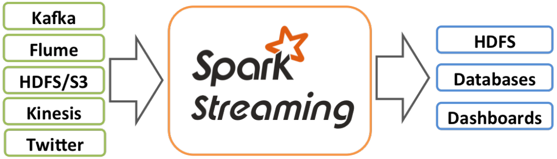
Big Data
Spark Streaming

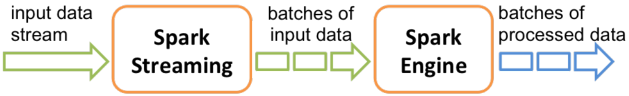
Dr. Wenceslao PALMA
wenceslao.palma@pucv.cl

Spark streaming



- An extension to the core api that enables scalable, high-throughput and fault tolerant stream processing of data streams.
- Data is ingested from many sources and can be processed using high level functions like map, reduce, join and window.
- The processed data can be stored to HDFS, databases and dashboards.

Spark streaming: batches



- A data stream is divided into batches.
- Batches are generated w.r.t an interval of time.
- The batch interval must considers the latency requirements of the applcation.
- Spark streaming provides a high level abstraction called Discretized stread (DStream).
- DStreams are created from input data streams or by applying hig-level operations on other DStreams.

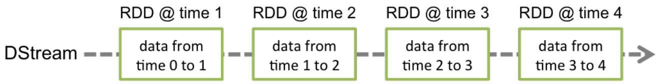
Spark streaming

- A StreamingContext object has to be created in order to initialize a Spark streaming program.
- Input sources are defined by creating input DStreams.
- The streaming computation is defined applying transformations and output operations to DSTreams.
- Using streamingContext.start() starts the data ingestion.
- The processing can be stopped manually or due to any error using streamingContext.awaitTermination().

Spark streaming: transformations and outputs

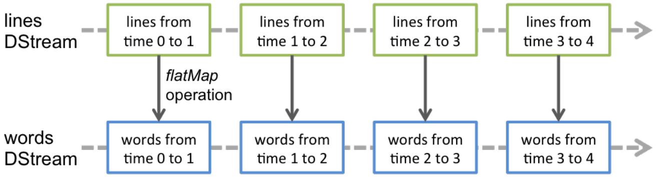
- transformations: DStreams support many of the transformations of Spark RDD's. Some of the common ones are the following: flatMap(), filter(), count(), union(), join(), reduceByKey(), etc
- outputs: similar to actions for RDDs, output operations trigger the execution of all the DStream transformations.
- using output operations DStream's data is pushed out to external systems (databases or filesystems). pprint(), saveAsTextFiles(), foreachRDD().
- a spark streaming program must include at least one output operation.

Spark streaming: DStreams



- A Dstream is inmutable and is represented by a continuous series of RDD's.
- Each RDD contains data from a certain interval.

Spark streaming: DStreams and transformations



- Any transformations applied on a Dstream operates on the underlying RDD's and generates a new DStream.
- All the RDD's transformations are computed by the Spark engine.

Spark streaming: an example

```
# spark-submit sparkStreaming-ncFilter.py 2>/dev/null

from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

conf = SparkConf()
conf.setMaster('yarn-client')
conf.setAppName('spark-streaming')
conf.set('spark.executor.instances', 3)

sc = SparkContext(conf=conf)

streamSC = StreamingContext(sc,60)

line = streamSC.socketTextStream("hadoop1",9999)

words = line.flatMap(lambda line: line.split(" "))
wPair = words.map(lambda word: (word,1))
wordCount = wPair.reduceByKey(lambda x, y: x+y)

wordCount.pprint()

streamSC.start()
streamSC.awaitTermination()
```

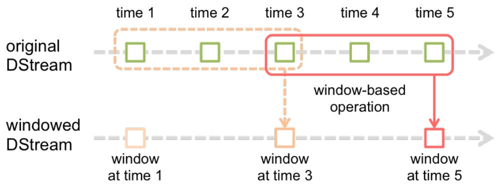
Spark streaming: instructions to execute the code

- Open two terminals.
- In the first terminal, a data server will be simulated using the Netcat utility ($nc -lk 9999).
- In the second server the python code is running ($ spark-submit sparkStreaming-nc.py).

Spark streaming: filter example

```
# spark-submit sparkStreaming-ncFilter.py 2>/dev/null

from pysaprk import SparkConf
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

conf = SparkConf()
conf.setMaster('yarn-client')
conf.setAppName('spark-streamingFilter')
conf.set('spark.executor.instances', 3)

sc = SparkContext(conf=conf)

streamSC = StreamingContext(sc,30)

line = streamSC.socketTextStream("hadoop1",9999)

filterDS = line.filter(lambda line: "sensor4" in line)
#print the first 10 elements of every batch of data in a DStream
filterDS.pprint()
filterDS.count().pprint()

streamSC.start()
streamSC.awaitTermination()
```

Spark streaming: Batches, windows and slide duration



- In the figure, the duration of the window is 3 (window length) and the interval at which the window operations is performed is 2 (sliding interval).
- Spark allows to apply transformations over a sliding window of data (windowed computations).
- RDDs that fall within the window are combined and operated to produce the RDDs of the windowed DStream.
- Some of the common window operations are the following: window(), countByWindow(), reduceByWindow(), reduceByKeyAndWindow(), countByValueAndWindow().

Spark streaming: Example (How the window slides)

```
from pyspark import SparkConf
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

conf = SparkConf()
conf.setMaster('yarn-client')
conf.setAppName('spark-streaming window map')
conf.set('spark.executor.instances', 3)

sc = SparkContext(conf=conf)
streamSC = StreamingContext(sc,10)

line = streamSC.socketTextStream("hadoop1",9998)

windw = line.window(30,10).pprint()

streamSC.start()
streamSC.awaitTermination()
-------------------------------------------------------------------
INPUT (nc -lk portNumber)               OUTPUT
10 (0th second)                           10
20 (10 seconds later)                     10 20
30 (20 seconds later)                     10 20 30
40 (30 seconds later)                     20 30 40
-------------------------------------------------------------------
```

Spark streaming: fault tolerance

- Remember that an RDD is an inmutable and deterministically re-computable distributed dataset.
- If any partition of an RDD is lost due to a worker node failure, then that partition can be re-computed from the original dataset using the lineage of operations.

All data transformations in Spark Streaming are based on RDD operations and as long as the input dataset is present all intermediate data can be recomputed.

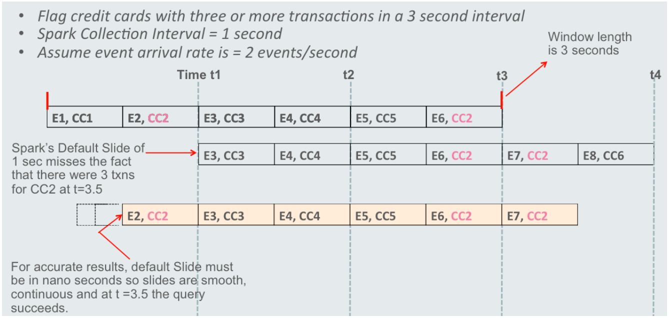
Spark streaming: failure of a worker node

- **Using HDFS files as inputs sources.** Data is reliable store in HDFS, thus no data will be lost and all data can be recomputed.
- **Using network-based data sources.** In this case, the input stream is replicated in memory between nodes of the cluster (default replication factor is 2).

Spark streaming: failure of the driver node

- Spark can periodically perform checkpointing by setting a checkpointing directory using the StreamingContext.
- Metadata checkpointing
  - Configuration
  - DStream operations
  - Incomplete batches

Spark streaming: some considerations about slide interval



- *Flag credit cards with three or more transactions in a 3 second interval*
- *Spark Collection Interval = 1 second*
- *Assume event arrival rate is = 2 events/second*

Window length is 3 seconds
Time t1
t2
t3
t4
E1, CC1
E2, CC2
E3, CC3
E4, CC4
E5, CC5
E6, CC2
E7, CC2
E8, CC6
Spark's Default Slide of 1 sec misses the fact that there were 3 txns for CC2 at t=3.5
For accurate results, default Slide must be in nano seconds so slides are smooth, continuous and at t =3.5 the query succeeds.