Extreme Computing
Second assignment

Based on an Assignment by Michail Basios and Stratis Viglas

Questions About The Assignment?

We answer frequent questions at
`http://www.inf.ed.ac.uk/teaching/courses/exc/assignment.html` .
Contact **one** of us:

Nikolay Bogoychev <n.bogoych at ed>
Rafael Karampatsis <R.M.Karampatsis at sms.ed>
Matt Pugh <s1459650 at staffmail.ed>
Kenneth Heafield <kheafiel at inf>

In this assignment you will address real-world cases where MapReduce can be used. Initially, you will deal with a problem related to information retrieval: you will build an inverted index and calculate the so termed tf-idf values for a group of terms. Next, you will analyse an HTTP request logs from a NASA Space Center server and extract information that may be useful to the Website administrator. Lastly, you will use a dataset from StackOverflow that contains user questions and answers and you will extract specific information from it. For all tasks, you should use the teaching Hadoop Cluster and any programming language you want.

For each part there are different data sets (on HDFS). Similarly to the first assignment, there are two versions of each input file that should be used in your program, a small one and a larger one. The small file version is for developing and testing your code; when you are happy that it works, you should use the large version. Unlike Assignment 1, we are not giving you sample output.

The files and folders on HDFS that you will need for each part follow:

Part 1

Folder	Number of files
`/data/assignments/ex2/task1/large/`	17
`/data/assignments/ex2/task1/small/`	5

Part 2

Files	Lines
`/data/assignments/ex2/task2/logsSmall.txt`	(195694)
`/data/assignments/ex2/task2/logsLarge.txt`	(1569898)

Part3

File	Number of lines	Number of words
`/data/assignments/ex2/task3/stackSmall.txt`	1000	106611
`/data/assignments/ex2/task3/stackLarge.txt`	492895	49871095

All your actual results should be produced using the larger files and for all tasks you should use Hadoop MapReduce and HDFS. Moreover, you are allowed to use multiple MapReduce jobs if you think that is necessary.

1 Tasks

1.1 Inverted Index and tf-idf

Task 1

◁ **Task**

Use the files of folder `/data/assignments/ex2/task1/large/` as input and produce an inverted index. For instance, given the following documents: **(4 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 :  {(d2.txt,1)}
```

For each term (word), there is a single record consisting of a number and a list of what are termed postings; the semicolon character (':') is used to delimit the fields of each record. 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`. Either the document name or the document full path can be used for identifying a document.

Task 2

◁ **Task**

Tf-idf is short for *term frequency-inverse document frequency*. It is a numerical statistic that is intended to reflect how important a word is to a document in a collection of documents or *corpus*.[1] For this task you will compute the simple version of tf-idf as follows:

$$\mathsf{tf}(t,d) = f(t,d) \tag{1}$$

where $f(t,d)$ is the frequency (the number of occurrences) of term t inside document d; and

$$\mathsf{idf}(t,D) = \log_{10} \frac{N}{1 + |d \in D : t \in d|} \tag{2}$$

where N is the total number of files in the corpus and $|d \in D : t \in d|$ is the number of documents where the term t appears.

The tf-idf score is then computed as:

$$\mathsf{tf\text{-}idf}(t,d,D) = \mathsf{tf}(t,d) \times \mathsf{idf}(t,D) \tag{3}$$

Calculate the tf-idf score for each term that appears in file:
`/data/assignments/ex2/terms.txt`
using the document found in:
`/data/assignments/ex2/task1/large/d1.txt`.
The final result should have the following format: **(4 marks)**

[1]See also `http://en.wikipedia.org/wiki/Tf%E2%80%93idf` for more information.

```
term1, d1.txt = score
term2, d1.txt = score
term3, d1.txt = score
```

***Note**: See the section "using side information" in Lab 1 as a reminder about loading another file.*

1.2 Log analysis

Task 3

◁ **Task**

For this task you will use as input the file `/data/assignments/ex2/task2/logsLarge.txt` that contains logged activity from a Nasa Space Center Web server.[2] The log records are stored as an ASCII file with one line per request, with the format:

host	*timestamp*	*request*	*reply*	*bytes*

and the following semantics:

- *host* is the address of the host making the request. The address may be a hostname when possible, or the IP address if the hostname could not be looked up.
- *timestamp* in the format "[DAY MON DD HH:MM:SS YYYY]", where DAY is the day of the week, MON is the name of the month, DD is the day of the month, HH:MM:SS is the time of day using a 24-hour clock, and YYYY is the year. The timezone is GMT-0400.
- *request* is escaped within quotes and contains the HTTP command, the file accessed, and the protocol being used.
- *reply* is the HTTP reply code.
- *bytes* is the number of bytes in the reply.

Given the above description the following line:

```
128.159.105.240 - - [03/Aug/1995:13:10:49 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 4673
```

is to be read as: host `128.159.105.240` made a request at `03/Aug/1995:13:10:49 -0400` to retrieve (`GET`) file `/shuttle/countdown/` using protocol `HTTP/1.0`; the server replied with code `200` and the result was `4673` bytes long. Note that the IP address of the host is delimited with the rest of the record with '`- -`'; and the entire request is within quotes.

For this task you will need to answer the following questions:

1. Find the most popular page of the Web server. (The one that appears most frequently in requests, regardless of command.) **(2 marks)**
2. Find the top 10 hosts that produced the most 404 HTTP errors. **(3 marks)**
3. Find the time difference between the first and the last visit for each host (if a host has visited the server only once than just print the timestamp of the visit). **(3 marks)**

1.3 Querying StackOverflow

Task 4

◁ **Task**

For this task you will use a dataset from StackOverflow and extract specific pieces of information. Initially, you should understand the format of the dataset, next you will need to do parse each post, and finally you will need to implement your MapReduce workflows.

[2]See also `http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html` for more details.

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 to access 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 we give the expected output format; lines beginning with '%' are only descriptive comments and you do not need to print them; you only need to print the actual output values.

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

```
% Question Id,  Count
659891,         17551
659892,         2131
```

2. Who was the user that answered the most questions and what were the `Id`s of these questions? **(3 marks)**

   Output Format:

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

3. Who was the user that had the most accepted answers and what were these answers? **(4 marks)**

   Output Format:

```
% OwnerUserId  ->  Number of accepted answers,  AnswerId, AnswerId, ...
1              ->  1,                           1432, 1643
```

2 Marking Guidelines

There are 25 marks available. Each task has marks allocated, which gives a rough guide of how much effort you should spend relevant to the total number of marks available. For programs, marks will be awarded for making use of Hadoop, efficiency and correctness. Bonus marks will be awarded if you are creative.

For efficiency we primarily look at how you structure the computation. Are you sending data around unnecessarily? Are you making appropriate use of MapReduce features? Would your task run out of RAM with more data? Is it sending too much data? Is the computation unbalanced?

You can still use Python or your favorite programming language, even if it's slower, but don't inefficiently use your chosen language.

3 Submission

Your work should consist of:

1. One plain text file, with sections for each part of this assignment. (This is so that it is possible to see which part of the file answers which question.) Include all programs you have written. Make sure you comment your code so that what you are doing can be understood. **Do not submit a Word document or a PDF file—only submit a plain text file**.

2. The output of each task stored in HDFS and named as `task_i.out` where `i` is the number of the task the folder corresponds to. **Do not delete these files from HDFS until you are told it is OK to do so.**

Your text file submission should be named `exc-mr.txt`. Organise your file in sections for each task. Each section should have a code block that should start with `Task i code begin` and finish with `Task i code end` where `i` is the task you are currently providing the solution for. The code block should contain the mapper code, the reducer code (if a reducer is used) and the command (or commands) for running it on Hadoop. It should then be followed by a result block starting with `Task i results begin` and finishing with `Task i results end` for task `i`. If the output files contain multiple lines per file (this depends on the task), only show the first 10 lines of your first output file that you have stored in HDFS. You can use the command:

```
hadoop dfs -cat filename | head -10
```

for showing just the first 10 lines of the file. *Hadoop will print `cat: Unable to write to output stream.` which you can ignore and not submit as part of the assignment. Hadoop is complaining that it cannot write the full file `head` has stopped reading.*

An example submission file should therefore look like:

```
Task 1 code begin
...
code for task 1
...
Task 1 code end

Task 1 results begin
...
first 10 lines of the result of task 1
...
Task 1 results end

Task 2 code begin
...
code for task 2
...
```

```
Task 2 code end

Task 2 results begin
...
first 10 lines of the result of task 2
...
Task 2 results end

...
code and results for remaining tasks
```

Use the `submit` program to submit the `exc-mr.txt` text file; for the rest of your results you only need to make sure they follow the outlined naming scheme and are accessible on HDFS. You can submit the `exc-mr.txt` file from the command line as:

```
submit exc 2 exc-mr.txt
```

4 Deadline

The deadline depends on when you submitted Assignment 1. If you took Choice A (submitted Assignment 1 by 4pm 28 October), then the deadline is **Friday 4 December, 4:00 pm**. If you took Choice B (submitted Assignment 1 by 4pm 30 October), then the submission deadline is **Wednesday 2 December, 4:00 pm**. You will receive e-mail in November reminding you of the choice you took.