# Spark Streaming

# Table of Content

# What is Streaming Data

# What is Streaming Data

- Streaming means continuous flow events
- Streaming data is data that is generated continuously by thousands of data sources
- Batch processing framework fail to process data of this nature
- Streaming data processing is beneficial in most scenarios where new, dynamic data is generated on a continual basis

# Intro to Spark Streaming

# Intro to Spark Streaming

- Spark Streaming API is used for processing the data in real-time

- Other Spark  abstractions available: Spark SQL, Spark Core components are not capable of handling the data that you receive in real-time

# Intro to Spark Streaming

- Spark Streaming is a processing engine to process data in real-time from sources and output data to external storage systems

- It has three main components:
  - Input Source
  - Streaming Engine
  - Sink

# Intro to Spark Streaming

- Input sources: Kafka, Flume, HDFS/S3, etc.
- Spark Streaming engine processes incoming data from various input sources
- Sinks store processed data from Spark Streaming engine like HDFS, relational databases, or NoSQL datastores

# Intro to Spark Streaming

- Spark uses various output modes to store the streaming data:
  - Append Mode
  - Update Mode
  - Complete Mode

# Intro to Spark Streaming

- In append mode, Spark will output only newly processed rows since the last trigger

- In update mode, Spark will output only updated rows since the last trigger

- In complete mode, Spark will output all the rows it has processed so far

# How Spark Streaming Works?

# How Spark Streaming Works?

- A spark streaming job gets data from multiple sources

- You need to implement two components in this scenario:
  - Receiver (for connecting to a streaming source)
  - DStream (to receive the data)

# How Spark Streaming Works?

- The receiver gets connected to multiple sources that consistently generate events in real-time

- The source streams the data and receiver is the interface in a spark streaming job to consume the data

- We create the receiver using SparkStreamingContext interface

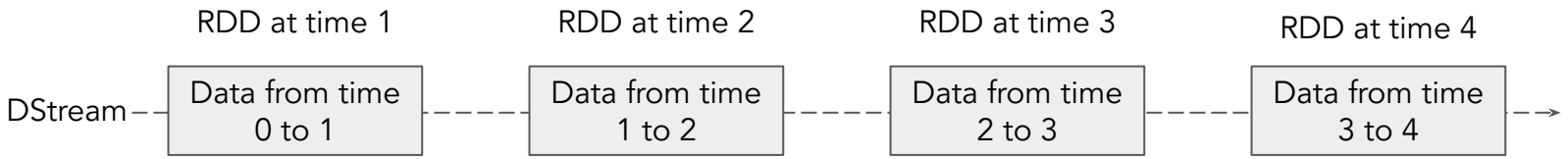
# How Spark Streaming Works?

- The second interface we implement to receive the data is Dstream (also known as *Discretized Streams*)

- When talking about continuous streams, the term “continuous” means that we never start or stop receiving data as part of the stream

- Dstream represents a continuous stream of data over time

# How Spark Streaming Works?



# How Spark Streaming Works?

- Spark Streaming accumulates the data in the form of batches
- The amount of data accumulated will depend on the batch interval defined in terms of time
- By defining the batch interval of 5 seconds, spark job will accumulate the data for 5 seconds and it will create 1 batch
- These batches gets generated continuously and the spark job runs on top of data accumulated in the batch interval of 5 seconds

# How Spark Streaming Works?

- You can apply transformations and actions (groupby, reduceby, map , flatmap, etc.) on top of these Dstreams

- Once Dstream gets generated, after that you job becomes the same as normal spark Job

# How Spark Streaming Works?

- DStream transformations can be categorized into two types:
  - Stateless Transformations
  - Stateful Transformations

# How Spark Streaming Works?

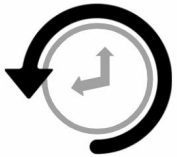
Stateless Transformations:

- Stateless transformations are transformations that applies on every batch in a DStream
- For any batch of data being processed, there is no dependency on the data from previous batches
- Ex. map(), filter(), reduceByKey(), etc.

# How Spark Streaming Works?

Stateful Transformations:

- Stateful transformations are transformations that tracks data across multiple batches
- For any batch of data being processed, there is either a partial or whole dependency on the data from previous batches
- It makes use of some data/results from previous batches and computes the result of the current batch of data
- Ex. map(), filter(), reduceByKey(), etc.



*Stateful Transformation are advanced topics that requires good understanding of sliding window and checkpointing. For more information please refer:*

*https://spark.apache.org/docs/latest/streaming-programming-guide.html*

# Streaming Sources

- The sources should be capable to stream the data to your spark job

- The sources can be (not just limited to):
  - Kafka
  - Flume
  - Socket Streaming
  - File

# Spark Streaming + Kafka Example

# Spark Streaming + Kafka Example

- We will count the number of words on the data you receive in real time
- Here we create a Kafka Producer program that writes messages to a Kafka Topic as bytes
- The objective is to create a Spark Streaming program as consumer that reads the data in real time and process it

# Spark Streaming + Kafka Example

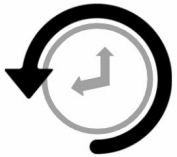
- Producer: Python program to publish data to a topic using KafkaProducer

```
from kafka import KafkaProducer
import json
import os

bootstrap = 'localhost: 9092'

#producer = KafkaProducer(bootstrap_servers = bootstrap)

json_producer = KafkaProducer(bootstrap_servers = bootstrap, \
                              value_serializer = lambda v: json.dumps(v).encode('utf-8'))

topic_name = 'kafkatest'
#producer.send(topic_name, "Hello there how are you")
#producer.send(topic_name, "Hello Hello Hello")

json_producer.send(topic_name, value="Hello Hello Hello")
producer.flush()
```

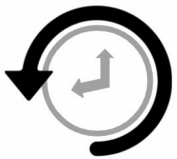
Bootstrap server address

# Spark Streaming + Kafka Example

- We use json_producer, if you want to write messages in the form of Json
- json_producer writes the data to a topic in the form of (key,value) pair
- We will have to create value_serializer since we are interested in values and not keys
- We also create another kafka producer with the same list of bootstrap server address for value we say
- for every value convert it into json and use utf-8 as the encoder



*keys in kafka is actually meant for accumulating all messages pertaining to a specific user or specific key by which you want to combine messages over single partition. In this case we don't have such requirement*



- Producer: To write to a Topic, you can also try this command as well

```
[hadoop@ip-172-31-37-192 bin]$ ./kafka-console-producer.sh --broker-list ec2-54-
145-95-53.compute-1.amazonaws.com:9092 --topic kafkatest
Hello World
Hello Hello Hello
Hello Hello Hello
Hello Hello Hello
```

# Spark Streaming + Kafka Example

- Consumer: Program

```
from pyspark.sql import SparkSession
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka import KafkaUtils


spark = SparkSession.builder.appName('appname').getOrCreate()

sc = spark.sparkContext

ssc = StreamingContext(sc, 10)

kvs = KafkaUtils.createDirectStream(ssc,topics=["kafkatest"],
                                    kafkaParams={"metadata.broker.list":"localhost:9092"})

lines = kvs.map(lambda x : x[1])

counts = lines.flatMap(lambda line: line.split(" ")).map(lambda word : (word, 1)).reduceByKey(lambda a,b: a+b)

counts.pprint()

ssc.start()
ssc.awaitTermination()

ssc.stop()
```

# Spark Streaming + Kafka Example

- A spark streaming context is created by providing two parameters
  - spark configuration
  - batch interval (in seconds)

- Once the spark streaming context is created, we need to create Dstream

- Dstream is the actual object that connects you to the source

# Spark Streaming + Kafka Example

- In the Dstream you will provide source
- Source would either be kafka, socket connection, file, flume, etc.
- The syntax would be different for different types of sources

# Spark Streaming + Kafka Example

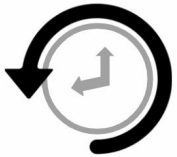
- Consumer: Result

```
-------------------------------------------
Time: 2021-06-24 12:36:55
-------------------------------------------
('Hello', 3)
```

# Intro to Structured Streaming

# Structured Streaming

- Structured Streaming is a stream API built on top of Spark SQL
- It lets you express computation on streaming data in the same way you express a batch computation on static data
- The Spark SQL engine performs the computation incrementally and continuously updates the result as streaming data arrives



*This model of streaming is based on Dataframe and Dataset APIs. Hence, with this library, we can easily apply any SQL query (using the DataFrame API) or Scala operations (using DataSet API) on streaming data.*

# Structured Streaming

- The structured streaming construct using DataFrames is an improvement on the use of DStreams as found in the original Apache Spark streaming model

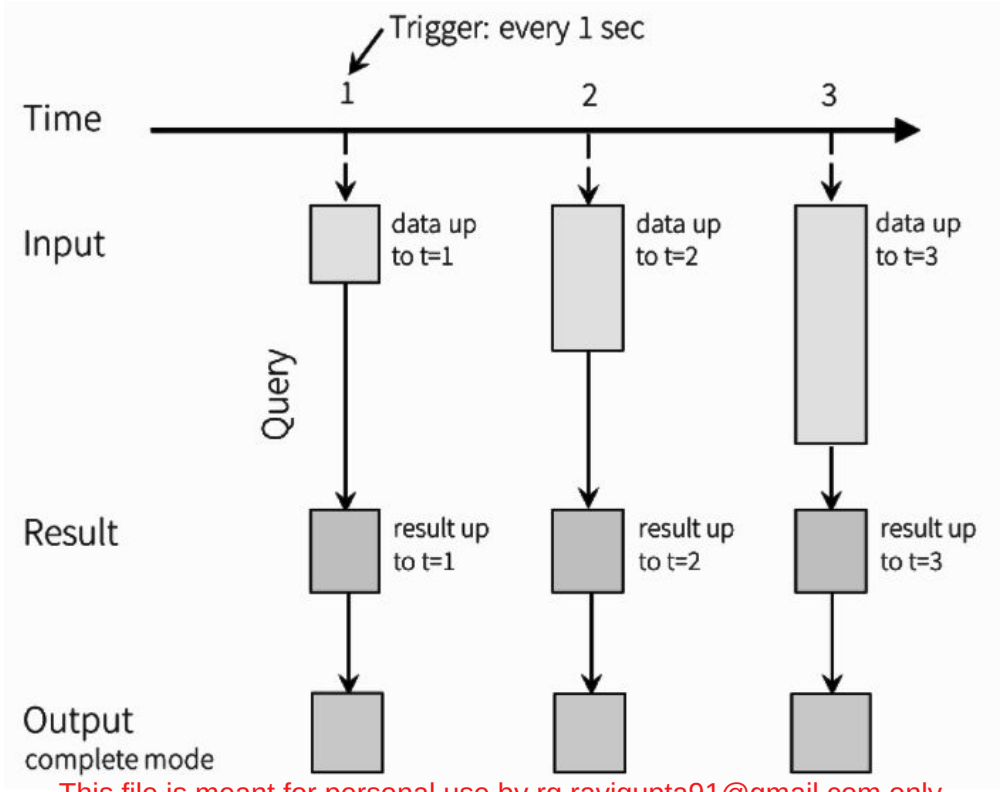
- Dtreams are fault-tolerant, but the division of streamed data into RDD blocks is slower than the DataFrames

- Hence, the processing and analysis of the data streams are slower, increasing the latency and reducing the message delivering reliability

# Structured Streaming vs. Spark Streaming

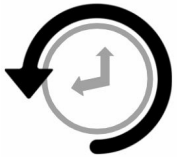
| Spark Streaming | Structured Streaming |
|---|---|
| Dstream based | Dataframe/Dataset based |
| Apache Spark streaming is a separate library in the Spark engine | Spark Structured streaming is part of the Spark 2.0 release built on top of Spark SQL |
| Slower | Faster |
| Spark Streaming works on something we call a micro batch | In Structured Streaming, there is no batch concept |

# Structured Streaming Programming Model



# Structured Streaming Programming Model

- Every data item that is arriving on the stream is like a new row being appended to the Input Table
- A query on the input will generate the “Result Table”
- Every trigger interval new rows get appended to the Input Table
- This Input Table updates the Result Table
- Whenever the result table gets updated, the changed result rows is written to an external sink in different Output modes



*Structured Streaming DOES NOT materialize the entire table. It reads the latest available data from the streaming data source, processes it incrementally to update the result, and then discards the source data. It only keeps around the minimal intermediate state data as required to update the result*

# Example

- Write a structured streaming query that processes the incoming csv file(in a folder) in real time a display the result on the console

- The data has following information about an employee:
  - emp_id
  - emp_name
  - job_name
  - manager_id
  - salary
  - dept_name

- The result should display the count of each job_name and should update the result as and when a file arrives in the folder

# Example

- Structured Streaming Program

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *

#create sparksession
spark = SparkSession.builder.appName("StructStream").getOrCreate()

#creating Schema
#Note:structured Streaming processing always requires the specification of a schema for
#the data in the stream
schema = StructType([StructField('emp_id', IntegerType(), True),
                     StructField('emp_name', StringType(), True),
                     StructField('job_name', StringType(), True),
                     StructField('manager_id', IntegerType(), True),
                     StructField('salary', DoubleType(), True),
                     StructField('dept_name', StringType(), True)])
```

# Example

```
#creating Streaming Dataframe that would read the data from a given folder
#Note: Please change the folder path accordingly
customer = spark.readStream.schema(schema)\

               .option("header", True)\

               .option("sep", ",")\

               .csv("hdfs:///users/hadoop")



#query to find the total count of employees in a particular profession
customer.groupBy("job_name").count().writeStream.format('console').outputMode('complete').
start().awaitTermination()
```

# Example

- Pass the csv files (data1.csv and data2.csv) in the *streamingdata* folder one at a time

- Result (after passing the first file)

```
-------------------------------------------
Batch: 0
-------------------------------------------
+---------+-----+
| job_name|count|
+---------+-----+
|  ANALYST|    2|
| SALESMAN|    3|
|    CLERK|    1|
|  MANAGER|    3|
|PRESIDENT|    1|
+---------+-----+
```

# Example

- Result (after passing the second file)

```
-------------------------------------------
Batch: 1
-------------------------------------------
+-----------------+-----+
|         job_name|count|
+-----------------+-----+
|          ANALYST|    2|
|         SALESMAN|    4|
|            CLERK|    4|
|SOFTWARE ENGINEER|    2|
|   DATA SCIENTIST|    4|
|          MANAGER|    3|
|        PRESIDENT|    1|
+-----------------+-----+
```

# Summary

- Streaming data is data that is generated continuously by thousands of data sources
- Spark Streaming API is used for processing the data in real-time
- Spark Streaming engine processes incoming data from various input sources
- You need to implement two components in Spark Streaming:
  - Receiver - for connecting to a streaming source
  - DStream - to receive the data
- Dstream represents a continuous stream of data over time (batch interval)

# Thank You