Stream Processing
Spark Streaming

Amin FARVARDIN

Ph.D. Computer Science
University Paris Dauphine

Dauphine | PSL TUNIS

Dauphine | PSL UNIVERSITÉ PARIS

Spark Streaming

Batch Processing

- It is one of the first use cases for big data technologies
- Data is collected for a period of time and processed in batches
- Data spanning from hours to years
- For example, some organizations run nightly batch processing jobs, which process data collected throughout the day by various systems
- Have high latency
- So, there is a long wait before you can see the results

Stream Processing

- Sometimes data needs to be processed and analyzed as it is collected
- For example,
  - fraud detection in an e-commerce system
  - network intrusion or security breach detection
  - application or device failure detection in a data center
- One of the challenges with live data stream processing:
  - handling high-velocity data in real time or near real time.
- A data stream processing application running on a single machine will not be able to handle high-velocity data.
- A distributed stream processing framework addresses this issue.

Spark Streaming – Use cases

- Uber uses Spark Streaming for real-time telemetry analytics by collecting data from its users mobile

- Pinterest uses Spark Streaming to provide immediate insight into how users are engaging with pins across the globe in real-time

- Netflix uses Spark Streaming to provide movie recommendations to its users

Introducing Spark Streaming

- A distributed data stream processing framework
- A distributed applications for processing live data streams in near real time
- a simple programming model
- enables an application to process high-velocity stream data
- allows the combining of data streams and historical data for processing

Spark Streaming Is a Spark Add-on

- A library that runs on top of Spark
- To extends Spark for data stream processing
- It provides higher-level abstractions for processing streaming data
- But under the hood, it uses Spark

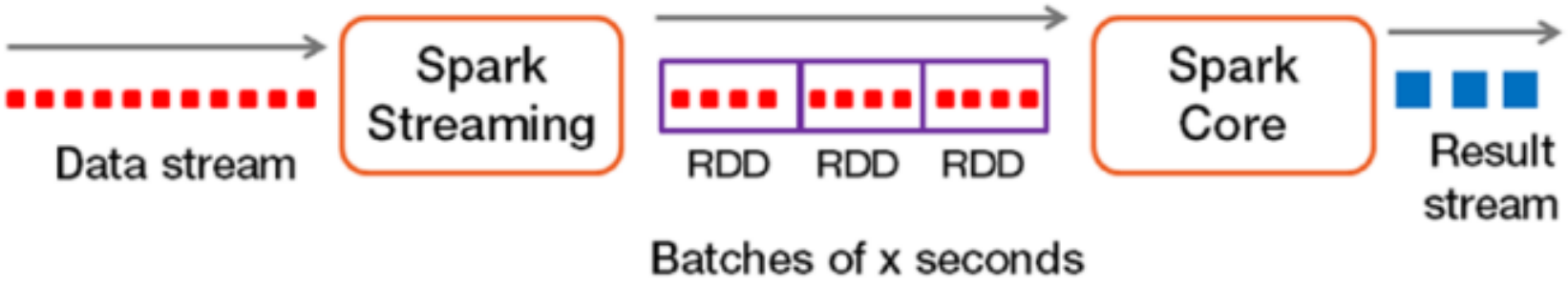
Spark Streaming runs on top of Spark core

Spark Streaming Is a Spark Add-on

- provides a scalable, fault-tolerant, and high-throughput distributed
- inherits all the features and benefits of Spark core
- The processing capability can be easily increased by adding more nodes to a Spark cluster
- Can be used along with other Spark libraries, such as
  - Spark SQL,
  - MLlib,
  - Spark ML, and
  - GraphX

High-Level Architecture

- Data stream processes in micro-batches
- By splits a data stream into batches of very small fixed-sized time intervals
- Data in each micro-batch is stored as an RDD (Resilient Distributed Datasets)
- Any RDD operation can be applied to an RDD created by Spark Streaming
- The results of the RDD operations are streamed out in batches



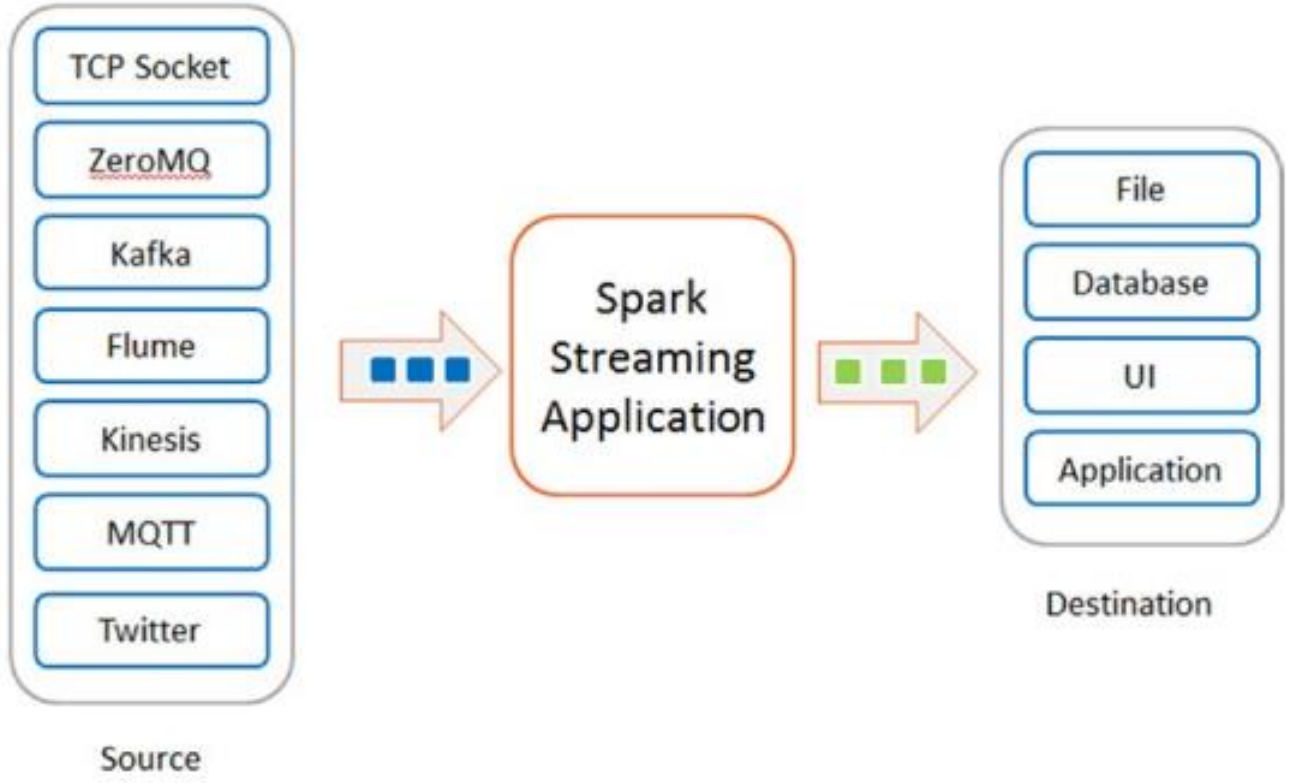
Data Stream Sources

The built-in streaming data sources are grouped into two categories:

- Basic data stream sources:
  - Include TCP sockets, Akka Actors, and files
  - Spark Streaming provides libraries for these sources
- Advanced sources
  - Include Kafka, Flume, Kinesis, MQTT, ZeroMQ
  - External libraries required that are not included with Spark Streaming

Data Stream Sources
Receiver

- A Receiver receives data from a streaming data source and stores it in memory
- Spark Streaming creates and runs a Receiver on a worker node for each data stream
- An application can connect to multiple data streams to process data streams in parallel



Stream data sources and destinations

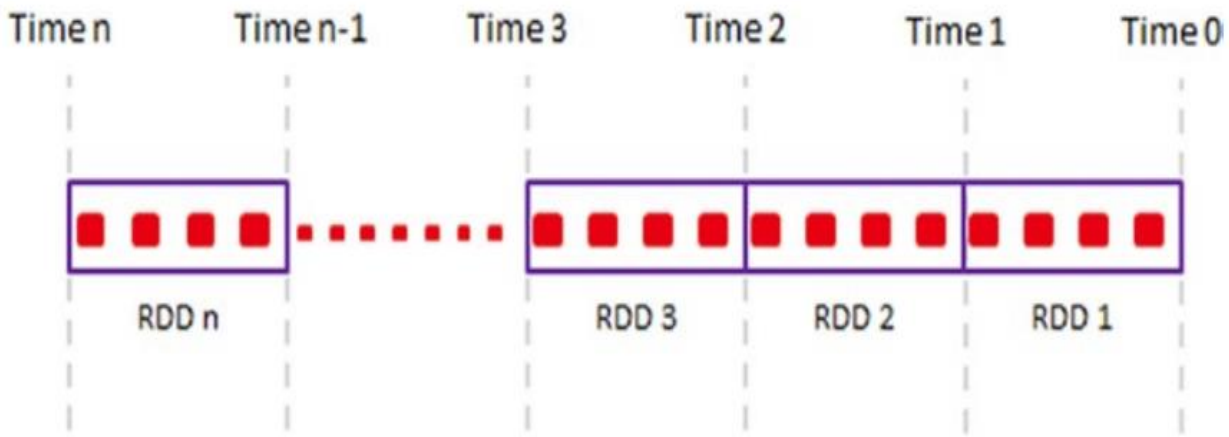
Data Stream Sources

Destinations

- A dashboard
  - may take some action or just display it
- May trigger cancellation of a transaction
  - A fraud detection application
- Be stored in a storage system such as a file or a database

Discretized Stream (DStream)

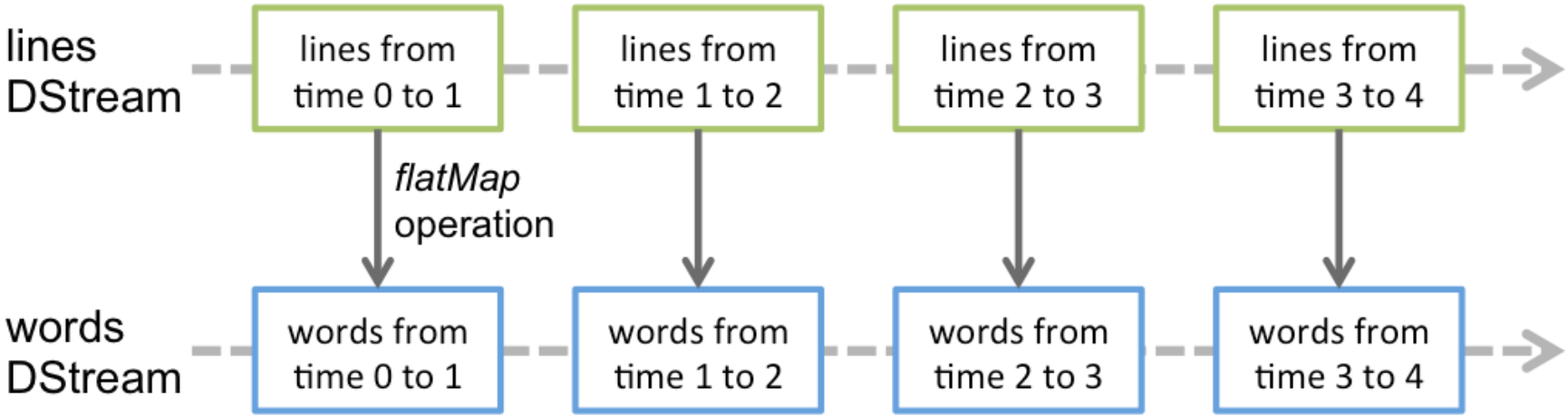
- An abstract class in the Spark Streaming library
- DStream is a sequence of RDDs
- DStream operations translate to operations on the underlying RDDs



DStream is a never-ending sequence of RDDs

Discretized Stream (DStream)

- Every transformation yield a new Dstream



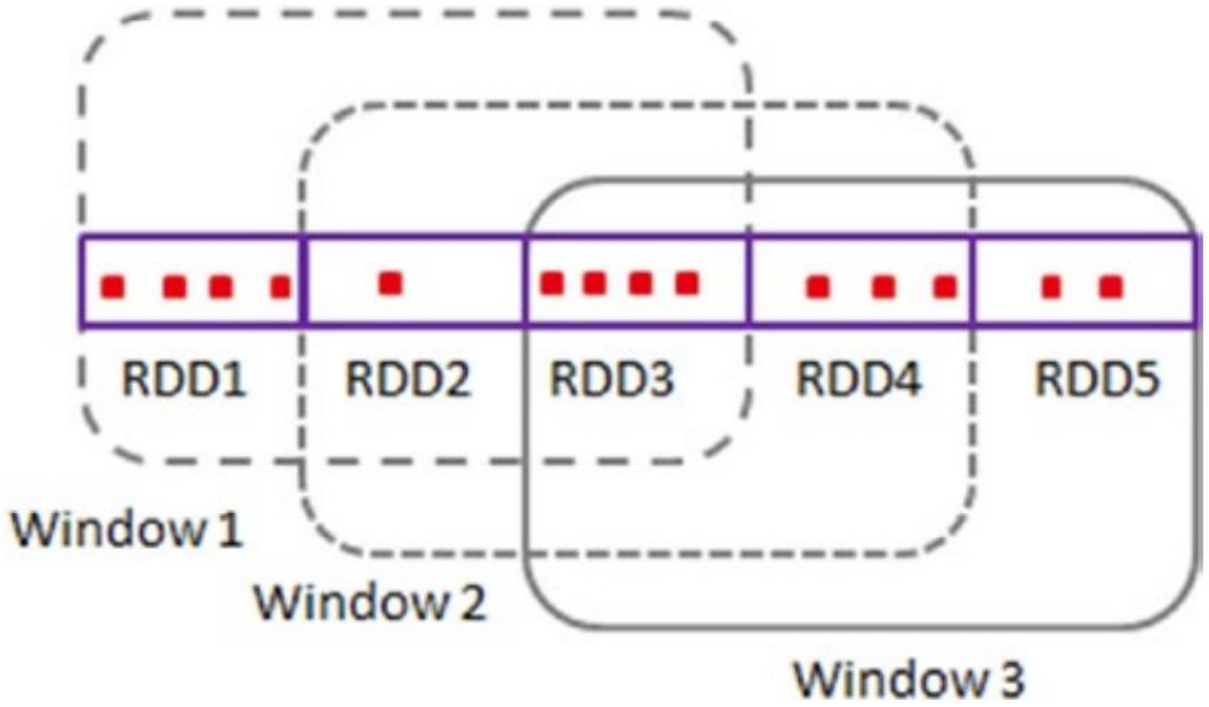
Spark Streaming

Transformation types

- Transformation has two different types:
  - **Stateless Transformations**: the processing of each batch does not depend on the data of its previous batches.
    - map(), filter(), and reduceByKey()
  - **Stateful Transformations**: use data from previous batches to compute the results of the current batch.
    - They include sliding windows, tracking state across time, etc.

Window Operation

- A DStream operation that is applied over a sliding window of data in a stream.
- Successive windows have one or more overlapping RDDs
- A window operation is a stateful DStream operation that combines data across multiple batches
- A window operation requires two parameters:
  - window length
  - sliding interval



DStream windows

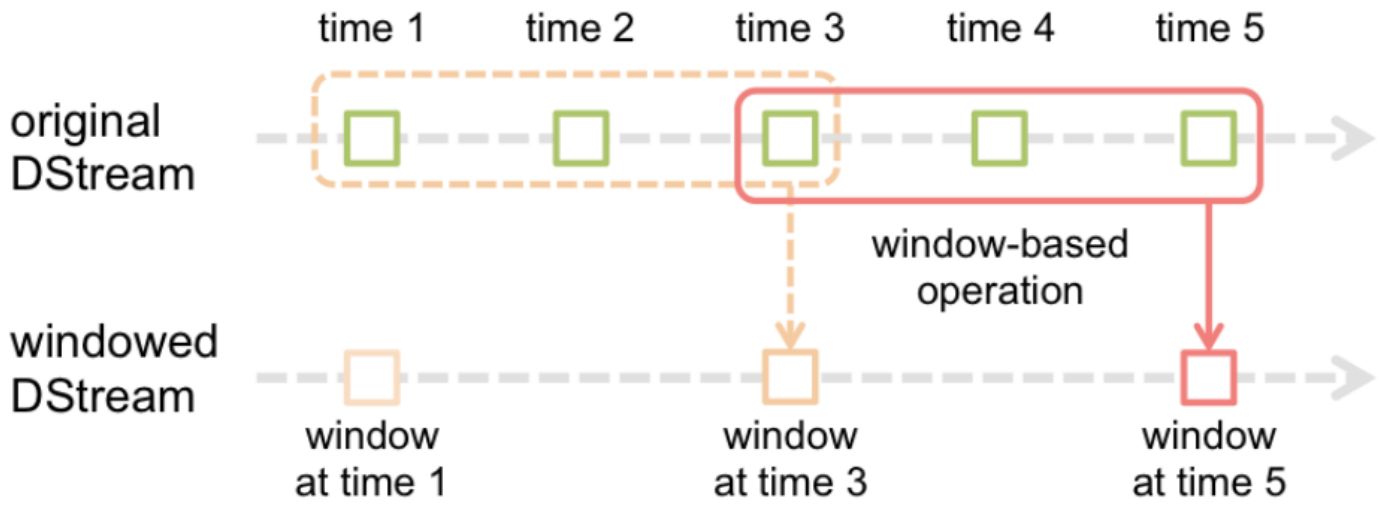
Window Operation

- Windowed computations allow you to apply transformations over a sliding window of data

Any window operation needs to specify two parameters:

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

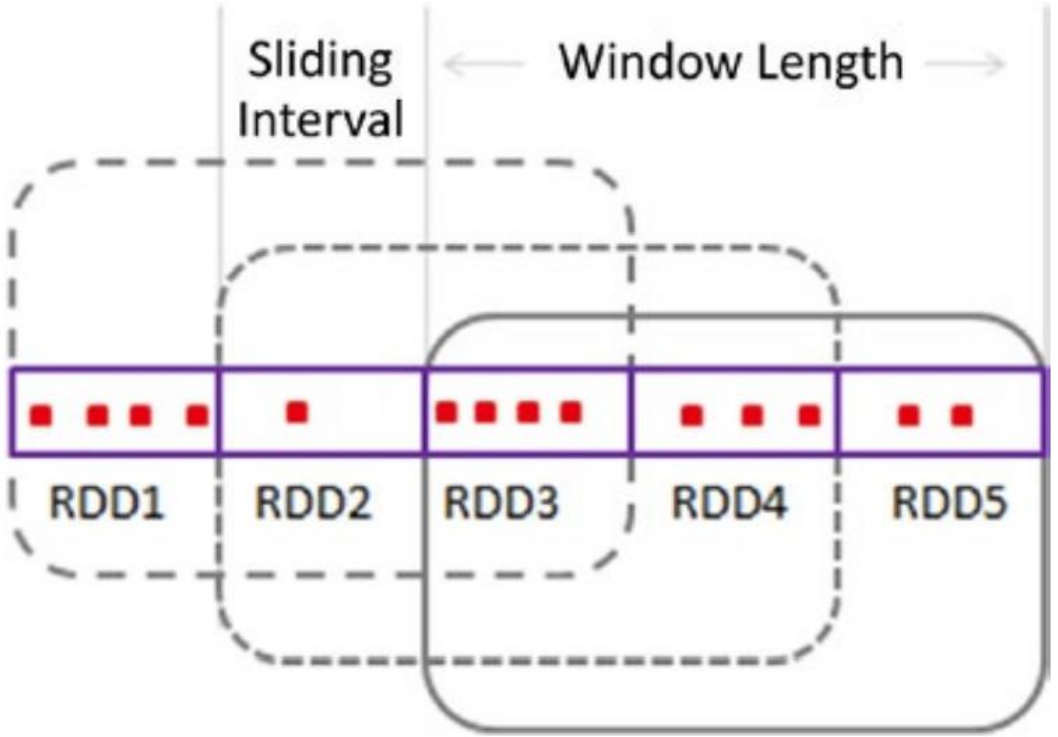
Note: These parameters must be multiples of the batch interval



Window Operation

- A window operation requires two parameters:
  - window length specifies the time duration over which a window operation is applied
  - sliding interval specifies the time interval at which a window operation is performed

- NOTE: It is the time interval at which new RDDs are generated by a window operation.



countByWindow

- Returns a DStream of single-element RDDs
- The single element in each returned DStream RDD is the count of the elements in a sliding window of a specified duration
- It takes two arguments, window duration, and sliding interval

```
ssc.checkpoint("checkpoint")
val lines = ssc.socketTextStream("localhost", 9999)
val words = lines flatMap {line => line.split(" ")}
val windowLen = 30
val slidingInterval = 10
val countByWindow = words.countByWindow(Seconds(windowLen), Seconds(slidingInterval))
countByWindow.print()
```

countByValueAndWindow

- Returns a DStream containing the counts of each distinct element within a sliding window that slides at the specified time interval.

```
ssc.checkpoint("checkpoint")
val lines = ssc.socketTextStream("localhost", 9999)
val words = lines flatMap {line => line.split(" ")}
val windowLen = 30
val slidingInterval = 10
val countByValueAndWindow = words.countByValueAndWindow(
      Seconds(windowLen),
      Seconds(slidingInterval))
countByValueAndWindow.print()
```

Hands-on practice

Socket Text Stream

https://github.com/afarvardin/dauphine_tunis/tree/main/socketTextStream