UNIVERSITY OF TARTU
Institute of Computer Science

(/)

Basics of Cloud Computing 2018/19 spring

Cus

- Main (https://courses.cs.ut.ee/2019/cloud/spring/Main/HomePage)
- Lectures (https://courses.cs.ut.ee/2019/cloud/spring/Main/Lectures)
- Practicals (https://courses.cs.ut.ee/2019/cloud/spring/Main/Practicals)
- Results (https://courses.cs.ut.ee/2019/cloud/spring/Main/Results)
- Submit Homework (https://courses.cs.ut.ee/2019/cloud/spring/Main/Homework)

Practice 5 - Designing MapReduce Applications

Goal of this exercise is to analyse an open dataset using MapReduce. You will create a new MapReduce application, download an example data set from open data repository and perform a series of data processing tasks.

References

Referred documents and web sites contain supportive information for the practice.

- MapReduce tutorial: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html (https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html)
- Hadoop API: http://hadoop.apache.org/docs/stable/api/ (http://hadoop.apache.org/docs/stable/api/)
- Hadoop wiki https://hadoop.apache.org/ (https://hadoop.apache.org/)

Dataset Description

The dataset that we will analyze using MapReduce is taken from UCI Machine Learning Repository and contains census information from a set of Adults living in USA. The original goal of the dataset was to use it for predicting whether the income of a person exceeds $50K per year.

- Name: UCI Adult dataset
- Location: http://archive.ics.uci.edu/ml/datasets/Adult (http://archive.ics.uci.edu/ml/datasets/Adult)
- Dataset attributes (column names) are:

```
0. age
1. workclass
2. fnlwgt
3. education
4. education-num
5. marital-status
6. occupation
7. relationship
8. race
9. sex
10. capital-gain
11. capital-loss
12. hours-per-week
13. native-country
14. Classification
```

Download the `adult.data` and `adult.test` files from the dataset **Data Folder**.

Exercise 5.1. Create a new MapReduce application

Goal of this exercise is to analyse UCI Adult dataset (http://archive.ics.uci.edu/ml/datasets/Adult) using MapReduce.

- Use your old IntelliJ MapReduce project or make a new one like in the last practice session
- Create folder named "input" inside your IntelliJ project
  - Download the `adult.data` and `adult.test` files from the dataset **Data Folder** page and move them inside the input folder.
    - You may have to delete 2 last empty lines in the file - or you will get parsing errors
- Create a new `ee.ut.cs.ddpc` package inside the project.
  - Do not use the old `org.apache.hadoop.examples` package as you will otherwise have issues later when trying to execute your program in the cluster.
- Copy the `WordCount.java` example from the last practice session into the `ee.ut.cs.ddpc` package folder and rename it as `Lab5MRApp.java` .
- Modify the copied class and change the Map output, Reduce input & output types into **Text** type.
  - **In the Mapper class:**

```
34 public class WordCount {
35
36   public static class TokenizerMapper
37        extends Mapper<Object, Text, Text, Text>{
38
39
```

  - **In the Reducer class:**

```
56   public static class IntSumReducer extends Reducer<Text,Text,Text,Text> {
57
58     public void reduce(Text key, Iterable<Text> values, Context context)
59            throws IOException, InterruptedException {
```

  - **In the main() method:**

```
83      job.setReducerClass(IntSumReducer.class);
84
85      job.setOutputKeyClass(Text.class);
86      job.setOutputValueClass(Text.class);
87
88      for (int i = 0; i < otherArgs.length - 1; ++i) {
```

- Also add the following lines to the `main()` method to reconfigure the map output key and value types to be `Text` :

```
job.setMapOutputKeyClass(Text.class);
job.setMapOutputValueClass(Text.class);
```

- NB! Remove the Combiner configuration statement in the main method. The Reduce function that you will create can not be used directly as a Combiner this time.
  - `//job.setCombinerClass(IntSumReducer.class);`

Exercise 5.2. Data analytics using MapReduce

Modify the application to perform the following tasks on the UCI Adult data set:

1. **For each native-country, calculate the average hours-per-week**
   - **Map method:**
     - Input **key-value pair** is: `(line_nr, line)`, where line is a single line from the csv file, which contains 14 comma separated values.
     - You should split the input line by commas and output native country as a **key** and hours-per-week as a **value**: `(native-country, hours-per-week)`
   - **In Reduce:**
     - Input is: `(native-country, [hours-per-week])`
     - Input **key** is unique native-country and **value** is a list-like iteratable object of all 'hours-per-week' **values** from this native-country.
     - Compute the average of all hours-per-week values.
     - Output should be `(native-country, avg_val)`
2. **In addition to average value, also find minimum and maximum values**
   - Instead of writing out a single **value** using `context.write()`, Reduce function should compute and output multiple **values**.
   - Output should be written as either:
     - 3 different **key-value** pairs, using 3 `context.write()` calls:
       - `("native-country,MIN", min_val)`
       - `("native-country,AVG", avg_val)`
       - `("native-country,MAX", max_val)`
     - Or as a single KeyValue pair where the value object contains multiple results: `(native-country, "min_val, avg_val, max_val")`.
3. **Perform the previous analysis for each unique native-country AND workclass pair.**
   - Use the Map output **key** to modify by which attributed the data is grouped by when it "arrives" in the Reduce method
   - You can create a combined **key**. For example: `("native-country,workclass", value)`
4. **And finally, instead of a specific column: hours-per-week, perform the analysis for every numerical column**
   - Create a loop inside the Map method that outputs a **(key, value)** pair for every column you want to process separately.
   - Use a unique **key** for each column to make sure they are grouped separately in Reduce function. Simple way to achieve this is to add the name of the numerical column as another component into the combined **key**.

Exercise 5.3. Passing user defined parameters to Map or Reduce tasks.

Now, lets change the MapReduce program to analyse entries only from a specific user defined **occupation** and ask users to provide the **occupation** value as the third argument to the program. The arguments to the program should become: `input_folder output_folder occupation`

Additional parameters can not be directly passed to the Map and Reduce tasks because we are not executing them directly when the application is running in a distributed manner (in a cluster). Instead, they should be written to the MapReduce **Job Configuration** before running the job. The **Job Configuration** object will be made available for the Map and Reduce processes through `context` and configuration parameter values can be extracted from there.

- Modify the run configuration of your new application to contain three arguments: `input_folder output_folder occupation` instead of just two
- Change how the `main()` method parses the input arguments as input and output folders. Currently it uses the last program argument as the output folder and all previous arguments as input folders to allow you to define multiple input paths. Instead change it to use first argument as input folder and second as output folder:
  - ```
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    ```
- Value of user defined `occupation` should be read from the program arguments in the `main()` method and written into the MapReduce job configuration so that it is passed along to all the Map and Reduce processes.
  - You can use the `conf.set(name, value);` method to write the user defined `occupation` value into the MapReduce configuration.
- Inside the **Map** method, use the `context.getConfiguration().get(name, default_value);` method to access the previously defined `occupation` value.
- Use the `occupation` value to filter out all input file entries that have the wrong **occupation** field/column value.

Deliverables

- MapReduce application source code.
- Output files (part-r-0000*) of your job.
- Answer the following questions:
  1. What are the respective advantages/disadvantages of using either:
     - separate **key-value** pairs to output the results
     - or using comma separated combined values

Task	lab 5
Current submission	ZIP (/course-2065/submissions/get/5/B91541_zip/B91541_4303_1) (20:24 17.03.2019)
	If your homework consists of multiple files (for example HTML + CSS + JS) it should be archived before submitting.
File	Choose File no file selected
Comment	I have uploaded my submission!
    ```

Submit

Solutions to common issues and errors.

- If you get an error about `ClassNotFoundException`.
  - Make sure you have selected the `Include dependencies with the "Provided" scope` option in the Run Configuration of your application.
- Make sure that you have removed the Combiner configuration statement in the main method. The Reduce function that you will create can not be used directly as a Combiner this time.
- If you get an error about running MapReduce example tests when creating the jar file, you can try deleting the tests (Test classes inside /test/java/ folder inside your project), run maven clean and run maven package again.
- NB! You will run into problems while running Hadoop MapReduce programs if your Windows username includes a space character. We would suggest that you create an alternative user in your computer without any white space characters and complete the exercise under that user.
  - You should also avoid spaces in folder paths.

NB! If you run into any issues not covered here then contact the lab supervisor.