HOMEWORK 6

EXERCISE ON MAPREDUCE

Submitted by

Arun Rajagopalan

A20360689

For CS422 Data Mining

Illinois Institute of Technology

EXERCISE ON MAPREDUCE

You are given a large number of files with integers. Design and write very detailed pseudocode for the solution of the following problem:

- *Find the average for those numbers across all files.*
- *Write the pseudocode for the mapper, reducer, provide very detailed comments for each line.*

MapReduce is a paradigm of programming and an associated implementation that enables us to massively scale our operations across several distributed systems that can function concurrently. In other words, it is used for manipulating and creating huge datasets with a parallel, distributed algorithm on a clustered system such as Hadoop or YARN.

As the name suggests, this model usually consists of a Mapper characterized by a map() method and a Reducer characterized by a reduce() method. Map method is used for filtering items and to sort them while the Reduce method is used to summarize the results of Map method. We can illustrate the working of this model by designing a mapreduce solution for the given problem of finding average of all integers in the files provided.

The high level approach to solve the problem is to have a mapper and a combiner for every document and have a single reducer to summarize the results of all the mapper-combiner pairs. Combiners are characterized by a function called combine which does exactly what the reduce method does, with the difference being the scale. Combiners reduce the network bandwidth usage and contention which shall be discussed later. It outputs summary of each document while the Reducer outputs summary of all documents. Summary operation is the calculation of sum and count in case of Combiners and average calculation in case of Reducers. Pseudocode of the mappers, combiners and reducers that solve the given problem are as follows.

Mapper:
The mappers get a file name or ID and its content as input. Let us assume that our file is delimited by comma. A custom function called parse is used to get every number from the block of content by iteratively splitting the block on every instance of a comma. Mapper emits all numbers in the provided document with the document id as key. We need not emit a count of 1 for each number, as we know mapper emits every occurrence of every number once.

Input: Key-Document ID, Value-Data block in the document
Output: Key-Document ID, Value-Every occurrence of every number

Pseudocode:

```
class Mapper
        method Map (docid id, block b)

                # Declaring and initializing a boolean value to true
                bool bl = true

                # Iterate as long as bl is true
                while(bl)
                        # Split the input block on instance of delimiter (assumed to
                        # be comma here), saving the value at the left of delimiter
                        # which is a number to n and saving the rest of block to
                        # residue. The function returns 1 when delimiter is found and
                        # 0 if not. This return value is stored in ret.
                        ret, n, residue = parse(b, ",")
                        # If delimiter is found, emit the number with docid as key
                        # Both num and count types are of datatype integer
                        if ret == 1
                                Emit (docid id, num int(n))
                                b = residue
                        # If delimiter is not found, then the residue has only a single
                        # number. Emit the residue with docid as key
                        else
                                Emit (docid id, num int(residue))
                                bl = false
```

Combiner:

Combiners get a document ID and an array of numbers after the mapper output is grouped. Sum of all numbers in the array is calculated and a tuple of this sum and count of numbers in the array is given as output value. Output key is going to be 1, since we need reducer to summarize sum and count values coming from all document handlers and the reducer will do so, only if all the reducer inputs have the same key value.

Combiners are mainly employed to use networks efficiently. If we directly sent all the mapper outputs to reducer, that will consume lot of bandwidth and might cause network contention. But if we summarize the mapper outputs at combiners and sent only a summary per document to reducers, then we would have relatively less network usage. Usually combiners are present along with mappers and hence network utilization to transfer data from mappers to combiners is minimal.

Input: Key-Document ID, Value-Array of numbers in the document
Output: Key-1, Value-Tuple containing sum and count of all numbers in the given document

Pseudocode:

```
class Combiner
        method Combine(docid id, array [num n, num l, num n, num f, ...])

                # Declaring and initializing integer variables to store the sum and
                # count of numbers in the given document
                sum s = 0
                count c = 0

                # For each number in the input array, add the number to the sum
                # variable s and increment the count variable c by 1
                for all numbers nm in [num n, num l, num n, num f, ...] do
                        s = s + nm
                        c++

                # Emit a key of 1 with a tuple of sum and count of the document so
                # that reducer can summarize this (sum, count) tuple from all
                # combiners.
                Emit (key 1, tuple(sum s, count c))
```

Reducer:

For all the mapper-combiners pairs (one for each document), there will be a single reducer alone. Reducer gets and input key of 1 with an array of (sum, count) tuples from each document for value. Reducer sums up all document sum and count values and divides the total sum by total count to get average of all numbers in all files.

Input: Key-1, Value-Array of tuples with sum and count of each document
Output: Key-Null, Value-Average of all numbers in all documents

Pseudocode:

```
class Reducer
        method Reduce(key 1, array_of_tuples [(sum s1, count c1), (sum s2, count
c2), (sum s2, count c2), ….])

                # Declaring and initializing integer variables to store the total sum
                # and total count of numbers in all documents and a float variable
                # to store the average of numbers in all documents
                sum total_s = 0
                count total_c = 0
                average avg = 0

                # For each tuple in input array, add the sum to the summary sum
                # variable total_s and count to the summary count variable total_c
                for all tuples (s, c) in [(sum s1, count c1), (sum s2, count c2), (sum
s2, count c2), ….] do
```

```
        total_s = total_s + s
        total_c = total_c + c

    # Calculate average from the total sum and total count of numbers
    # in all documents and store it in the float variable avg.
    avg = total_s / total_c

    # Emit the average as value. Key is insignificant since there are no
    # further operations and a null can be emitted for key.
    Emit (Null, average avg)
```

Example:

Lets us consider two files A and B. A has 1, 32, 54, 76, 98 as data block and B has 2, 43, 68, 87 as data block. The inputs and outputs of mappers, combiners and reducers are as follows.

	Input	Output
Mapper for A	(A, file content)	(A,1), (A,32), (A,54), (A,76) and (A,98)
Combiner for A	(A, [1 32 54 76 98]) after group by key	(1,(261,5)) where 261 is the sum of all numbers in A & 5 is the count of integers in A.
Mapper for B	(B, file content)	(B,2), (B,43), (B,68) and (B,87)
Combiner for B	(B, [2 43 68 87]) after group by key	(1,(200,4)) where 261 is the sum of all numbers in A & 5 is the count of integers in A.
Reducer	(1,[(261,5),(200,4)]) after group by key	(Null, 51.23)

Special Case:

Lets us consider two files C and D with same number count. C has 3, 23, 45, 77, 76 as data block and D has 4, 31, 51, 81, 101 as data block. For cases like these, we can redesign our combiners and reducers as below.

Combiners:
Instead of sending a tuple of sum and count to reducer, we can calculate the average of every document in the combiner itself and send only (1, average) to reducer. This also minimizes network usage by a small degree. **But note that this approach will work only if all the files have same number count.**

Input: Key-Document ID, Value-Array of numbers in the document
Output: Key-1, Value-Average of numbers in each document

Pseudocode:

```
class Combiner
    method Combine(docid id, array [num n, num l, num n, num f, ...])
        sum s = 0
        count c = 0
        # Declaring and initializing one more float variable to calculate and
        # and store average of each document in combiner itself
        average avg = 0

        for all numbers nm in [num n, num l, num n, num f, ...] do
            s = s + nm
            c++

        # Calculate average from the sum and count of numbers
        avg = s / c

        # Emit a key of 1 with average of document as value
        Emit (key 1, average avg)
```

Reducers:

In this case, our reducers will only need to sum all the input averages and divide the sum by number of files to get the final average.

Input: Key-1, Value-Array of tuples with sum and count of each document
Output: Key-Null, Value-Average of all numbers in all documents

Pseudocode:

```
class Reducer
    method Reduce(key 1, array [avg1, avg2, avg3, ...])

        # average is of float datatype and count is of int datatype
        average sum_avg = 0
        average final_avg = 0
        count c_of_files = 0

        # For each average in input array, add that to the total average and
        # increment the count of files by 1
        for all averages a in [avg1, avg2, avg3, ...] do

            sum_avg = sum_avg + a
            count c_of_files++

        # Calculate final average by dividing the sum of averages by number
        # of files
        final_avg = sum_avg / c_of_files

        # Emit final average as value. Key is insignificant as there are no
```

 # further operations and a null can be emitted for key.
Emit (Null, average final_avg)

The inputs and outputs of mappers, combiners and reducers for files C and D are as follows. Remember C has 3, 23, 45, 77, 76 as data block and D has 4, 31, 51, 81, 101 as data block

	Input	Output
Mapper for A	(C, file content)	(C,3), (C,23), (C,45), (C,77) and (C,76)
Combiner for A	(C, [3 23 45 77 76]) after group by key	(1,44.8) where 44.8 is the average of all numbers in C
Mapper for B	(D, file content)	(D,4), (D,31), (D,51), (D,81) and (D,101)
Combiner for B	(D, [4 31 51 81 101]) after group by key	(1,53.6) where 53.6 is the average of all numbers in D
Reducer	(1,[44.8,53.6]) after group by key	(Null, 49.2) where 49.2 is the sum of 44.8 and 53.6 divided by 2.

Thus, the mapreduce paradigm is studied and illustrated with examples.