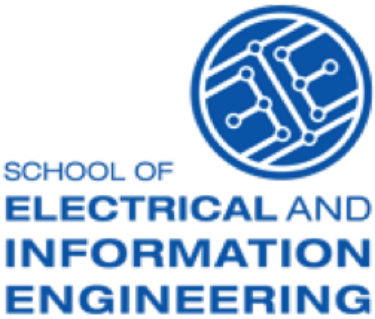

University of the Witwatersrand
School of Electrical and Information Engineering

ELEN4020: Data Intensive Computing

Laboratory Exercise 3

Authors:

Kayla-Jade Butkow
714227

Jared Ping
704447

Lara Timm
704157

Matthew van Rooyen
706692

Date Handed In: 10th April, 2018

1. Matrix Multiplication

The input to the `mapper` function is a text file containing a input matrix written in Matrix-Market format. The first row of the file provides the dimensions of the matrix. The following rows contain three columns, with the first two representing the row and column index of the element and the third the value of the element.

The product matrix is represented as $P_{ik} = M_{ij} * N_{jk}$, where M_{ij} and N_{jk} are the input matrices. In order to perform the multiplication, the first row of M is multiplied by the first column in N and the resultant values are summed. This process is performed for every combination of rows in M and columns in N.

2. MapReduce & MrJob

In order to perform matrix multiplication for large matrices, a MapReduce algorithm was implemented. The algorithm consists of two distinct parts, namely the `mapper` function and `reducer` function.

The `mapper` function takes as an input a key and a value and emits key-value pairs. The `reducer` function then makes use of the output from the `mapper` function. It takes as inputs a key and a list of values associated with the key and performs an action of the values. It also emits key-value pairs.

MrJob is a MapReduce framework which provides an environment which allows multi-step MapReduce jobs to be written in Python and run on a variety platforms [1]. A *job* is defined by class which inherits from `MRJob`, and which contains the functions which define the *steps* of the job. Each *step* consists of a `mapper`, a `combiner` and a `reducer`, of which you must use at least one [1].

The input data is read in one line at a time. The `mapper` function performs the role of map in MapReduce allocating key-value pairs to each line of text input. Similarly, the `reducer` performs the reduce task in order to output a single value for each key-value pair. Each yield function found at the end of the mapper and reducer is responsible for a single line of output code.

MrJob is intended as an easy way to write Python programs that run on Hadoop. Using MrJob, one is able to test the code locally, without installing Hadoop, before you deploy the code on your cluster. For the purpose of this laboratory, this local approach was used. Additionally, MrJob was preferred over other Python MapReduce libraries owing to its relative ease of implementation and availability of documentation.

3. Multiplication

Two algorithms for performing matrix multiplication with MapReduce were implemented. Two input matrices are used as input to the job.

In both the algorithms, the `mapper` function transforms the input file into a format which can be used by the `reducer` to perform the necessary calculations. The input data to the `mapper` function is a line from the text file in the form *row column value*. The data is then assigned a key so that a key-value pair can be yielded from the function.

The `reduce` function performs computations on the key-value pairs produced by the `mapper` function in order to produce the product matrix.

The implemented algorithms make use of the MapReduce framework in two different ways. The first uses multiple mapping and reducing stages, while the second only makes use of a single mapper stage with two reducer stages. The use of these algorithms allows one to determine whether having more MapReduce steps results in a performance increase, or if having more computations within fewer steps performs better.

3.1 Algorithm A

In Algorithm A, two MapReduce steps are used, meaning that a `mapper` and `reducer` stage are used, followed by a second `mapper` and `reducer` stage.

In the first step, the algorithm takes in two matrices. The first input matrix is represented as $M(I, J, V)$ with tuples (i, j, m_{ij}) and the second as $N(J, K, W)$ with tuples (j, k, n_{jk}). These tuples represent the (row index, column index, value).

The value of each element in the two matrices is assigned a key according to it's j value which is common to both matrices. For the first matrix (M), the key j is emitted corresponding to the value $(0, i, m_{ij})$, where 0 is used as an identifier for matrix M. Likewise, for matrix N, the key j is emitted and its corresponding value is $(1, k, n_{jk})$. For matrix N, 1 is used as an identifier. Since only one input matrix is used, for each line of the input file, both of the above mentioned key-value pairs are emitted.

In the reduce function in the first step, the values corresponding to the key are divided into two matrices: matrix0 and matrix1 according to the first entry of the value. If the first entry of the value is 0, the value is appended onto matrix0 and likewise for matrix1. Thereafter, for each key j, the corresponding values (m_{ij} and n_{jk} from matrix0 and matrix1 respectively) are multiplied and the key-value pair j, $(i, k, m_{ij}n_{jk})$ is emitted.

The output of the reduce function is then used as the input to the map function in the second step.

In the second step, a key-value pair of the form (i,k) is created for each element and assigned the combined value $m_{ij}n_{jk}$ in the mapper stage. The i value represents the row index of the product matrix and k represents the column index.

In the final reducer stage, all the values corresponding to each key are summed. This allows for the resultant value at each matrix index to be calculated. Then, a final key-value pair is emitted where the key represents the row index and column index of the entry in the product matrix, and the value is the final value at each index.

3.2 Algorithm B

Algorithm B makes use of a combined map and reduce step and a reduce step. In the `mapper` stage, for matrix M, the key-value pair is $((i, k), (0, j, m_{ij}))$ for each value m_{ij} of M. The variable k goes from 1 to the number of columns of N. In order to implement this, a for loop was created ranging from 0 to the number of columns of in. Within the for loop, a key-value pair is emitted for each value of k.

Likewise, for matrix N, the key-value pair is $((i, k), (1, j, n_{jk}))$ for each value n_{jk} of N. The variable i goes from 1 to the number of rows of M. The key-value pairs were then emitted as described above.

The output of the map function is then used as the input to the reducer.

In the first reducer, the key (i,k) represents the row and column index of the product matrix. First, the values corresponding to the key are divided into two matrices: matrix0 and matrix1 according to the first entry of the value. If the first entry of the value is 0, the value is appended onto matrix0 and likewise for matrix1. Next, for each key, the values in matrix M are sorted according to their j values. This is then repeated for matrix N. Finally, the values m_{ij} and n_{jk} at the j_{th} index of each list are the multiplied together for each value of j.

In the second reducer, the values corresponding to each key are calculated. Then, a final key-value pair is emitted where the key represents the row index and column index of the entry in the product matrix, and the value is the final value at each index.

4. Pairs of Nodes Connected by Paths of Length 3

4.1 Overview

According to [2], the total number of paths of length 3 from j to i in G is:

$$N_{ij}^3 = \sum_{k,l=1}^{n} A_{ik}A_{kl}A_{ij} = [G^3]_{ij} \tag{1}$$

Where: $G = (v, e)$ is an unweighted directed graph of $\mid v \mid$ nodes and $\mid e \mid$ edges

From Equation 1, it is evident that the most efficient way to represent G is in matrix form. Since the majority of the values in the G matrix are 0, all zero value entires should be emitted from the input file.

From Equation 1, in order to calculate the pairs of nodes that are connected by paths of length 3, the input matrix *File2ForLab3.txt* must be cubed.

In order to perform the calculation, multiplication algorithm A was used. First, the input text file was multiplied with itself to result in A^2. Then, the resulting matrix was multiplied with the input text file to result in A^3. In order to count the number of paths, the resulting matrix entries were summed, and this value indicates the total number of paths of length 3.

4.2 Implementation

Since the input file is a symmetric matrix given in upper triangular form, the lower triangle must first be produced. This is done by letting M_{ji} equal M_{ij}, where i is the row index and j is the column index. The input file with the full matrix is then used as the input to the procedure to determine the number of paths of length 3.

In order to implement the above mentioned procedure, two multiplication classes were required. The first (*lab3-multiplicationPart1.py*) makes use of Algorithm A as discussed in *Section 3.1.* The result of the first multiplication is written to the text file *partialOutputQ6.txt.* The second class (*lab3-multiplicationPart2.py*) makes use of Algorithm A, but takes two in input files. This is necessary for the second multiplication in which the resultant matrix is multiplied by the input matrix. In this class, the text files are differentiated by name using the MRJob `os.environ['mapreduce_map_input_file']` function which gives the program access to the input file name. By using the file name, all data that originates from the file *File2ForLab3.txt* is assigned a 0 as an identifier, and all data from the other file is assigned a 1.

To calculate the total number of paths of length 3, a third mapper and reducer stage was added to the second class. In the mapper stage, all the entries of the resulting matrix were assigned the key *Number of paths:* . This ensures that all the values have the same key and can thus be summed in the reducer. Within the reducer stage, the values are summed and the final count is written to the output file.

In order to run the two classes, a bash script was created. Within this script, the two classes are called with their respective input files. The final resulting matrix is written to the text file *outputQ6.txt.*

5. Results

Table 1: Time taken to multiply outA1 and outB1 matrices using Algorithm A and B

Algorithm	**Time taken (s)**
A	52.20356583595276
B	106.5891628265380

Table 2: Time taken to multiply outA2 and outB2 matrices using Algorithm A and B

Algorithm	**Time taken (s)**
A	25809.8268780708
B	

Table 3: Time taken to multiply outA3 and outB3 matrices using Algorithm A and B

Algorithm	**Time taken (s)**
A	47.3928279876709
B	91.8078939914703

Table 4: Time taken to determine number of paths by cubing outNetwork matrix using Algorithm A

Algorithm	**Time taken (s)**
A	2105.412

From the above results, it is evident that Algorithm A far outperforms Algorithm B. For all of the input matrices, the time taken to run Algorithm A is less than half that of Algorithm B. It can thus be concluded that an algorithm that makes use of more MapReduce steps has a far superior performance than one that has more computations in fewer steps.

REFERENCES

[1] S. Johnson. "mrjob Documentation." Release 0.6.2.dev0.
[2] M. Newman. *Networks: An Introduction*, p. 136. Oxford, 2010.