Big Data Systems and Analysis

Saurab S

9 September 2021

1 Introduction

Note: The repository associated with this submission can be found here.

This assignment (Problem1-Option1) is an experimentation with the MapReduce paradigm to solve problems. For installation and setup, please refer to *README.MD*. For the problem statement and other problems, please refer to *2021-ProgrammingAssignment-1.pdf*.

2 Hardware Specifications

We are running the MapReduce programs on a Apple M1 Chip with 16GB of RAM. It has 8 cores running at 3.2GHz, with 2.6 TFlops of throughput.

3 Overview

We performed high-level tasks as a part of this assignment:

1. Install Hadoop and run HDFS and MapReduce
2. Run the vanilla WordCount program given in the hadoop tutorial on four datasets.
3. Run a MapReduce based TopN program to find top 100 words across an entire corpus, for four different sizes of corpus.

Installation of Hadoop and screenshots of execution are covered in *README.MD*. So we will only cover the analysis of second and third tasks in this report.

4 WordCount using MapReduce

We ran the vanilla WordCount program over 5,10,15,20 files of similar sizes to observe the scaling trend followed by MapReduce programs in general.
Dataset size: 4KB per file, 80KB for all 20 files combined.

These are the runtimes for the given words (approximated through average).

Words	Mappers Runtime(ms)	Reducers Runtime(ms)
3000	23328	3019
6000	56345	5224
9000	86295	5787
12000	102106	10909

The graph showing how the runtime scales as the words increase.

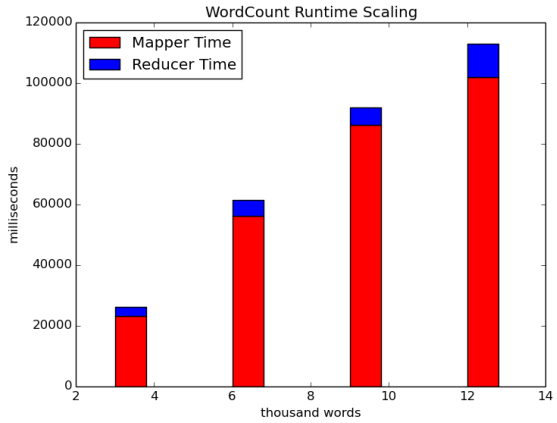

Figure 1: Runtime scaling of WordCount Program

As we can see, it scales roughly linearly in our specific example. This is the expected behavior since we are using a single node as our cluster. As the files we are dealing with range in small-mid range, the miscellaneous overhead costs are roughly constant (like FileIO setup) and the execution scales alomst linearly with word count. So, as the mappers increase, since each mapper roughly does the same work across all datasets, more mappers running for a program would mean linear runtime scaling. Keep in mind that we are reporting the *total* runtime of all mappers and some of this processing might happen parallely on different cores.

5 TopN using MapReduce

We also ran a Top100 java code that uses MapReduce paradigm on files in Input/gutenberg. For this task, the books (and parts of books) written by Mark Twain were considered. On average, each file is in the size of 16,000 words.

Dataset size: 110KB per file, 2.2MB for all 20 files combined.

Here are the runtimes for 4 executions. These are for the given number of words (approximated through average).

Words	Mappers Runtime(ms)	Reducers Runtime(ms)
80000	28335	3329
160000	52939	4773
240000	85551	5411
320000	110659	11860

The graph showing how the runtime scales as the words increase.

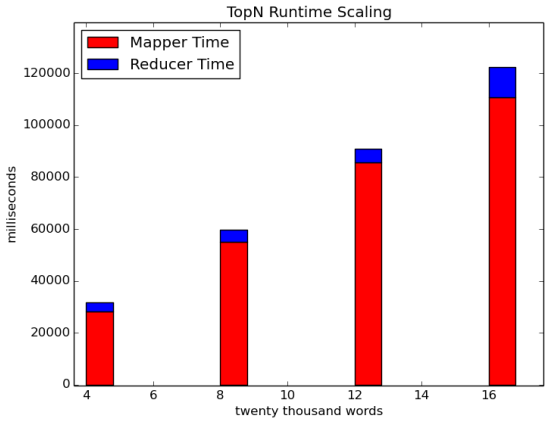

Figure 2: Runtime scaling of Top-100 Program

As we can see, it scales alomst linearly here too, due to similar reasons as the abovementioned WordCount program. For screenshots of the runtime for each of 8 executions, refer to *Screenshots* folder in the repository.