PARALLEL DATA PROCESSING IN MAPREDUCE – CS 6240
Prof. Mirek Riedewald
VIJET BADIGANNAVAR

A. Pseudo-code to compute mean min temperature and mean max temperature by station for a single year.

a. No combiner

```
// map function emits the record if the record is of TMIN or TMAX type
map(offset B, Line l){
        // l is a record that contains stationId, year, type, value
        // type is an enum {TMAX, TMIN, SNOW...}
        // Station s refers to {type, value}

        if l is of type TMAX or TMIN
                emit(stationId, {type, value})
}

// for each station, computes the Mean TMIN and TMAX
reduce(Text stationId,[s1,s2,s3…]{
        //stationId is the id of station
        // s1,s2..sn refer to inputlist
        // type is an enum {TMAX,TMIN}
        // value refers to the temp. value
        // Station s refers to {type, value}

        maxTempSum = 0.0;
        maxTempCount = 0;
        minTempSum = 0.0;
        minTempCount = 0;

        for(s in inputlist){
                if(type(s) == TMIN){
                        minTempSum += s.value;
                        minTempCount +=1;
                }else{
                        maxTempSum += s.value;
                        maxTempCount +=1;

                }
        }
        avgMinTemp = minTempSum/ minTempCount;
        avgMaxTemp = maxTempSum/ maxTempCount;

        emit(stationId, {avgMinTemp, avgMaxTemp});
}
```

b. With Combiner

```
// map function emits the record if the record is of TMIN or TMAX type
map(offset B, Line l){
        // l is a record that contains stationId, year, type, value
        // type is an enum {TMAX, TMIN, SNOW...}
        // station is {maxTemp, maxCount, minTemp, minCount}

        if l is of type TMAX
                emit(stationId, {value, 1, NULL, NULL })
        if l is of type TMIN
                emit(stationId, { NULL, NULL, value, 1 })

}

// Combiner that aggregates the TMIN and TMAX for a particular Station along with count
combiner(Text stationId, [s1,s2…sn]){
        //stationId is the id of station
        // s1,s2,s3…sn refers to inputlist
        // type is an enum {TMAX,TMIN}
        // value refers to the temp. value
        // Station s is {maxTemp, maxCount, minTemp, minCount}

        maxTempSum = 0.0;
        maxTempCount = 0;
        minTempSum = 0.0;
        minTempCount = 0;

        for( station s in inputlist){
                // aggregate the MAX temp and corresponding count
                if(s.maxTemp != NULL){
                        maxTempSum  += s.maxTemp;
                        maxTempCount +=s.maxCount;
                }
                // aggregate the MIN temp and corresponding count
                if(s.minTemp != NULL){
                        minTempSum  += s.minTemp;
                        minTempCount +=s.minCount;
                }
        }
        emit(stationId, { maxTempSum, maxTempCount, minTempSum, minTempCount });
}
```

```
// Reducer that computes the mean TMIN and TMAX for each station by aggregating the
//TMIN and TMAX for a particular station
reducer(Text stationId, [s1,s2,s3…..]{
        //stationId is the id of station
        // [s1,s2…sn is inputlist]
        // Station s is {maxTemp, maxCount, minTemp, minCount}

        maxTempSum = 0.0;
        maxTempCount = 0;
        minTempSum = 0.0;
        minTempCount = 0;

        for(Station s in inputlist){
                // aggregate the MAX temp and corresponding count
                if(s.maxTemp != NULL){
                        maxTempSum  += s.maxTemp;
                        maxTempCount +=s.maxCount;
                }
                // aggregate the MIN temp and corresponding count
                if(s.minTemp != NULL){
                        minTempSum  += s.minTemp;
                        minTempCount +=s.minCount;
                }
        }
        avgMinTemp = minTempSum/ minTempCount;
        avgMaxTemp = maxTempSum/ maxTempCount;

        emit(stationId, { avgMinTemp, avgMaxTemp});
}
```

c. With InMapperCombiner

Mapper:

```
// Sets up the map that temperory stores all the temperature of station per map task
setup(){
// Station s is {maxTemp, maxCount, minTemp, minCount}
  h = new hashmap // stores the stationId -> { maxTemp, maxCount, minTemp, minCount }
}
```

```
// map function reads each line and updates the map with the corresponding temperature
value  and count accordingly.
map(offset B, Line l){
        // l is a record that contains stationId, year, type, value
        // type is an enum {TMAX, TMIN, SNOW...}
        // station is {maxTemp, maxCount, minTemp, minCount}

        if l is of type TMAX or TMIN
                if h contains an entry with stationId
                        update the values by adding to existing values of TMIN or TMAX and
                        increment the count by 1 correspondingly
                else
                        create an entry in h based on the type and count

}

// Writes all the map to the output
cleanup(){
        for each entry in h
                emit(stationId, { maxTemp, maxCount, minTemp, minCount });
}
```

Reducer:

```
// Reducer that computes the mean TMIN and TMAX for each station by aggregating the TMIN and
TMAX for a particular station

Reducer(Text StationId, [s1,s2…sn]){
        //stationId is the id of station
        // [s1,s2…sn is inputlist]
        // station s is {maxTemp, maxCount, minTemp, minCount}

                maxTempSum = 0.0;
                maxTempCount = 0;
                minTempSum = 0.0;
                minTempCount = 0;

                for(Station s in inputlist){
                        // aggregate the MAX temp and corresponding count
                        if(s.maxTemp != NULL){
                                maxTempSum  += s.maxTemp;
                                maxTempCount +=s.maxCount;
                        }
                        // aggregate the MIN temp and corresponding count
                        if(s.minTemp != NULL){
                                minTempSum  += s.minTemp;
                                minTempCount +=s.minCount;
                        }
                }
                avgMinTemp = minTempSum/ minTempCount;
                avgMaxTemp = maxTempSum/ maxTempCount;
                emit(stationId, { avgMinTemp, avgMaxTemp});
```

```
}
```

d. Secondary Sort

Mapper:

```
// Sets up the map that temperory stores all the temperature of station per map task

setup(){
        // Station s is {maxTemp, maxCount, minTemp, minCount}
        h = new hashmap // stores the {stationId,year} -> { maxTemp, maxTempCount,
minTemp, minTempCount }
}

// map function reads each line and updates the map with the corresponding temperature
//value  and count accordingly.
map(offset B, Line l){
        // l is a record that contains stationId, year, type, value
        // type is an enum {TMAX, TMIN, SNOW...}
        // station is {maxTemp, maxCount, minTemp, minCount}

        if l is of type TMAX or TMIN
                if h contains an entry with {stationId,year}as key
                        update the values by adding to existing values of TMIN or TMAX and
                        increment the count by 1 correspondingly
                else
                        create an entry in h based on the type and count

}
// write the contents of h in output
cleanup(){
        for each entry in h
                emit({stationId,year}, { maxTemp, maxCount, minTemp, minCount });
}
```

Grouping Comparator

```
// sorts by stationId
groupComparator({stationId,year},{stationId,year}){
        // Station s refers to {stationId,year}
        sort by stationId
}

// KeyComparator that sorts first by stationId then by year
KeyComparator ({stationId,year},{stationId,year}){
        sort by stationId,year
}
```

```
// partitioner that partions based on the hashcode of the stationId, so that all the station data goes to the // single reducer
Partitioner ({stationId,year},{maxTemp, maxCount, minTemp, minCount}){
        // n stands for no of reducers
        return hashcode(stationId)%n;
}
```

Reducer:

```
// Reducer that aggregates the mean Min temp and mean max temp corresponding to station per //year. As we are using the grouping comparator, there will be single reduce call for each station and // the years will be already sorted according to ascending order.

 Reducer(({stationId,year}, [s1,s2,s3…]){
                //stationId is the id of station
                // [s1,s2…sn is inputlist]
                // station s is {maxTemp, maxCount, minTemp, minCount}

                maxTempSum = 0.0;
                maxTempCount = 0;
                minTempSum = 0.0;
                minTempCount = 0;

                for each year corresponding to one stationId in s{
                        // aggregate the MAX temp and corresponding count
                                if(s.maxTemp != NULL){
                                        maxTempSum  += s.maxTemp;
                                        maxTempCount +=s.maxCount;
                        }
                        // aggregate the MIN temp and corresponding count
                        if(s.minTemp != NULL){
                                        minTempSum  += s.minTemp;
                                        minTempCount +=s.minCount;
                        }
                }

                avgMinTemp = minTempSum/ minTempCount;
                avgMaxTemp = maxTempSum/ maxTempCount;

                emit(stationId, { avgMinTemp, avgMaxTemp});
        }
```

Explanation

The mapper is going to emit the records with {stationId, year} as the key. The key comparator then sorts the keys according to {stationId,year} (both in ascending order). The Partitioner partions the records based on the hashcode of the stationId so that all the records related to the particular station goes to the single reducer. On the reducer side, the grouping comparator groups the records by stationId. I.e. {s1,y1},{s1,y2},{s1,y3}.. are grouped together and are given as single reduce call for s1. y1,y2,y3 will also be sorted in ascending order as per the key comparator logic. Thus, for each reduce call all the records corresponding to the particular station will be processed according to the increasing order of year.

EMR Running Time:

MapReduce – without – combiner

Trial 1 – 88 sec
Trail 2 – 88 sec

MapReduce -with-combiner

Trail 1 – 84 sec
Trail 2 – 86 sec

MapReduce – with – inMapper - combiner

Trail 1 – 80 sec
Trail 2 – 80 sec

Questions!

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

Yes, the combiner was surely called in the program. In the logs of the run that ran with the combiner I see that "Combiner input records: 8798241", "Combiner output records: 223782". While, the logs that ran without the combiner these values are set to 0.
I ran the program by keeping the custom count to check how many times the Combiner was called. The results are as follows:

No of times the map call was called: 30865324
Combiner Count: 586429

This shows that Combiner may or may not be called during each map task. During a map task each time the buffer overflows the combiner is called before it is written to the spill. If the map task does not fills the buffer, then the combiner might not be called.

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

The running time of program that used the combiner is slightly better than NoCombiner program. The "raw materialized bytes" value which indicates the number of actual bytes produced by the mapper is less (4139071 bytes) in program with Combiner compared to the program without combiner which is 55947397 bytes. The number of input records for reducer in case of NoCombiner is 8798241 while its 223782 for Combiner indicating the combiner has aggregated some of the records at the mapper side.

3. *Was the local aggregation effective in InMapperComb compared to NoCombiner?*

Yes, the running time of the program with InMapperCombiner(80sec) is better than NoCombiner (88sec). As mentioned above the total amount of data that is produced in NoCombiner as indicated by the "raw materialized bytes" is 55947397 bytes compared to InMapper combiner that is 4139071 bytes, which means the amount of data that's needs to transferred is less in InMapperCombiner. Also, the number of input records for reducer is 8798241 for NoCombiner, while, its 223782 for InMapperCombiner that suggests its effective.

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

With the results of the experiments above, the InMapperCombiner is better when compared to Combiner. Also, in InMapperCombiner the programmer has control over combining process while it's not possible using the Combiner. Combiner can be used only when the collecting values are of commutative or associative in nature. In Combiner there will be more IO operations compared to InMapper combiner. As long as in mapper combiner don't blow up the heap, InMapper combiner can be used.

5. *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 sequential implementation took 16.672 sec. While, the map reduce program took 56 sec to complete. As the size of data is small the time taken by the sequential program is less when compared to the map reduce job, but when the size of the data is huge, map reduce should beat the sequential time considerably.
   The output of both the programs are same and correct.