THE UNIVERSITY OF SYDNEY

School of Information Technologies
A/Prof Uwe Roehm

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

Week 8: Flink Practice

04.05.2017

Flink on local Hadoop cluster

Apache Flink is installed on all nodes of our cluster. The version installed is flink-1.2.1. Flink has several cluster deployment modes depending on how resource is managed and allocated for Flink applications. Currently, Flink applications can be managed by YARN, Mesos, or Flink's own standalone resource management system. We will be using YARN mode as we've already been using a YARN cluster to execute your MapReduce jobs, so we can take advantage of that.

Although Flink is already installed, there are some environment variables that you need to set to work with it:

```
export PATH=/usr/local/flink/bin:$PATH

# Make sure you're compiling with the correct version of Java
export PATH=/usr/local/jdk1.8.0_40/bin:$PATH
```

Data Set

All Flink 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 Flink 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 `flinkML.jar` in the current directory:

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

# This is the command you should use to compile Java
mvn clean install -Pbuild-jar
```

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 rating data into a tuple with 2 values using `readCsvFile` operations, which also perform the type conversions (i.e. to convert the rating to `Double`). It also converts the movie data into a tuple with 2 values using `flatMap` operation. Next, the two data sets are joined on the common field `movie-id`. The `join` result is mapped to a new data set with just the genre and rating as the movie ID is not required. A `groupBy` and `reduceGroup` operation is applied to this data set to compute the sum and number of ratings per genre, and thus output the average.

Run the application using the following command:

```
flink run -m yarn-cluster -yn 3 \
  -c ml.MovieLensGenreAverageRating \
  target/ml-1.0-SNAPSHOT.jar \
  --movie-dir hdfs:///share/movie
```

`-yn 3` specifies that 3 YARN containers should be provisioned to the Flink program to be used as Task Managers. One additional container is provisioned for the Job Manager.

View the job in the application master while the application is running and try to relate the DAG that's being displayed with the code that's executing.

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

`flinkML.tgz` 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 graph of operations involved in computing the result. Note that you can view the same graph (and zoom in/out) by running the Flink application and viewing the Job's overview.

a) Inspect the source code in your favorite text editor or IDE. The archive file contains `pom.xml` file to be used with `mvn` (Maven) building tool.

   It's not trivial to use Flink with an IDE, however they have provided instructions if you wish to do so: `https://ci.apache.org/projects/flink/flink-docs-release-1.2/internals/ide_setup.html#eclipse`

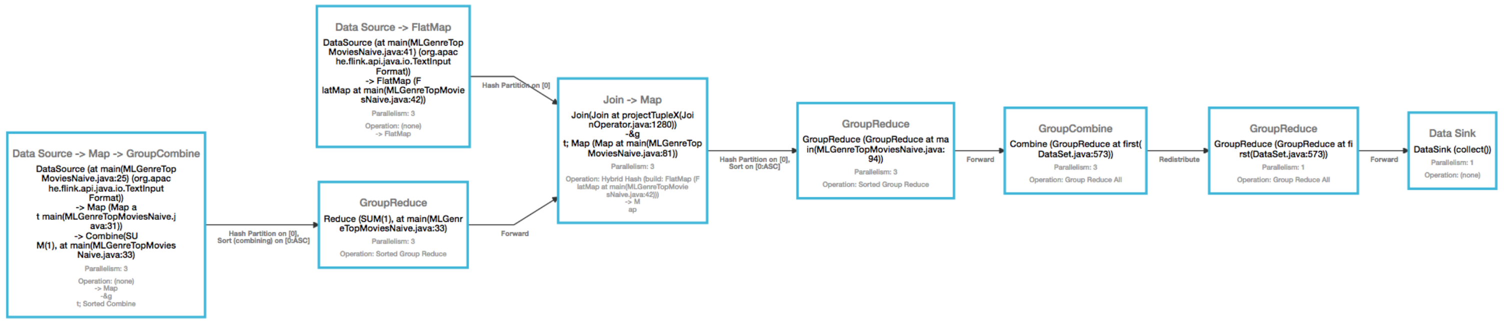
Data Source -> Map -> GroupCombine
DataSource (at main(MLGenreTopMoviesNaive.java:25) (org.apache.flink.api.java.io.TextInputFormat)) -> Map (Map at main(MLGenreTopMoviesNaive.java:31)) -> Combine(SUM(1), at main(MLGenreTopMoviesNaive.java:33))
Parallelism: 3
Operation: (none) -> Map -&gt; Sorted Combine
Hash Partition on [0], Sort (combining) on [0:ASC]
Data Source -> FlatMap
DataSource (at main(MLGenreTopMoviesNaive.java:41) (org.apache.flink.api.java.io.TextInputFormat)) -> FlatMap (FlatMap at main(MLGenreTopMoviesNaive.java:42))
Parallelism: 3
Operation: (none) -> FlatMap
GroupReduce
Reduce (SUM(1), at main(MLGenreTopMoviesNaive.java:33)
Parallelism: 3
Operation: Sorted Group Reduce
Hash Partition on [0]
Forward
Join -> Map
Join(Join at projectTupleX(JoinOperator.java:1280)) -&gt; Map (Map at main(MLGenreTopMoviesNaive.java:81))
Parallelism: 3
Operation: Hybrid Hash (build: FlatMap (FlatMap at main(MLGenreTopMoviesNaive.java:42))) -> Map
Hash Partition on [0], Sort on [0:ASC]
GroupReduce
GroupReduce (GroupReduce at main(MLGenreTopMoviesNaive.java:94))
Parallelism: 3
Operation: Sorted Group Reduce
Forward
GroupCombine
Combine (GroupReduce at first(DataSet.java:573))
Parallelism: 3
Operation: Group Reduce All
Redistribute
GroupReduce
GroupReduce (GroupReduce at first(DataSet.java:573))
Parallelism: 1
Operation: Group Reduce All
Forward
Data Sink
DataSink (collect())
Parallelism: 1
Operation: (none)

Figure 1: Flink Naive Application Graph

b) The main differences in this application and the previous is the `sum` aggregation performed early on with the ratings, the usage of a `MovieRatingCount` class to hold some data together, and the implementation of the top 5 in the `reduceGroup` near the end.

   Try to understand the flow of the data as it comes from the input files, is grouped together, movied into the `MovieRatingCount` class and followed through to completion.

c) Run the application using the following command:

```
flink run -m yarn-cluster -yn 3 \
  -c ml.MLGenreTopMoviesNaive \
  target/ml-1.0-SNAPSHOT.jar \
  --movie-dir hdfs:///share/movie
```

   Inspect the application as it's running to view it's execution graph, subtasks, task managers and such.

Question 3: Write your own Flink program

`MLGenreTopMoviesNaive` class represents a naive solution of finding top five movies per genre from the given data set. The implementation of the top 5 within the `reduceGroup` to find out the final result is not efficient in terms of memory usage and perhaps other factors. Additionally, there are numerous inefficiencies throughout the program that could be improved with an understanding of the available transformations.

Use Flink's documentation to try and improve the implementation of the program.