Extreme Computing
Second assignment

Jozef Mokrý, Kenneth Heafield
(partly based on an assignment by Michail Basios and Stratis Viglas)

Due Monday 20 November at 4 pm. All questions should go on Piazza

`https://piazza.com/class/j7m5dr4ns4dta`

in a "hw2" folder. If your question or reply reveals an answer, ask privately.

The assignment is worth 25 marks in total. We mark for correctness, efficiency, and proper use of tools. That includes, but is not limited to, running your solution in a scalable way. In many cases, it is possible to get the right answer with a less scalable implementation, in which case you may receive partial marks. Most marks are allocated to efficiency.

Tasks 1 through 4, inclusive, require you to use `HDFS` and `Hadoop`. As a hint, all of the solutions take less than an hour to run. You should kill any jobs that take longer:

```
mapred job -kill $jobid
```

We may use a bot to kill long-running jobs so that everyone can run on the cluster.

This assignment is divided into three parts and eight tasks. The first part deals with using MapReduce for building an inverted index. The second part deals with parsing and analysing data from StackOverflow. The last part deals with randomized and approximate algorithms. In all tasks, except 7, it is ok to use the output of a previous task as input to subsequent tasks.

You should use the teaching Hadoop Cluster[1] and any programming language you want. If you are logging in from outside Informatics, first

```
ssh s12345678@student.ssh.inf.ed.ac.uk
```

or use the Informatics VPN:

`http://computing.help.inf.ed.ac.uk/openvpn`

Once inside Informatics, connect to a random machine in the cluster:

```
ssh scutter$(seq -w 1 12 | shuf -n 1)
```

For each part there are different data sets (on HDFS). There are two versions of each input file that should be used in your program, a small one and a larger one. The **small** version file is for developing and testing your code; when you are happy that it works, you should use the **large** version.

Reference outputs for small data are not provided in this assignment. The files (on HDFS) that you will need for each part follows:

Part 1

Folder	Number of files
`/data/assignments/ex2/part1/small/`	5
`/data/assignments/ex2/part1/large/`	17

[1] You can run your own, but we won't support it and you need to copy output back to the teaching one.

Part 2

File	Number of lines
`/data/assignments/ex2/part2/stackSmall.txt`	981
`/data/assignments/ex2/part2/stackLarge.txt`	468273

Part 3

On HDFS (useful for Task 4):

File	Lines	Words
`/data/assignments/ex2/part3/webSmall.txt`	95350	6311838
`/data/assignments/ex2/part3/webLarge.txt`	1897987	123588873

On the UNIX file system (useful for Tasks 5–7):

File	Lines	Words
`/afs/inf.ed.ac.uk/group/teaching/exc/ex2/part3/webSmall.txt`	95350	6311838
`/afs/inf.ed.ac.uk/group/teaching/exc/ex2/part3/webLarge.txt`	1897987	123588873

Part 4

On the UNIX file system:

File	Lines	Words
`/afs/inf.ed.ac.uk/group/teaching/exc/ex2/part4/queriesSmall.txt`	10000	10000
`/afs/inf.ed.ac.uk/group/teaching/exc/ex2/part4/queriesLarge.txt`	1000000	1000000

1 Tasks

1.1 Inverted index with MapReduce

Task 1

◁ **Task**

Use the files in the folder `/data/assignments/ex2/task1/large/` as input and produce an inverted index using MapReduce. For instance, given the following documents: **(3 marks)**

```
d1.txt:  cat dog cat fox
d2.txt:  cat bear cat cat fox
d3.txt:  fox wolf dog
```

you should build the following full inverted index.

```
bear :  1 :  {(d2.txt,1)}
cat :  2 :  {(d1.txt, 2), (d2.txt, 3)}
dog :  2 :  {(d1.txt, 1), (d3.txt, 1)}
fox :  3 :  {(d1.txt, 1), (d2.txt, 1), (d3.txt, 1)}
wolf :  1 :  {(d3.txt,1)}
```

For each term (anything separated by spaces), there is a single record consisting of a number and a list of what are termed postings; the colon character (‘:’) is used to delimit the fields of each record. There are also colons in the document, but just leave them as-is. The first field is a number that represents the number of documents that contain the term. Then a list of postings follows where each posting is a pair consisting of the document name and the frequency of the word in that specific document. Note that terms are sorted alphabetically and also that the items inside lists are also sorted alphabetically by document identifier. For example, the following line:

```
cat :  2 :  {(d1.txt, 2), (d2.txt, 3)}
```

indicates that the word `cat` appears in two documents, two times in document `d1.txt` and three times in document `d2.txt`. Instead of `hdfs://scutter01.inf.ed.ac.uk:8020/data/incredibly/long/path/d1.txt`, just use `d1.txt`.

To get the full path to the input file in Hadoop streaming, read the `mapreduce_map_input_file` environment variable. In Python, that's `os.environ["mapreduce_map_input_file"]`. Use a single space between elements in your inverted index (not a tab, and not double-spaces)

1.2 Parsing StackOverflow

For tasks 2 and 3, you will use a dataset from StackOverflow (`stackLarge.txt`) and extract specific pieces of information. Initially, you should understand the format of the dataset, next you will need to parse each post, and finally you will need to implement your MapReduce workflows. Use MapReduce for tasks 2 and 3.

The dataset contains a number of post records, one record per line. Each record consists of comma-separated key-value pairs, which are then pointlessly wrapped in an XML element. That is, a record looks like:

```
<row attribute1=value1, attribute2=value2, ..., attributeN=valueN />
```

Each record has its own identifier stored in a field named `Id` and a type, indicated by the value of a field `PostTypeId`. If the value of `PostTypeId` is 1, than the post refers to a question, otherwise is the value of `PostTypeId` is 2 the post refers to an answer.

An example of a question post is:

```
<row Id="2155" PostTypeId="1" AcceptedAnswerId="2928"
CreationDate="2008-08-05T12:13:40.640" Score="25" ViewCount="17551"
Body="The question content" OwnerUserId="371" LastEditorUserId="2134"
LastEditorDisplayName="stackoverflowGuy"
LastEditDate="2008-08-23T18:09:09.777"
LastActivityDate="2013-09-19T15:39:43.160" Title="How do I?"
Tags="&lt;asp.net&gt;" AnswerCount="6" CommentCount="0"
 FavoriteCount="12" />
```

You will need to parse the record into a structure that will allow access to the value of each attribute by name. In this example, `Id="2155"` represents the unique identifier given to the post; `PostTypeId="1"` means that this post is a question; `AcceptedAnswerId="2928"` means that the accepted answer from the user for this query is the answer with `Id="2928"`; and so on.

An example of a post that corresponds to an answer is:

```
<row Id="659891" PostTypeId="2" ParentId="659089"
CreationDate="2009-03-18T20:07:44.843" Score="1"
Body="Description of the problem"
OwnerUserId="45756" OwnerDisplayName="terminator"
LastActivityDate="2009-03-18T20:07:44.843" CommentCount="0" />
```

The attribute-value pair `Id="659891"` represents the unique identifier given to this post. The value for `ParentId` represents the identifier of the question this answer applies to, and the value for `OwnerUserId` represents the user who wrote the answer for this question.

You do not need to know exactly what each attribute means, but you will need to be able to access the value of each attribute, given a record. You will need to write a parser for each record and then answer the following questions. For each question (ie task 2 and task 3) we give the expected output format; lines beginning with '#' are only descriptive comments and you are not required to print them; you only need to print the actual output values.

Task 2

◁ **Task**

Which are the 10 most popular questions according to their view counts (attribute `ViewCount` in a question post)? Output Format: **(2 marks)**

```
#Count Id
17551 659891
2131 659892
1782 314159
...
```

The columns are count and question id. Ties may be broken arbitrarily. Sort in decreasing order of count. Use a single space between count and id.

Task 3

◁ Task

Who was the user that answered the most questions and what were the `Ids` of these questions? Output Format: **(3 marks)**

```
# OwnerUserId   ->   PostId, PostId, PostId, ...
1342            ->   23, 26, 531
```

Use a single space in your actual output: "`1342 -> 23, 26, 531`".

1.3 Sifting Web Data

Task 4

◁ Task

In lectures, you saw how to sample uniformly a single line from a stream of lines efficiently on a single machine. For large data, running reservoir sampling on a single machine would take too long. Implement a MapReduce version of reservoir sampling which uniformly samples only a *single* line and uses MapReduce to do so. Use your implementation to sample a single line from the file `webLarge.txt`. Your output should contain only a single line. Do not implement the method that generates a random number for each line then takes the largest one. **(3 marks)**

Task 5

◁ Task

Extend the basic version of reservoir sampling such that it can sample multiple lines uniformly without replacement and run on a *single* machine (i.e. no MapReduce). This means that if we want to sample k lines from a file with n lines in total, then each of the $\binom{n}{k}$ possible outcomes has equal probability. Implement a program that will sample 100 lines from the file `webLarge.txt`. Run your program locally (not as a MapReduce job). **(2 marks)**

Task 6

◁ Task

Make your own implementation of a Bloom filter. We leave the choice of a hashing function up to you. Write a program that uses your implementation of Bloom filter to *approximately* de-duplicate the lines in the file `webLarge.txt`. The output of an approximate de-duplication contains no duplicate lines, but some lines from the input might not appear at all in the output. You should think carefully about the number of hashing functions and the size of the Bloom filter you use. The probability that a line (and its duplicates) from the input does not appear at all in the output should be less than 1%. You can assume that your hashing functions produce every value equally likely. When choosing the size of your Bloom filter and number of hashing functions, you should assume the worst case in which all lines are unique. The number of lines should be a command-line parameter to your program. **(4 marks)**

Task 7

◁ Task

Use GNU parallel to build a Bloom filter on all of `webLarge.txt` in parallel. The final Bloom filter should be the same as the one you built in the previous task. This task is exempt from submitting 20 lines of output, but you should still submit your code and shell script. Put your final Bloom filter, in raw form, on HDFS per the submission instructions. You must use GNU parallel in non-trivial way(s) to receive marks for this task. **(5 marks)**

1.4 Mining Query Logs

Task 8 ◁ **Task**

Imagine you are Google and you want to know which search queries (if any) form at least 1% of all queries in total. In the file `queriesLarge.txt` each line is a hash of a query and queries occurred in the order as listed in the file. Implement the *lossy counting* algorithm and run it on the file `queriesLarge.txt`. Your output should contain all queries that form at least 1% of all queries and no query that formed less than 0.9% of all queries. **(3 marks)**

2 Submission

To submit your work, please do the following:

1. Make sure that you store the output for ALL tasks in HDFS. For tasks which did not use MapReduce you can use the command:

   ```
   hdfs dfs -copyFromLocal src dst
   ```

   Please store the output for the Xth task in the folder: `/user/sXXXXXXX/assignment2/taskX` (replace `sXXXXXXX` with your student number). This way we can easily check whether your output is correct. When marking, we will only be interested in the output for the **large** version inputs. Do not put the output for the small version inputs in the same folder as it can be mistaken for your output for the large version. **Do not delete these files from HDFS until you are told it is OK to do so.**

2. To submit your code, please prepare a directory for submission (it can have any name) that will contain one directory for each task and name the directory for the Xth task `taskX`. Inside each task folder, please include:

   - For all tasks except Task 7, one file with `.out` extension. This file should contain the first 20 lines of your output. If you used MapReduce to solve the task and Hadoop split your output into multiple files, please use the first twenty lines from the first output file (usually called `part-00000`). You can use the command:

     ```
     hdfs dfs -cat filename | head -20
     ```

     for showing just the first 20 lines of the file stored in HDFS. If the whole output has less then 20 lines together, then your `*.out` file should contain the whole output.
   - Files with your code. This includes all code that you wrote to solve the task. **Do not submit a Word document or a PDF file—only submit a plain text file**. It is not required, but it will make marking easier if the file name with mapper, combiner, reducer code starts with `mapper`, `combiner` and `reducer` respectively.
   - Include one file with `.sh` extension. It could execute your MapReduce job(s), run GNU parallel, or just be a trivial invocation of your code in another language. You can see an example below:

     ```
     submissionFolder/
     -- task1/
        -- run.sh
        -- output.out
        -- mapper.py
        -- mapper2.py
        -- reducer.py
        -- reducer2.py
        -- combiner.py
        -- combiner2.py
     -- task2/
     ```

```
   -- run.sh
   -- output.out
...
```

This is an artificial example, it does not mean that you need to code in python or that you need two MapReduce jobs with combiners to solve the first task.

Once you have prepared your submission directory with the specified structure, please run

```
/afs/inf.ed.ac.uk/group/teaching/exc/submit-assignment2.sh path/to/submission_folder
```

This will give you friendly warnings to help you spot if a folder (e.g. `task8`) or a file is missing. You will be asked whether you want to submit your submission directory (regardless of whether any warnings were given).

The late policy is at http://www.inf.ed.ac.uk/teaching/courses/exc/assignment.html.