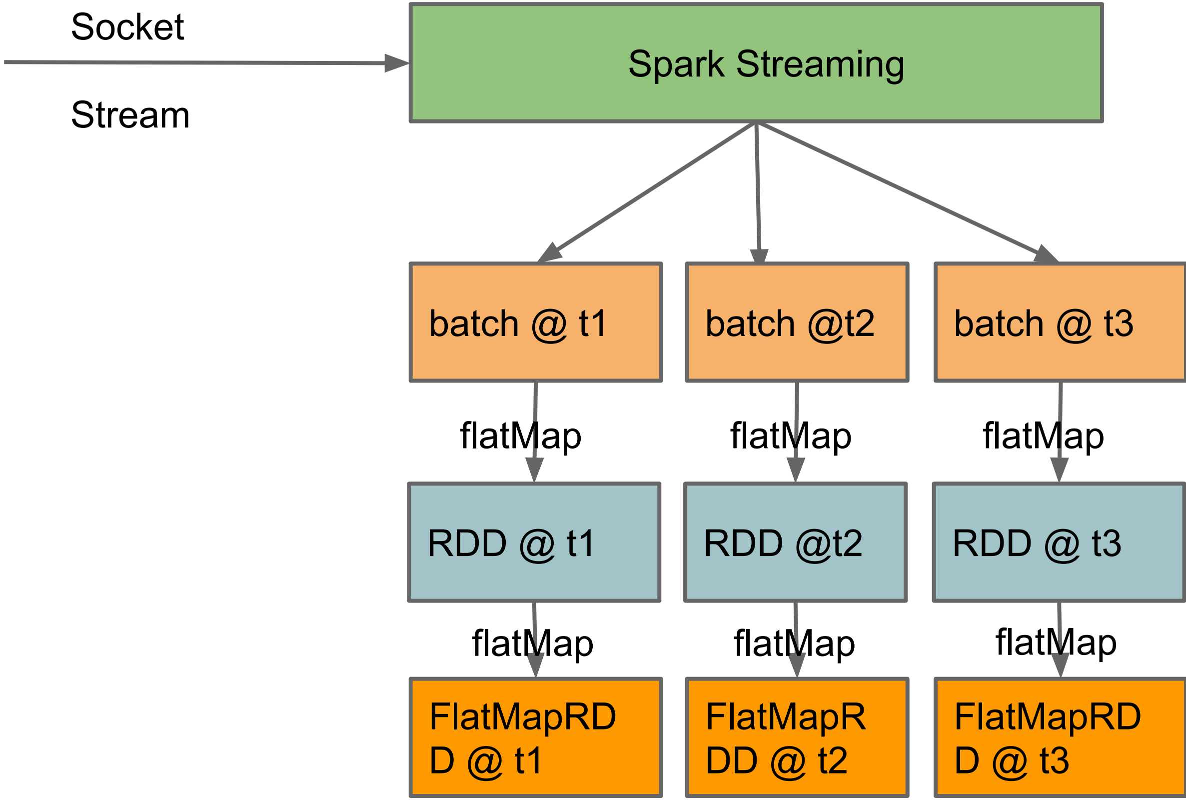
- Each batch of Dstream is represented as RDD underneath
- These RDD are replicated in cluster for fault tolerance
- Every DStream operation result in RDD transformation
- There are API’s to access these RDD is directly
- Can combine stream and batch processing

DStream transformation

```
val ssc = new StreamingContext(args(0), "wordcount", Seconds(5))

val lines = ssc.socketTextStream("localhost",50050)

val words = lines.flatMap(_.split(" "))
```



Socket stream

- Ability to listen to any socket on remote machines
- Need to configure host and port
- Both Raw and Text representation of socket available
- Built in retry mechanism

Wordcount example

File Stream

- File streams allows for track new files in a given directory on file system
- Whenever there is new file appears, spark streaming will pick it up
- Only works for new files, modification for existing files will not be considered
- Tracked using file creation time

FileStream example

Stateful operations

- Ability to maintain random state across multiple batches
- Fault tolerant
- Exactly once semantics
- WAL (Write Ahead Log) for receiver crashes

StatefulWordcount example

How stateful operations work?

- Generally state is a mutable operation
- But in functional programming, state is represented with state machine going from one state to another

  fn(oldState,newInfo) => newState

- In Spark, state is represented using RDD.
- Change in the state is represented using transformation of RDD’s
- Fault tolerance of RDD helps in fault tolerance of state

Using state for real world use cases

- Implementing cart discount
  CartDiscount.scala

- Recovering from failures for state
  RecoverableCardDiscount

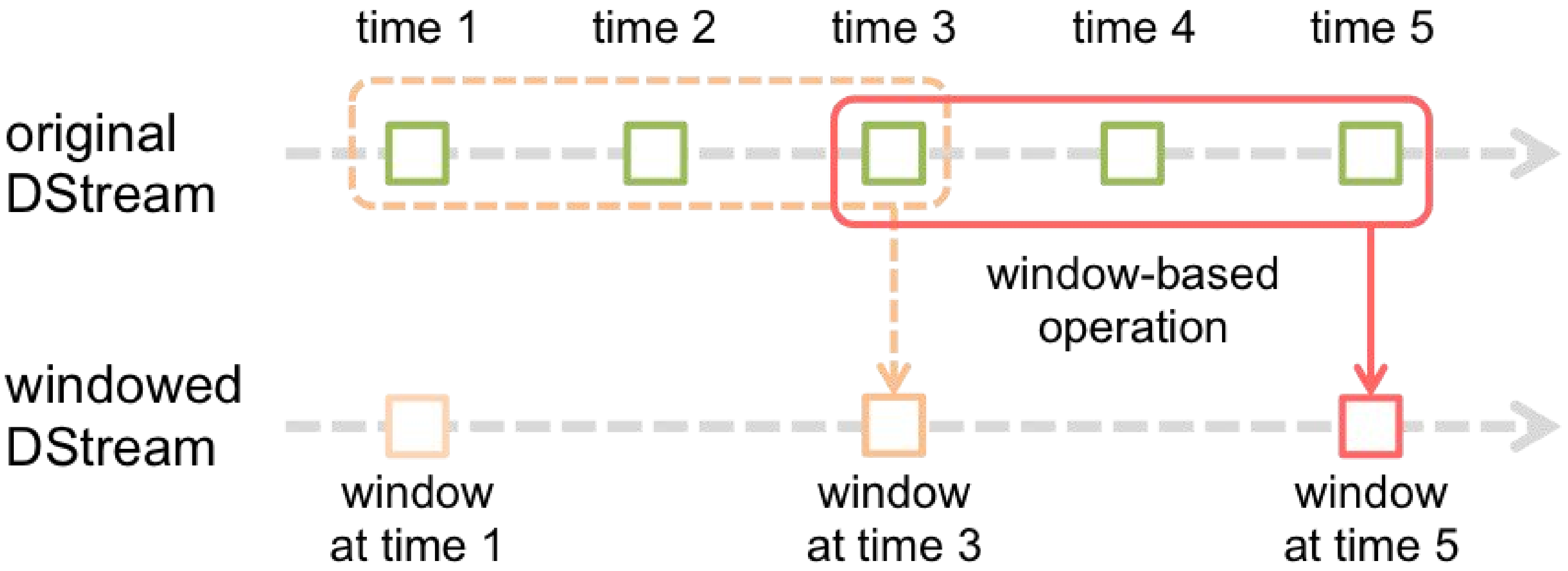
Transform API

- In stream processing, ability to combine stream data with batch data is extremely important
- Both batch API and stream API share RDD as abstraction
- transform api of DStream allows us to access underneath RDD’s directly

Ex : Combine customer sales data with customer information

CartCustomerJoin example

Window based operations



Window wordcount

Receiver architecture