Spark streaming

Igor Yakushin
ivy2@uchicago.edu

December 27, 2019

Streaming

- Streaming allows to process continuously arriving data
- There are two streaming interfaces in Spark:
  - Structured Streaming - DataFrame API
  - Spark Streaming (DStreams) - RDD API
- We shall only consider Structured Streaming.
- Structured Streaming is a scalable and fault-tolerant stream processing engine built on the Spark SQL engine.
- One can express streaming computation the same way one would express a batch computation on static data.
- The Spark SQL engine will take care of running it incrementally and continuously and updating the final result as streaming data continues to arrive
- Structured Streaming can use as sources: files, Kafka, socket (for testing)

Streaming: Lab 6: word_count.py

```
from pyspark.sql.functions import explode, split
import sys
port = int(sys.argv[1]); host = sys.argv[2]
spark = SparkSession.builder.getOrCreate()
lines = spark.readStream.format("socket") \
    .option("host",host).option("port",port).load()
words = lines.select(
   explode(
       split(lines.value, " ")
   ).alias("word")
)
wordCounts = words.groupBy("word").count()
query = wordCounts \
    .writeStream \
    .outputMode("complete") \
    .format("console") \
    .start()
query.awaitTermination()
```

Streaming: Lab 6: word_count.py

- Generate random port number from 9000 to 10000 and write the corresponding commands used below into the files mync.sh and mystream.sh.
- Open the second terminal on the same host and run netcat in it:

```
source env.sh
make nc
```

- netcat will connect to `localhost:<port>`
- In the first terminal, start pyspark streaming program:

```
make word_count
```

- Now pyspark listens to input on `localhost:<port>`
- In the second terminal type some words. Once you hit Enter, all the words including the current ones are reprocessed by pyspark. Keep entering new lines and observe the output in the first terminal.
- To stop both processes, type Ctrl+C in the window with pyspark.

Streaming: Lab 6: age.py

```
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType
from pyspark.sql.functions import avg

spark = SparkSession.builder.getOrCreate()

userSchema = StructType().add("name", "string")\
                         .add("age", "integer")
csvDF = spark.readStream.option("sep", ";") \
    .schema(userSchema).csv("input_csv")

averageAge = csvDF.select(avg(csvDF.age))

query = averageAge.writeStream.outputMode("complete")\
                              .format("console").start()
query.awaitTermination()
```

Streaming: Lab 6: age.py

- In this lab data is processed as files are added into `input_csv` directory
- First start pyspark by running

  ```
  make age
  ```

- In the second terminal

  ```
  mv tmp/1.csv input_csv/
  ```

- Observe average age computed in the first terminal
- Next

  ```
  mv tmp/2.csv input_csv/
  ```

- Notice: it is important to use `mv` because files for Spark Streaming are supposed to be immutable. `mv` simply renames a file instantaneously while something like `cp` might take time. As a result Spark might process the first part of the file it notices and ignore the rest.