Map-Reduce Algorithm

The pseudo code for the different map reduce patterns is as follows

No-Combiner Approach:

```
Method map (Key k, Value v):
    Extract StationId, RecordType, temperature from v
    If (recordType = "TMAX" or recordType = "TMIN"):
            emit (stationId, (recordType, temperature,1);

Method reduce (Key k, values [(recordType1, temperature1,1),……. ]):
     //key is the station-id
    // value consists of record-type , temperature reading and count
      maxTempCount ← 0
      maxTempSum ← 0
       minTempCount ← 0
       minTempSum ←0
       minAverage ← 0
       maxAverage ← 0

       for each record v in values:
            if (v.recordType = "TMAX"):
                maxTempSum ← maxTempSum + v.temperature
                maxTempCount ← maxTempCount + 1

            if (v.recordType = "TMIN"):
                minTempSum ← minTempSum + v.temperature
                minTempCount ← minTempCount + 1

      minAverage ← minTempSum/minTempCount
      maxAverage ← maxTempSum/maxTempCount
       emit(stationId, (minAverage, maxAverage))
```

Combiner Approach:

```
Method map (Key k, Value v):
      Extract StationId, RecordType, temperature from v
      If (recordType = “TMAX” or recordType = “TMIN”):
                emit (stationId, (recordType, temperature,1);
end


Method combine ((Key k, values [(recordType1, temperature1,1),……. ])):
       // key consists of station-id
      // value consists of record-type, temperature reading and count
        maxTempCount ← 0
        maxTempSum ← 0
        minTempCount ← 0
         minTempSum ←0

         for each record v in values:
              if (v.recordType = “TMAX”):
                   maxTempSum ← maxTempSum + v.temperature
                   maxTempCount ← maxTempCount + 1

              if (v.recordType = “TMIN”):
                   minTempSum ← minTempSum + v.temperature
                   minTempCount ← minTempCount + 1

        emit(stationId, (“TMAX”, maxTempSum))
        emit(stationId, (“TMIN”, minTempSum))
end
```

```
method reduce (Key k, values [(recordType1, temperature1,count1),……. ]):
        // key consists of station-id
       // value consists of record-type, temperature reading and count

         maxTempCount ← 0
         maxTempSum ← 0
          minTempCount ← 0
          minTempSum ←0
         minAverage ← 0
         maxAverage ← 0

          for each record v in values:
                if (v.recordType = “TMAX”):
                    maxTempSum ← maxTempSum + v.temperature
                    maxTempCount ← maxTempCount + v.count

                if (v.recordType = “TMIN”):
                    minTempSum ← minTempSum + v.temperature
                    minTempCount ← minTempCount + v.count

        minAverage ← minTempSum/minTempCount
        maxAverage ← maxTempSum/maxTempCount
        emit(stationId, (minAverage, maxAverage))
end
```

In-Mapper Combining Approach:

```
class map {

        method initialize () :
                H-max ← Initialize HashMap;
                H-min ← Initialize HashMap;

        method map(Key k , Value v):
               sum ← 0
                total ← 0
                Extract StationId, RecordType, temperature from v
                If (recordType = “TMAX”):
                    sum ← H-max{stationId}.temperature + temperature
                    total ← H-max{stationId}.count + 1
                    H-max.add(stationId,(sum,total))

                If (recordType = “TMIN”):
                    sum ← H-min{stationId}.temperature + temperature
                    total ← H-min{stationId}.count + 1
                    H-min.add(stationId,(sum,total))
         end


       method cleanup():

                  for each key in H-max :
                        emit(key, (H-max{key}.sum, H-max{key}.total)

                  for each key in H-min :
                        emit(key, (H-min{key}.sum, H-min{key}.total)
      end


}
```

```
class reduce{

method reduce (Key k, values [(recordType1, temperature1,count1),……. ]):
        maxTempCount ← 0
        maxTempSum ← 0
         minTempCount ← 0
         minTempSum ←0
        minAverage ← 0
        maxAverage ← 0

         for each record v in values:
              if (v.recordType = “TMAX”):
                   maxTempSum ← maxTempSum + v.temperature
                   maxTempCount ← maxTempCount + v.count

              if (v.recordType = “TMIN”):
                   minTempSum ← minTempSum + v.temperature
                   minTempCount ← minTempCount + v.count

        minAverage ← minTempSum/minTempCount
        maxAverage ← maxTempSum/maxTempCount
         emit(stationId, (minAverage, maxAverage))
end

}
```

Secondary Sort :

// map function converts the value into a key and value pair . The key is an object of CustomKey class

```
method map(Key k, value v) :
        from value v extract station-id, year, record-type and temperature
        if(record-type = “TMAX”):
                emit( (station-id,year), (year, 0,0,temperature,1))
        if(record-type= “TMIN”) :
                emit((station-id,year),(year,temperature,1,0,0))
  end
```

//The key is an object of type CustomKey. Custom key consists of two attributes. stationId and year.

```
class CustomKey {

  stationId
   year

  method  compareTo(Custom key k1 , Custom Key k2)
            compare the stationId’s .
            if(same):
               compare year
   end

}
```

//The partitioner takes the key which is of type CustomKey and returns an appropriate partition based on stationId . All records with a particular station-id go to same reducer

```
method partitioner(key):
        return hash(key.stationId)
end
```

//The combiner takes two parameters , key of the type CustomKey and list of values having same key

```
method combiner(key, values[(year, maxTempSum0,maxTempCount0,
          minTempSum0,minTempCount0),…]):

          maxTempSum←0
          minTempSum ←0
          maxTempCount←0
          minTempCount←0

          for each v in values:
                maxTempSum ← maxTempSum + v.maxTempSum
                minTempSum ← minTempSum + v.minTempSum
                maxTempCount ← maxTempCount + v.maxTempCount
                minTempCount ← minTempCount + v.minTempCount

           emit(key,   (year, maxTempSum, maxTempCount, minTempSum,
                minTempCount)

end
```

// The grouping comparator groups data by station id and all records having same stationId are sent to same reducer.

```
method customGroupComparator (Key k1, Key k2):
        //Key consists of station-id and year
        compareValue ← compare(k1.station-id, k2.station-id)
        return compareValue
end
```

//The reduce function takes two parameters . A key of type CustomKey and list of values having same key . Each value contains five components . year , TMIN sum , TMIN, count , TMAX sum, TMAX count. The values are received in the increasing order of year

```
method reduce(key k, values[(year, minTemp, maxTemp, minTempCount,
        maxTempCount)] ):

    maxTempSum←0
    minTempSum ←0
    maxTempCount←0
    minTempCount←0
    year ← key.year

    for each v in values:
        if(v.year is not equal to year)
          emit(key,  (year, maxTempSum, maxTempCount, minTempSum,
              minTempCount)
         maxTempSum ← 0
        minTempSum ← 0
        maxTempCount ← 0
        minTempCount ← 0
  maxTempSum ← maxTempSum + v.maxTempSum
  minTempSum ← minTempSum + v.minTempSum
  maxTempCount ← maxTempCount + v.maxTempCount
  minTempCount ← minTempCount + v.minTempCount

end
```

The mapper emits records in the increasing order of stationId. This is achieved by overriding the inbuilt compareTo function. We defined our own custom compareTo function in CustomKey class . In this compareTo function , we first compare by stationId . If stationId's are equal, then we compare the years. Thus mapper emits records in increasing order of keys
The grouping comparator the groups all the records having similar stationId and sends them to the reducer. Since we have defined our own custom comparator,

the records in the reducer will be already present in the increasing order of year(In custom comparator , we compare by year when stationId's are equal).

Thus we make use of map reduce's sorting ability to prevent explicit sorting of values in reducer. This in-turn eliminates the need for complex data structures needed to sort.

Performance Comparison

The performance for different design patterns of map-reduce for two executions using 6 m4.large machines is as follows

No Combiner :

Execution 1: Execution 1 completed in 88 seconds

```
INFO total process run time: 88 seconds
2017-02-12T04:12:06.771Z INFO Step created jobs: job_1486872401215_0001
2017-02-12T04:12:06.771Z INFO Step succeeded with exitCode 0 and took 88 seconds
```

Execution 2: Execution 2 completed in 84 seconds

```
INFO total process run time: 84 seconds
2017-02-12T04:43:20.296Z INFO Step created jobs: job_1486874379995_0001
2017-02-12T04:43:20.297Z INFO Step succeeded with exitCode 0 and took 84 seconds
```

Combiner Design Pattern :

Execution 1 : Execution 1 completed in 82 seconds

```
INFO waitProcessCompletion ended with exit code 0 : hadoop jar /mnt/var/lib/hadoop/steps/
s-2YL0FHTW...
INFO total process run time: 98 seconds
2017-02-11T21:03:14.458Z INFO Step created jobs: job_1486846755603_0001
2017-02-11T21:03:14.458Z INFO Step succeeded with exitCode 0 and took 98 seconds
```

Execution 2: Execution 2 completed in 82 seconds

```
INFO total process run time: 82 seconds
2017-02-12T05:36:59.869Z INFO Step created jobs: job_1486877623798_0001
2017-02-12T05:36:59.869Z INFO Step succeeded with exitCode 0 and took 82 seconds
```

In-Mapper Design Pattern

Execution 1: Execution 1 completed in 76 seconds

```
INFO total process run time: 76 seconds
2017-02-11T22:29:34.415Z INFO Step created jobs: job_1486851932347_0001
2017-02-11T22:29:34.415Z INFO Step succeeded with exitCode 0 and took 76 seconds
```

Execution 2: Execution 2 completed in 80 seconds

```
INFO total process run time: 80 seconds
2017-02-11T22:41:51.068Z INFO Step created jobs: job_1486852705270_0001
2017-02-11T22:41:51.068Z INFO Step succeeded with exitCode 0 and took 80 seconds
```

Was the Combiner called at all in program Combiner? Was it called more than once per Map task?

Combiner was called on both the executions in combiner program. The entries in log confirm our observations

Combiner Execution 1:

```
    Map-Reduce Framework
            Map input records=30868726
            Map output records=8798241
            Map output bytes=255148989
            Map output materialized bytes=6029332
            Input split bytes=1620
            Combine input records=8798241
            Combine output records=525992
            Reduce input groups=14135
            Reduce shuffle bytes=6029332
            Reduce input records=525992
            Reduce output records=14135
            Spilled Records=1051984
            Shuffled Maps =180
            Failed Shuffles=0
            Merged Map outputs=180
            GC time elapsed (ms)=16585
            CPU time spent (ms)=198280
            Physical memory (bytes) snapshot=17119162368
            Virtual memory (bytes) snapshot=107879714816
            Total committed heap usage (bytes)=15439233024
```

Combiner Execution 2:

```
        Map-Reduce Framework
                Map input records=30868726
                Map output records=8798241
                Map output bytes=255148989
                Map output materialized bytes=6029332
                Input split bytes=1620
                Combine input records=8798241
                Combine output records=525992
                Reduce input groups=14135
                Reduce shuffle bytes=6029332
                Reduce input records=525992
                Reduce output records=14135
                Spilled Records=1051984
                Shuffled Maps =180
                Failed Shuffles=0
                Merged Map outputs=180
                GC time elapsed (ms)=16546
                CPU time spent (ms)=191850
                Physical memory (bytes) snapshot=16824655872
                Virtual memory (bytes) snapshot=107836256256
                Total committed heap usage (bytes)=15233187840
```

What difference did the use of a Combiner make in Combiner compared to NoCombiner?

By using the combiner , we can see that the network traffic is greatly reduced. Since combiner combines the intermediate records, we can see from the logs that number of keys moving between mapper and reducer is reduced. These observations can be seen in logs as shown below

Combiner :

```
    Map-Reduce Framework
            Map input records=30868726
            Map output records=8798241
            Map output bytes=255148989
            Map output materialized bytes=6029332
            Input split bytes=1620
            Combine input records=8798241
            Combine output records=525992
            Reduce input groups=14135
            Reduce shuffle bytes=6029332
            Reduce input records=525992
            Reduce output records=14135
            Spilled Records=1051984
            Shuffled Maps =180
            Failed Shuffles=0
            Merged Map outputs=180
            GC time elapsed (ms)=16585
            CPU time spent (ms)=198280
            Physical memory (bytes) snapshot=17119162368
            Virtual memory (bytes) snapshot=107879714816
            Total committed heap usage (bytes)=15439233024
```

No Combiner :

```
Map-Reduce Framework
        Map input records=30868726
        Map output records=8798241
        Map output bytes=219956025
        Map output materialized bytes=49005105
        Input split bytes=1620
        Combine input records=0
        Combine output records=0
        Reduce input groups=14135
        Reduce shuffle bytes=49005105
        Reduce input records=8798241
        Reduce output records=14135
        Spilled Records=17596482
        Shuffled Maps =180
        Failed Shuffles=0
        Merged Map outputs=180
        GC time elapsed (ms)=17122
        CPU time spent (ms)=211990
        Physical memory (bytes) snapshot=16907845632
        Virtual memory (bytes) snapshot=107945103360
        Total committed heap usage (bytes)=15397289984
```

Was the local aggregation effective in InMapperComb compared to NoCombiner?

Local aggregation in “In-Mapper combining” provides several benefits

a) The number of records that are emitted between mapper and reducer is greatly reduced . We can see these observations in the log records shown below

No Combiner :

```
Map-Reduce Framework
        Map input records=30868726
        Map output records=8798241
        Map output bytes=219956025
        Map output materialized bytes=49005105
        Input split bytes=1620
        Combine input records=0
        Combine output records=0
        Reduce input groups=14135
        Reduce shuffle bytes=49005105
        Reduce input records=8798241
        Reduce output records=14135
        Spilled Records=17596482
        Shuffled Maps =180
        Failed Shuffles=0
        Merged Map outputs=180
        GC time elapsed (ms)=17122
        CPU time spent (ms)=211990
        Physical memory (bytes) snapshot=16907845632
        Virtual memory (bytes) snapshot=107945103360
        Total committed heap usage (bytes)=15397289984
```

In-Mapper Combining:

```
Map-Reduce Framework
        Map input records=30868726
        Map output records=223783
        Map output bytes=8056188
        Map output materialized bytes=4018316
        Input split bytes=1598
        Combine input records=0
        Combine output records=0
        Reduce input groups=14135
        Reduce shuffle bytes=4018316
        Reduce input records=223783
        Reduce output records=14135
        Spilled Records=447566
        Shuffled Maps =153
        Failed Shuffles=0
        Merged Map outputs=153
        GC time elapsed (ms)=19103
        CPU time spent (ms)=153200
        Physical memory (bytes) snapshot=15585722368
        Virtual memory (bytes) snapshot=98027868160
        Total committed heap usage (bytes)=14260109312
```

From the above two images we can see a significant difference in “Map output records” field between In mapper combining and no Mapper combining

b) The time taken to execute in-mapper combining is lesser compared to no-combining. This can be attributed to the fact that less records travel between mapper and reducer. We can see the difference in execution time between the no combiner approach and as follows

In Mapper Combining:

Execution 1: Execution 1 completed in 76 seconds

```
INFO total process run time: 76 seconds
2017-02-11T22:29:34.415Z INFO Step created jobs: job_1486851932347_0001
2017-02-11T22:29:34.415Z INFO Step succeeded with exitCode 0 and took 76 seconds
```

Execution 2: Execution 2 completed in 80 seconds

```
INFO total process run time: 80 seconds
2017-02-11T22:41:51.068Z INFO Step created jobs: job_1486852705270_0001
2017-02-11T22:41:51.068Z INFO Step succeeded with exitCode 0 and took 80 seconds
```

No Mapper Combining :

Execution 1: Execution 1 completed in 88 seconds

```
INFO total process run time: 88 seconds
2017-02-12T04:12:06.771Z INFO Step created jobs: job_1486872401215_0001
2017-02-12T04:12:06.771Z INFO Step succeeded with exitCode 0 and took 88 seconds
```

Execution 2: Execution 2 completed in 84 seconds

```
INFO total process run time: 84 seconds
2017-02-12T04:43:20.296Z INFO Step created jobs: job_1486874379995_0001
2017-02-12T04:43:20.297Z INFO Step succeeded with exitCode 0 and took 84 seconds
```

From the above two images, we can see that in both the executions, in-mapper combining approach improves the execution time.

Which one is better, Combiner or InMapperComb? Briefly justify your answer.

In Map-reduce, the extent to which efficiency can be increased through local-aggregation depends on size of intermediate key space, distribution of keys and number of key-value pairs that are emitted by each individual map task. Generally, In-Mapper combiner is better due to two reasons,

1) The number of records emitted by map function is much lesser in "in-mapper combining" design pattern than in combiner design pattern. This is due to the fact that in "in-mapper" design pattern, data is aggregated before being emitted. We have observed the same in our experiments. Below is the snapshot of the part of log records. We can see that the number of records emitted by map in "in-mapper combining" is significantly lesser than the number of records emitted by map in combiner approach.

In-Mapper Combining

```
Map-Reduce Framework
        Map input records=30868726
        Map output records=223783
        Map output bytes=8056188
        Map output materialized bytes=4018316
        Input split bytes=1598
        Combine input records=0
        Combine output records=0
        Reduce input groups=14135
        Reduce shuffle bytes=4018316
        Reduce input records=223783
        Reduce output records=14135
        Spilled Records=447566
        Shuffled Maps =153
        Failed Shuffles=0
        Merged Map outputs=153
        GC time elapsed (ms)=19103
        CPU time spent (ms)=153200
        Physical memory (bytes) snapshot=15585722368
        Virtual memory (bytes) snapshot=98027868160
        Total committed heap usage (bytes)=14260109312
```

Combiner Approach

```
Map-Reduce Framework
        Map input records=30868726
        Map output records=8798241
        Map output bytes=255148989
        Map output materialized bytes=6029332
        Input split bytes=1620
        Combine input records=8798241
        Combine output records=525992
        Reduce input groups=14135
        Reduce shuffle bytes=6029332
        Reduce input records=525992
        Reduce output records=14135
        Spilled Records=1051984
        Shuffled Maps =180
        Failed Shuffles=0
        Merged Map outputs=180
        GC time elapsed (ms)=16585
        CPU time spent (ms)=198280
        Physical memory (bytes) snapshot=17119162368
        Virtual memory (bytes) snapshot=107879714816
        Total committed heap usage (bytes)=15439233024
```

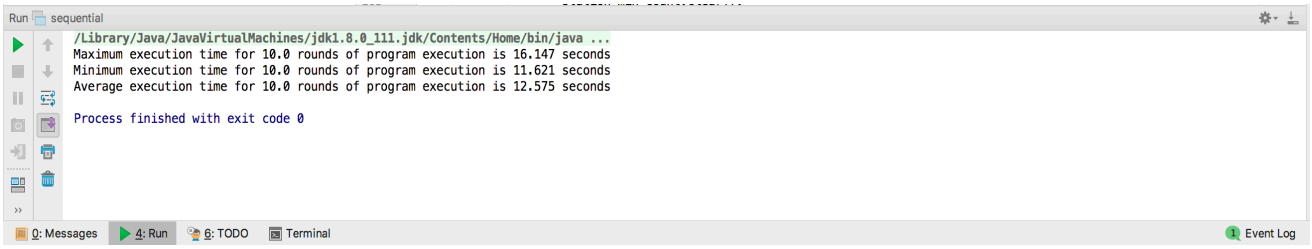
2) There is no guarantee that the combiner will be executed by the map-reduce framework. Hence it is not a good practice to rely on combiner approach. With “In-mapper combining” technique, we can ensure that combining always happens and hence we can ensure that our job runs efficiently.

However there are certain cons for in-mapper combining approach. They are

a) This approach breaks the functional programming underpinnings of map-reduce since state is being preserved across multiple input key-value pairs
b) Preserving statement across multiple input instances means that algorithmic behavior may depend on order in which input key-value pairs are encountered. This creates order dependent bugs which are difficult to debug on large datasets.
c) There is a fundamental scalability bottleneck associated with in-mapper combining data pattern. It critically depends on having sufficient memory to store intermediate results until mapper has completely processed all key-value pairs in an input split

How do the running times and accuracy of these MapReduce programs compare to the sequential implementation of per-station mean temperature? Modify, run, and time the sequential version of your HW1 program on the 1991.csv data. Make sure to change it to measure the end-to-end running time by including the time spent reading the file. Tip: Modify your code to read and process the data line by line (i.e., instead of reading it all into memory). Finally, compare the MapReduce output to the sequential program output to verify and report on its correctness.

The running time of the modified sequence program is 12.575s. The accuracy for both program seems to be correct.



Execution of Program 2

Execution 1: Execution 1 completed in 58 seconds

```
INFO total process run time: 58 seconds
2017-02-12T00:56:57.396Z INFO Step created jobs: job_1486860854922_0001
2017-02-12T00:56:57.396Z INFO Step succeeded with exitCode 0 and took 58 seconds
```

The map reduce execution details for program 2 are as follows:

```
Map-Reduce Framework
        Map input records=7003910
        Map output records=1470207
        Map output bytes=72040143
        Map output materialized bytes=66929
        Input split bytes=940
        Combine input records=1470207
        Combine output records=2417
        Reduce input groups=441
        Reduce shuffle bytes=66929
        Reduce input records=2417
        Reduce output records=441
        Spilled Records=4834
        Shuffled Maps =90
        Failed Shuffles=0
        Merged Map outputs=90
        GC time elapsed (ms)=6827
        CPU time spent (ms)=63910
        Physical memory (bytes) snapshot=9817706496
        Virtual memory (bytes) snapshot=74876817408
        Total committed heap usage (bytes)=8852602880
```