CS6240, Assignment 2, Rakesh Buchan Krishna Murthy

Weather Analysis – No Combiner:

```
map(key k, input_data value){
    String[] itr ← split value based on "\n"
    Loop over itr[]
        String[] StationReading ← split itr based on "," // since it's a comma separated file
        StationID = StationReading[0];
        TemperatureAccumulator dataStructure ← new (1/0 , StationReading[3]) // 1 for TMAX
records and 0 for TMIN records and StationReading[3] is the temperature value.
        emit(StationID, TemperatureAccumulator)
}

reduce(Text StationID, TemperatureAccumulator records){
    for each record in records
        if record.type == 1
            sum_tmax =  sum_tmax +1;
            countTmax++;
        if record.type == 0
            sum_tmin =  sum_tmin +1;
            countTmin++;
    avg_tmax = sum_tmax/countTmax (or) 0 if countTmax = 0
    avg_tmin = sum_tmin/countTmin (or) 0 if countTmin = 0
    emit(StationID + ","+avg_tmin +","+avg_tmax , null)
}
```

Weather Analysis – Combiner:

```
map(key k, input_data value){
    String[] itr ← split value based on "\n"
    Loop over itr[]
        String[] StationReading ← split itr based on "," // since it's a comma separated file
        StationID = StationReading[0];
        If it's a TMAX record
            TemperatureAccumulator dataStructure ← new (1 , StationReading[3], 0 , 0.0)
// StationReading[3] is the temperature value.
        If it's a TMIN record
            TemperatureAccumulator dataStructure ← new (0, 0.0, 1 , StationReading[3]) //
StationReading[3] is the temperature value.
        emit(StationID, TemperatureAccumulator)
}
```

```
combine(Text StationID, TemperatureAccumulator records){
        for each record in records
                if record.tmaxTempCount == 1
                        sum_tmax =  sum_tmax +1;
                        countTmax++;
                if record.tminTempCount == 1
                        sum_tmin =  sum_tmin +1;
                        countTmin++
        emit(StationID, TemperatureAccumulator)
}

reduce(Text StationID, TemperatureAccumulator records){
        for each record in records
                if record.tmaxTempCount >= 1
                        sum_tmax =  sum_tmax +1;
                        countTmax++;
                if record.tminTempCount >= 1
                        sum_tmin =  sum_tmin +1;
                        countTmin++;
        avg_tmax = sum_tmax/countTmax (or) 0 if countTmax = 0
        avg_tmin = sum_tmin/countTmin (or) 0 if countTmin = 0
        emit (StationID + “,”+avg_tmin +”,”+avg_tmax , null)
}
```

Weather Analysis – In Mapper Combiner:

```
Class mapper{
        Declare Hashmap mapComputations

        setup(Context c){
                initialize Hashmap mapComputations
        }

        map(key k, input_data value){
                String[] itr ← split value based on “\n”
                Loop over itr[]
                        String[] StationReading ← split itr based on “,” // since it’s a comma separated
                        file
                        StationID = StationReading[0];
                        If (mapComputations contains StationID){
                                If it’s a TMAX record
                                        mapComputations .
                                update(TemperatureAccumulator.updateTmax(StationReading[3]) )
```

```
                    If it's a TMIN record
                        mapComputations.
                     update(TemperatureAccumulator.updateTmin(StationReading[3]) )
                }else{
                    If it's a TMAX record
                        mapComputations .add(TemperatureAccumulator
                dataStructure ← new (1 , StationReading[3], 0 , 0.0)) // StationReading[3] is the
                temperature value.
                    If it's a TMIN record
                        mapComputations .add(TemperatureAccumulator
                    dataStructure ← new (0, 0.0, 1 , StationReading[3])) //
                    StationReading[3] is the temperature value.
    }

    Cleanup(Context c){
        For each key- value pair in
            emit(StationID, TemperatureAccumulator)
    }
}

reduce(Text StationID, TemperatureAccumulator records){
    for each record in records
        if record.tmaxTempCount >= 1
            sum_tmax =  sum_tmax +1;
            countTmax++;
        if record.tminTempCount >= 1
            sum_tmin =  sum_tmin +1;
            countTmin++;
    avg_tmax = sum_tmax/countTmax (or) 0 if countTmax = 0
    avg_tmin = sum_tmin/countTmin (or) 0 if countTmin = 0
    emit (StationID + ","+avg_tmin +","+avg_tmax , null)
}
```

CS6240, Assignment 2, Rakesh Buchan Krishna Murthy

Weather Analysis – Secondary Sort:

```
map(key k, input_data value){
        String[] itr ← split value based on "\n"
        Loop over itr[]
                String[] StationReading ← split itr based on "," // since it's a comma separated file
                StationID = StationReading[0];
                Year = (StationReading[1].substring(0, 4))
                Key <- StationID, Year
                If it's a TMAX record
                        TemperatureAccumulator dataStructure ← new (1 , StationReading[3], 0 , 0.0)
// StationReading[3] is the temperature value.
                If it's a TMIN record
                        TemperatureAccumulator dataStructure ← new (0, 0.0, 1 , StationReading[3]) //
StationReading[3] is the temperature value.
                emit(Key, TemperatureAccumulator)
}


combine(Text Key, TemperatureAccumulator records){
        for each record in records
                if record.tmaxTempCount == 1
                        sum_tmax =  sum_tmax +1;
                        countTmax++;
                if record.tminTempCount == 1
                        sum_tmin =  sum_tmin +1;
                        countTmin++
        emit(Key, TemperatureAccumulator)
}

Partitioner(){
        Finds the hashcodes of all keys
        Split the keys based on the available machines
}

keyComparator(){
        sort the keys based on stationID, for same stationIDs, sort it based on Year
}


groupingComparator(){
        group the keys based on stationID
}
```

```
reduce(Text Key, TemperatureAccumulator records){
    for each record in records
        year = key.year
        previousYear = year
        if(year == previousYear)
            if record.tmaxTempCount >= 1
                sum_tmax =  sum_tmax +1;
                countTmax++;
            if record.tminTempCount >= 1
                sum_tmin =  sum_tmin +1;
                countTmin++;
        else
            year_avg_tmax = sum_tmax/countTmax (or) 0 if countTmax = 0
            year_avg_tmin = sum_tmin/countTmin (or) 0 if countTmin = 0
    emit (StationID + “,”+year”,”+year_avg_tmin +”,”+year_avg_tmax , null)
}
```

For Secondary Sort, I have used a composite key of StationID and Year
So the mapper emits a composite key for each input line
Then the combiner at mapper level combines any similar StationID-Year records with updated temps and emit the latest values.
Next Partition will distribute these keys across the available systems ensuring that all records with same key land on the same machine
Next Key Comparator compares these keys and sort them in StationId and then with the year
Next group comparator groups all stationIDs together
Thus the reduce call gets triggered with composite key – StationID, Year all sorted in the order of hash function implemented in the key class for a particular partition.

Running Times

Execution Type	First Run	Second Run
No Combine	80 sec	83 sec
Combine	78 sec	81 sec
In mapper Combiner	68 sec	74 sec
Secondary Sort	45 sec	50 sec

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

Yes the Combiner was called because the following details can be found in the syslog file:

CS6240, Assignment 2, Rakesh Buchan Krishna Murthy

Map input records=30870343
Map output records=8798758
Combine input records=8798758
Combine output records=223795
Reduce input records=223795
Reduce output records=14136
Here the Combine input records is same as Map Output records and combine reduces it to a way much records by combining records with same key and outputs them to the reducer.

The number of times the combiner gets called is decided by map-reduce framework and it is not configured by the programmer, hence we cannot say the number of times combiner gets called either in total or by each map

Was the local aggregation effective in InMapperComb compared to NoCombiner?
InMapperCombiner stats:
Map input records=30870343
Map output records=223795
Reduce input records=223795
Reduce output records=14136

No Combiner stats:
Map input records=30870343
Map output records=8798758
Reduce input records=8798758
Reduce output records=14136

From this we can infer that the records transferred over network is higher in case of no combiner, also the time required to execute is also higher by around 10 sec in case of NoCombiner compared to InMapperCombiner for the input data of 1.1 GB