CS 434: Parallel and Distributed Computing

Percy Brown: 40382022

April 20, 2021

Lab Assignment 4

1 Part 1

1.1 Selected approach of MapReduce Framework

My selected MapReduce framework was MrJob. One of the reasons for choosing MrJob was that I am very conversant with python, thus preferred a framework that would use python as the programming language. MrJob is the easiest route to writing Python programs that run on Hadoop. If you use mrjob, you'll be able to test your code locally without installing Hadoop or run it on a cluster of your choice. Once you're set up, it's as easy to run your job in the cloud as it is to run it on your laptop. MrJob ultimately makes MapReducing Jobs easier because:

1. It keeps all MapReduce code for one job in a single class.
2. It switches input and output formats with a single line of code
3. It automatically downloads and parses error logs for Python tracebacks
4. It puts command line filters before or after your Python code

1.1.1 Installation

Installing MrJob required only one statement - "pip install mrjob" to be executed, hence installation was very easy. It took a while to grasp some aspects of the documentation, and to practice writing my first job.

1.1.2 Word Counting Algorithm

The word count algorithm counts the frequency of words for one step job. The algorithm first imports MRJob from the mrjob.job module and inherits its class and regular expressions to be able to identify words. It uses methods such as:

1. mapper(self, , line) - which defines the mapper for a one-step job. It takes a key and value parsed from input.
2. combiner(self, word, counts) - which defines the combiner for one-step job. It takes the key which was yielded by the mapper and a value which yields all values yielded. This methods sums the words that have been seen so far.
3. reducer(self, word, counts) - which defines the reducer for one-step job. It outputs a key, value pair consisting of words and their frequencies.

Requirements for running the program include:

1. python3 or later installed

2. Linux system - terminal

A moderate text found in a text file called "input.txt" was used to test our word count algorithm. Therefore, to run the program, use the command:
python3 mrJob_wordCount.py < input.txt.
Kindly note that you can replace the text file with yours to test the algorithm. If run appropriately, the output, consisting of key, value pairs of each word and its frequency in the text will be displayed on the terminal.

1.2 Part2

In this section, I design and implement MapReduce Algorithms for multiplying matrices using MrJob. The matrices involved are square matrices with dimensions N X N, with P as the number of processors, where P is an even square number.

1.2.1 Serial Implementation

The *serial_multiplication.py* file contains the algorithm for the serial implementation of matrix multiplication using the method - *matrix_multiplication*. It returns one matrix generated as both matrix A and B. The matrix created is written to a text file called *matrix.txt*. This was done to ensure easy access of matrix during the MapReduce implementation, in order to easily access performances of the two approaches using the same matrix at a particular time. The resulting matrix C is also written to an output text file called *serial_result.txt*.

To run the serial algorithm, use the command:
time python3 serial_multiplication.py matrix_multiplication.py N, where N is the size of the matrices. Therefore, if you want to run the serial algorithm using N as 8, then the command will be:
time python3 serial_multiplication.py matrix_multiplication.py 8.
The matrix generated will be displayed in *matrix.txt* file. The matrix multiplication result(of the matrix generated multiplied by itself) will be displayed in *serial_result.txt*.

NB: It is important to note that any matrix size specified for the square matrices A and B when executing the serial algorithm will be the same matrices and dimensions used during the MrJob execution, based on retrieval from *matrix.txt*

1.2.2 MrJob Implementation

The rows and columns are each split into $\sqrt{P}$ bands so that the number of tasks in the both the mapping and reduce phases are P = $\sqrt{P} \times \sqrt{P}$ tasks. Matrix A is partitioned into row-order while Matrix B is partitioned into column-order. The file *partitions.py* contains the methods that are responsible for partitioning into row-order and column. The file *mrJob_multiplication.py* contains the algorithm for multiplying the matrices using MrJob. It uses the matrix displayed

in the *matrix.txt* file for its execution. It contains helper methods such as:

1. configure_args(self) - which configures the command line arguments and allows users to type a number of processors to use for the block partitioning.
2. mapper_raw(self,input_file, input_uri) - which maps partitions into key value pairs
3. reducer(self, key, values) - which aggregates results from the mapper and yields local multiplications.

Algorithm 1 mapper_raw(self,input_file, input_uri)

```
//Method maps matrix with block partitioning into key-value pairs
  count ← 1
  for i in row_order(matrix, bands, offset) do
    for j in column_order(matrix, bands, offset) do
      yield str(count), i
      yield str(count), j
    end for
  end for
```

Algorithm 2 reducer(self, key, values)

```
//Method aggregates results from the mapper and yields local multiplications.
  local_array ← []
  value ← []
  for i in values do
    Add i to localarray
  end for
  for row in local_array[:1][0] do
    for col in local_array[1:][0] do
      value.append(row*col)
    end for
  end for
  yield key, value
```

The values are written to a file called *mrjob*_output.

To run the MapReduce algorithm, use the command:
time python3 mrJob_multiplication.py matrix.txt - -P=p, where p is the number of processors. Therefore if we want to run the algorithm with 16 processors, we use the command:
time python3 mrJob_multiplication.py matrix.txt - -P=16

1.2.3 Performance

The table below shows the performance (in seconds) of both serial and parallel(MapReduce) algorithm, as the size of matrices and number of processors increase.

No = N1	P	Serial	MapReduce
16	64	0.0036	0.1301
80	256	0.2707	0.2419
120	576	0.9066	0.6206
280	784	11.6891	5.2446
512	1024	77.7672	26.0642