ELEN4020A: Data Intensive Computing Lab 3

Jared Gautier (820687), Nick Raal (793528), Sasha Berkowitz (818737), Arunima Pathania()

I. Introduction

The objective of the lab was to perform matrix multipication using various frameworks and MapReduce. The MapReduce framework was introduced by Google to support distributed computing on large data sets onto clusters of computers. The data is replicated multiple times in parallel on the system for increased efficiency, reliability and availability. This was done by using the MrJob framework adapted to two different algorithms in the the Python language.

II. Algorithms

The reason for using the MapReduce framework is due to the benefits it provides. MapReduce has the ability to take a query over a data set, divide it, then run the query in parallel over multiple nodes. This benedits in removing the issue of small computers processing data too large to handle, using multiple servers and the Batch processing model.

A. Map

The main purpose of the map function is to generate a key,value pair. The map function takes individual tasks and transforms the input records into intermediate records, which can be processed by the multiplication algorithm. Each of these transformations occur in parallel. The map function shuffles the key, value pairs based on the first key to re-organize the output.

B. Reduce

The reduce function takes the re-organized data from the map function and reduces them to a summarized data set, the desired output. The reduce function, in terms of this laboratory, performs matrix multiplication using the generated key,value pair.This is done by multiplying the values with the keys and storing them. The final value is the sum of the different products obtained in the previous step.

C. Algorithm A

The first algorithm uses the MrJobs MapReduce framework. The mapping function, *mapFn* shown in algorithm 1, takes the input matrix *.txt* files and generates the key, value pairs in terms of columns for matrix A, and the key, value pairs in terms of rows for matrix B. From these generated pairs, the reduce function, *reduceFn* show in algorithm 2, appends the key, value pairs into an array, then multiplies the matricies. This multiplied value is then added using the *addition* function to get the final result.

1) Reducer:

D. Algorithm B

1) Reducer:

Algorithm 1 mapFn(_ , line)

```
Require: Mapping function to generate key, value pairs
  $row, col, value \leftarrow line.split$
  $filename \leftarrow os.environ["inputfile"]$
  if filename = "matrix1.txt" then
    $yield\ col, (0, row, value)$
  else if filename = matrix2.txt then
    $yield\ row, (1, col, value)$
  end if
```

Algorithm 2 reduceFn(y,value)

```
Require: Uses output of mapFn function to perform matrix multiplication
  rowVals[]
  colVals[]
  for $x \leftarrow 0$ to $values$ do
    if x[0] = 0 then
      rowVals.append(x)
    end if
    if x[0] = 1 then
      colVals.append(x)
    end if
    for $a, b, row \leftarrow 0$to $rowVals$ do
      for $a, key, col \leftarrow 0$to $colVals$ do
        $yield\ (b, key), (int(row) * int(col))$
      end for
    end for
  end for
```

III. Results

A. Algorithm A

For a 10 x 10 matrix multiplied by a 10 x 10 matrix, the algorithm took 1.226556 seconds to compute. For a 100 x 100 matrix multiplied by a 100 x 100 matrix, the algorithm took 20.135463 seconds to compute.

Algorithm 3 The mapper function

```
Require: Map function to produce and return key, value pairs

  for $value\_A \leftarrow 0$ to $A$ do
    $k \leftarrow 1$ to $B$
    $((i, k), (A, j, value\_A))$ for each value of $k$
  end for
  for $value\_B \leftarrow 0$ to $B$ do
    $i \leftarrow 1$ to $A$
    $((i, k), (B, j, value\_B))$ for each value of $i$
  end for
  return $(key, value)$ pair
```

Algorithm 4 The Reducer Function

```
Require: Uses output of Mapper function to perform matrix multiplication
  for $(i, k) \leftarrow 0$ to $A$ do
    sort $(A, j, value\_A)$ by $j$
    sort $(B, j, value\_B)$ by $j$
    $value\_A * j \leftarrow multiA$
    $value\_B * j \leftarrow multiB$
    $Answer \leftarrow multiA + multiB$
    Return $(i, k), Answer$
  end for
```

IV. Unweighted Directed Graphs

Directed graphs involve a series of interconnected nodes, the nodes are connected by edges. In order to calculate all of the groups of paths of length 3, an adjacency matrix data structure must be implemented. Where the adjacency matrix is given by a matrix of ones and zeroes, where ones indicate an edge between a vertex and 0s represent no connection. This can be implemented by means of an adjacency list data structure, which is a list of interconnected vertices that is similar to how a 2D array would be represented using data structures [1]. The preferable data structure would be the data list, this is because the space of the list would be 0(n+m), as opposed to the adjacency matrix which has a space compexity of Once this data structure is implemented, finding the paths of length three would be searching the list for the links that have a length greater than or equal to three.

References

[1] Author unknown. 2015. *Directed and Undirected Graphs*. [ONLINE] Available at: http://www.inf.ed.ac.uk/teaching/courses/cs2/LectureNotes/CS2Bh/ADS/lecture9.pdf [Accessed 5 April 2018].