THE UNIVERSITY OF SYDNEY

School of Information Technologies
Dr Ying Zhou and A/Prof. Uwe Roehm

COMP5349: Cloud Computing **1.Sem./2015**

Assignment 1: Hadoop MapReduce Programming

Group Work: 20% **02.04.2015**

Introduction

In this assignment, you are asked to write Hadoop jobs to reorganize and analyze a data set that originates from flickr.com. The analysis is separated into two workloads, each deals with a slightly different problem. You will practice basic Hadoop programming features and also observe the Hadoop performance under various conditions. The data set is the same as used in the labs.

Data Set Description

The main input data can be found in the HDFS directory `/share/large` in our Hadoop cluster. The directory has several text files, all with names like `n0*.txt`. All of them have the same format, with each line representing a photo record. These data files had been downloaded from the Flickr website using their REST API.

Each photo record contains the following information delimited by `tab` (\t):

photo-id \t owner \t tags \t date-taken \t place-id \t accuracy

An additional input file `place.txt` can be found in the HDFS directory `/share`. This file contains place data: each line represents a place record with the following information delimited by `tab` (\t):

place-id \t woeid \t latitude \t longitude \t place-name \t place-type-id \t place-url

Both `place-id` (Flickr place identifier) and `woeid` (Where-On-Earth identifier) can uniquely identify a place on earth. A photo record only contains `place-id` information. Details about a place are stored in `place.txt`. Places can be specified at different levels, indicated by `place-type-id`. The valid `place-type-id`s in decreasing specificity are:

- **22**: neighbourhood (roughly equivalent to place of interest)
- **7**: locality (roughly equivalent to city)
- **8**: region (roughly equivalent to state/province)
- **12**: country
- **29**: continent

`place-name` and `place-url` are alternative ways of naming a place. They both carry upper level place information. For instance, a neighbourhood level place "Coogee" has a `place-url` "/Australia/NSW/Sydney/Coogee" and a `place-name` "Coogee, Sydney, NSW, AU, Australia".

Note that the data set may not have a consistent way of using acronyms in `place-name` and `place-url`. For instance, it may use either "NSW" or "New South Wales" to refer to the same region level place. You do not have to handle the mapping of acronyms to full names in this assignment. It is OK for you to treat them as different places.

The size of all input files is about 10GB. Carefully design the map/reduce tasks to minimize the number of scans of the original data set.

Analysis Tasks

We are interested in getting some simple summary information of the given data set. You are asked to program two separate MapReduce jobs to find out the following:

1. The **top 50 locality-level places** based on the number of photos taken in each locality, plus the corresponding **top-10 photo tags** at each place. Note that photos can be geotagged at various neighbourhood levels that belong to the same locality level. For instance, the locality level place name "Sydney, NSW, Australia" has 500+ neighbourhood level geotags associated with it such as "Coogee, Sydney, NSW, AU, Australia", "Circular Quay, Sydney, NSW, AU, Australia". You need to count all photos geotagged at locality level and neighbourhood levels belonging to that locality. For each top locality level place, find out the top 10 tags based on frequency of tags assigned to photos taken in this place. List these top-10 tags in descending order of frequency, and if two tags have the same frequency, then order them alphabetically. The output should be a list of the format:

   PlaceName \t numberOfPhotos \t (tag:freq)+

2. The **top 10 locality-level places** for all countries appearing in the data set. The locality level places should be ranked by the number of unique users having photos taken there. Order the list of placeNames first in descending order of popularity, then alphabetically for places with same popularity. Similar as analysis task 1, you need to count all users with photos geotagged at locality level and neighbourhood levels belonging to that locality. For each top **locality** place, also find out the top **neighbourhood** level place with the highest the number of unique users having photos on it. The output should be a list of the format:

   CountryName \t (placeName:NumOfUsers)+ \t (neighbourhoodName:NumOfUsers)+

Implementation Requirements

Your solution will be marked according to correctness, job design, and code quality. In particular we are interested in a cleanly written, well-design MapReduce solution that consequently applies the MapReduce principles and hence provides good scalability for large clusters and data sets. Some tips:

1. Implement your jobs using small, independent, and well-defined tasks that build on each other.
   While it certainly is is possible to implement the above tasks using a sequential program, this would require loading all data in memory and organize it as a Hashmap or Treemap structure. You are asked to avoid using this type of approach either for the whole data set or for the partial data set. Instead, you should follow the divide-and-conquer principles of MapReduce framework to achieve most of the functions. Both workloads would require more than one jobs to produce the final result.

2. Always test your code using a small data set before applying it to the large ones.

3. Do not copy the entire input data to your HDFS or Linux home directory.

Deliverables

There are two deliverables: **source code** and brief **report** (up to 4 pages). Both are due in Week 7. Please submit the source code and a soft copy of performance report as a zip or tar file in eLearning. You need to demo your implementation in Week 7 using a small data set. Please also submit a hard copy of your performance report during Week 7's demo.

The report should contain the following sections:

Cover Sheet. A signed individual assignment cover sheet
(http://sydney.edu.au/engineering/it/current_students/undergrad/policies/assignment_sheet_group.pdf)

MapReduce Job Design. Basic design/implementation information about the MR jobs. For each job, briefly describe the purpose and show a pseudo code for the map/reduce task. Describe your combiner function as well if you have one.

Performance Evaluation. In this section, analyse the execution time of the two MR workloads under various input sizes: 1G, 2G, 4G, 6G and 10G. The desirable input sizes can be achieved by a combination of various input files. Plot the execution time (total, map, reduce) against input data sizes on a chart. If you have more than one job in each workload, only plot those jobs that read the original input data. Interpret any trends or interesting patterns you observe. The execution time information can be obtained programmatically or from the job history WebUI.

Appendix. List the HDFS location of your final output files for the various executions.

Assessment Criteria

Your team's final grade will depend on how sophisticated and correct your solution is. The main criteria are: Implementation correctness of Task 1 and Task 2, Job Design and Efficiency, Job Description, and Evaluation. Each team member must also be able to answer questions to any part of the code during the demo to the tutor. You can find the detailed marking rubric in eLearning.

Grade Descriptors

High Distinction 85 – 100%	The solution demonstrates an outstanding understanding of the Hadoop MapReduce computational model, including a discussion of some design problems and a thorough, well-presented performance evaluation with a very good interpretation of the results; the submission is a complete solution of both data analysis tasks with excellent quality of all deliverables.
Distinction 75 – 84%	Thorough understanding of programming with Hadoop MapReduce; medium level of original thinking and critical discussion of at least one design problem; the submission is a complete solution of all data analysis tasks and includes a thorough, well-presented performance evaluation with a good interpretation of the results.
Credit 65 – 74%	Good understanding of programming with Hadoop MapReduce; complete solution of all data analysis tasks with average code quality; includes a performance evaluation with useful graphs and some general description of the results; average quality of deliverables.
Pass 50 – 64%	The submitted code compiles and runs and executes both analysis tasks with at most a few minor mistakes; documentation with some form of evaluation results provided.
Fail below 50%	Falls short of the basic requirements for a Pass.

Academic Honesty

IMPORTANT: Policy relating to Academic Dishonesty and Plagiarism.
All teams must declare that the work is original and not plagiarised from the work of others. In assessing a piece of submitted work, the School of IT may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program. A copy of the assignment may be maintained by the service or the School of IT for the purpose of future plagiarism checking.