Cloud Computing Practical 4
Map-Reduce Programming Model

Eoghan Hogan
17335293
Eoghan.hogan@ucdconnect.ie

October 23, 2020

Contents

1 Part 1

1.1 Part 1 - Question 1 - Calculate Size MapReduce Signatures

The Mapper Function will take in the Corpus Name and the Corpus (As it expects key, value pairs)

It will output the URL of a page in the Corpus along with its Size

Algorithm 1 The Mapper Function Input and Output

```
procedure MAP(CorpusName, Corpus)
                                        ▷ Do something to get the url and size...
    emit (HostURL, Size)
end procedure
```

The reduce Will take the Urls and Size and emit the sum of the sizes for specific URL's

Algorithm 2 The Reducer Function Input and Output

```
procedure REDUCE(HostURL, Sizes)        ▷ Sizes will be an iterator
    emit (HostURL, sizeSum)
end procedure
```

1.2 Part 1 - Question 2 - Calculate Size MapReduce

The implementations of the MapReduce algorithms is fairly simple we simply need to go through the corpus and filter the lines to get the information we are looking for.

Algorithm 3 The Mapper Function

```
procedure MAPPER(corpusName, Corpus)
    for line in Corpus do
        HostURL ← split(line, ",")[0]
        Size ← split(line, ",")[1]
        Size ← bytes(Size)
        emit (HostURL, Size)
    end for
end procedure
```

In the reducer function we just iterate over the values and do the summation as required.

Algorithm 4 The Reducer Function

```
procedure REDUCER(HostURL, Sizes)                ▷ Sizes will be an iterator
    sizeSum ← 0
    for bytes in Sizes do
        sizeSum ← bytes + sizeSum
    end for
    emit (HostURL, sizeSum)
end procedure
```

1.3 Part 1 - Question 3 - Multiplication Signatures

For matrix Multiplication done via Map reduce things get a bit more Complicated. I found it best to start with the Output Matrix and work my way back. to give a little visual help to the algorith lets start from the end and work our way backwards.

So we have two Matrices A and B

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, B = \begin{pmatrix} e & f \\ g & h \end{pmatrix}$$

and the product is this:

$$A * B = C = \begin{pmatrix} a*e+b*g & a*f+b*h \\ c*e+d*g & c*f+d*h \end{pmatrix}$$

But we can rewrite the product by indexing the original Matrices

$$C = \begin{pmatrix} A_{0,0} * B_{0,0} + A_{0,1} * B_{1,0} & A_{0,0} * B_{0,1} + A_{0,1} * B_{1,1} \\ A_{1,0} * B_{0,0} + A_{1,1} * B_{1,0} & A_{1,0} * B_{0,1} + A_{1,1} * B_{1,1} \end{pmatrix}$$

okay now lets say we want to compute
$A_{0,0} * B_{0,0} + A_{0,1} * B_{1,0}$

but how do we know which $A_{0,0}$ or $A_{0,1}$ we are dealing with? they both appear twice! the row of the final matrix. lets rewrite the final product but include the indexes also

$$C = \begin{pmatrix} (0,0)A_{0,0} * B_{0,0} + A_{0,1} * B_{1,0} & (0,1)A_{0,0} * B_{0,1} + A_{0,1} * B_{1,1} \\ (1,0)A_{1,0} * B_{0,0} + A_{1,1} * B_{1,0} & (1,1)A_{1,0} * B_{0,1} + A_{1,1} * B_{1,1} \end{pmatrix}$$

Now the pattern easily jumps out of us. When we are finding the answer for $C_{i,j}$ we are going to be using all $A_{i,k1}$ and all $B_{k2,j}$. this means we will need to replicate every value in a the same number of times as columns in B since that's the amount of Columns in C and every value in B the same number of times of Rows in A as that's the number of Rows in C

Algorithm 5 The Matrix Multiplication Mapper Function Signature

▷ A is a LxM matrix anf B is an MxN matrix
procedure MAP(A, B)
▷ duplicate values in A as many times as columns in B
▷ k = column index B, i = row index A, j = col index A
    emit $((i, k), (\text{"}A\text{"}, j, A_{i,j}))$
▷ duplicate values in B as many times as rows in A
▷ i = row index A, j = row index B, i = col index B
    emit $((i, k), (\text{"}B\text{"}, j, B_{k,j}))$
end procedure

For the reducer we now know we are going to take in values that look like $("M", j, M_{(iork),j}))$ and that all our Values that the reduce takes in will represent one value in the final matrix. so we need to sort the Values to so that we end up all the A values in one array and all the B values in another

Then we can simple do this with a for loop over j and indexing the arrays:
$A_{i,j} * B_{k,j} + A_{i,j} * B_{k,j}$

Algorithm 6 The Matrix Multiplication Reducer Function Signature

▷ positions= (row, col), tuples = (matrix_id, idx, value)
procedure REDUCE(*positions*, *tuples*)
▷ sort values by matrix
▷ do the summation and multiplication
    emit (*positions*, S)
end procedure

1.4 Part 1 - Question 4 A - Matrix Multiplication Mapper

Algorithm 7 The Matrix Multiplication Mapper Function

```
                                   ▷ A is a LxM matrix and B is an MxN matrix
procedure MAP(A, B)
    L, M ← shape(A)
    M, N ← Shape(B)
                         ▷ duplicate values in A as many times as columns in B
    for k ← 0, N do
        for i ← 0, L do
            for j ← 0, M do
                   ▷ k = column index B, i = row index A, j = col index A
                emit ((i, k), ("A", j, A_{i,j}))
            end for
        end for
    end for
                           ▷ duplicate values in B as many times as rows in A
    for i ← 0, L do
        for j ← 0, M do
            for k ← 0, N do
                  ▷ i = row index A, j = row index B, k = column index B
                emit ((i, k), ("B", j, B_{j,k}))
            end for
        end for
    end for
end procedure
```

1.5 Part 1 - Question 4 B - Matrix Multiplication Reducer

Algorithm 8 The Matrix Multiplication Reducer Function

```
                              ▷ positions= (row, col), tuples = (matrix_id, idx, value)
procedure REDUCE(positions, tuples)
    l ← length(tuples)/2
    a_vals ← array(0, l)
    b_vals ← array(0, l)
                                                                  ▷ sort values by matrix
    for t in tuples do
        idx ← t[1]
        if t[0] = "A" then
            a_vals[idx] ← t[2]
        else
            b_vals[idx] ← t[2]
        end if
    end for
                                            ▷ do the summation and multipication
    s ← 0
    for i in 0, l do
        s ← s + (a_vals[i] ∗ b_vals[i])
    end for
    emit (positions, s)
end procedure
```

1.6 Part 1 - Question 5 A - Matrices Don't Fit

What if the Matrices do not fit into The Mappers memory?

Well luckily we can keep the Reducer the same we just need to change the input to the mapper and modify what it does. Basically the strategy to deal with this would be to Pre-Processes the Matrices before suplying them to the map functions.

to Pre-Process matrix A we would need to split off the rows such that they are 1xN 2d Arrays. then we would send that to the mapper with the row index and the array "A" id. We also would need The number of columns of B.

To Pre-Process matrix B we would need to split B by its columns so our mapper would take in the Nx1 array representing the column, the index of that column and the number of rows of A.

Its actually quite simple conceptually when we look at the Original Mapper function.

The original Mapper function went through All of A col(B) times and all of B row(A) times, however this time we don't have all of A or B. we only have vectors from them and we are not even sure how many. we Do know however if the vectors are from A or B.

So thinking about the Original Function. while we don't have the whole matrix we do know which column of row the vector belongs to. so we can fix that value and essentially go over it rows(A) or Cols(B) times like would've happened to that specific vector in the original Function.

Thus we can emit pretty much the same information just with a fixed value

Algorithm 9 The Matrix Multiplication Mapper Function Signature

matrixID is "A" or "B"
tuple[0] = vector that was split off
tuple[1] = col index if B or row index if A
tuple[2] = cols(A) if vector is from A
tuple[2] = Rows(B) if vector is from B
tuple[3] = cols(B) if vector is from A
tuple[3] = Rows(A) if vector is from B
procedure MAP($matrixID, tuple$)
    if $matrixID =$ "A" **then**
        ▷ for each tuple duplicate the values in vector cols(B) times
        ▷ k = column index B, i = row index A, j = col index A
        emit $((i,k),(\text{"}A\text{"}, j, A_{i,j}))$
    else
        ▷ for each tuple duplicate the values in vector rows(A) times
        ▷ i = row index A, j = row index B, i = col index B
        emit $((i,k),(\text{"}B\text{"}, j, B_{k,j}))$
    end if
end procedure

1.7 Part 1 - Question 5 B - New Mapper

This is the implementation of the Mapper function I discussed on the last page.

Algorithm 10 The Matrix Multiplication Mapper Function

```
procedure MAP(matrixID, tuple)
    if matrixID = "A" then
                ▷ for each tuple duplicate, the values in vector, cols(B) times
                    ▷ k = column index B, i = row index A, j = col index A
        i ← tuple[1]
        for t in tuple do
            vec = t[0]
            for k ← 0, t[3] do
                for j ← 0, t[2] do
                    emit ((i, k), ("A", j, A_{0,j}))
                end for
            end for
        end for
    else
                ▷ for each tuple duplicate, the values in vector, rows(A) times
                    ▷ i = row index A, j = row index B, k = column index B
        k ← tuple[1]
        for t in tuple do
            vec = t[0]
            for i ← 0, t[3] do
                for j ← 0, t[2] do
                    emit ((i, k), ("B", j, B_{j,0}))
                end for
            end for
        end for
    end if
end procedure
```

2 Part 2

2.1 Part 2 - Question 1

From the lecture notes We have the following Function Signatures which I have changed a bit to make more sense to me

K means makes alot more sense than matrix multiplication and thus is easier to follow.

The Mapper will take in a data point and the list of K centroids. then all wee need to do is compare the distance from each centroid to the data point and return the centroid that is closest to the data point.

Algorithm 11 The K-Means Mapper Signature

procedure MAP($dataPoint, [centroid]$)
    emit $(centroid, dataPoint)$
end procedure

The Reducer will take then take the centroids as the keys so that all the datapoints that were closest to that centroid will be the values. The all we need to do is calculate the new centroid by getting the mean of the datapoints.

Algorithm 12 The K-Means Reducer Signature

procedure REDUCE($centroid, [dataPoint]$)
    emit $(centroidornewCentroid, [dataPoints])$
end procedure

we Then continue this until there is no change in the centroid position

2.2 Part 2 - Question 2

The implementation details of the Mapper and Reducer Functions

Algorithm 13 The K-Means Mapper

procedure MAP($dataPoint, [centroids]$)
  $min \leftarrow centroids[0]$
  for $c \leftarrow centroids$ **do**
    if **Distance**($dataPoint, c$) $\leq$ **Distance**($dataPoint, min$) **then**
      $min \leftarrow c$
    end if
  end for
  emit ($min, dataPoint$)
end procedure

Algorithm 14 The K-Means Reducer

procedure REDUCE($centroid, dataPoints$)
  $newCentroid \leftarrow$ **mean**($dataPoints$)
  if $centorid \neq newCentroid$ **then**
    emit ($newCentroid, dataPoints$)
  else
    emit ($centroid, dataPoints$)
  end if
end procedure