Extreme Computing
Second Assignment

Michail Basios (m.basios@sms.ed.ac.uk)
Stratis Viglas (sviglas@inf.ed.ac.uk)

This assignment is divided into three parts and eight tasks. The first part deals with taking messy Web data, cleaning it up and performing simple queries over it. The second part focuses on how huge matrices can be transposed by using MAPREDUCE. Finally, the last part deals with how a relational join operation should be done by using MAPREDUCE. You should use the teaching Hadoop Cluster and any programming language you want.

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.

The files (on HDFS) that you will need for each part follows:

Part 1

File	Number of lines	Number of words
/user/s1250553/ex2/webSmall.txt	10000	149552
/user/s1250553/ex2/webLarge.txt	5000000	69760377

Part 2

Files	Dimensions
/user/s1250553/ex2/matrixSmall.txt	(20, 10)
/user/s1250553/ex2/matrixLarge.txt	(4000, 3000)

Part3

File	Number of lines	Number of words
/user/s1250553/ex2/uniSmall.txt	2000	6987
/user/s1250553/ex2/uniLarge.txt	2000000	6985354

All your actual results should be produced using the larger files and for all tasks you should use Hadoop MAPREDUCE and HDFS.

1 Tasks

1.1 Processing Web data

Task 1

◁ **Task**

Take file `webLarge.txt`, and produce a version which is all *UPPER-case*. For example, the sentence `John loves Mary.` would become `JOHN LOVES MARY.` Call this the *UPPER-case* version. **It does not matter if the output is a single file or multiple files.** **(2 marks)**

Task 2

◁ **Task**

Remove duplicated sentences from the produced *UPPER-case* version. Call this the *deduplicated* version. Your approach should be exact (make no mistakes). The output will be used for some of the following tasks. **(3 marks)**

Task 3

◁ **Task**

Look at the Unix command `wc`. Implement an exact version of it (which only counts words and lines) for Hadoop, using MAPREDUCE and any appropriate **Unix commands**. You should assume the input data and results are stored in HDFS. The output results from HDFS should be merged into a **single file** (on Unix) with the total number of words and lines. For example, the following file `example.txt`:

```
bob had a little lamb and a small cat
alice had one tiger
mary had some small dogs and a rabbit
```

should have the following result (21 is the total number of words and 3 is the total number of lines):

```
21 3
```

(2 marks)

Task 4

◁ **Task**

Implement a randomised version of the previous task which uses probabilistic counting, as taught during the lectures. **(3 marks)**

Task 5

◁ **Task**

On the deduplicated version, find all three-word sequences and their counts. For example, the two sentences:

```
mary had a little lamb
mary had a little tiger
```

have the following three-word sequences and counts:

Sequence	Count
`mary had a`	2
`had a little`	2
`a little lamb`	1
`a little tiger`	1

(3 marks)

Task 6

◁ **Task**

What are the top twenty most frequent three-word sequences? **(2 marks)**

1.2 Matrix Transpose

Task 7

◁ **Task**

Create a program that uses MAPREDUCE and takes as an input a large file containing a matrix `/user/s1250553/ex2/largeMatrix.txt` and returns an output file that has it transposed. The final result can be either a single file or multiple files (in which case, when concatenated, they should give the same result as a single file).

For example, the input matrix:

$$\begin{matrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{matrix}$$

should be transposed to:

$$\begin{matrix} 1 & 5 & 9 \\ 2 & 6 & 10 \\ 3 & 7 & 11 \\ 4 & 8 & 12 \end{matrix}$$

(5 marks)

1.3 Relational Join using MAPREDUCE

In this task you will perform a join operation in Hadoop. Let us assume that we have the relations **student(**`studentId`**,** `name`**)** and **marks(**`courseId`**,** `studentId`**,** `mark`**)** as shown below:

students

`studentId`	`name`
1	George
2	Anna

marks

courseId	studentId	mark
EXC	1	70
EXC	2	65
TTS	1	70
ADBS	1	80

and need to join them on the `studentId` field. Traditionally, this is an easy task when we deal with relational databases and can be performed by using the relational **join operator**. However, the way this join operation is performed drastically changes, when we assume our input is into a **single file** that stores information from both relations.

Assume the format of such a single input file storing data from two relations is as follows:

```
student   1     George
mark      EXC   1       70
student   2     Anna
mark      ADBS  1       80
mark      EXC   2       65
mark      TTS   1       80
```

The first column is a **tag** that shows from which relation the data comes from. Depending on this tag, we can assign meaning to the other columns. When the tag used is `mark`, we know that the second column refers to the `courseId`, the third to the `studentId` and the fourth refers to the `grade` the student took in this specific course. On the other hand, if the tag is `student`, we know that there are only two other columns, one with the `studentId` and one with the student `name`.

Task 8

◁ **Task**

Use the `uniLarge.txt` file perform a join operation on the `studentId` key and produce an output that will have the grades of each student as follows:

`name --> (course1, mark1) (course2, mark2) (course3, mark3) ...`

For example, for the previous input file your algorithm should return:

```
George --> (ADBS, 80) (EXC, 70) (TTS, 80)
Anna --> (EXC, 65)
```

(5 marks)

2 Marking Guidelines

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

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 `sXXXXXXX_task_i.out` where `XXXXXXX` is your matriculation number and `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`. The result block should contain the first 20 lines of your first output file that you have stored in HDFS. This is because the output files will be very big. You can use the command:

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

for showing just the first 20 lines of the file. 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 20 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 20 lines of the result of task 2
...
Task 2 results end

...
code and results for Tasks 3-8
```

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 submission deadline is **Wednesday 4 December, 4:00 pm**.