# Homework #10 – Spark Streaming Top-K

Yoke Kai Wen, 2020280598

May 27, 2021

## 1 Introduction

In this assignment, I implemented a Top-K (K=100) program using Spark Streaming to process a new text file from a HDFS directory every minute, for a duration of five minutes. As the text file data is being streamed, I find the top-100 most frequent words (accumulated from time zero) and print them out in descending order every minute.

## 2 Implementation overview

As per the instructions, I ran the file generator `bash generator.sh` to simulate the streaming data source. Once the generator is running, I ran `bash submit.sh` to run my `SimpleApp.py`, which contains the SparkStream implementation for finding the top-100 most frequent words every minute.

## 3 Description of solution: `SimpleApp.py`

I modified the given sample project code script of the same name `SimpleApp.py`, which listens to a HDFS directory, receives a new file per minute, count the words in the current file, and then prints out the word count result.

The main modifications I made were:

1. After counting the number of words for the current file, combine and accumulate the previous count history with the current count.
2. Sorting and printing the top 100 words in descending order.
3. Saving the word counts every minute.

### 3.1 Accumulate word count

In order to accumulate the word count across RDDs from different batches, stateful transformations (`updateStateByKey`) have to be executed. This requires Checkpointing to be enabled, and this is done by saving checkpoint information in a specified directory. The `updateStateByKey` operation allows us to continuously update an arbitrary state with new information. To use this operation, I first define a state update function `updateFunc` which specifies how to update the state using the previous state and the new values from an input stream. Finally, I update the `running_counts` state (DStream data type) with the new word count using `.updateStateByKey(updateFunc)`.

```
sc = SparkContext(appName="Py_HDFSWordCount")
ssc = StreamingContext(sc, 60)
ssc.checkpoint("hdfs://intro00:9000/user/2020280598/hw10output/checkpoint")

def updateFunc(new_values, last_sum):
    return sum(new_values) + (last_sum or 0)

lines = ssc.textFileStream("hdfs://intro00:9000/user/2020280598/stream")  # you should change path to your own directory on hdfs

running_counts = lines.flatMap(lambda line: line.split(" "))\
               .map(lambda x: (x, 1))\
               .updateStateByKey(updateFunc)
```

### 3.2 Sorting and printing in descending order

I applied the transform operation on the `running_counts` DStream, to sort each RDD in the DStream in descending order, and subsequently applied `pprint` to print the top 100 word counts.

```
sorted_ = running_counts.transform(lambda rdd: rdd.sortBy(lambda x: x[1], ascending=False))
sorted_.pprint(100)
```

### 3.3 Saving word count results every minute

The `saveAsTextFiles` operation allows DStream to be saved in a specified output directory, e.g. `hw10output/`. The output directory can then be copied from the HDFS directory to the main directory with `/hadoop/bin/hdfs dfs -get hw10output/`. However, I was unable to figure out how to automatically limit the outputs to only the the first 100 word counts in code and format the outputs. Hence, I manually saved the 5 results files by copying the printed terminal output and pasting into a text editor.

```
sorted_.saveAsTextFiles("hdfs://intro00:9000/user/2020280598/hw10output/")
```