ISIT312 Big Data Management

SIM S2 2020

Assignment 1

All files left on Moodle in a state "Draft(not submitted)" will not be evaluated. Please refer to the submission dropbox on Moodle for the submission due date and time.

This assignment contributes to **10%** of the total evaluation in the subject. This assignment consists of 3 tasks. Specification of each task starts from a new page.

It is a requirement that all Laboratory and Assignment tasks in this subject must be solved individually without any cooperation with the other students. If you have any doubts, questions, etc. please consult your lecturer or tutor during lab classes or office hours. Plagiarism will result in a FAIL grade being recorded for that assessment task.

Deliverables

You must submit three PDF files for three tasks (one for each task) and a Java source file for Task 3.

Each PDF file must be no more than 5 pages. The answers must be presented in a readable manner that facilitates evaluation. Poor presentation will result in a reduction of marks.

The Java sourcecode must be correct and compilable. The environment for Task 1 and Task 3 is the BigDataVM virtual machine in ISIT312.

Task 1. PutCombine Application (3 marks)

This task is based on the sourcecode PutCombine.java, and the data in FilesToBeMerged.zip (available on the Moodle site).

The PutCombine application extends Hadoop's own functionality. The motivation for this application comes when we want to analyse fragmented files such as logs from web servers. We can copy each file into HDFS, but in general, Hadoop works more effectively with a single large file rather than a number of smaller ones. Besides, for analytics purposes we think of all the data as one big file, even though it spreads over multiple files as an incidental result of the physical infrastructure that creates the data.

One solution is to merge all the files first and then upload the combined file into HDFS. Unfortunately, the file merging will require a lot of disk space in the local machine. It would be much easier if we could merge all the files on the fly as we upload them into HDFS.

What we need is, therefore, a "put-and-combine"-type of operation. Hadoop's command line utilities include a `getmerge` command for merging a number of HDFS files before copying them onto the local machine. What we're looking for is the exact opposite, which is not available in Hadoop's file utilities. The attached sourcecode of PutCombine is a Java application that fulfils this purpose.

i. Compare PutCombine with the FileSystemPut and FileSystemPutAlt applications in the lecture note and describe the difference. You must link the difference to the sourcecode. (1.5 mark)

ii. Compile PutCombine.java and create a jar file, and use to it to upload and merge the files in FilesToBeMerged.zip (unzip this file first). Produce a report as a PDF file that includes all the commands that you enter to a Zeppelin notebook or the Terminal as well as brief explanations of those commands (e.g., the purpose of each command). (1.5 mark)

Task 2. MapReduce Model (3 marks)

(a) (1.5 marks) Suppose two sets of English letters "a" to "d" are stored in two nodes (**A1** and **A2**):

- *D1* = {"a", "b", "c", "b", "c", "d", "a"} in Node **A1**,
- *D2* = {"a", "a", "a", "d", "d", "c"} in Node **A2**.

There is a MapReduce job that processes the sets *D1* and *D2*. This job is a "word-count" application, namely, it counts the total number of English letter(s) in both nodes.

In this job, two Map tasks run in Node **A1** and Node **A2**, and one Reduce task runs in another node, say, Node **A3**.

Describe the key-value data transferred from Node **A1** and Node **A2** to Node **A3**, respectively, *with and without a Combiner*, respectively.

Explain how a Combiner can improve a MapReduce job in this example.

(b) (1.5 marks) The following table (named X) has a column of keys and a column of values:

key	value
k1	1
k1	2
k2	3
k2	4

Explain how to implement the following SQL-like query in a MapReduce model:

"SELECT key, SUM(value) FROM X GROUBBY key"

You need to specify the key-value data in the input and output of the Map and Reduce stages.

Produce a report as a PDF file that documents your solutions to both questions above.

Task 3. Average Patent Claim Applications (4 marks)

This task is process the dataset apat63_99.txt (in the dataset folder in the Desktop of the VM). This dataset contains information about almost 3 million U.S. patents granted between January 1963 and December 1999. See http://www.nber.org/patents/ for more information.

The following table describes (some) meta-information about this data set.

Attribute Name	Description
PATENT	Patent number
GYEAR	Grant year
GDATA	Grant date, given as the number of days elapsed since January 1, 1960
APPYEAR	Application year (available only for patents granted since 1967)
COUNTRY	Country of first inventor
POSSTATE	State of first inventory (if country is U.S.)
ASSIGNEE	Numeric identifier for assignee (i.e., patent owner)
ASSCODE	One-digit (1-9) assignee type. (The assignee type includes U.S. individual, U.S. government, U.S. organization, non-U.S. individual, etc.)
CLAIMS	Number of claims (available only for patents granted since 1975)
NCLASS	3-digit main patent class

Develop the a MapReduce application AvgClaimsByYear.java in Java which

- computes *the average number of claims per patent by year*;
- includes ToolRunner and a Partitioner;
- sorts the outcome into four groups according to the year, i.e., (1) before 1970, (2) 1970-1979, (3) 1980-1989, and (4)1990-1999.

You should transfer the apat63_99.txt file to HDFS and run the application to process it.

Produce a report as a PDF file that includes the commands that you use, a brief explanation of each command and the output of your application.

Also submit the sourcecode of your application.