CS1003 Week 11 Practical: Data Processing & MapReduce

This practical is worth **20%** of the overall coursework mark. It is due at **9pm** on **Thursday 26th April (week 11)**. As for every practical, you should arrive in the lab having prepared in advance, by studying this specification and reviewing relevant course material.

Skills and Competencies

- using MapReduce to process JSON data
- expressing an algorithm in the MapReduce style
- choosing appropriate classes and methods from the MapReduce API
- choosing appropriate classes and methods to process JSON data
- testing and debugging
- writing clear, tidy, consistent and understandable code

Setting Up

As in previous practicals, we are using the automated checker `stacscheck`. For this practical, your program must have a `main` method in a class called `W11Practical` which is in a file `W11Practical.java` that is located inside your `src` directory in your `W11-Practical` directory.

If you can't remember how to create a suitable project setup, have a look at the "Setting Up" instructions for the week 3 practical at:

https://studres.cs.st-andrews.ac.uk/CS1003/Practicals/W03/W03-Practical-File-Processing.pdf

Using a suitable copy command from a terminal window in the lab, you should probably copy the basic MapReduce example directory from StudRes and then adapt and use it as a starting point for your own submission. For example, in a terminal window you might type the following commands

```
mkdir W11-Practical ; cp -r /cs/studres/CS1003/Examples/W10-1-Map-Reduce/W10-1-MapReduce/* W11-Practical/.
```

The example code contains all the Hadoop library code needed to run a Hadoop job to count the occurrence words in a plain text file. Make sure you can compile and run the example as sown in the `READ_ME.txt` file prior to moving on.

Background

Twitter is a micro-blogging service that is fairly popular and on a typical day carries some 500 million messages (https://blog.twitter.com/2013/new-tweets-per-second-record-and-how, http://www.internetlivestats.com/twitter-statistics/). In this assignment you will write a program to produce statistics on Twitter data taken from a 1% sample of Tweets made in January 2017 and conduct performance experiments with your programs.

Requirements

The objective is to build a program that uses Apache Hadoop (http://hadoop.apache.org) to implement a Map-Reduce to extract some possibly interesting information from a small subset of Tweets made in January 2017 in JSON format by counting the occurrence of the `expanded_url` version of URL links that Twitter has recorded users including in the body of their Tweets. When developing your application, you should make use some small test data at

https://studres.cs.st-andrews.ac.uk/CS1003/Practicals/W11/data

Each file contains a sequence of JSON objects corresponding to Tweets. You should examine the format used to represent Tweets as described at https://dev.twitter.com/overview/api/tweets. You should study examples of using Hadoop to count the occurrence of words as shown in lectures and at

https://studres.cs.st-andrews.ac.uk/CS1003/Examples/W10-1-Map-Reduce/

You will have to write code (probably in your *Mapper* class) to parse a line of text representing a Tweet (in JSON format) and find the `expanded_url` version of URL links which users have included in the body of their Tweets. The relevant `urls` array can be found in the `entities` object in Tweets (study the Tweet format at https://dev.twitter.com/overview/api/tweets).

Your program should be capable of processing Twitter data from an *input path* (to a whole directory or a specific file) specified as the first command-line argument (`args[0]`) and write a file to the *output directory* specified as the second command-line argument (`args[1]`). See examples of compiling and running your program shown below. The output produced by your Hadoop program for a specified input should be a count of all URL links (in `expanded_url` form) included by users in the Tweet body (as recorded by Twitter), ordered by URL.

Once you have finished developing your program and are satisfied that it works on the smaller data, you may wish to download some more data for your program to examine and to check that it works with larger data sets. For example, you might take 15 minutes, half an hour, or even an hour of data from some date and see whether the URLs and counts shown by your program are those that you would expect given what was going on at that time. You are supplied with a fairly sizeable amount of compressed 2017 Twitter data on Student Resources at

https://studres.cs.st-andrews.ac.uk/Library/TwitterSample/2017/

The *Tweet* files for the whole Twitter sample on Student Resources are compressed using bzip2 (http://www.bzip.org/) and stored in a directory hierarchy according to the date and time they were composed, namely as:

year/month/day/hour/minute.json.bz2

Please note, each file (i.e. a minute of data) may take up roughly 12MB when it is uncompressed. A day can take up roughly 17GB. When developing your program, you should probably only copy a minute (or two) of data to your local system and decompress the file(s) (using e.g. `bunzip2 *.bz2` in a terminal window) onto your local file system.

If you want to experiment with a larger amount of data (i.e. an hour or more of data), please make sure you don't store them in your home space which is held on a networked storage server. Storing the files on your home server will not only exhaust your disk quota very quickly but also severely slow down access to the files, stress our networks, and will almost certainly invalidate any results from the experiments mentioned as part of extension activities below. If you are working on the machines in the lab, you should copy files to `/cs/scratch/<your-username>/` (where you should replace `<your-username>` with your own username).

Compiling and Running your program

In order to be compiled by the stacscheck auto checker, it must be possible to compile your program from the `W11-Practical` directory using the command below

```
mkdir -p bin
javac -cp "lib/*" src/*.java -d bin
```

Below are some examples showing how your program should be run and the expected output when no command-line arguments are supplied. The commands assume you are working on a Linux lab machine, that the current directory is the `W11-Practical` directory (which contains your `src` directory) and that you are running the Hadoop job locally (rather than on a cluster). Please also make sure that the output directory specified as command-line argument does NOT exists prior to running your program, the Hadoop job will terminate if the directory already exists.

```
java -cp "lib/*:bin" W11Practical
Usage: java -cp "lib/*:bin" W11Practical <input_path> <output_directory>

java -cp "lib/*:bin" W11Practical /cs/studres/CS1003/Practicals/W11/data/data-very-small/00.json output00
```

The first invocation above does not pass any command-line arguments to the program and results in the required usage message being displayed. The second invocation does pass the expected command-line arguments to the program and as a result would produce some Hadoop debug output to the terminal and more importantly, an output file `output00/part-r-00000` containing the count for any URLs included by users in their Tweet body in the specified input file as shown below. Note the quotation marks surrounding the URL in the output.

```
"http://ow.ly/c48d300PjTY"    1
```

Also note that only a single URL with a count of 1 was found to have been included in the Tweet body by Twitter and recorded in the `entities urls` array, despite the file containing 3 Tweets, each of which could contain many URLs. As such, you will have to write code which works for any number of URLs being included in the body of a Tweet.

Running your program using

```
java -cp "lib/*:bin" W11Practical /cs/studres/CS1003/Practicals/W11/data/data-very-small/01.json output01
```

should produce the output file `output01/part-r-00000` containing the count for another URL being included in the Tweet body by a user in the file as shown below.

```
"http://etsy.me/1yjEIyw"      1
```

Similarly, running your program using

```
java -cp "lib/*:bin" W11Practical /cs/studres/CS1003/Practicals/W11/data/data-very-small output
```

should produce the output file `output/part-r-00000` containing the count for all Tweets included in the Tweet bodies (as recorded by Twitter) in all the files in the `data-very-small` directory as shown below.

```
"http://etsy.me/1yjEIyw"      1
"http://ow.ly/c48d300PjTY"    1
```

Hadoop creates the `part-r-00000` output files above and separates URLs from the counts using a TAB. Try running your program on the files in the `data-very-small` directory and try to debug errors before moving on. There are two other data directories at https://studres.cs.st-andrews.ac.uk/CS1003/Practicals/W11/data which contain 1 minute and 10 minutes of data respectively taken from January 20th 2017 at 12:00. These are also used by the auto checker to test your program.

You should use Hadoop Map-Reduce wherever applicable to compute the above, over conventional methods. You can reuse any code from examples or previous practicals, so long as you clearly identify it in your code and report.

Running the Automated Checker

As usual, please use the automated checker to help test your attempt. Please make sure you have a `W11-Practical` directory containing

- your `src` directory containing all your source code including a `W11Practical` class containing your `main` method in `W11Practical.java`
- a `lib` directory with all the Hadoop jars as shown in the example code on studres

If your `W11-Practical` directory does not contain these the auto checker will fail.

The checker is invoked from the command line on the Linux Lab Machines in your `W11-Practical` folder:

```
stacscheck /cs/studres/CS1003/Practicals/W11/Tests
```

As in previous practicals, the tests use the Unix `diff` command to print a message describing how a line (or set of lines) in your output should be changed to match the required output. If some of the tests are failing, examine the `diff` output and try to identify why your program is not compiling, running, or producing the output as expected. By examining the test files at

https://studres.cs.st-andrews.ac.uk/CS1003/Practicals/W11/Tests/basic

you can also see which data files are used in the tests (by looking in the `test.sh` file for a particular test). The data files are all in the data directory at

https://studres.cs.st-andrews.ac.uk/CS1003/Practicals/W11/data

You can also see the expected output by examining the `expected-output.txt` file for a particular test and then compare it yourself with the `part-r-00000` file produced by your Hadoop program when you run it on the same input as used in the test. If nothing is working and you don't know why, please don't suffer in silence, ask one of the demonstrators in the lab.

Testing

You are encouraged to create and document your own tests to test your program's functionality. Paste terminal output from each of your tests into your report in the Testing section. Be clear what each test demonstrates.

Deliverables

Hand in via MMS to the *Week 11 Practical* slot, a `.zip` file containing your entire `W11-Practical` directory which should contain:

- All your Java source files in the `src` directory as specified above.
- A **PDF** report containing the usual sections for *Overview*, *Design & Implementation*, *Testing*, *Examples* (example program runs), *Evaluation* (against requirements), *Conclusions* (your achievements, difficulties and solutions, and what you would have done given more time). Your report should also contain an accurate summary stating which files or code fragments you have written and any code you have modified from that which was given out in class or which you have sourced from elsewhere.
- Please also include in your report, output from running the auto checker on your program.

Extensions

If you wish to experiment further, you could try any or all of the following:

- Give the user an option to order the URLs by occurrence frequency. Hint: You could use the 'Most Popular Words' approach as outlined in the *W10-1-Map-Reduce* lecture slides.
- Perform additional queries on the data, such as:
  - establishing the most frequently re-tweeted Tweets
  - establishing the users associated with the most frequently re-tweeted Tweets
  - establishing any other additional statistics/queries about the data you could generate using Hadoop
- Vary the number of concurrently running map tasks on your system and measure the impact on performance (time to run the job) – you could plot the time taken over the number of current map tasks and discuss the results in relation to the number of processor cores on the lab machine you are using. Hint: look at the W10-2-MapReduce_Multiple_Map_Tasks example on studres.

Don't forget to document your extension work in your report. You are also free to implement your own extensions, but clearly state these in your report. If you use any additional libraries to implement your extensions, ensure you include these in your submission, with clear instructions to the marker on how to resolve any dependencies and run your program.

Marking

Your submission will be marked using the standard mark descriptors in the School Student Handbook:

https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors

However, more specifically, for this practical:

- 1-6: Very little evidence of work, software does not compile or run, or crashes before doing any useful work. You should seek help from your tutor.
- 7-10: A decent attempt which gets part of the way toward a solution, but has serious problems such as not compiling (at the low end), or lacking in robustness and maybe crashing during execution or failing to use Hadoop but obtaining the correct output (at the high end).
- 11-13: A solution which is mostly correct in using Hadoop and going some way towards fulfilling basic requirements, but with issues such as not always printing out the correct values or demonstrating poor use of JSON processing.
- 14-15: A correct solution using Hadoop to solve the problem accompanied by a good report, but which can be improved in terms of code quality, for example: poor method/OO structure, lack of comments, or lack of reuse.
- 16-17: A very good, correct solution to the main problem containing clear and well-structured code achieving all required basic functionality, demonstrating very good design and code quality, good method and class decomposition, elegant implementation of methods with reuse where appropriate, and accompanied by a very well-written report.
- 18-19: As above, but implementing at least one or more extensions, accompanied by an excellent report. However, as mentioned before, it is key that you focus on getting a very good solution to the basic requirements prior to touching any of the extension activities. A weak solution with an attempt at extensions is still a weak solution.
- 20: As above, but with multiple extensions, outstanding code decomposition and design, and accompanied by an exceptional report.

Lateness

The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):

https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties

Good Academic Practice

The University policy on Good Academic Practice applies:

http://www.st-andrews.ac.uk/students/rules/academicpractice/

Hints

Here is one possible sequence for developing your solution. It is recommended that you make a new method or class at each stage, so that you can easily go back to a previous stage if you run into problems. **Please use demonstrator support in the labs whenever possible.**

1. You will need to examine the structure of the Tweet data files and the structure as outlined at https://dev.twitter.com/overview/api/tweets, to see how Tweets are stored in the files, what each Tweet can contain etc.
2. Examine lecture notes, example code, exercises that you attempted, and practical code in which JSON data is processed. Write code to extract `entities` `expanded_urls` from a single line (Tweet) of the `.json`, for example, the 3rd row in `data/data-very-small/00.json`) contains such a URL. Check whether you can find the same URL as shown above for this file. You might write a separate test program which loads a `.json` file and tries to print out the `expanded_urls` it encounters in the `entities`.
3. Move your code to find the URLs into a suitable *Mapper* class which can then output the URLs and count for a single tweet. Remember, each instance of the *Mapper* class will be given exactly one line (Tweet) from an input file.
4. Design and implement a suitable *Reducer* class which can aggregate and output the count for each URL.
5. Write code to setup the Hadoop job using your own *Mapper* and *Reducer* classes and check whether it works on the very small data.
6. It might be worth writing debug statements to a debug file to help you understand your program's behaviour.
7. Test your program first manually and then using stacscheck.