Spark Structure Streaming

1. Load the data

scala> val activity\_data = spark.read.json("/sparkLab/activity-data/")

1. Visualize the data

scala> activity\_data.show(5)

1. See the Schema

scala> val activity\_schema = activity\_data.schema

1. Specify the Source

scala> val activity\_data\_stream = spark.readStream.schema(activity\_schema).option("maxFilesPerTrigger",1).json("/sparkLab/activity-data/")

1. Specify the transformation and action

scala> val activity\_count\_stream = activity\_data\_stream.groupBy("gt").count()

1. Specify the Sink and start the streaming

scala> val activityQuery = activity\_count\_stream.writeStream.queryName("activity\_count\_stream\_query").format("console").outputMode("complete").start()

1. Await termination

activityQuery.awaitTermination()

**Checkpoint:**

1. Load the data

scala> val activity\_data = spark.read.json("/sparkLab/activity-data/")

1. Visualize the data

scala> activity\_data.show(5)

1. See the Schema

scala> val activity\_schema = activity\_data.schema

1. Specify the Source

scala> val activity\_data\_stream = spark.readStream.schema(activity\_schema).option("maxFilesPerTrigger",1).json("/sparkLab/activity-data/")

1. Specify the transformation and action

scala> val activity\_count\_stream = activity\_data\_stream.groupBy("gt").count()

1. Specify the Sink and start the streaming

scala> val activityQuery = activity\_count\_stream.writeStream.queryName("activity\_count\_stream\_checkpoint\_query").format("console").outputMode("complete").**option("checkpointLocation","/sparkLab/checkpoint").**option("truncate",false).start()

Explore checkpoint directory:

hadoopuser@hadoopuser-VirtualBox:~$ hdfs dfs -ls /sparkLab/checkpoint

Stateful Processing Practical’s

**Checkpoint:**

1. Load the data

scala> val activity\_data = spark.read.json("/sparkLab/activity-data/")

1. Visualize the data

scala> activity\_data.show(5)

1. See the Schema

scala> val activity\_schema = activity\_data.schema

1. Specify the Source

scala> val activity\_data\_stream = spark.readStream.schema(activity\_schema).option("maxFilesPerTrigger",1).json("/sparkLab/activity-data/")

1. Specify the transformation and action

import org.apache.spark.sql.functions.{window, col}

scala> val activity\_count\_stream = activity\_data\_stream.**selectExpr( "\*", "cast(cast(Creation\_Time as double)/1000000000 as timestamp) as event\_time").**groupBy**(window(col("event\_time"),"10 minutes")** ).count()

**Window size :** 10 mins – because analyzing for every 10 minutes

**Trigger:** every time a new file comes, new file come every second, so it will run every second. But we don’t want it to run every second, because it will unnecessarily take lot of resources. We want to run it **every 5 minutes**

1. Specify the Sink and start the streaming

scala> val activityQuery = activity\_count\_stream.writeStream.queryName("activity\_count\_stream\_window\_query").format("console").outputMode("complete")**.**option("truncate",false).start()

**Small change in above (Change in E and F) – Sliding window**

1. Specify the transformation and action

import org.apache.spark.sql.functions.{window, col}

scala> val activity\_count\_stream = activity\_data\_stream.**selectExpr( "\*", "cast(cast(Creation\_Time as double)/1000000000 as timestamp) as event\_time").**groupBy**(window(col("event\_time"),"10 minutes","5 minutes")** ).count()

**Window size :** 10 mins – because analyzing for every 10 minutes

**Trigger:** every time a new file comes, new file come every second, so it will run every second. But we don’t want it to run every second, because it will unnecessarily take lot of resources. We want to run it **every 5 minutes**

1. Specify the Sink and start the streaming

scala> val activityQuery = activity\_count\_stream.writeStream.queryName("activity\_count\_stream\_slidewindow\_query").format("console").outputMode("complete")**.**option("truncate",false).start()

**Small change in above (Change in G and H) – Watermark**

1. Specify the transformation and action

import org.apache.spark.sql.functions.{window, col}

scala> val activity\_count\_stream = activity\_data\_stream.**selectExpr( "\*", "cast(cast(Creation\_Time as double)/1000000000 as timestamp) as event\_time").** withWatermark("event\_time", "30 minutes").groupBy**(window(col("event\_time"),"10 minutes","5 minutes")** ).count()

**Window size :** 10 mins – because analyzing for every 10 minutes

**Trigger:** every time a new file comes, new file come every second, so it will run every second. But we don’t want it to run every second, because it will unnecessarily take lot of resources. We want to run it **every 5 minutes**

1. Specify the Sink and start the streaming

scala> val activityQuery = activity\_count\_stream.writeStream.queryName("activity\_count\_stream\_watermarkslidewindow\_query").format("console").outputMode("complete")**.**option("truncate",false).start()

**Small change in above (Change in I and J) – Watermark**

1. Specify the transformation and action

import org.apache.spark.sql.functions.{window, col}

scala> val activity\_count\_stream = activity\_data\_stream.**selectExpr( "\*", "cast(cast(Creation\_Time as double)/1000000000 as timestamp) as event\_time").dropDuplicates("User", "event\_time").** withWatermark("event\_time", "30 minutes").groupBy**(window(col("event\_time"),"10 minutes","5 minutes")** ).count()

1. Specify the Sink and start the streaming

scala> val activityQuery = activity\_count\_stream.writeStream.queryName("activity\_count\_stream\_duplicatewatermarkslidewindow\_query").format("console").outputMode("complete")**.**option("truncate",false).start()