Laboratory Exercise 3

1 Outcome

The objective of the exercise is to introduce the principles of the MapReduce framework for processing big data. The main features in such a framework are the reliable and fault-tolerant processing techniques employed. While not all problems are solvable by the MapReduce techniques, a large number of decomposable problems can be divided up to be solved by the MapReduce technique.

The main outcomes will be:

1. Learning how to decompose a big data processing problem into sub-tasks executed by workers but coordinated by a master task. This phase is referred to as the *mapping* phase.
2. Learning how the results from workers performing the sub-tasks are eventually merged into a final result. This phase is referred to as the *reduction* phase.
3. Learning to use the fundamental principles of how MapReduce works using an alternate approach to Hadoop, e.g., Phoenix++, Mrs-MapReduce, or MrJob. Hadoop is an open source implementation from Apache of the MapReduce framework written in Java. The original MapReduce concept came from Google. Pheonix-2 and Pheonix++ are C and C++ implementations respectively. The rest are Python equivalent implementations.

2 Problem Description

2.1 Work Schedule

The work involves:

- Designing and implementing MapReduce algorithms for a variety of common data processing tasks. The required algorithms are:
  1. A simple word count algorithm of a text. This gives the frequencies of occurence of words in a text. You need not include Stop Words, e.g, “for, as, the, is, at, which, on”. Please include your list of Stop Words used in your repository. Consider words to be *case-insensitive*.
  2. Top-K query. The K most frequently occurring words, ignoring stop words, for K = 10, 20.
  3. An inverted index of the text. This involves listing, for a given list of words, the line numbers of the text that the words occur. Only list out about 50 lines of the distinct words: i.e. the first 50 lines with that word.

- Some Python-Based or C/C++-Based MapReduce frameworks are given below. There are other C++-based and Python3-based MapReduce frameworks available. It is recommended that you choose one from the following.

  C/C++: Phoenix++

  - `https://github.com/kozyraki/phoenix`
  - `https://github.com/conradhaupt/phoenix`
    - Contains CMake build scripts that work. Library is identical.
  - `https://csinparallel.org/csinparallel/modules/PhoenixMRIntro.html`

  Python: Through pip.

  - Mrs-MapReduce: `https://pythonhosted.org/mrs-mapreduce/`
  - MrJob: `https://mrjob.readthedocs.io/en/latest/`
    - Ensure to run using `-r local` to use parallelisation.
  - ~~DISCO:~~ **Does not work on the cluster. You may use it if you can run it on your personal computers.**

- Conduct some program tests with a small and large text files, provided to you at `http://jaguar1.eie.wits.ac.za/lab3/`. If you would like to, there is also a very large text file with which you can further test your implementation.

- The first task of the word-count algorithm is the most common algorithm used in explaining MapReduce. Reading the material on the listed websites and linked codes will assist you to get going. The subsequent tasks will require you to think a bit more on how to solve the tasks.

- Your implementation need not exploit distributed parallelisation and is only required to have multiple threads on a single machine.

3 Deliverables

- A lab-report analysing the performance of the implementation for both the small and large files. Max 2 pages and one report per group.
  - Provide a short description of your algorithm to the word-count, top-k, and inverted index problems, as well as simple pseudo-code of your routines.
  - Lists of the Top-K words for K = 20, and their inverted indices, for both files. These lists, and only these lists, may be included in appendices and will not count towards the two page limit.

- Your code for marking in your initialised group `git` repository in the Github Organisation.
  - Ensure your repository has an appropriate `README` describing the build process.

- You may commit the input files to your git repo. If you do so, we will verify they are the same as the originals. If you do not commit them, we will test your submission using the original hosted copies.

- If you are using a C/C++ library, you may include its source-code in your repo so you can compile and link appropriately. Do not commit binaries.