Real-Time Data Processing using Spark Streaming

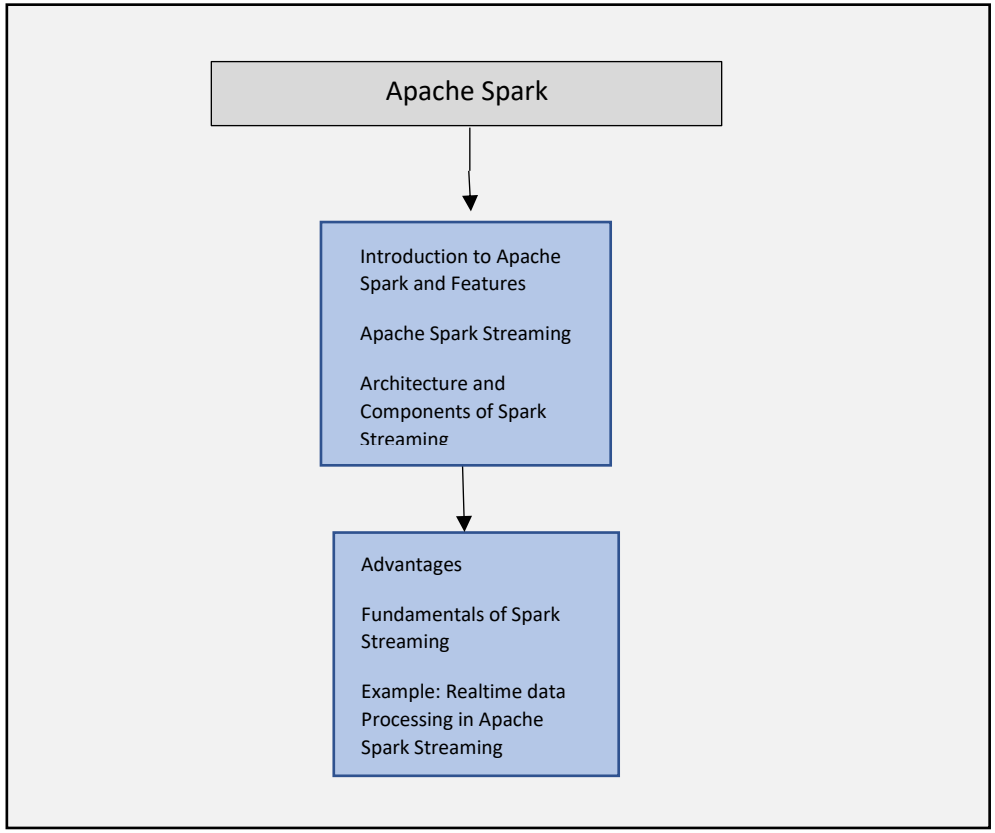
Apache Spark Streaming is an integral part of Spark core API to perform real-time data analytics. It allows us to build a scalable, high-throughput, and fault-tolerant streaming application of live data streams.

As a part of Realtime Data Processing with Apache Spark, you covered:

- Introduction to Apache Spark and Features
- Apache Spark streaming and Architecture
- Advantage and Fundamentals of Spark Streaming
- Example: Realtime data streaming with Apache Spark Streaming

Common Interview Questions:

1. What are some of the features of Spark?
2. What is Apache Spark Streaming?
3. Describe how Spark Streaming processes data?
4. What are DStreams?
5. What is a StreamingContext object?
6. What are the two different types of built-in streaming sources provided by Spark Streaming?
7. What are some of the common transformations on DStreams supported by Spark Streaming?
8. What are the output operations that can be performed on DStreams?



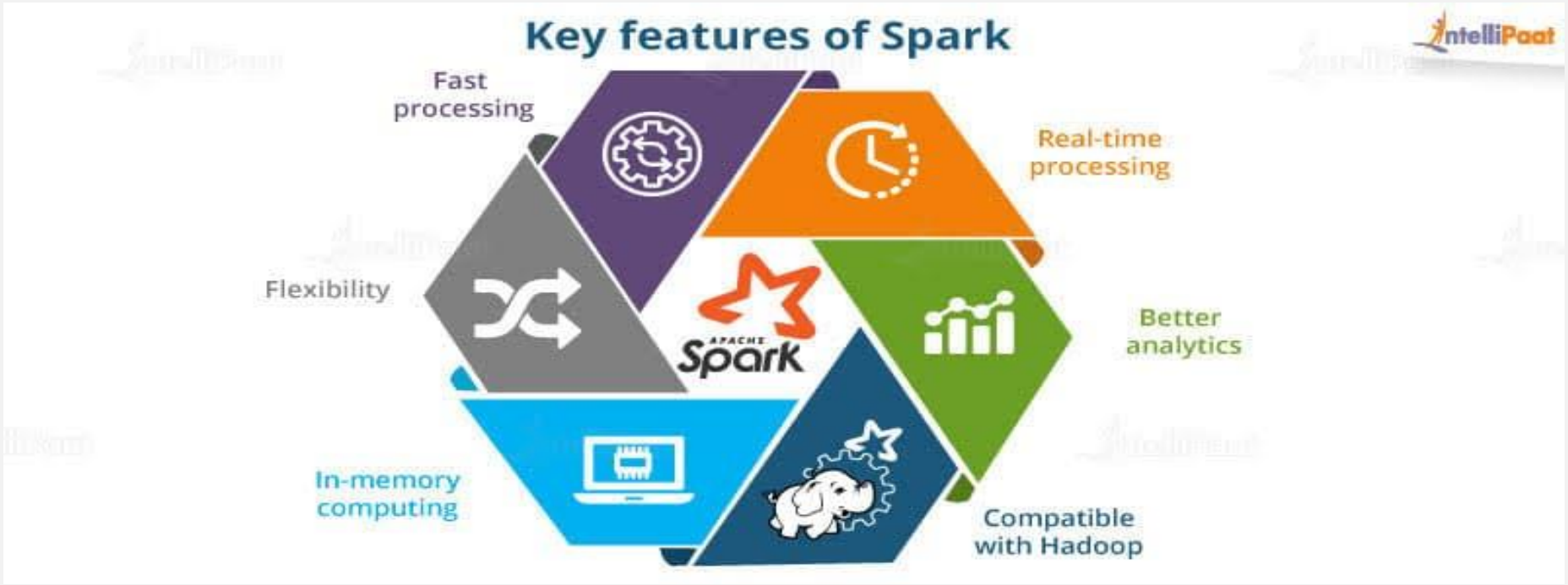
Real-Time Data Processing using Spark Streaming

Apache Spark:

Apache Spark is an open-source data-processing engine for large data sets. It is designed to deliver the computational speed, scalability, and programmability required for Big Data, specifically for streaming data, graph data, machine learning, and artificial intelligence (AI) applications.

It scales by distributing processing work across large clusters of computers, with built-in parallelism and fault tolerance. It even includes APIs for programming languages that are popular among data analysts and data scientists, including Scala, Java, Python, and R.

Features of Apache Spark:

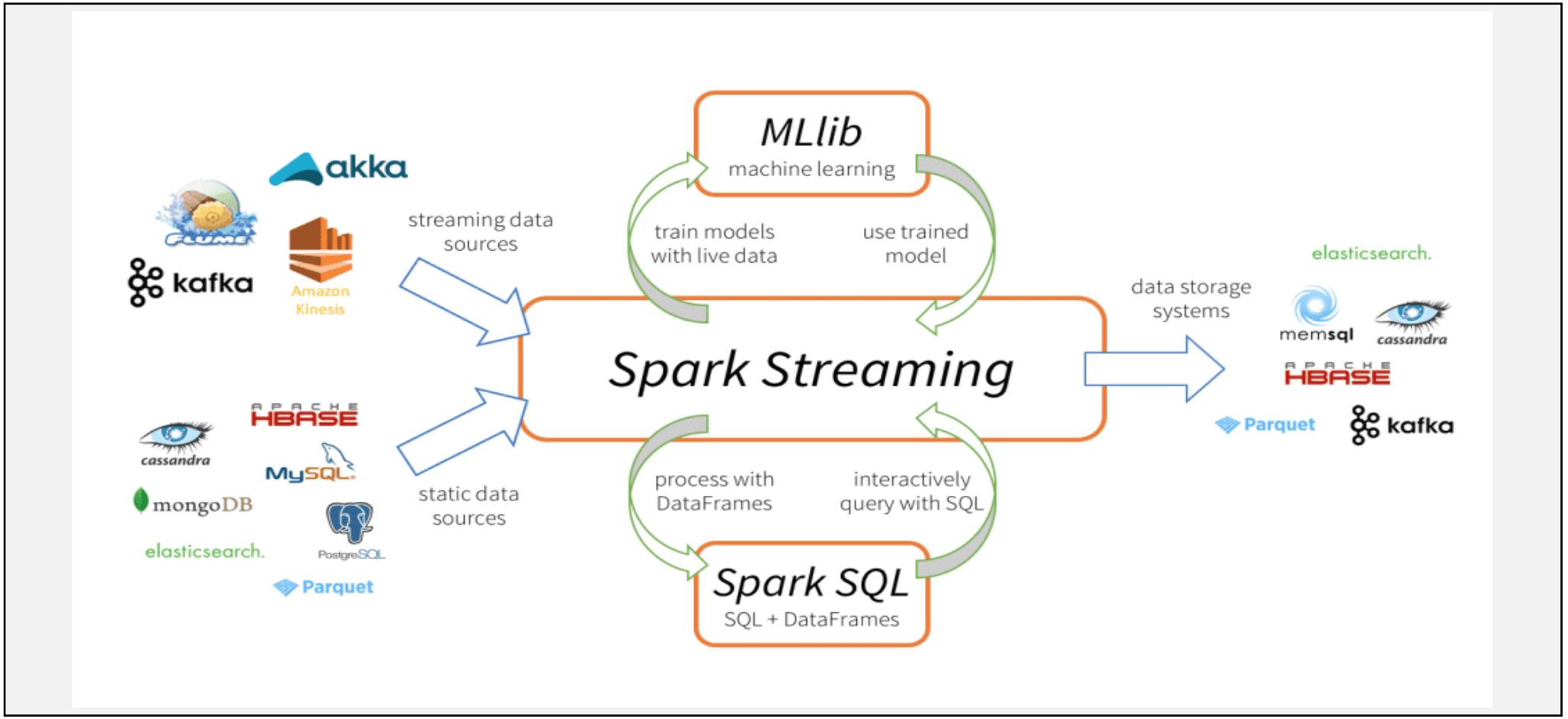


Real-Time Data Processing using Spark Streaming

Spark Streaming:

Spark Streaming is an integral part of Spark core API to perform real-time data analytics. It allows us to build a scalable, high-throughput, and fault-tolerant streaming application of live data streams.

Spark Streaming supports the processing of real-time data from various input sources and storing the processed data to various output sinks.



Real-Time Data Processing using Spark Streaming

Components of Spark Streaming:

Components of Spark Streaming:

Input data sources:

Streaming data sources (like Kafka, Flume, Kinesis, etc.), static data sources (like MySQL, MongoDB, Cassandra, etc.), TCP sockets, Twitter, etc.

Spark Streaming engine:

To process incoming data using various built-in functions, complex algorithms. Also, we can query live streams, apply machine learning using Spark SQL and MLlib respectively.

Output Sinks:

Processed data can be stored to file systems, databases(relational and NoSQL), live dashboards etc.

Advantages of Spark Streaming:

Advantages of Spark Streaming:

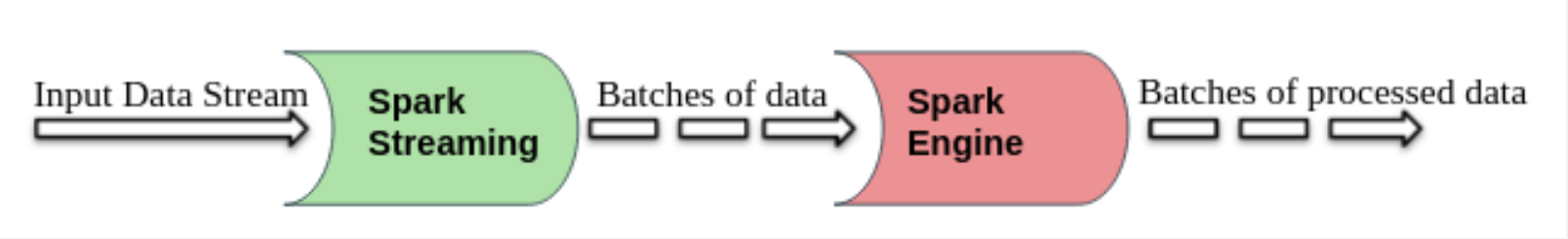
- Unified streaming framework for all data processing tasks(including machine learning, graph processing, SQL operations) on live data streams.
- Dynamic load balancing and better resource management by efficiently balancing the workload across the workers and launching the task in parallel.
- Deeply integrated with advanced processing libraries like Spark SQL, MLlib, GraphX.
- Faster recovery from failures by re-launching the failed tasks in parallel on other free nodes.

Real-Time Data Processing using Spark Streaming

Fundamentals of Apache Spark Streaming:

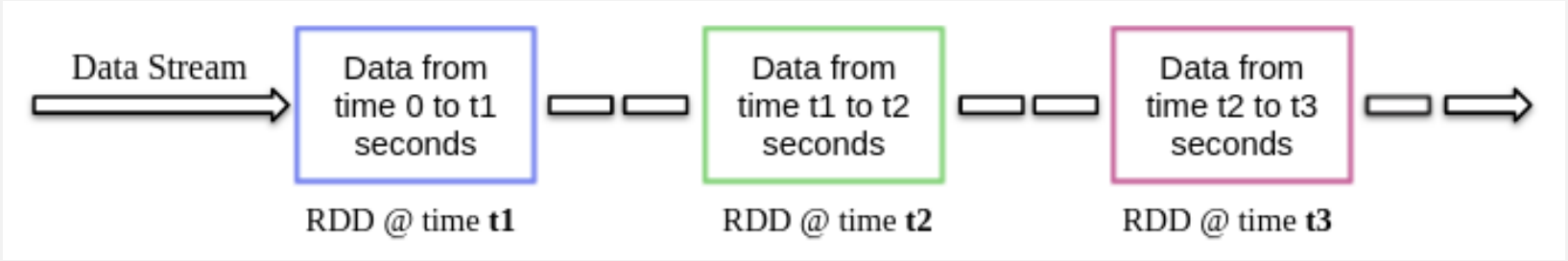
Spark Streaming receives live input data streams and divides the data into batches, which are then processed by the Spark engine to generate the final stream of results in batches.

Spark Streaming provides a high-level abstraction called discretized stream or DStream, which represents a continuous stream of data.



DStream (Discretized Stream)

They can be created either from streaming data sources (like Kafka, Flume, Kinesis, etc.) or by performing high-level operations on other DStreams. Internally DStream is a sequence of RDDs and this phenomenon allows Spark Streaming to integrate with other Spark components like MLlib, GraphX, etc



While creating a streaming application we also need to specify the batch duration to create new batches at regular time intervals. Typically, the batch duration varies from 500 ms to several seconds. For e.g. it is 3 seconds then the input data is collected every 3 seconds.

Apache Spark Streaming common Operations:

Some common Apache Spark Streaming Transformation Operations:

map(func) - map() transformation returns a new distributed dataset from a source dataset formed by passing each element of the source through a function func.

filter() - filter() transformation returns a new distributed dataset from a source dataset formed by selecting the elements of the source on which func returns true.

flatMap() - flatMap() transformation is similar to map() function. In flatMap() each input item can be mapped to 0 or more output items.

union() - union() transformation returns a new dataset that contains the union of the elements in the source dataset and the dataset that is passed as argument to the function.

intersection() - intersection() transformation returns a new distributed dataset that contains the intersection of elements in the source dataset and the dataset that is passed as argument to the function.

distinct() - distinct() transformation returns a new distributed dataset that contains the distinct elements.

Output operations that can be performed on DStreams:

Output operations on DStreams pushes the DStream's data to external systems like a database or a file system.

Following are the key operations that can be performed on DStreams.

saveAsTextFiles() - Saves the DStream's data as text file.

saveAsObjectFiles() - Saves the DStreams data as serialized Java objects.

saveAsHadoopFiles() - Saves the DStream's data as Hadoop files.

foreachRDD() - A generic output operator that applies a function, func, to each RDD generated from the DStream.

Real-Time Data Processing using Spark Streaming

Output Modes:

Append Mode	Update Mode	Complete Mode
It involves writing only the new incoming data to the sink. So this can be used when it is required to insert only the new data but not update the previous state of data. It is the default output mode. It does not support aggregation operations since aggregation depends on old data.	It involves writing the data records that are either new or for which the old value is updated. So this mode can be used when it is required to have the "upsert" mode of operation doing some aggregation. If no aggregation is applied, the update mode works the same as the append mode.	It involves writing the complete data again. This means that every time the entire data is written to the result table. So this mode can be used when it is required to overwrite all the previous data. This mode can be used only when aggregations are applied.

Types of Triggers in Spark:

	Default	Once	Processing Time	Continuous
Processing Interval	ASAP in seconds	Only once	Custom in seconds	ASAP in Milliseconds
Task type	Periodic	Once	Periodic	Long running
Latency	Low (seconds)	Batch	Custom (Seconds)	Very Low (Milliseconds)
Throughput	Medium	High	Medium – High	Low
Commit type	Synchronous	Synchronous	Synchronous	Asynchronous
Write Ahead Log (Commit)	Before	Before	Before	After
Supported Semantics	All	All	All	At least once
Epoch markers	Not used	Not used	Not used	Used

Real-Time Data Processing using Spark Streaming

Streaming Join in spark:

Streaming Join in spark:

In Spark Structured Streaming, a streaming join is a streaming query that was described (build) using the high-level streaming operators:

- Dataset.crossJoin
- Dataset.join
- Dataset.joinWith
- SQL's JOIN clause

Streaming joins can be stateless or stateful:

- Joins of a streaming query and a batch query (stream-static joins) are stateless and no state management is required.
- Joins of two streaming queries (stream-stream joins) are stateful and require streaming state (with an optional join state watermark for state removal).

Watermarking in spark:

Watermarking is a useful method which helps a Stream Processing Engine to deal with lateness. Basically, a watermark is a threshold to specify how long the system waits for late events. If an arriving event lies within the watermark, it gets used to update a query. Otherwise, if it's older than the watermark, it will be dropped and not further processed by the Streaming Engine.

Since Spark 2.1, watermarking is introduced into Structured Streaming API. You can enable it by simply adding the withWatermark-Operator to a query:

withWatermark(eventTime: String, delayThreshold: String): Dataset[T]

It takes two Parameters, a) an event time column (must be the same as the aggregate is working on) and b) a threshold to specify for how long late data should be processed (in event time unit).

The state of an aggregate will then be maintained by Spark until max eventTime — delayThreshold > T , where max eventTime is the latest event time seen by the engine and T is the starting time of a window. If late data fall within this threshold, the query gets updated eventually. Otherwise, it gets dropped and no reprocessing is triggered

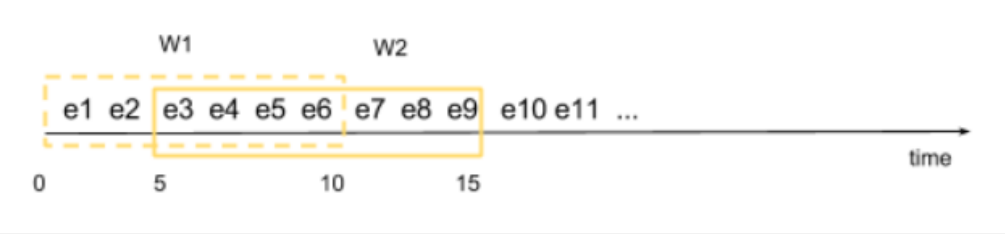
Real-Time Data Processing using Spark Streaming

Types of windows:

Sliding window:

In a sliding window, tuples are grouped within a window that slides across the data stream according to a specified interval. A time-based sliding window with a length of ten seconds and a sliding interval of five seconds contains tuples that arrive within a ten-second window. The set of tuples within the window are evaluated every five seconds. Sliding windows can contain overlapping data; an event can belong to more than one sliding window.

In the following image, the first window (w1, in the box with dashed lines) contains events that arrived between the zeroth and tenth seconds. The second window (w2, in the box with solid lines) contains events that arrived between the fifth and fifteenth seconds. Note that events e3 through e6 are in both windows. When window w2 is evaluated at time t = 15 seconds, events e1 and e2 are dropped from the event queue.



Tumbling window:

In a tumbling window, tuples are grouped in a single window based on time or count. A tuple belongs to only one window.

For example, consider a time-based tumbling window with a length of five seconds. The first window (w1) contains events that arrived between the zeroth and fifth seconds. The second window (w2) contains events that arrived between the fifth and tenth seconds, and the third window (w3) contains events that arrived between tenth and fifteenth seconds. The tumbling window is evaluated every five seconds, and none of the windows overlap; each segment represents a distinct time segment.

Real-Time Data Processing using Spark Streaming

Example: Creating a Real-Time Data Processing with Apache Spark Streaming:

Example: Simple streaming program to receive text data streams on a particular port, perform basic text cleaning (like white space removal, stop words removal, lemmatization, etc.), and print the cleaned text on the screen.

Step 1: Creating Streaming Context and Receiving data stream

1.1 Instantiating StreamingContext class from pyspark.streaming module.

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

1.2 While creating StreamingContext we can specify the batch duration, for e.g. here the batch duration is 3 seconds.

```
sc = SparkContext(appName = "Text Cleaning")
strc = StreamingContext(sc, 3)
```

1.3 Once the StreamingContext is created, we can start receiving data in the form of DStream through TCP protocol on a specific port. For e.g. here the hostname is specified as “localhost” and port used is 8084.

```
text_data = strc.socketTextStream("localhost", 8084)
```

Real-Time Data Processing using Spark Streaming

Step 2: Performing operations on data streams

After creating a DStream object, we can perform operations on it as per the requirement. Here, we wrote a custom text cleaning function.

```
import re
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
def clean_text(sentence):
    sentence = sentence.lower()
    sentence = re.sub("s+"," ", sentence)
    sentence = re.sub("W"," ", sentence)
    sentence = re.sub(r"httpS+", "", sentence)
    sentence = ' '.join(word for word in sentence.split() if word not in stop_words)
    sentence = [lemmatizer.lemmatize(token, "v") for token in sentence.split()]
    sentence = " ".join(sentence)
    return sentence.strip()
```

Real-Time Data Processing using Spark Streaming

Step 3: Starting the Streaming service

The streaming service has not started yet. Use the start() function on top of the StreamingContext object to start it and keep on receiving streaming data until the termination command (Ctrl + C or Ctrl + Z) is not received by *awaitTermination() function.*

strc.start()

strc.awaitTermination()

Now first we need to run the 'nc' command (Netcat Utility) to send the text data from the data server to the spark streaming server. So run the following nc command in the terminal.

nc -lk 8083

Similarly, run the pyspark script in a different terminal using the following command in order to perform text cleaning on the received data.

spark-submit streaming.py localhost 8083

As per this demo, any text written in the terminal (running netcat server) will be cleaned and the cleaned text is printed in another terminal after every 3 seconds (batch duration).