COMP38120: Documents, Services and Data on the Web: Lab Manual - 1

This document describes the laboratory work that is to be taken at the end of the first semester and at the start of the second semester. The laboratory focuses on supporting functionality addressed within "Documents on the Web" using big data processing techniques associated with Cloud computing introduced in the part of the unit on "Services on the Web".

There are three exercises in this laboratory, which are assigned to the 4 timetabled lab sessions as follows:

- *Exercise 1.1: Getting started with MapReduce*. The aim of this exercise is to become familiar with how MapReduce jobs are written using Java and run using Hadoop. In essence, you will run an existing WordCount program using Hadoop and make a modest change to it. Thus here you should become familiar with the environment that will be used in the later exercises. This exercise is not assessed.
- *Exercise 1.2: Inverted index in MapReduce*. The aim of this exercise is to develop a MapReduce program that builds an inverted index. Here you should become familiar with a code base that will be the foundation for Exercise 1.3, and should obtain some experience writing a new (but fairly straightforward) application that supports document processing. This exercise is not assessed.
- *Exercise 1.3: Document indexing in MapReduce*. The aim of this exercise is to develop your own document indexing application using MapReduce. Here you will select from a range of possible functionalities, and implement these using MapReduce, as an extension to your inverted index from Exercise 1.2. This exercise *is* assessed.

These labs will be supported by 4 Laboratory Sessions in the normal timetabled slots for the unit (in which there will be academics and Teaching Assistants present). Make the most of these. I will not be trying to debug your programs over email, and do not expect people to come to see me with lab questions at other times. Thus these face-to-face sessions are the only way of getting help with your labs, so please make the most of them. In the past we ran additional consultancy sessions, but these were poorly attended, which suggests that the labs provide sufficient support.

Exercise	Laboratory Sessions
1.1	8th Dec 2017: 2-4
1.2	15th Dec 2017: 2-4
1.3	2nd Feb 2018: 1-3 9th Feb 2018: 1-3

Exercise 1.1 Specification

Getting started with MapReduce

Duration

1 x 2 hour Lab, plus additional time to complete as required.

Aim

To introduce students to the lab environment, and in particular to Hadoop, and to obtain some experience coding up a familiar MapReduce program

Learning Outcomes

A student who successfully completes this exercise will be able to

- Use the software that the labs require.
- Read the control flow of a MapReduce program.
- Compile and run a MapReduce job.
- Make some modifications to a MapReduce program.

Summary

For this exercise, you must import the framework to be used for developing Hadoop applications into Eclipse, and build the framework using ant. Once built, you should run the WordCount program and evaluate the output.

Once you have the basic framework running, you should then make some modifications to the WordCount program to increase the cleanliness of the output.

Description

You should carry out the following steps for this lab:

1. Run Eclipse under linux.
2. From Eclipse, compile and run the supplied WordCount program by following the instructions in: *How To Compile and Run Hadoop Programs in Eclipse*. The WordCount program for this laboratory is available from http://www.cs.man.ac.uk/~norm/COMP38120/COMP38120-Lab1.1.zip. If you would rather not use Eclipse, you can use another IDE that supports Ant, or you can compile and run using Ant directly, as described in: *How To Build and Run Projects Using Ant*.
3. Use Eclipse to browse the source of the program, and in the file system look at the input and the output. What is the word count for *Bart*? How easy is that question to answer?
4. Modify the program to change the *map* operation so that it cleans up the data it is acting on, for example by removing non-alphabetic characters and converting all letters to lower case. What is the word count for *Bart*? How easy is that question to answer?

Assessment

This Exercise is not assessed, but should be completed prior to attempting Exercises 1.2 and 1.3

Exercise 1.2 Specification

Writing a MapReduce program that builds an inverted index.

Duration

1 x 2 hour Lab, plus additional time to complete as required.

Aims

To introduce students to programming MapReduce jobs, using an example of direct relevance to Documents on the Web.

Learning Outcomes

A student who successfully completes this exercise will have:

- Designed the map and reduce operations for an inverted index.
- Understood how to integrate these into a Java template for use with Hadoop.
- Run the resulting application using Hadoop.
- Made explicit some issues with a (most likely fairly) naïve implementation of an inverted index.

Summary

For this exercise you import the framework that will be the starting point for this exercise and Exercise 1.3 into Eclipse, and build it using ant. Once built, you should complete the inverted index application and evaluate the output.

Description

"Your boss is keen on The Simpsons. However, it can be difficult to remember the names of specific episodes. You have been asked to build an inverted index from the wikipedia pages of each episode that will allow searches to be made for particular episodes using a search tool your colleague is building."

You must write a MapReduce job that creates an inverted index from a set of the wikipedia entries for The Simpsons episodes (supplied to you in the *input* folder). The tokens for the inverted index can be create by splitting a string on spaces. Punctuation and stop words need not be taken into consideration at this point in time.

You should carry out the following steps for this lab:

1. Run Eclipse under linux.

2. From Eclipse, compile and run the supplied inverted index template program in the same way as you compiled and ran WordCount in Exercise1.1. The starting point for this laboratory is available from http://www.cs.man.ac.uk/~norm/COMP38120/COMP38120-Lab1.2.zip.
3. Design the map and reduce operations for building an inverted index. The suggestion here is that you write them first using pseudo-code, as in the workshop, and then convert this pseudo-code to Java.
    - Hint 1: the comments for the map and reduce operations give an insight as to their expected complexity for this exercise (e.g. these versions need not do any counting).
    - Hint 2: various of the tasks that need to be undertaken here have analogues in WordCount.
    - Hint 3: look at all the operations you have been provided with in the template, as some contain code that does things for you that most likely you do not know how to do!
4. Implement the MapReduce application in Java, compile it, and run it over the datasets from Wikipedia that have been provided.
5. Review your implementation, to identify functionality or performance limitations.

Deliverables

A basic inverted index built from a set of wikipedia pages.

Assessment

This exercise is not assessed, but must be completed prior to completing Exercise 1.3, for which it is the starting point.

Exercise 1.3 Specification

Developing a more sophisticated document index using MapReduce

Duration

2 x 2 hour Lab + additional time outside the timetabled labs

Aims

To apply understanding from the workshops on Documents on the Web in the development of an index-building program using MapReduce, and to discuss the principal features, strengths and limitations of the result.

Learning Outcomes

A student who successfully completes this exercise will have:

- Designed map and reduce operations for document indexing.

- Run the resulting indexing program using Hadoop, and studied the results.
- Discussed functionality and performance characteristics of the solution.

Summary

You must extend your MapReduce program from Exercise 1.2 to construct an index from a corpus that can support efficient and effective search, and critically evaluate the resulting program and index in a report of up to 1000 words.

The functionalities to be supported are divided into two parts. If you are not able to complete all the functionalities, start with those in Part 1, but note that you cannot get a good mark if you have only completed Part 1. Please use the libraries that have been made available to you, for example for representing the data to be passed from map to reduce.

Description

Part 1:

"Whilst your boss was impressed with your initial index from Exercise 1.2 some searches using the index were less effective than would be ideal. Thus you have been asked to clean up the inverted index so that searches match more entries."

You must implement the following functionality into your MapReduce to cleanse your data and increase performance:

- Stemming.
- Stop Word Removal.
- In-Mapper Aggregation.
- Case folding (should all terms be lower case, or do some need to remain upper case).

Part2:

"Your colleague has extended the search to rank the results. As such you are now being asked to support positional indexing, so that terms that are closer together score higher, TFIDF scores can be derived, and terms that lie in the title of the document need to be identified."

You should implement functionality such as the following into your program to aid in search over the document:

- Positional indexing (where the token lies in the document). Note that documents could be bigger than a file split.
- Document and term frequency (which combine to make TFIDF): here you should consider carefully what it is appropriate to store.
- Flagging of important terms (e.g. the first line of the document represents the title, the rest is the article).

Note that this exercise is not intended to have a single correct answer in terms of what is developed or how. Thus you can choose (and then explain/motivate in your report) different

functionalities from the above, or you can apply other techniques from the Documents on the Web part of the unit. You could even support the same functionalities in different ways and then compare them.

Deliverables

You should submit the following in a single Word or pdf document, in the following order:

1. A report of not more than 1000 words that discusses issues such as the following. These issues and questions can be used to provide a structure to your report (the marking scheme is structured this way). Note that quite a few marks require you to go beyond bookwork. What are the specific features of *your* implementation, how does it do with The Simpsons test data set, etc.
    - Functionality
        i. What searches could be performed successfully and why. For example, if you google "google search help" (or "yahoo search help"), you will get to a page that describes the sorts of search you can carry out with google (or yahoo).
        ii. What features were implemented, what they did to improve the index, and what problems may they create?
        iii. The limitations of the design or its implementation.
    - Performance
        iv. What map reduce design patterns were used in the code, and what effect did they have on the performance of the program?
        v. What factors are likely to limit the performance of your application, and why?
2. A fragment of the output of your program that occupies not more than 50 lines, and that includes complete index entries for terms that illustrate well the properties of your index. These need not be 50 contiguous lines from the output. You can refer to this sample from (1).
3. The complete listing of your BasicInvertedIndex.java program.

Assessment

This exercise is worth 25% of a student's overall mark for this course unit. Credit will be given taking into account: (i) the amount of functionality supported; (ii) the elegance and effectiveness of the design and implementation; and (iii) the clarity and depth of insight in the report. A few marks are reserved for additional functionality in the code that goes beyond that suggested in some way, and for particularly insightful comments in the report. The available marks will be distributed fairly evenly between the code and the report, but each may influence the other (e.g. it may be helpful for your code mark if the report makes clear why things have been implemented in the way they have).

A first class level submission will provide comprehensive functionality, with few mistakes, implemented in an elegant and efficient way, with an insightful discussion of the strengths and weaknesses of decisions made.

An upper second level submission will provide significant functionality, with few mistakes, implemented in a generally appropriate way, with a reasonable discussion of the strengths and weaknesses of the features supported.

A lower second level submission will provide reasonable functionality, but may well include some mistakes, implemented in a way that provides significant opportunities for improvement, with a rather superficial discussion of the features supported.

Below lower second class honours, there is likely to be limited functionality, which likely contains some errors, with significant weaknesses in the implementation, and a discussion that rarely goes beyond bookwork.

Submission Deadline: **Monday 12th February 2018 at 5:00pm**. This is a hard deadline; extensions will only be granted as a result of formally processed Mitigating Circumstances (http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=4272). Marks for late submissions will be reduced in line with the university's rather punitive policy (http://documents.manchester.ac.uk/display.aspx?DocID=24561).

Submission Guidelines

The deliverables should be submitted using Blackboard.

Norman Paton

npaton@manchester.ac.uk