Class Number: SEC 01(BOS-2-TR) (CRN: 13036)

HW Number: 2 **Name: Anoop Pai**

Pseudo-code for Part 1:

NoCombiner Program:

Mapper:

```
map(offset B, line L) {
        for each observation_type in L do
                if observation_type == "TMAX"
                        temperature = getTemperature() from L
                        type = "TMAX"
                        stationID = getStationID() from L
                        emit(stationID, (temperature,  type, 1))
                if observation_type == "TMIN"
                        temperature = getTemperature() from L
                        type = "TMIN"
                        stationID = getStationID() from L
                        emit (stationID, (temperature, type, 1))
}
```

Reducer:

```
reduce(StationID s, [(temp1, type1, count1), (temp2, type2, count2) … (tempn, typen, countn)])
{
        average_min =0, average_max =0
        temp_min =0, temp_max =0
        count_min =0, count_max=0
        for each tempi, typei in input list do
                if typei == "TMIN"
```

```
                temp_min + = tempi
                count_min ++
            if typei ==”TMAX”
                temp_max += tempi
                count_max ++
        average_min = temp_min / count_min
        average_max = temp_max/ count_max
        emit ( s, (average_min, average_max))
}
```

Combiner Program:

Mapper:

```
map(offset B, line L) {
        for each observation_type in L do
            if observation_type == “TMAX”
                temperature = getTemperature() from L
                type = “TMAX”
                stationID = getStationID() from L
                emit (stationID, (temperature,  type, 1))
            if observation_type == “TMIN”
                temperature = getTemperature() from L
                type = “TMIN”
                stationID = getStationID() from L
                emit (stationID, (temperature, type, 1))
}
```

Combiner:

```
combine(StationID s, [(temp1, type1, count1), (temp2, type2, count2) … (tempn, typen, countn)])) {
        temp_min =0, temp_max=0
        count_min=0, count_max=0
        for each tempi, typei in inputlist do
                if typei == “TMIN”
                        temp_min += tempi
                        count_min++
                if typei == “TMAX”
                        temp_max += tempi
                        count_max++
        emit (s, (temp_min, count_min, TMIN)
        emit (s, (temp_max. count_max, TMAX)
}
```

Reducer:

```
reduce(StationID s, [(temp1, type1, count1), (temp2, type2, count2) … (tempn, typen, countn)])
{
        average_min =0, average_max =0
        temp_min =0, temp_max =0
        count_min =0, count_max=0
        for each tempi, typei, counti in input list do
                if typei == “TMIN”
                        temp_min + = tempi
                        count_min += counti
                if typei ==”TMAX”
                        temp_max += tempi
```

```
                    count_max += counti
        average_min = temp_min / count_min
        average_max = temp_max/ count_max
        emit ( s, (average_min, average_max))
}
```

InMapper Program:

Mapper:

```
setup() {
        m1 = new HashMap
        m2 = new HashMap
}
map(offset B, line L) {
        count_min =0, count_max = 0
        temperature_min =0, temperature_max =0
        for each observation_type in L do
                stationID = getStationID() from L
                if observation_type == “TMAX”
                        temperature = getTemperature() from L
                        if stationID present in map1
                                count = map1.getCount
                                count_max += count
                                temperature = map1.getTemperature
                                temperature_max += temperature
                                map1.put(stationID, (temperature_max, count_max, TMAX))
                        else
                                map1.put(stationID, (temperature, 1, TMAX)
```

```
            if observation_type == “TMIN”
                temperature = getTemperature() from L
                if stationID present in map2
                    count = map2.getCount
                    count_min += count
                    temperature = map2.getTemperature
                    temperature_min += temperature
                    map2. put(stationID, (temperature_min, count_min, TMIN))
                else
                    map2.put(stationID, (temperature, 1, TMIN))
}

cleanup()
    for every key in map1
        emit (key, map1(key))
    for every key in map2
        emit (key, map2(key))
```

Reducer:

```
reduce(StationID s, [(temp1, type1, count1), (temp2, type2, count2) … (tempn, typen, countn)])
{
    average_min =0, average_max =0
    temp_min =0, temp_max =0
    count_min =0, count_max=0
    for each tempi, typei, counti in input list do
        if typei == “TMIN”
            temp_min + = tempi
```

```
                count_min += counti
            if typei ==”TMAX”
                temp_max += tempi
                count_max += counti
        average_min = temp_min / count_min
        average_max = temp_max/ count_max
        emit ( s, (average_min, average_max))
}
```

Secondary Sort :

Mapper:

```
map(offset B, line L) {
    for each observation_type in L do
        year = getYear() from L
        if observation_type == “TMAX”
            temperature = getTemperature() from L
            type = “TMAX”
            stationID = getStationID() from L
            emit( (stationID, year) , (temperature,  type, 1))
        if observation_type == “TMIN”
            temperature = getTemperature() from L
            type = “TMIN”
            stationID = getStationID() from L
            emit ((stationID, year) , (temperature, type, 1))
}
```

Partitioner:

```
getPartition( (station, year), (temperature, type, count), numreducetasks) {
        // partition by stationID
        return stationID % (numreducetasks)
}
```

KeyComparator:

```
keyComparator(StationID, year){
        //Sort in increasing order of StationID first
        //If the StationID is equal, sorts in increasing order of year next
}
```

```
// A grouping comparator
// This comparator decides which keys are grouped together for a single reduce call Reducer.reduce()
function
// For example, all records with same station ID end up in the same reduce call.
// Particularly, an input record (AGE00135039, 1884, V1) and (AGE00135039, 1885, V2) end up being
// a part of the same reduce call.
// Without a Grouping Comparator, there would be one reduce call per distinct (StationID, year) pair
```

GroupingComparator:

```
groupComparator(StationID, year) {
        //Sorts in increasing order of StationID
        //Does not consider year for sorting
        //Hence two keys with the same StationID are considered identical
        //no matter the year value
}
```

Reducer:

```
reduce((stationID, year), [(temp1, type1, count1), (temp2, type2, count2) … (tempn, typen,
countn)]) {
        average_min =0, average_max =0
```

```
        temp_min =0, temp_max =0
        count_min =0, count_max=0
        partialoutput = new list
        prevyear = year
        for each tempi, typei in input list do
                if prevyear != year
                        avg_min = temp_min/ count_min
                        avg_max = temp_max / count_max
                        partialoutput.add((year, avg_min, avg_max))
                        avg_min =0, avg_max =0
                        temp_min=0, temp_max =0
                        count_min =0, count_max =0
                prevyear = year
                if typei == "TMIN"
                        temp_min + = tempi
                        count_min ++
                if typei =="TMIN"
                        temp_max += tempi
                        count_max ++
        average_min = temp_min / count_min
        average_max = temp_max/ count_max
        emit ( s, (average_min, partialoutput))
}
```

By using a grouping comparator to perform secondary sort, we can group data within a reduce task. Thus, secondary sort is used to sort the value attributes (year), while the key is inherently sorted by the mappers before starting reducers.

In the case of this example, with the help of a grouping comparator, an entry such as ((AGE00135039, 1886, V1) (AGE00135039, 1881, V2) (AGE00135039, 1883, V3) would appear in Reduce's input list as (AGE00135039, (1881, V2), (1883, V3) (1886, V1)) as the years have been sorted in increasing order.

Running times:

	Runtime1	Runtime2
NoCombiner	69 secs	70 secs
Combiner	68 secs	68 secs
InMapper	64 secs	63 secs

1) As per the syslog file, it is clear that the Combiner was called atleast once during the execution of the program. This can be backed up by the numbers in the syslog file reports which are as below :

   Combine input records=8798241

   Combine output records=447564

In general, the number of times that a Combiner was called can be determined by the number of spills to disk. The drawback of using a Combiner is that its execution cannot be controlled by the user but is instead controlled by the MapReduce system. It could be called once, twice or sometimes not at all. Although it is difficult to conclude how many times a Combiner was called per map task from the syslog file, theoretically, a Combiner can only be called once per Map task as a combiner is a process that runs locally on each Mapper machine.

2) Having a Combiner definitely improved the performance of the job as shown in the running times of the two versions of the program above. Also, from the syslog file, the following observation can be made:

   NoCombiner : Combiner:

   Reduce input records=8798241 vs Reduce input records=447564

There is a significant difference between the input records processed by the reducers when the NoCombiner program is compared with the Combiner approach. Clearly, the combiner reduces the number of records that is to be input to the reducer by

performing aggregation. Therefore, for example, instead of sending a (AGE003579, (10, TMAX, 1)) and a (AGE003579, (20, TMAX, 1)) separately which would've happen in the case of the NoCombiner, the Combiner aggregates these two entries and sends a (AGE003579, (30, TMAX, 2)) entry to the reducer. The reduce input groups value found in both the approaches remain the same (14135) but the Combiner only aggregates most of the entries output from the mapper and thereby saving some computational effort in the Reducer.

3) The local aggregation performed in the case of InMapperComb is definitely effective compared to NoCombiner. The following observation can be made from the syslog file found in each versions of the program :

NoCombiner:	InMapperComb:
Map output records=8798241	Map output records=445200
Map output bytes=246350748	Map output bytes=12465600

This statistic shows that the InMapperComb emits the records to the Reducer only once it has performed local aggregation within the mapper. It does so with the help of a data structure, accumulating all values pertaining to the same key in memory and then emitting these accumulated values in the data structure to the Reducer. These records also co-relate to the bytes output by each of these programs. Also, the same value for map output records is reflected in reduce input records.

4) As mentioned earlier, a Combiner is a process that runs locally on each Mapper machine to pre-aggregate data before it is shuffled across the network to various cluster Reducers. It is also beyond the control of the user and hence makes it difficult to specify how many times it has to be called.

The InMapper performs a similar aggregation only difference being that it is local within the mapper. This means these aggregations are not written to disk and occur in-memory. It does so with the help of a setup() and cleanup() function within the Mapper which initializes the data structure(s) and emits the records to the Reducers at the beginning and end of the Mapper phase respectively. Following is the observation from the syslog files :

Combiner:	InMapperComb:
Reduce input records=447564	Reduce input records=445200

From the above stats, it is evident that the InMapperComb does a better job at emitting fewer records to the Reducer thereby performing better aggregation while producing the same results. Although it is only a slight difference, this difference could increase as the size of the data would increase. Moreover, InMapperComb accomplishes this by doing all the work in

memory instead of writing to output files(Combiner). Given these reasons and unless there is a constraint on the heap memory, it is clear that the InMapperComb approach is better than Combiner.

5) Upon modifying, running and calculating the running times for both the sequential version of the program and the MapReduce program, the sequential program measured to be running for about 18 seconds while the MapReduce program lasted for about a minute. This significant difference in running time can be explained by the time taken to spawn the mappers, transfer the output from the mappers to the reducers, number of reducers used amongst other several factors. This performance comparison might seem misleading but the MapReduce program is still better. This is because, as the input size goes up to hundreds of TBs or Peta Bytes, MapReduce will be the better performer if the program is written efficiently and optimizations are made on using an InMapperCombiner, several reducers and so on. As far as the correctness was concerned, both the versions of the program ended up giving correct results.