THE UNIVERSITY OF SYDNEY

School of Information Technologies
A/Prof Uwe Roehm

COMP5349: Cloud Computing **1.Sem./2017**

Week 8: Spark Practice

04.05.2017

Spark on local Hadoop cluster

Spark is installed on all nodes of our cluster. The version installed is spark-1.6.0. Spark has several cluster deployment modes depending on how resource is managed and allocated for Spark applications. Currently, Spark applications can be managed by YARN, Mesos, or Spark's own standalone resource management system. YARN and standalone are the popular deployment modes for Spark cluster. The standalone mode consists of a master node and many worker nodes. We will be using YARN mode as we've already been using a YARN cluster to execute your MapReduce jobs.

Data Set

All Spark sample applications in this lab use a movie rating data set downloaded from `http://grouplens.org/datasets/movielens/`. The relevant data files have been uploaded to HDFS under `/share/movie`. It contains the `movielens` dataset of 22,000,000 ratings and 580,000 tags applied to 33,000 movies by 240,000 users. The format of each file is described in `http://files.grouplens.org/datasets/movielens/ml-latest-README.html`

Lab exercise

Question 1: Sample Spark Java 8 Application

SSH to one of the slave nodes.

Run the following commands to download a tar file, extract the content, and build a jar file `sparkML.jar` in the current directory:

```
wget https://2017sem1comp5349.blob.core.windows.net/res/sparkML.tgz
tar xzf sparkML.tgz
cd sparkML
ant
```

This sample application computes the average rating of each genre. It is the sample application in week 8'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`. The `join` result is 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.

The application has one job and 4 stages, each with a variable number of tasks.

Run the application using the following command (remember to put your actual login name in the last line)

```
spark-submit  \
  --class ml.MovieLensLarge \
  --master yarn \
  --num-executors 3 \
  sparkML.jar \
  hdfs://soit-hdp-pro-1.ucc.usyd.edu.au:8020/share/movie/ \
  hdfs://soit-hdp-pro-1.ucc.usyd.edu.au:8020/user/<yourLogin>/spark/
```

Inspect the history of this application to see how many jobs, stages and tasks there are. We specify in the submit command to have 3 executors to run the application. This number does not include the extra one to be used for the driver, so there are actually 4 executors in total. How many tasks are there for each executor?

Question 2: Understand lineage graph, job, stage and task

`sparkML.tar` contains the source code of another application: `MLGenreTopMoviesNaive`. This application finds out the top movies per genre based on the number of ratings a movie receives. It runs on the same movie rating data set. Figure 1 shows the lineage graph of operations involved in computing the final RDD.

a) Inspect the source code in your favorite text editor or IDE. The tar file contains a `build.xml` and a `pom.xml` file to give you options of using `ant` or `maven` building tool. If you prefer `ant` and `Eclipse`, you should download `http://web.it.usyd.edu.au/~comp5349/2016/code/spark-assembly.jar` and add it to the project's build path.

b) This application has similar lineage structure to the previous one's. Two new operations `reduceByKey` and `groupByKey` are used. Both operations have wide dependencies between the parent and child RDD.

   The `reduceByKey` operation is applied on rating data to compute the total number of ratings per movie. The `mapToPair` and `reduceByKey` operation sequence closely mimics the classic word count `MapReduce` program.

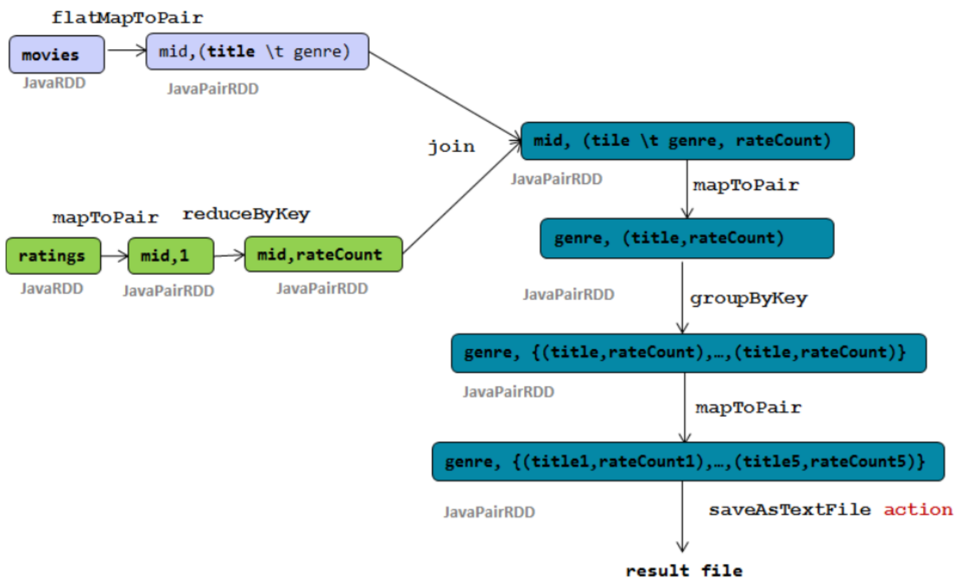

Figure 1: Spark Naive Application Lineage Graph

The `groupByKey` operation is used to group all movies for each genre. A custom type `MovieRatingCount` is used to store movie title and rating count. Note that all custom types used in operations should be serializable. Spark by default uses Java serialization framework and can work with any class you create that implements `java.io.Serializable` interface. Spark can also use the Kryo library `https://github.com/EsotericSoftware/kryo` to serialize objects more quickly.

A `mapToPair` operation is applied on the result RDD of the `groupByKey` operation. It uses Java's `Collections` class to sort all movies in a genre based on its rating count in decending order. The top 5 movies are retured as the final result.

c) Run the application using the following command (remember to put your actual login name in the last line)

```
spark-submit  \
  --class ml.MLGenreTopMoviesNaive \
  --master yarn \
  --num-executors 3 \
  sparkML.jar \
  hdfs://soit-hdp-pro-1.ucc.usyd.edu.au:8020/share/movie/ \
  hdfs://soit-hdp-pro-1.ucc.usyd.edu.au:8020/user/<yourLogin>/spark/
```

Inspect the history of this application to see how many jobs, stages and tasks there are.

d) Draw boxes on the lineage graph to show which operations are included in which stage. Highlight the stage boundary when shuffle happens. Also highlight operations that are executed in pipeline style in one task.

e) Modify the source code to make the `join` operation to have only one partition. Build the jar and re-run the application. Make sure you either delete the previous output directory or specify a different output directory for the second run. Inspect the application history to see how many stages and tasks there are. Draw boxes on the lineage graph to show which operations are included in which stage for this run. Explain the difference in stage and job numbers between the two runs.

Question 3: Write your own Spark program

`MLGenreTopMoviesNaive` class represents a naive solution of finding top five movies per genre from the given data set. The sequence of applying `groupByKey` then `mapToPair` operation to find out the final result is not efficient in terms of network tranfer and computation. The `groupByKey` operation requires shuffling all movie data across the network and the `mapToPair` operation computes a full order of all movies belonging to a genre in order to get the top five.

A general performance tunning rule is to avoid using `groupByKey` operation if you need to do further processing of the value list. In most cases, `reduceByKey` and `aggregateByKey` can achieve the same results with much less data shuffling across the network. Both `reduceByKey` and `aggregateByKey` perform the merging locally on each partition before sending results to a reducer, similarly to a "combiner" in `MapReduce`.

Design and implement a better solution to find top five movies per genre using `aggregateByKey` operation. There is a general `Top N` MapReduce design pattern that does no require total ordering. It is applicable when N is relatively small. It finds local `top N` records from each partition of data in the map phase and uses a single reducer to find global `top N` from all local `top Ns` produced in the map phase. You should design your Spark application following the same principle. You may start with finding top one movie in each genre to get used to the functional parameters of `aggregateByKey` operation.

Spark Python Application

Spark provides APIs in `Scala`, `Java` and `Python`. You can download the Python version of this tutorial code from `http://web.it.usyd.edu.au/~comp5349/2016/code/python/top_5_movies_per_genre_naive.py` and `http://web.it.usyd.edu.au/~comp5349/2016/code/python/average_rating_per_genre.py`. As again, Python version is provided by our tutor Andrian Yang. Application submission instruction is included in the files as well.