

## Overview

I was asked to make a program that using hadoop extracted some information from Tweeter data.

## Design

I implemented the practical in two parts – first the basic deliverable and then the extensions.

For the basic deliverable I was using the example code as guide and changed the mapper and reducer in the appropriate manner.

For the mapper I used a JsonReader in order to read the json object and get the hashtags from them; these I find in the hashtags arrays in the entitites object. As there are some unexpected characters (delete character in the 1-minute data) I have created a try-catch block for a JsonException which prints to the console the error message.

For the reducer and main classes I am using basically the same code – the reducer sums the counts for all keys and outputs them and the main class creates the job and runs it.

### Extensions:

For the extensions I created a new folder. I made a fully-functioning GUI application, which would take all user input via graphical windows and display everything (including exceptions) in graphical windows.

### GUIRunner:

This is the main GUI class that builds the main window and is responsible for handling all events happening.

From there you can change the flag of the ExceptionGUI class for graphical display of the exceptions – if it is false they would be printed out in the terminal. In the beginning the JLabel that holds the name of the directory has an initial value “Directory ...” and until it is not changed the run function button would give an error. When the directory is chosen the “Choose directory” button changes text to “Change directory” and the functions drop down becomes active; I have made the run function button give an error instead of being inactive for better user experience (more details in the code comments).

From clicking the button an action event is triggered and the class deals with it using switch – for each different function it creates the corresponding job and runs it. In the end it outputs the result – a table for the first 3 and a graph for the benchmark.

### JobBuilder:

I had to create a lot of jobs (6 at total) and I was facing the problem how to do it succinctly and without code duplication. So I decided to use the build pattern and create a builder class. Each

method in this class adds a new property to the job and returns the JobBuilder object (which holds the job). When these are chained one can easily create a new job.

## JobExecutor:

The job executor is the class that uses the job builder – it has four methods for the separate specific jobs; for each method it creates a job with the appropriate parameters and then executes. It also has a method that is displaying a “Job <job\_name> is ready” to the user.

## Mappers and reducers:

These are pretty straightforward and well documented in the code. The only more interesting ones are for finding the most retweeted tweets. For this I am first reading the json strings in the mapper and finding all retweeted\_status objects, from which I extract the id\_str of the user who wrote that tweet and the number of retweets of the particular post. In the end I sum all numbers for the user and use the MostPopular function in order to sort them in decreasing order.

## Extensions

- Sort the hashtags by count in decreasing order.
- Performing additional tasks – get the retweeted posts sorted by retweet count in decreasing order and the id of the user who wrote them.
- Created a benchmark that is varying automatically the number of running map tasks and displaying the results on a graph.
- Creating a GUI for the whole program .
- Creating a table to display the results of the map-reduce task.

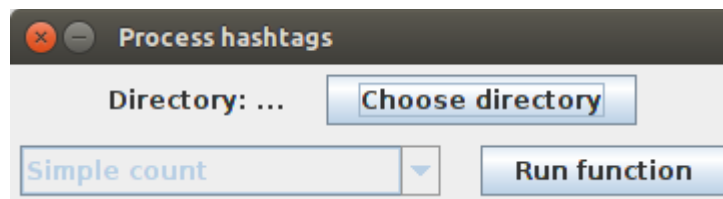
## Testing

I tested the core deliverable using stackcheck and it passed all tests. I also ran some custom tests on the extensions, but due to the nature of the program and the things it is doing I can only provide screenshots for only a few of those.

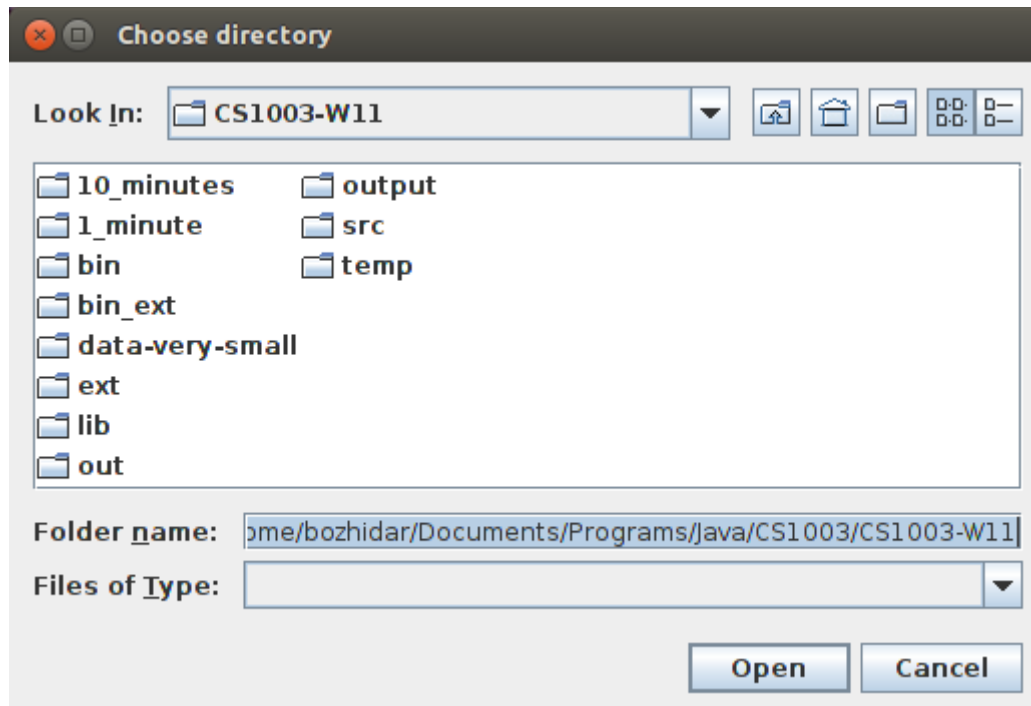
## Examples

```
Terminal
bash-4.3$ stacccheck /cs/studres/CS1003/Practicals/W11/Tests
Testing CS1003 Week 11 Practical
- Looking for submission in a directory called 'src': found in current directory
* BUILD TEST - basic/build : pass
* COMPARISON TEST - basic/Test01_no_arguments/progRun-expected.out : pass
* TEST - basic/Test02_data-very-small-00/test : pass
* TEST - basic/Test03_data-very-small-01/test : pass
* TEST - basic/Test04_data-very-small-all-files/test : pass
* TEST - basic/Test05_1_minute_of_data/test : pass
* TEST - basic/Test06_10_minutes_of_data/test : pass
* INFO - basic/Test0_CheckStyle/infoCheckStyle : pass
--- submission output ---
Starting audit...
[WARN] /cs/home/bg43/Documents/CS1003/Practicals/W11/W11Practical/src/./TwitterMapper.java:5: Using the '.*' form of
import should be avoided - javax.json.*. [AvoidStarImport]
[WARN] /cs/home/bg43/Documents/CS1003/Practicals/W11/W11Practical/src/./TwitterMapper.java:16:19: 'static' modifier o
ut of order with the JLS suggestions. [ModifierOrder]
[WARN] /cs/home/bg43/Documents/CS1003/Practicals/W11/W11Practical/src/./TwitterMapper.java:22:20: Name 'DEL' must mat
ch pattern '^[a-z][a-zA-Z0-9_]*$'. [LocalVariableName]
[WARN] /cs/home/bg43/Documents/CS1003/Practicals/W11/W11Practical/src/./TwitterMapper.java:22:51: '127' is a magic nu
mber. [MagicNumber]
[WARN] /cs/home/bg43/Documents/CS1003/Practicals/W11/W11Practical/src/./TwitterMapper.java:22:51: 'typecast' is not f
ollowed by whitespace. [WhitespaceAfter]
[WARN] /cs/home/bg43/Documents/CS1003/Practicals/W11/W11Practical/src/./W11Practical.java:15:5: Missing a Javadoc com
ment. [JavadocMethod]
[WARN] /cs/home/bg43/Documents/CS1003/Practicals/W11/W11Practical/src/./W11Practical.java:16:11: 'if' is not followed
by whitespace. [WhitespaceAround]
[WARN] /cs/home/bg43/Documents/CS1003/Practicals/W11/W11Practical/src/./log4j.properties:0: File does not end with a
newline. [NewlineAtEndOfFile]
Audit done.
---
8 out of 8 tests passed
bash-4.3$
```

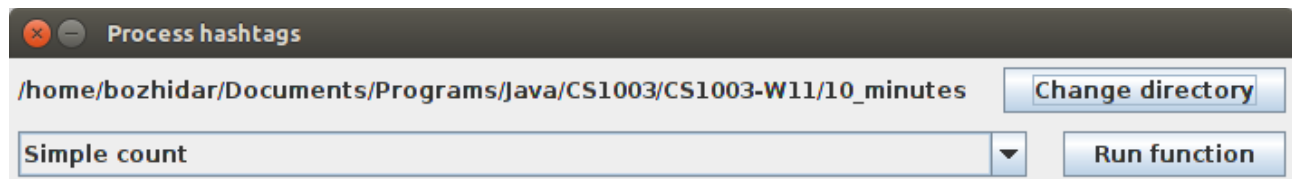
Extension window:



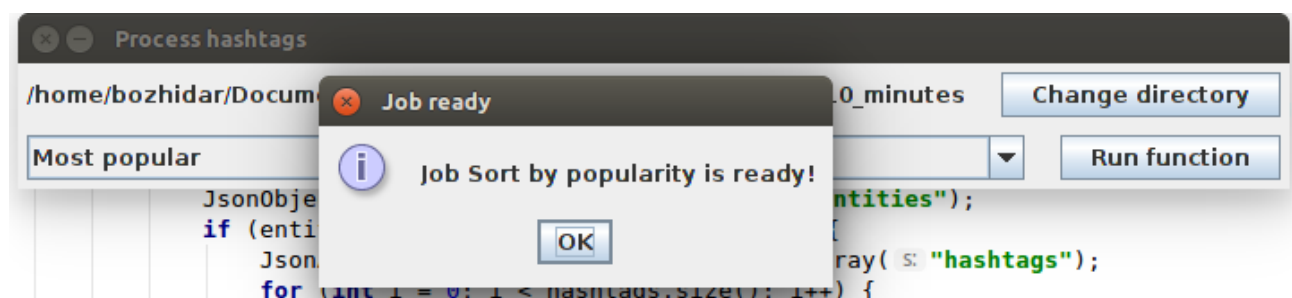
Choose directory:



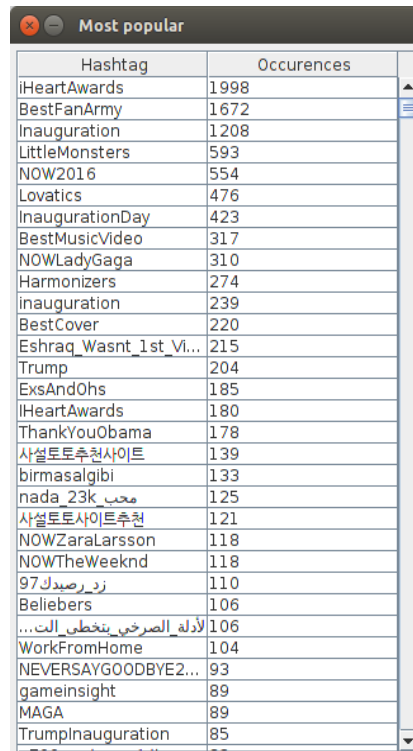
Chosen directory:



Job ready:

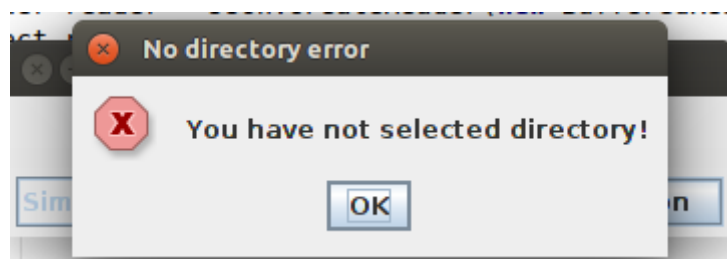
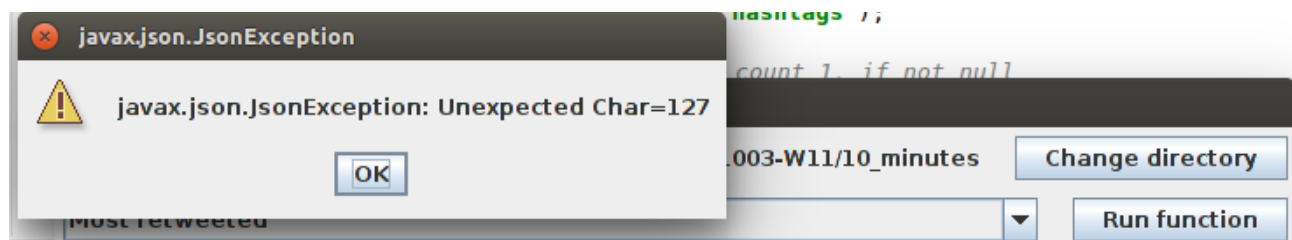


Results table:



Hashtag	Occurences
iHeartAwards	1998
BestFanArmy	1672
Inauguration	1208
LittleMonsters	593
NOW2016	554
Lovatics	476
InaugurationDay	423
BestMusicVideo	317
NOWLadyGaga	310
Harmonizers	274
inauguration	239
BestCover	220
Eshraq_Wasnt_1st_Vi...	215
Trump	204
ExsAndOhs	185
iHeartAwards	180
ThankYouObama	178
사설토추천사이트	139
birmasalgibi	133
nada_23k محب	125
사설토사이트추천	121
NOWZaraLarsson	118
NOWTheWeeknd	118
97 رصيدك	110
Beliebers	106
...أدلة الصرخي يتخطى الت...	106
WorkFromHome	104
NEVER SAY GOODBYE2...	93
gameinsight	89
MAGA	89
TrumpInauguration	85

Error:



## Evaluation

Looking at the task I was given, I believe I did excellent job in creating the program. It works with the test data, it is optimised for user experience and can perform extra tasks.

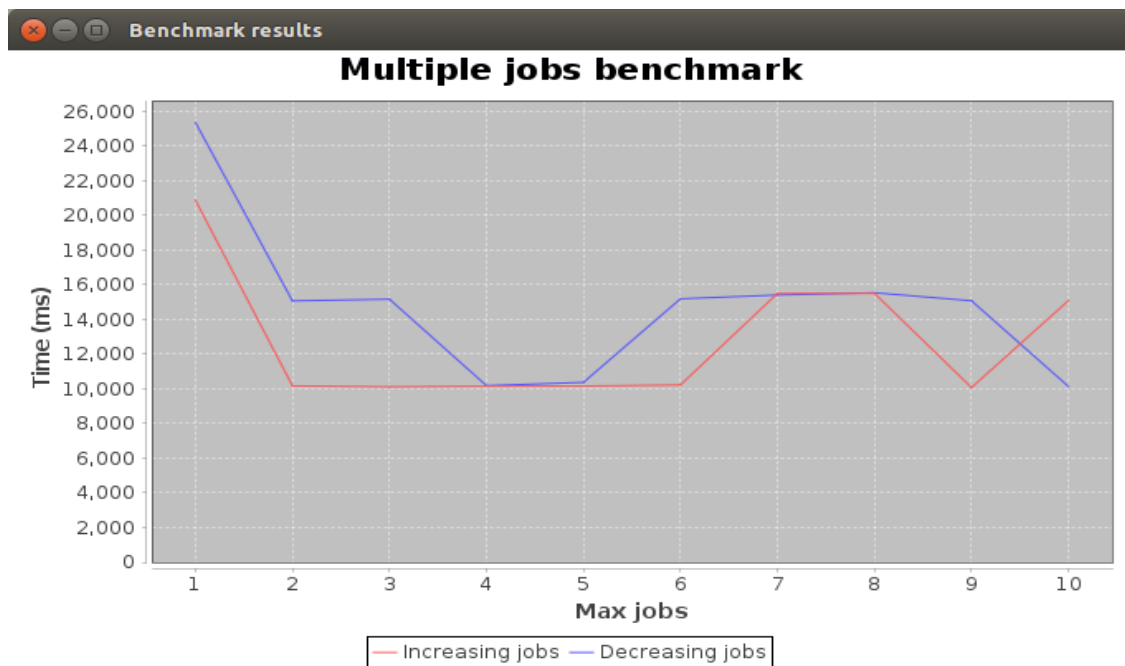
## Summary of twitter statistics:

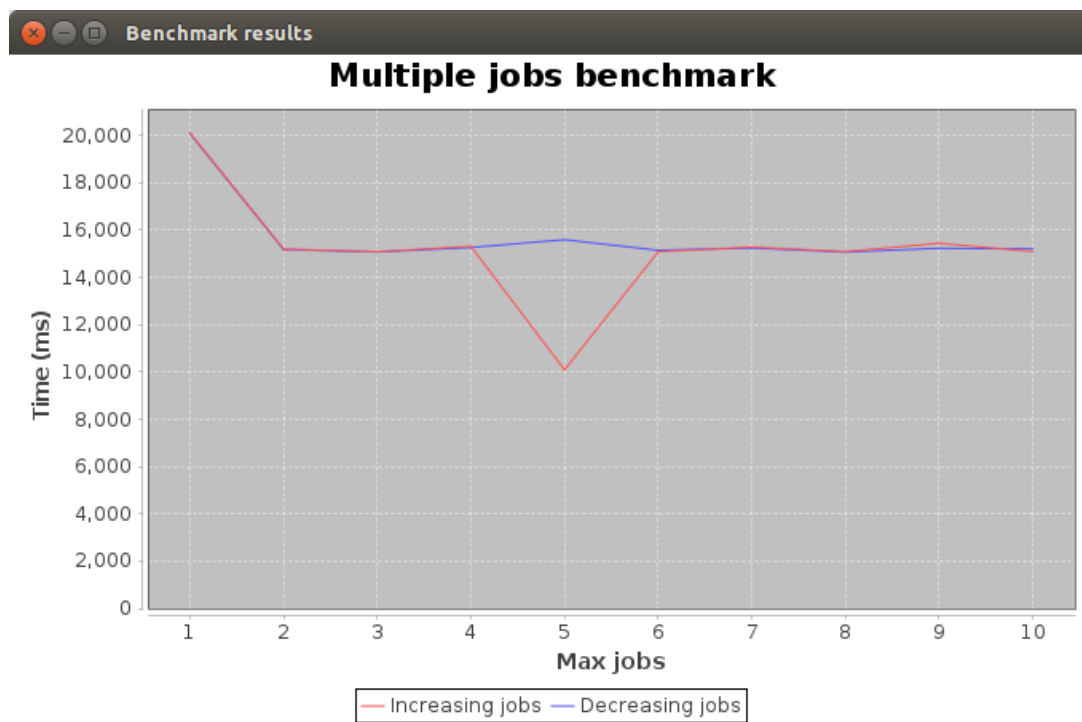
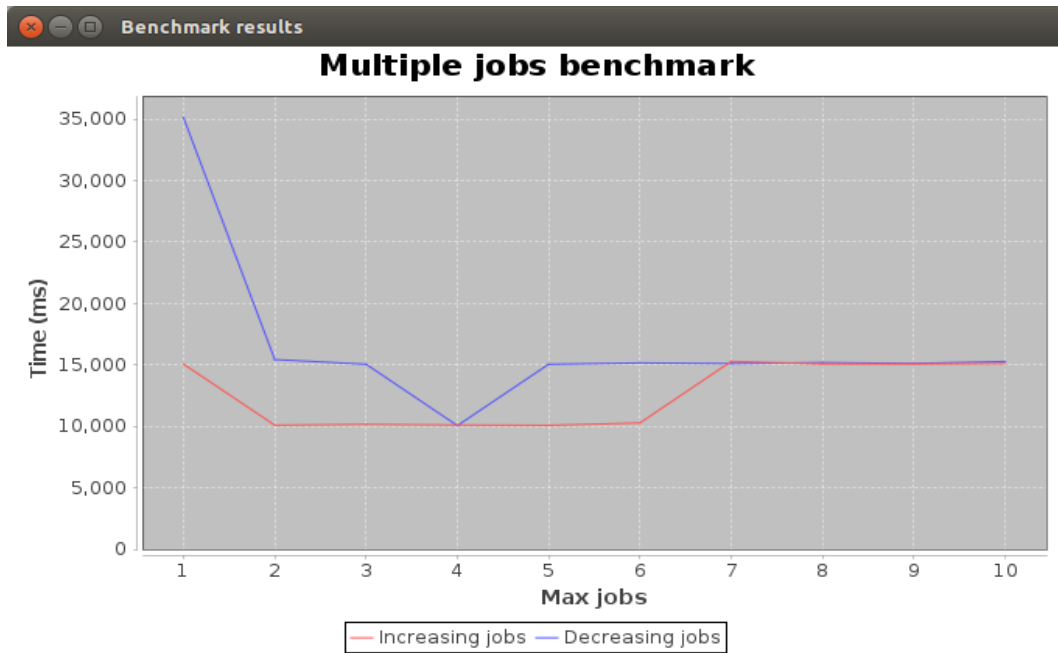
If we look at the statistics for the 10 minute data (picture in the examples above) we can see that three hashtags are predominant: iHeartAwards, BestFanArmy and inauguration. They certainly fit, as the data was taken on the 20<sup>th</sup> of January this year when the I Heart Awards list was up and people were in the peak of their excitement (the list was published just 2 weeks before this date) and Best Fan army is a award related to these awards. Also, on the same day Donald Trump was inaugurated, which explains the peak of this tag (and of InaugurationDay). Some of the other leading tags are also connected to the iHeartRadio music awards (like Lovatics, representing the Demi Lovato fans).

So to sum up, this data definitely fits the current time.

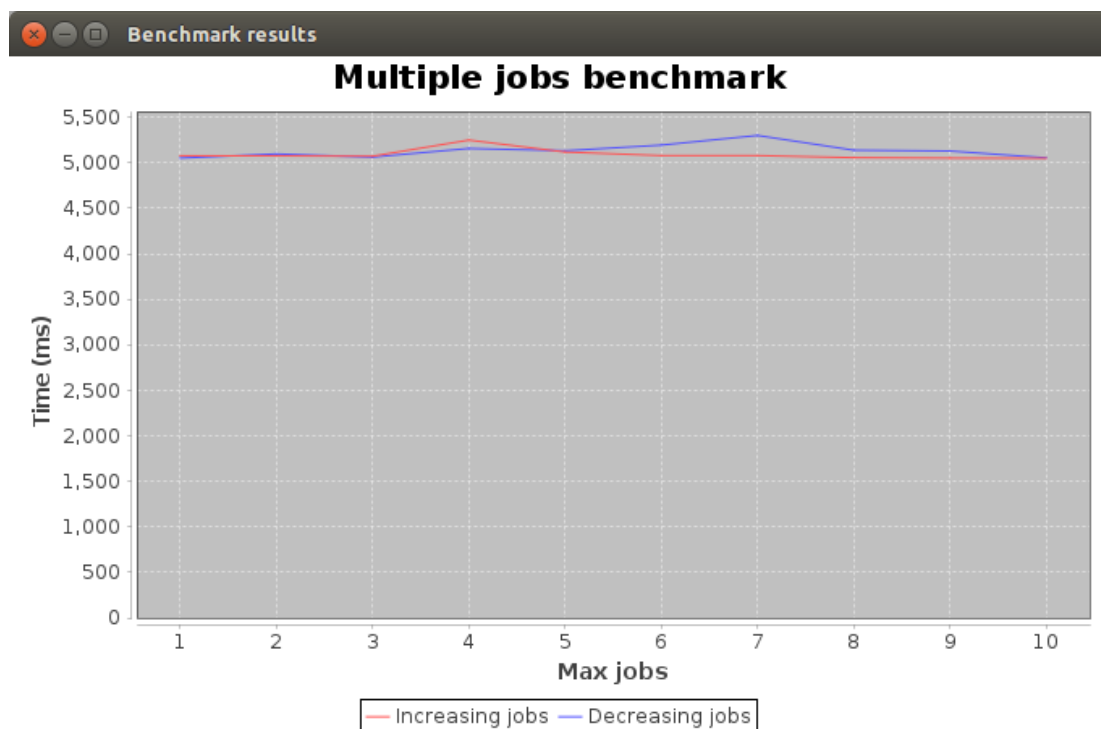
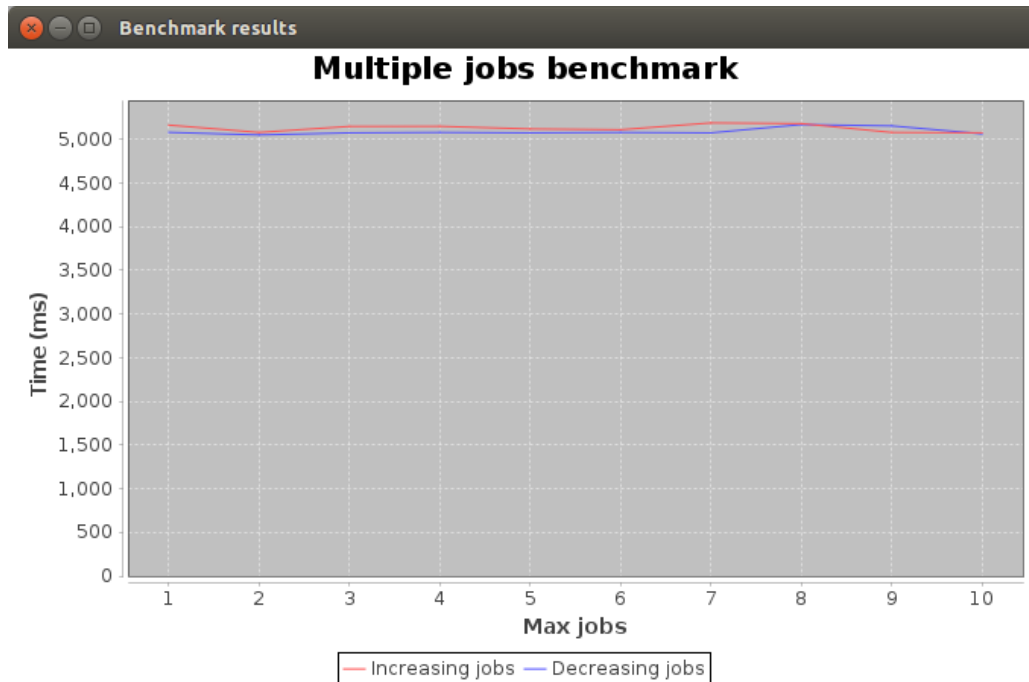
## Discussion of the benchmark results:

Output for 10-minute data:





Output for 1-minute data:



As we can see from the test runs there is almost no difference in the time for the 1 minute data.

The results are little more curios when we look at the 10-minute data.

On the three runs we can see huge inconsistencies (once the blue graph takes 35 seconds with 1 task, the next time only 20). Also the two graphs are going against intuition – instead to have the



same value for time at the same number of tasks (the red is the time when the number of tasks was increased and the blue – when decreased), they almost always are far apart.

Therefore there cannot be made a solid conclusion of this data; the only similar thing between the three graphs is one thing – the time always drops from 1 to 2 tasks with a noticeable amount.

After googling and finding these websites :

<http://ercoppa.github.io/HadoopInternals/AnatomyMapReduceJob.html>

<http://stackoverflow.com/questions/6885441/setting-the-number-of-map-tasks-and-reduce-tasks>

I have only one explanation – as hadoop uses the same number of mappers as there are splits, it needs only one mapper as the 1-minute data is small and in one file, which could explain the consistency; on the other hand the 10-minute one is a lot bigger and then hadoop will need more mappers as there would be more splits (from the graphs I would conclude 2 splits), therefore with setting the max map jobs at 1 the program is slowed down and when it is  $\geq 2$  it is working at its fastest.

(As a side note I want to add that setting the max jobs at higher number, that is 20 or more, made hadoop crash and the output folder was empty. I really have no explanation for this.)

## Conclusion

I am extremely happy with the program I managed to create. My only difficulty was trying to figure out the discrepancy in the graph results and if I had more time I would have probably downloaded bigger chunks of data and tested my hypothesis on them.