Map-Reduce Algorithm (40 points)

No-Combiner :

```
/*Mapper */
map(K1,V1)
Extract station_id , min_or_max, val(value of temperature) from
each record
If(min_or_max==”TMAX” or min_or_max==”TMIN”)
Emit (station_id,(min_or_max,val))

/*Reducer */
//key is the station id
//value has the type (min/max)  and the val of the temperature
reduce(Key k,values[(min_or_max,val)…….])

minSum=0
maxSum=0
minCount=0
maxCount=0
avgMax=0
avgMin=0

for each record v in values
      if (v.min_or_max.equals("TMIN"))
                                    {
                                    minSum+=v.temp;
                                    minCount++;
                                    }
       else if (v.min_or_max.equals("TMAX")){
                                maxCount++;
                                maxSum+=v.temp;
                                }
```

```
avgMin=minSum/minCount
avgMax=maxSum/maxCount
emit (stationId,(avgMin,avgMax))
```

***Combine*r**

```
/*Mapper */
map(K1,V1)
Extract station_id , min_or_max, val(value of temperature) from
each record
If(min_or_max==”TMAX” or min_or_max==”TMIN”)
emit (station_id,(min_or_max,val))

/*Combiner */
//key is the station id
//value has the type (min/max)  and the val of the temperature
combine(Key k,values[(min_or_max,val,1)…….])

minSum=0
maxSum=0
minCount=0
maxCount=0

for each record v in values
 if (v.min_or_max.equals("TMIN"))
                                    {

                                minSum+=v.val;
                                minCount++;

                                    }
                            else if (v.min_or_max.equals("TMAX")){

                                maxCount++;
```

```
                    maxSum+=v.val;
                    }

k2= stationId

emit (K2,(“TMIN”,minSum,minCount))
emit (K2,(“TMAX”,maxSum,maxCount))


/*Reducer */
//key is the station id
//value has the type (min/max)  and the val of the temperature
reduce(Key k,values[(min_or_max,val)…….])

minSum=0
maxSum=0
minCount=0
maxCount=0
avgMax=0;
avgMin=0;

for each record v in values
 if (v.min_or_max.equals("TMIN"))
                        {

                    minSum+=v.temp;
                    minCount++;

                        }
                else if (v.min_or_max.equals("TMAX")){

                    maxCount++;
                    maxSum+=w.temp;
                    }
avgMin=minSum/minCount
```

avgMax=maxSum/maxCount
emit (stationId,(avgMin,avgMax))

InMapper

class mapper{

method initialize()
HashMap<key,value > map
//key is stationId
//value is writable of maxcount,mincount,sumMaxTemp,sumMintemp

/*Mapper */
map(Key k,Value v)
Extract station_id , min_or_max, val(value of temperature) from each record
If(min_or_max==”TMAX”)
v2.maxCount.set(v2.getMaxCount()+1);
v2.maxSum.set(v2.getMaxSum()+val);
map.put(k,v2)

If(min_or_max==”TMIN”)
v2.minCount.set(v2.getMinCount()+1);
v2.minSum.set(v2.getMinSum()+val);
map.put(k,v2)

method cleanup()
for(String key:map.keySet())
{
emit(key,map.get(key));
}
/*Reducer */
//key is the station id
//value has the type (min/max) and the val of the temperature
reduce(Key k,values[(min_or_max,val)…….])

```
minSum=0
maxSum=0
minCount=0
maxCount=0
avgMax=0;
avgMin=0;

for each record v in values
 if (v.min_or_max.equals("TMIN"))
                                    {

                                minSum+=v.temp;
                                minCount++;

                                    }
                        else if (v.min_or_max.equals("TMAX")){

                                maxCount++;
                                maxSum+=w.temp;
                                }
avgMin=minSum/minCount
avgMax=maxSum/maxCount
emit (stationId,(avgMin,avgMax))
```

Secondary Sort:

```
        /*Mapper*/
        //map emits key ,value pair
        //where key is an object of CustomKEY
        //and value is
        //(year,TmaxValue,TmaxCount,TminValue,TminCount)

        //map function
        method map(key,value)
        extract station_id,min_or_max,val(temp)
```

```
        if (min_or_max==”TMAX”)
        emit((stationId,Year),year,0,0,val,1)
if (min_or_max==”TMAX”)
emit((stationId,Year),year,val,1,0,0)


//The key is an object of CustomKey that’s overrides its own
//CompareTo method.
//The Key is a combination of stationId and Year
//First it compare stationId s , if same compares the year
 class customKey{
stationId
year
method compareTo(k1,k2)
{
compare stationId
if  k1.stationId = =k2.stationId
compare year
}
}


//The partioner makes sure that all the  records with same station Id go to the same reducer .It does this by calling a simple hashing function
method partitioner(key)
return hash(key.StationId)

//A combiner can  be used to aggregate the results from the mapper , but for performance tradeoff its optional

//The grouping comparator groups data by station id ignoring all the year value , this ensures all the records with same stationId goes to same reduce call

method customGroupingComparator (key1,key2)
```

```
compare(key1.stationID and key2.stationId)
/*Reducer*/
//The key to the reduce is our custom type (StationId,Year)
//and value is a list having same keys
//Because of the grouping Comparator year is ignored and same
//stationId s with diferent year are clubbed in the same call
//Value consists of  minSum maxSum, minCount,maxCount, //year
//Because we exploited MR frameworks sorting  of keys the //important
thing to notice is that the records to the reducer call //is sorted in
ascending order of year .

    reduce(Key k,values[(minSum, maxSum,
    minCount,maxCount,year)…….])

    minSum=0
    maxSum=0
    minCount=0
    maxCount=0
    year=v.year

    for each record v in values
     if (year is not equal to year)//year changes
    //flush
    emit (Null,(yaer,avgMin,avgMax))

    //reset
    minSum=0
    maxSum=0
    minCount=0
    maxCount=0
    year=v.year
    else
        year=v.year;
        maxSum += v.maxSum;
        maxCount += v.maxCount;
```

```
minSum += v.minSum;
minCount += v.minCount;
```

Explanation:
We make use of map reduces sorting mechanism to avoid explicit sorting in the reducer .
We achieve this by adding year as a part of the key and then use Key comparator and grouping comparator
The keycomprator emits records with ascending order of both station id and year and group comparator ignores the year and sends all records with same station id to one reduce call
Thus the input to the reducer is a list of
Records for one stationId in increasing order year eg:

(A 1998)(A 1998)(A 1999)(A 2000) in one reduce call

Performance Comparison (24 points total)

Nocombiner :Attempt 1
INFO total process run time: 80 seconds
2017-10-08T17:45:40.754Z INFO Step created jobs: job_1507484521305_0001
2017-10-08T17:45:40.755Z INFO Step succeeded with exitCode 0 and took 80 seconds
Nocombiner :Attempt 2

INFO total process run time: 78 seconds
2017-10-08T18:07:17.499Z INFO Step created jobs: job_1507485836804_0001
2017-10-08T18:07:17.499Z INFO Step succeeded with exitCode 0 and took 78 seconds

combiner: Attempt 1
INFO total process run time: 80 seconds

2017-10-08T18:27:53.066Z INFO Step created jobs:
job_1507487059743_0001
2017-10-08T18:27:53.066Z INFO Step succeeded with exitCode 0 and took 80 seconds

combiner: Attempt 2

INFO total process run time: 80 seconds
2017-10-08T18:50:53.077Z INFO Step created jobs:
job_1507488446451_0001
2017-10-08T18:50:53.077Z INFO Step succeeded with exitCode 0 and took 80 seconds

InMapper: Attempt 1

INFO total process run time: 68 seconds
2017-10-08T18:54:10.458Z INFO Step created jobs:
job_1507488669046_0001
2017-10-08T18:54:10.459Z INFO Step succeeded with exitCode 0 and took 68 seconds

InMapper: Attempt 2

INFO total process run time: 68 seconds
2017-10-08T19:09:09.288Z INFO Step created jobs:
job_1507489556983_0001
2017-10-08T19:09:09.288Z INFO Step succeeded with exitCode 0 and took 68 seconds

SecondarySort: Attempt 1

INFO total process run time: 62 seconds
2017-10-08T08:54:52.844Z INFO Step created jobs:
job_1507452734343_0001
2017-10-08T08:54:52.844Z INFO Step succeeded with exitCode 0 and took 62 seconds

SecondarySort: Attempt 2

INFO total process run time: 60 seconds

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

Yes,The combiner was called in the program as its evident from the following figures:
Since there was almost 1/10 reduction in the number of records I am inferring that it was called more than one map task.

Combiner Syslog:

Map-Reduce Framework

Map input records=30870343
Map output records=8798758
Map output bytes=255163982
Map output materialized bytes=7192081
Input split bytes=1992
Combine input records=8798758
Combine output records=629332
Reduce input groups=14136
Reduce shuffle bytes=7192081
Reduce input records=629332
Reduce output records=14136
Spilled Records=1258664
Shuffled Maps =264
Failed Shuffles=0
Merged Map outputs=264
GC time elapsed (ms)=23390
CPU time spent (ms)=230700
Physical memory (bytes) snapshot=19726614528
Virtual memory (bytes) snapshot=130353664000
Total committed heap usage (bytes)=18189123584

Was the local aggregation effective in InMapperComb compared to NoCombiner?

1)The number of records passed to reducer in Inmapper is much lesser

223795 vs 8798758

InMapper:

Map-Reduce Framework

Map input records=30870343
Map output records=223795
Map output bytes=8056620
Map output materialized bytes=4426315
Input split bytes=1581
Combine input records=0
Combine output records=0
Reduce input groups=14136
Reduce shuffle bytes=4426315
Reduce input records=223795
Reduce output records=14136
Spilled Records=447590
Shuffled Maps =187
Failed Shuffles=0
Merged Map outputs=187
GC time elapsed (ms)=16655
CPU time spent (ms)=151670
Physical memory (bytes) snapshot=16612216832
Virtual memory (bytes) snapshot=107270967296
Total committed heap usage (bytes)=15055978496

NoCombiner

Map-Reduce Framework

Map input records=30870343
Map output records=8798758
Map output bytes=228767708
Map output materialized bytes=50738537

Input split bytes=1992
Combine input records=0
Combine output records=0
Reduce input groups=14136
Reduce shuffle bytes=50738537
Reduce input records=8798758
Reduce output records=14136
Spilled Records=17597516
Shuffled Maps =264
Failed Shuffles=0
Merged Map outputs=264

2)Faster Runtime in Inmapper

the following records confirm to that:

InMapper Runtime
INFO total process run time: 68 seconds
2017-10-08T19:09:09.288Z INFO Step created jobs:
job_1507489556983_0001
2017-10-08T19:09:09.288Z INFO Step succeeded with exitCode 0 and took 68 seconds

Nocombiner :Runtime

INFO total process run time: 80 seconds
2017-10-08T17:45:40.754Z INFO Step created jobs:
job_1507484521305_0001
2017-10-08T17:45:40.755Z INFO Step succeeded with exitCode 0 and took 80 seconds

Program 2: Secondary sort

INFO total process run time: 62 seconds

2017-10-08T08:54:52.844Z INFO Step created jobs: job_1507452734343_0001
2017-10-08T08:54:52.844Z INFO Step succeeded with exitCode 0 and took 62 seconds

Scala :
Briefly discuss where in your program—and why—you chose to use which of the following data representations: RDD, pair RDD, DataSet, DataFrame. (4 points) Show the Spark Scala programs you wrote for part 1 (mean min and max temperature for each station in a single year). If you do not have a fully functional program, discuss the Scala commands your program should use. (4 points)

) No Combiner
We use pair RDD to imitate the Key, Value pair Hadoop ‘s map function emits. The aggregate function used is **groupByKey** as it groups all the values with same key.

```
/*Pseudocode*/

val pairRDD = sc.textFile("input")
val maxPair       =pairRDD.map(line => line.split(","))
val minPair       =pairRDD.map(line => line.split(","))
val mins     =minPair.filter(fields => fields(2) ==“TMIN” )
val  maxs         =maxPair.filter(fields => fields(2) ==“TMAX” )
val   minMap    =mins.map(fields => (fields(0),
(Integer.parseInt(fields(3))))
val   maxMap    =maxs.map(fields => (fields(0),
(Integer.parseInt(fields(3))))
//Calculate Min
val MaxsumByKey = minMap groupByKey((sum, temp) => temp
+ sum)
val MaxcountByKey = minMap.foldByKey(0)((ct, temp) => ct + 1)
```

```
val MinsumCountByKey = MinsumByKey.join(countByKey)
val MinavgByKey = MinsumCountByKey.map(t => (t._1, t._2._1 /
t._2._2.toDouble))
//Calculate Max
val MaxsumByKey = maxMap groupByKey((sum, temp) => temp
+ sum)
val MaxcountByKey = maxMap.foldByKey(0)((ct, temp) => ct +
1)
val MaxsumCountByKey = MaxsumByKey.join(countByKey)
val MaxavgByKey = MaxsumCountByKey.map(t => (t._1, t._2._1
/ t._2._2.toDouble))
//join mining and maxavg
MinMaxAvg= MaxavgByKey.join(MinavgByKey)
MinMaxAvg.saveAsTextFile("output")
```

2)InMapper

We use pair RDD To imitatate the Key Value map emits in Hadoop For Inmapper we use the Reduce By key Aggregate since it is an equivalent to Inmapper in Hadoop It adds all temperature and counts. InMapper

```
/*Pseudocode*/
val pairRDD = sc.textFile("input")
val maxPair  = pairRDD.map(line => line.split(","))
val minPair = pairRDD.map(line => line.split(","))
val mins    =minPair.filter(fields => fields(2) ==“TMIN” )
val  maxs        =maxPair.filter(fields => fields(2) ==“TMAX” )
val   minMap    =mins.map(fields => (fields(0),
(Integer.parseInt(fields(3))))
val   maxMap    =maxs.map(fields => (fields(0),
(Integer.parseInt(fields(3))))
//Calculate Min
```

```
val MaxsumByKey = minMap reduceByKey((sum, temp) => temp + sum)
val MaxcountByKey = minMap.foldByKey(0)((ct, temp) => ct + 1)
val MinsumCountByKey = MinsumByKey.join(countByKey)
val MinavgByKey = MinsumCountByKey.map(t => (t._1, t._2._1 / t._2._2.toDouble))
//Calculate Max
val MaxsumByKey = maxMap groupByKey((sum, temp) => temp + sum)
val MaxcountByKey = maxMap.foldByKey(0)((ct, temp) => ct + 1)
val MaxsumCountByKey = MaxsumByKey.join(countByKey)
val MaxavgByKey = MaxsumCountByKey.map(t => (t._1, t._2._1 / t._2._2.toDouble))
//join mining and maxavg
MinMaxAvg= MaxavgByKey.join(MinavgByKey)
MinMaxAvg.saveAsTextFile("output")
```

3)Combiner
We use pair RDD To imitatate the Key Value map emits in Hadoop
The aggregate Data structure used is **combineByKey since**
It is an equivalent for combiner in hadoop

```
/*Pseudocode*/
val tempByKey = sc.textFile("input", 3) .
val maxPair =pairRDD.map(line => line.split(","))
val minPair =pairRDD.map(line => line.split(","))
val mins =minPair.filter(fields => fields(2) ==“TMIN” )
val maxs =maxPair.filter(fields => fields(2) ==“TMAX” )
val minMap =mins.map(fields => (fields(0), (Integer.parseInt(fields(3))))
val maxMap =maxs.map(fields => (fields(0), (Integer.parseInt(fields(3))))
val sumCtByKey = tempByKey.combineByKey((v:Int) => (v, 1),
```

(a:(Int,Int), v) =>
(a._1 + v, a._2 + 1), (a1:(Int,Int), a2:(Int,Int)) => (a1._1 + a2._1, a1._2 + a2._2))

Secondary Sort
We need to use RDD pair since
Partitioning only useful only when RDD can be used in key-oriented operation.

*/*Pseudocode*/*

//map function

val tempByKey = sc.textFile("input", 3) .
val maxPair =pairRDD.map(line => line.split(","))
val minPair =pairRDD.map(line => line.split(","))
val mins =minPair.filter(fields => fields(2) ==“TMIN”)
val maxs =maxPair.filter(fields => fields(2) ==“TMAX”)
val minMap =mins.map(fields => (fields(0), year(Integer.parseInt(fields(3))))
val maxMap =maxs.map(fields =>
*((**fields(0),year**)(Integer.parseInt(fields(3))))*
*val sumCtByKey = tempByKey.**combineByKey**((v:Int) => (v, 1),*
(a:(Int,Int), v) =>
(a._1 + v, a._2 + 1), (a1:(Int,Int), a2:(Int,Int)) => (a1._1 + a2._1, a1._2 + a2._2))