COMP38211:
Documents and Data on the Web Coursework

This document outlines the specifications for the COMP38211 Coursework which focusses on supporting functionality for analysing "Documents on the Web" using big data processing techniques associated with cloud computing.

The first part of this document describes Preparatory Exercises that are *not* assessed but should be completed prior to attempting the Coursework.

- *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.
- *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 functionality that will be the foundation for the Coursework, and should obtain some experience writing a new (but fairly straightforward) application that supports document processing.

The second part of this document describes the Coursework, *Document indexing in MapReduce*. The aim of this coursework 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 above.

This *is* assessed.

The coursework will be supported by laboratory sessions (in which there will be academics and Teaching Assistants present). Please make the most of these, as they are the primary way of obtaining support for the coursework. We may also have some drop in "office hours" sessions to support the lab, but will work out the details nearer the time, when we understand better how everything is going and what might be most suitable. The lab sessions are as follows:

Week	Activity
w/c 5th and 12th October	Please make sure you are up and running with VirtualBox. Alternatively, you could install Hadoop on your computer directly, but please assume you are on your own with this.
w/c 19th October	1 hour lab session: Exercise 1.1
w/c 26h October	1 hour lab session: Exercise 1.2
w/c 2nd November	1 hour lab session: Coursework Part 1
w/c 9th November	1 hour lab session: Coursework Part 2
w/c 16th November	3 hour lab session: Finalising the lab / report

Preparatory Exercises

Exercise 1.1: Getting started with MapReduce

Aim

To introduce the development environment, and in particular Hadoop, and to obtain some experience extending a familiar MapReduce program

Learning Outcomes

A student who successfully completes this exercise will be able to

- use the software that the exercises 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. 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 the course unit site on Blackboard (the item named *Exercise1.1*). 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*.
2. 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?
3. 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?

Exercise 1.2: Inverted index in MapReduce

Aim

To program a 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 the Coursework 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 created 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. From Eclipse, compile and run the supplied inverted index template program in the same way as you compiled and ran WordCount in Exercise 1.1. The starting point for this exercise is available from the course unit site on Blackboard (the item named *Exercise1.1*).
2. 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 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!
3. Implement the MapReduce application in Java, compile it, and run it over the datasets from Wikipedia that have been provided.
4. Review your implementation, to identify functionality or performance limitations. You will need to address these in the Coursework.

Coursework: Document indexing in MapReduce

Aims

To apply understanding from the lectures 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 above) 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 and improve 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
- Stopword removal
- In-mapper aggregation
- Case folding (should all terms be lower case, or do some need to remain upper case).

Additionally, here you should consider carefully what types of index terms are appropriate, e.g., single-word, multi-word terms.

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, and derivation of TFIDF scores."

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).

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 an archive (i.e., a zip file): (1) your source code including your BasicInvertedIndex.java program, and (2) a Word or PDF file that includes:

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 – you should discuss your implementation and its properties, illustrated with reference to the Simpsons example where this helps to make things concrete.
    - Functionality
        i. What features were implemented, what did they do to improve the index, and what problems may they create?
        ii. What are the limitations of the design and/or its implementation.
    - Performance
        iii. What MapReduce design patterns were used in the code, and what effect did they have on the performance of the program?
        iv. What factors are likely to limit the performance of your application, and why?
2. A fragment of the output of your program (i.e., the index) 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).

Assessment

This exercise is worth 60% 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 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: **Tuesday 24th November at 18:00pm**.

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=4271). 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.