Saloni Shah
NUID: 001617242

CS6240 : Assignment 2

Map-Reduce Algorithm (Pseudo Code)

1.

1.1 No Combiner:

```
class Mapper{
   map(key, (stationId,date,obsType,obsValue,...)){
         if(obsType = 'TMAX' or obsType = 'TMIN'){
               if(obsType = 'TMAX')
                     emit(stationId, (obsValue, 1, 0, 0))
               else
                     emit(stationId, (0, 0, obsValue, 1))
         }
   }
}

class Reducer{
   reduce(stationId, [ (tMax, tMaxCt, tMin, tMinCt), …. ]){
         Initialize tMaxSum , tMinSum , tMaxTotalCt , tMinTotalCt
         for each value in list{
               tMaxSum += tMax
               tMaxTotalCt += tMaxCt
               tMinSum += tMin
               tMinTotalCt += tMinCt
         }

         emit(stationId, tMinSum/tMinTotalCt, tMaxSum/tMaxTotalCt)
   }
}
```

1.2. Combiner:

```
class Mapper{
     map(key, (stationId, date, obsType, obsValue,...)){
          if(obsType = 'TMAX' or obsType = 'TMIN'){
               if(obsType = 'TMAX')
                    emit(stationId, (obsValue, 1, 0, 0))
```

```
                else
                        emit(stationId, (0, 0, obsValue, 1))
            }
        }
    }

    class Combiner{
        reduce(stationId, [ (tMax, tMaxCt, tMin, tMinCt), …. ]){
            Initialize tMaxSum , tMinSum , tMaxTotalCt , tMinTotalCt
            for each value in list{
                tMaxSum += tMax
                tMaxTotalCt += tMaxCt
                tMinSum += tMin
                tMinTotalCt += tMinCt
            }

            emit(stationId, (tMaxSum, tMaxTotalCt, tMinSum, tMinTotalCt))
        }
    }

    class Reducer{
        reduce(stationId, [ (tMax, tMaxCt, tMin, tMinCt), …. ]){
            Intialize tMaxSum , tMinSum , tMaxTotalCt , tMinTotalCt
            for each value in list{
                tMaxSum += tMax
                tMaxTotalCt += tMaxCt
                tMinSum += tMin
                tMinTotalCt += tMinCt
            }

            emit(stationId, tMinSum/tMinTotalCt, tMaxSum/tMaxTotalCt)
        }
     }
```

1.3. InMapper Combiner:

```
    class Mapper{
        HashMap hMap

        setup(){
```

```
            Initialize hMap
        }

        map(key, (stationId, date, obsType, obsValue,...)){
            Initialize value
            if(obsType = 'TMAX' or obsType = 'TMIN'){
                if(obsType = 'TMAX')
                    value = (obsValue, 1, 0, 0)
                else
                    value = (0, 0, obsValue, 1)

                hMap.add(stationId, value)
            }
        }

        cleanup(){
            for each station in hMap
                emit(stationId, hMap[stationId])
        }
    }

    class Reducer{
        reduce(stationId, [ (tMax, tMaxCt, tMin, tMinCt), …. ]){
            Intialize tMaxSum , tMinSum , tMaxTotalCt , tMinTotalCt
            for each value in list{
                tMaxSum += tMax
                tMaxTotalCt += tMaxCt
                tMinSum += tMin
                tMinTotalCt += tMinCt
            }

            emit(stationId, tMinSum/tMinTotalCt, tMaxSum/tMaxTotalCt)
        }
    }
```

2. Secondary Sort:

```
    class Mapper{
        map(key, (stationId,date,obsType,obsValue,...)){
            if(obsType = 'TMAX' or obsType = 'TMIN'){
```

```
                        if(obsType = ‘TMAX’)
                                emit((stationId, date.year), (obsValue, 1, 0, 0))
                        else
                                emit((stationId, date.year), (0, 0, obsValue, 1)) ’
                }
        }
  }

class Combiner{
        reduce((stationId, year) , [ (tMax, tMaxCt, tMin, tMinCt), …. ]){
                Initialize tMaxSum , tMinSum , tMaxTotalCt , tMinTotalCt
                for each value in list{
                        tMaxSum += tMax
                        tMaxTotalCt += tMaxCt
                        tMinSum += tMin
                        tMinTotalCt += tMinCt
                }

                emit((stationId, year) , (tMaxSum, tMaxTotalCt, tMinSum, tMinTotalCt))
        }
}

  class HashPartitioner{
        getPartition(key, value){
                return (key.stationId.hashCode() % numReduceTasks)
        }
  }

  class NaturalComparator{
        compare(key k1, key k2){
                //sorts based on (stationId, year)
                //sorts year in ascending order
                compare k1.stationId and k2.stationId
                if equal compare k1.year and k2.year
        }
  }

  class GroupComparator{
        compare(key k1, key k2){
```

```
            //sorts based on stationId only, ignores year
            compare k1.stationId and k2.stationId
    }
}

class Reducer{
    reduce((stationId, year), [(tMax, tMaxCt, tMin, tMinCt), ...]){
            Initialize outputValue to key.stationId
            Initialize tMaxSum , tMinSum , tMaxTotalCt , tMinTotalCt
            Initialize currentYear to key.year
            for each value in list{
                    if key.year != currentYear{
                            outputValue += (currentYear, tMinSum/tMinCt,
tMaxSum/tMinCt)
                            Reinitialize tMaxSum, tMinSum, tMaxTotalCt, tMinTotalCt
                            currentYear = key.year
                    }
                    tMaxSum += tMax
                    tMaxTotalCt += tMaxCt
                    tMinSum += tMin
                    tMinTotalCt += tMinCt
            }
            outputValue += (currentYear, tMinSum/tMinCt, tMaxSum/tMinCt)

            emit(outputValue, NULL)
    }
}
```

Here, reduce function call will process records having same stationId, since the GroupComparator will group values in the list based solely on stationId. Hence all records with same stationId will be grouped in single reduce call. Also, the key will be sorted by year in ascending order (NaturalComparator), hence all the values for same year will appear together in the values list.

Performance Comparison

Running Times for Program 1:

1. NoCombiner - Run 1: 84 seconds
   Run 2: 90 seconds

2. Combiner - Run 1: 80 seconds
   Run 2: 92 seconds
3. InMapper Combiner - Run 1: 82 seconds
   Run 2: 76 seconds

Running Time for Program 2: 58 seconds

Answers:

1. Yes the Combiner was called in Combiner program. It can be verified by the looking at number of records passed to combiner input, and the number of records emitted combiner. Based on the log files (of Combiner program), we can see that:
   Map input records=30868726
   Map output records=8798241
   Combine input records=8798241
   Combine output records=223783
   Reduce input records=223783
   Reduce output records=14135
   Map output records are same as Combine input records. Hence it shows that the combiner was called.
   How many times a Combiner is called, or on which records is it called, is controlled by Hadoop. Even though we can see that the number of records emitted by Mapper is equal to number of records received by Combiner, we can't infer how many times the Combiner was called per each Map task.

2. Because of the Combiner, the Reducer had to process fewer records (as compared to NoCombiner), since Combiner combined many records emitted by the Mapper. While in the program that did not have a combiner (NoCombiner), the number or records to be processed by Reducer was same as the number of records emitted by Mapper. Hence introduction of Combiner reduced the overhead on the Reducer. Also, due to this the number of bytes that had to be transferred from Mapper to Reducer decreases. This observation can be backed by the following measurements observed from log files:
   NoCombiner Program Log file:
   Map input records=30868726
   Map output records=8798241
   Combine input records=0
   Combine output records=0
   Reduce input records=8798241
   Reduce output records=14135

Combiner Program Log file:
Map input records=30868726
Map output records=8798241
Combine input records=8798241
Combine output records=223783
Reduce input records=223783
Reduce output records=14135
Based on the above values, we can see that in NoCombiner program, the number of records to be processed by Reducer is 8798241, while in Combiner program, the number of records to be processed by Reducer is 223783, which is quite less than the one in NoCombiner.

3. Yes, the local aggregation in InMapperComb is effective as compared to NoCombiner. By local aggregation, we can reduce the amount of data transfer between Mapper and Reducer. Hence, the number of records to be processed by Reducer decreases, reducing the overhead. But the memory overhead increases in Mapper in InMapperComb because of the use of HashMap. This can be observed from the log files:
NoCombiner program Log file:
Map input records=30868726
Map output records=8798241
Map output bytes=246350748
Reduce input records=8798241
Reduce output records=14135
Physical memory (bytes) snapshot=13596729344
Virtual memory (bytes) snapshot=97934639104
Total committed heap usage (bytes)=12432441344

InMapperComb program Log file:
Map input records=30868726
Map output records=223783
Map output bytes=6265924
Reduce input records=223783
Reduce output records=14135
Physical memory (bytes) snapshot=14212345856
Virtual memory (bytes) snapshot=97899589632
Total committed heap usage (bytes)=13056868352
Based on the above observation, we can see that in NoCombiner the number of records emitted by Mapper is 8798241 and number of bytes that have to be transferred over to Reducer is 246350748. While in InMapperComb, the number of records emitted by Mapper is 223783 and number of bytes transferred over to

Reducer is 6265924, which is quite less than NoCombiner. Also, the total heap usage in InMapperComb is more by 624427008 bytes.

4. In InMapperComb, by using local aggregation we are reducing the data transfer between Mapper and Reducer, but this also increases overhead on Mapper because of data structures to perform local aggregation and a little complexity in Mapper code. Hence if data input is very huge, the program might crash because of insufficient memory. While in Combiner, the combiner class is called many times to perform the aggregation. Due to this, even though we do not use local aggregation in Mapper, combiner program runs slower than InMapperComb because of multiple calls to combiner class. Also, it is never known before-hand whether the Combiner will be called or not. So, if memory is not a constraint, Combiner is better option. If not, InMapperComb will give a faster execution.

5. The running time for sequential execution (from HW1) is 97.13 seconds. Compared to all the programs from this homework, it runs the slowest. This is expected because sequential program process each input line, performs it's sum and count, adds (or updates) the value for that stationId in HashMap, and then move on to processing next line. Hence this approach is very slow on big data. Correctness or accuracy of both the program is same i.e they give the same output.