School of Information Technologies
Dr. Ying Zhou

COMP5349: Cloud Computing **Sem. 1/2018**

Week 5: Spark Tutorial

12.04.2018

In this lab, you will practice SPARK RDD based Programming on locally installed Spark. The Spark version we have installed is `spark-2.2.1-bin-hadoop2.7` and it is installed under `/usr/local`.

Similar to MapReduce, Spark can run locally or in a cluster. Spark cluster can be managed by different resource management systems. Recent Spark releases support four such systems: Standalone, Mesos, YARN and Kubernetes. Please note that Spark Standalone refers to spark's own cluster management system. It comes with Spark installation. Mesos, YARN and Kubernetes are all third party, more general cluster management systems that works with not just Spark. Through out the semester, we will use the local mode and the YARN mode. This lab focuses on local mode.

Question 1: Prepare the Environment

Spark also need some configuration, including environment variables as well as configuration files. The login script you obtained from last week's lab also set the Spark environments variables: SPARK_HOME, SPARK_CONF and SPARK_LOG.

Spark provides a web based history server to enable visually viewing running statistics. The following configuration deals mainly with seting up and starting the history server.

A sample spark-conf has been pushed to git. Pull from lab-commons to obtain the update. Then

- copy `spark-conf` directory from `lab_commons` to `comp5349`
- open `spark-defaults.conf` in a text editor and replace all hard-coded unikey such as 'yzho8449' with your own unikey
- create a directory `spark-logs` under `comp5349`
- create a directory `spark-history` under `comp5349`

Start spark history server with the following command:

```
$SPARK_HOME/sbin/start-history-server.sh
```

Spark is able to talk to any storage source supported by Hadoop, including the local file system, HDFS, Cassandra, HBase, Amazon S3, etc. It can load input file from or write output to those sources. The input/output location is specified by URI. Each resource uses a different protocol. The protocol is included as part of the URI. For instance, `hdfs://` refers to hdfs location; `s3n://` refers to Amazon S3 location. Local file system is indicated by `file://`.If no protocol name is specified, Spark uses a default protocol. The actual value of default protocol depends on various factors. For instance, if Hadoop configuration is present, HDFS is the default storage protocol. In that case, if no protocol name is specified, Spark assumes you are referring to HDFS. It will search the HADOOP_CONF directory to find the HDFS location. It is always a good practice to use absolute path including the protocol prefix to specify the input and output location.

The following exercises assumes HDFS as the default protocol. If you have configured HDFS properly in week 4 tutorial. You can start it with

```
    $HADOOP_HOME/sbin/start-dfs.sh
```

All Spark sample applications in this lab use a movie rating data set downloaded from `http://grouplens.org/datasets/movielens/`. The two files: `movies.csv` and `ratings.csv` have been pushed to `lab-commons` repository under `data/week5`.

The `movies.csv` file contains movie information. Each row represents one movie, and has the following format:

```
movieId,title,genres
```

The `ratings.csv` file contains rating information. Each row represents one rating of one movie by one user, and has the following format:

```
userId,movieId,rating,timestamp
```

Use HDFS command `hdfs dfs -put [source] [target]` to put these files in HDFS under your home directory. Below is an example commands:

```
hdfs dfs -put week5 week5
```

Question 2: Running Sample Example

This sample application computes the average rating of each genre. It is the sample application in week 5's lecture. In summary, the application first converts the movies data into a key value pair RDD using `flatMapToPair` operation; it also converts the rating data into a key value pair RDD using `mapToPair` operation. Next, the two RDDs are joined on the common key `movie-id`. In Pyspark version, the join result is a pair RDD; In java version, the `join` result is an RDD of tuple and needs to be mapped to a new pair RDD with genre as the key and rating as the value. An `aggregateByKey` operation is applied to this pairRDD to compute the sum and number of ratings per genre. The average rating is then computed for each genre using a `mapToPair` operation.

a) **Python Version** Two python versions are provided a script one (`AverageRatingPerGenre.py`) and a jupyter notebook one (`AverageRatingPerGenre.ipynb`). They use a common module (`ml_utils.py`) which defines all functions to be used. A simple script (local_submit.sh) is provided to submit the script to local Spark. The script uses default input and output values. If you need to change the input and output. Use options `--input` and/or `--out` respectively.

   To run the notebook version, open a terminal window and change the present working directory to where the notebook file resides. Type `jupyter notebook` to start the notebook server. This will print some information about the notebook server in your terminal, including the URL of the web application. It will then open your default web browser to this URL. The notebook dashboard will show a list of the notebooks, files, and subdirectories in the directory where the notebook server was started. Click `AverageRatingPerGenre.ipynb` to open it in a new tab. Click `ml_utils.py` also to view all functions defined.

b) **Java Version** The Java version again comes with an ant file. Run `ant` command on the terminal would compile the source code into a JAR. To submit spark application locally, run the following command in a terminal:

```
spark-submit  \
  --class ml.MovieLensLarge \
  --master local(4) \ 修改为[4]
  sparkML.jar \
  week5/ \
  week5-out-java/
```

   The command assumes that you have downloaded the lab data and uploaded it in your local HDFS under your home directory in a directory called “week5”

Question 3: Obtaining Execution Data from History Server

Spark history server provides lots of useful execution data about the application. In this week we will only inspect the execution DAG. A spark program may contain many jobs. A job is chain of RDD transformations. The history server has nice visualization of the RDD transformation chain organized in a direct acyclic graph (DAG). Figure 1 is the execution DAG of the Java sample application. The flow is similar to what you have seen in the lecture slides.

It is worth noting that the execution DAG of Python application may be quite different to the actual operations used in the code. We suspect it has something to do with how PySpark implements the RDD transformations. Figure 2 shows an example of the Python sample application.

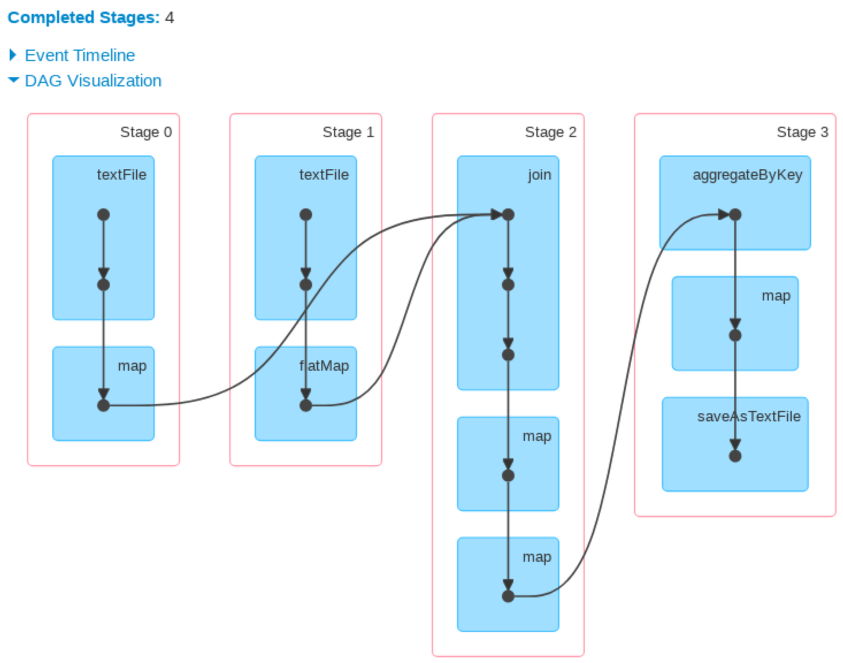

Figure 1: DAG of The Sample Java Application

Question 4: Write your own code

Assume we are interested in the most popular movies in a particular category. We measure popularity by the number of ratings a movie receives. Write your own program to find the most popular five movies in a given category, for instance “Documentar”.

You may reuse some functions of the sample program to process the input file. You will need to use a few other operations, for instance, `filter`, `countByKey`, `sortBy`,`top`. Read the API document of Java RDD and Pyspark RDD to find out their respective usage.

For python users, a template notebook file (`TopMovieGivenGenre_Student.ipynb`) as a starting point.

References

- Spark Submitting Applications
  https://spark.apache.org/docs/latest/submitting-applications.html
- Spark Java RDD API
  https://spark.apache.org/docs/latest/api/java/org/apache/spark/api/java/JavaRDD.html

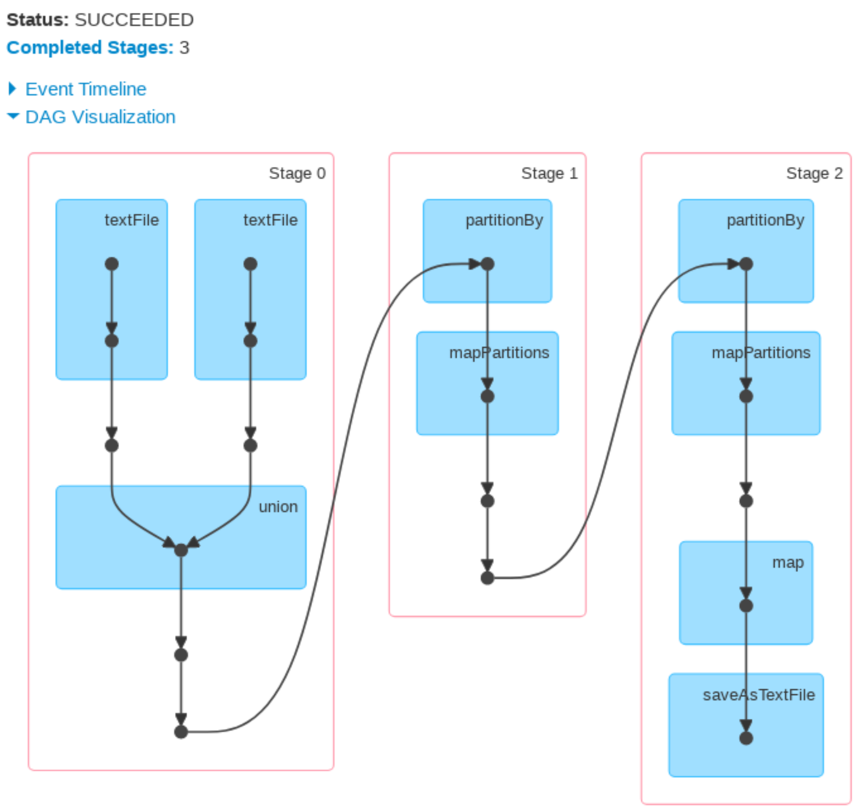
Status: SUCCEEDED
Completed Stages: 3

Figure 2: DAG of The Sample Python Application

- Spark Java PairRDD API
  `https://spark.apache.org/docs/latest/api/java/org/apache/spark/api/java/JavaPairRDD.html`
- Pyspark RDD API
  `https://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD`