MAPREDUCE FRAMEWORK IMPLEMENTATION IN C++ FOR MATRIX MULTIPLICATION

Massi Mapani (783767) Thabiso Magwaza (836403)

School of Electrical & Information Engineering, University of the Witwatersrand, ELEN4020A Electrical Engineering Data Intensive Computing

Abstract:

Key words:

1. Introduction

Being able to process large amounts of data; who's results require access in a couple of seconds has become a necessary commodity in social media, banking, cellular networks, etc. Millions to billions of people use an application, transfer money, communicate from one continent to another, all the time and service providers need to provide speedy services in this regard to stay competitive.

"Big Data" is characterized by the volume, value, variety and velocity of the data[1]. Large quantities of data for processing required large storage space and take up a lot of computation time for processing the data, these were the problems that were evident with "Big Data". To combat the processing computation time length, frameworks were made that are able to break down the large amounts of data, then process it with sub-tasks, distribute the subtasks to parallel computation tools; which are considerable fast and then present the results of the process in a practically efficient time.

MapReduce is a framework that is able to process large amounts of data in parallel and is the focus of this report.It's different implementations and their efficiency are compared in the implementation of matrix multiplication.

2. MapReduce

MapReduce is a programming model and a associated implementation for processing and generating large data sets. The programmer of a MapReduce program has the task of specifying two functions namely the *map* and *reduce* functions. In it's simplest for, the *map* functions take in a set of *(key,value)* pairs and produces another set of intermediate *(key,value)* pairs. These are then sent to a *reduce* function, also specified by the user, who's task is to combine the set of intermediate *(key,value)* pairs key wise to give an output of final *(key,value)* pairs. Users can specify multiple MapReduce steps depending on the application of their program. Programs written in the MapReduce funtional style are automatically parallelized and executed on a large cluster of commodity machines. MapReduce takes care of partitioning the input data, co-ordinating the program's execution over many machines, handling machine failure and managing the communication between the machines. Most implementation of MapReduce are designed to be highly scalable and operate using cheap commodity machines. MapReduce allows for programmers with little experience in parallel programming to take advantage of parellelization and code execution of clusters of machines.

3. Matrix Multiplication

Matrix multiplication is a common problem that fits perfectly into the MapReduce style of solving problems. Given two matrices M and N with the elements in each matrix denoted by m_{ij} and n_{jk} respectively, the elements in the product matrix $C = M \times N$ are given by $\sum_j m_{ij} n_{jk}$. This product can be implemented in MapReduce using two algorithms: one which uses two MapReduce steps and another does it in one. The pseudo-code for the algorithms are presented in Algorithm 1 and Algorithm 2.

Algorithm 1 Matrix multiplication with two MapReduce steps

```
Map step 1:
For each $m_{ij}$ in M
  $emit(j, (M, i, m_{ij}))$
ENDFOR
For each $n_{jk}$ in N
  $emit(j, (N, k, n_{jk}))$
ENDFOR
Reduce step 1:
For each key $j$
  $emit((i, k), m_{ij} n_{jk})$
ENDFOR
Map step 2:
For each key $(i, k)$
  $emit((i, k), v)$
ENDFOR
Reduce step 2:
For each key $(i, k)$
  $emit((i, k), sum(v))$
ENDFOR
```

4. Evaluation and Results

The two MapReduce algorithms were implemented using the python implementation of MapReduced MRJob. The algorithms were tested first on small matrices to verify the correctness of the answers they produce and make estimates about their performance on larger (big data) ma-

Algorithm 2 Matrix multiplication in one MapReduce step

```
Map step:
For each $m_{ij}$ in M
    For each $k$ in N
        $emit((i,k),(M,j,m_{ij})$
    ENDFOR
ENDFOR
For each $n_{jk}$ in N
    For each $k$ in M
        $emit((j,k),(M,j,n_{jk}))$
    ENDFOR
ENDFOR

Reduce: For each key (i,k)
    $sort$ values begin with M by $j$ in $list_M$
    $sort$ values begin with N by j in $list_N$
    multiply $m_{ij}$ and $n_{jk}$ for $j_{th}$ value for each list
    sum up $m_{ij} \times n_{jk}$
    $emit((i,k), sum_j m_{ij} \times n_{jk})$
ENDFOR
```

trices. Algorithm 2 performed better than Algorithm 1 for the smaller matrices and the algorthms were found to implement the matrix multiplication correctly. The algorithms were then tested on relatively larger matrices and their performances in terms of time are presented in Tables 1, 2 and 3. The algorithms were run in the Hornet01 lynx machine which resulted is some of the matrices running an indefinite amount of time due to the machine's hardware limitations. The trend, however, is seen to be that even for large matrices Algorithms 2 runs faster than 1. This is because although Algorithm 2 has more work to do in the mapping and reduction steps it still does less MapReduce steps which is enough to make it run faster than Algorithm 1.

Table 1 : Performance table for Matrix product A[100][100] and B[100][100]

Algorithm-A	83.67s
Algorithm-B	38.28s

Table 2 : Performance table for Matrix size A[1000][500] and B[500][1000]

Algorithm-A	indefinite
Algorithm-B	indefinite

Table 3 : Performance table for Matrix size A[1000][500] and B[500][1]

Algorithm-A	20min
Algorithm-B	12min

5. Conclusion

MapReduce has been applied to the problem of matrix multiplication. Two algorithms that solve the problem using one and two MapReduce steps respectively are presented. It is found that the algorithm that does the matrxi multiplication is one step runs faster than the one that does it in two steps due the it having less steps in the the computation.

REFERENCES

[1] Aditya Joshi. Hadoop - a gentle introduction, 2013.