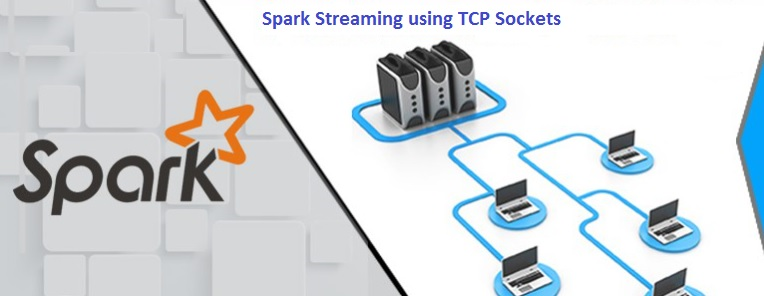
Session 22

Assignment 1

|  |  |
| --- | --- |
| **Prepared For:** | AcadGild |
|  |  |
| **Document Approval:** | **AcadGild** |
|  |  |
|  |  |
|  |  |
|  |  |
| **Project Title:** | Session 22 – Assignment 1 |
|  |  |
| **Prepared By:** | Duncan Burgess |
|  |  |
|  | dburgess@duncb.com |
|  |  |
| **Primary Engineer:** | Duncan Burgess |
|  |  |
| **Document Reference:** | **Session 21 – Assignment 1** |
|  |  |
| **Start Date:** | 22/10/2017 |
|  |  |
|  |  |



# 

# Contents

[Contents 2](#_Toc496434259)

[Change History 3](#_Toc496434260)

[1. Problem Statement 4](#_Toc496434261)

[2. Spark Streaming using TCP Socket 4](#_Toc496434262)

[2.1. Spark Streaming 4](#_Toc496434263)

[2.2. Code written 5](#_Toc496434264)

[2.3. NetCat 6](#_Toc496434265)

[2.4. Results 6](#_Toc496434266)

# Change History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Document Revision** | **Date** | **Authored By** | **Authorised By** | **Sections Affected** | **Reason for Change** |
| Rev 01 | 22/10/2017 | Duncan Burgess |  | All | Initial release. |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# Problem Statement

Follow the following blog

# Spark Streaming using TCP Socket

Learn the Spark streaming concepts by performing its demonstration with TCP socket.

Spark Streaming is an extension of core Spark API, which allows processing of live data streaming. In layman’s terms, Spark Streaming provides a way to consume a continuous data stream, and some of its features are listed below.

* Enables scalable, high throughput, and fault-tolerant data processing.
* Supports many input sources like TCP sockets, Kafka, Flume, HDFS/S3, etc.
* Uses a micro-batch architecture.

The internal working of Spark Streaming.

## Spark Streaming

*Figure:* spark streaming

Spark Streaming continuously receives live input data streams and divides the data into multiple batches.

* These new batches are created at regular time intervals, called **batch intervals.** The application developer can set batch intervals according to their requirement.
* Any data that arrives during an interval gets added to the batch.
* At the end of a batch interval.

In this assignment I will discuss the Spark streaming concepts by performing its demonstration with TCP socket.

This was run in Spark Eclipse

Spark Streaming is an extension of core Spark API, which allows processing of live data streaming. In layman’s terms, Spark Streaming provides a way to consume a continuous data stream, and some of its features are listed below.

* Enables scalable, high throughput, and fault-tolerant data processing.
* Supports many input sources like TCP sockets, Kafka, Flume, HDFS/S3, etc.
* Uses a micro-batch architecture.
* Spark engine processes these batches.

Spark Streaming is built on an abstraction called Discretized Stream or DStream. It represents the sequence of data arriving with time. Internally, each DStream is represented as a sequence of RDDs. A DStream is created from StreamingContext.

**Note Restriction**

We can only have one StreamingContext per JVM.

Once a DStream is created, it allows two kinds of operations: ***Transformation*** and ***Output operation****.*

A demo of Spark Streaming from a TCP socket. In this, we will perform the task of counting words in text data received from a data server listening on a TCP socket.

## Code written

**package** com.duncb.spark

**import** org.apache.spark.\_

**import** org.apache.spark.streaming.\_

**import** org.apache.log4j.\_

**object** streaming {

Logger.getLogger("org").setLevel(Level.ERROR)

**def** main(args:Array[*String*]) {

**val** SparkConf = **new** SparkConf().setAppName("streaming").setMaster("local[2]")

// Create a local StreamingContext with batch interval of 10 second

**val** ssc = **new** StreamingContext(SparkConf, Seconds(10))

/\* Create a DStream that will connect to hostname and port, like localhost 9999. As stated earlier, DStream will get created from StreamContext, which in return is created from SparkContext. \*/

**val** lines = ssc.socketTextStream("localhost",8000)

// Using this DStream (lines) we will perform transformation or output operation.

**val** words = lines.flatMap(\_.split(" "))

**val** wordCounts = words.map(x => (x, 1)).reduceByKey(\_ + \_)

wordCounts.print()

ssc.start() // Start the computation

ssc.awaitTermination() // Wait for the computation to terminate

}

## NetCat

In parallel in another terminal, I ran ncat server using port 8000 (**Ncat –l –p8000**) to run netcat as a data server.

**Command Run**

C:\Users\duncanbu>"C:\Program Files (x86)\Nmap\ncat.exe" -l -p 8000

This terminal acts as a server where we will continuously feed the words, and our Spark Streaming code will count the number of occurrences (in a batch interval of 10 sec).

We are running the code from IDE. Check outputs displayed below.

**Text typed into Ncat server**

This is a test

Bingo Bango bongo

Duncan Duncan Duncan

As the interval has been set at 10 sec in the first batch interval **Text typed** was captured and the count results were given as:

## Results

The results outputted are as follows:

Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties

-------------------------------------------

Time: 1508656970000 ms

-------------------------------------------

-------------------------------------------

Time: 1508656980000 ms

-------------------------------------------

(is,1)

(This,1)

(test,1)

(a,1)

-------------------------------------------

Time: 1508656990000 ms

-------------------------------------------

(Bango,1)

(Bingo,1)

(bongo,1)

-------------------------------------------

Time: 1508657010000 ms

-------------------------------------------

-------------------------------------------

Time: 1508657020000 ms

-------------------------------------------

(Duncan,3)

-------------------------------------------

Time: 1508657030000 ms

-------------------------------------------