CS 6240 Assignment 2 Section 1 **Priyanka Shenoy**

Running Times of Programs are as follows-

ProgramName	**Time_1**	**Time_2**
No Combiner	94 seconds	96 seconds
Combiner	84 seconds	84 seconds
In Mapper Combiner	78 seconds	78 seconds
Secondary Sort	56 seconds	-

Answering questions with respect to Performance Comparison

1. Yes the Combiner was called in the Combiner program. Our syslog is an indication for the same.
Map output records=8798241
Reduce input records=223783
There has to be the combiner which crunched the number of records between the Map and the reducer.
We cannot find out how many times it was called by using only the syslog since we only know the records input and output by each task. Also innately, calling the combiner solely depends on the Map Reduce program and is out of reach for the programmer.

2. Combiner reduced the number of records being transferred to the reducer significantly. This causes less time in network transfer as well which indeed causes program to be executed faster.
Combiner – Map output records=8798241 Reduce input records=223783
NoCombiner – Map output records=30868726 Reduce input records=30868726
The running times of the Combiner and NoCombiner differ by ~10 seconds for only 1GB of data.

3. Yes local aggregation was very effective.
NoCombiner – Map output records=30868726 Reduce input records=30868726
InMapper – Map input records=30868726 Map output records=223783 Reduce input records=223783

The number of records reduced significantly, so did the the time to transfer these records ~15 seconds for 1 GB of data. InMapper reduces the overhead required for calculation at the reducer end.

CS 6240 Assignment 2 Section 1 **Priyanka Shenoy**

4. In our scenario, in mapper combiner and normal combiner seem to be effectively running with similar record outputs.

InMapper – Map input records=30868726 Map output records=223783 Reduce input records=223783
Combiner – Map input records=30868726 Map output records=8798241 Reduce input records=223783

However time taken by the InMapper is ~6 seconds less than combiner, since each call to combiner has an overhead (creating objects of class, methods etc) whereas having an HashMap is just adding data to in memory object. *Yes, it consumes more memory but in an ideal situation if memory is unlimited, in mapper is perfect solution to resolve the over head issue. If memory is a constraint, then using normal combiner would also suffice since it is still better than transferring huge data load to the reducers*.

5. Sequential data run takes ~ 8 seconds which is significantly lesser as compared to InMapper combiner. We need to take the following criteria into consideration -

- The amount out data we ran the results for is only 1GB which is very less as compared to MapReduce capabilities
- When data increases, like real time, sequential time of execution increases exponentially. Where as in MapReduce, parallel data processing on distributed systems continues at the same rate
- Currently the entire processing on AWS is distributed and it takes lot of time to transfer data from map to reduce and process.
- *Sequential processing is good for static small data range where as MapReduce is better when processing big data chunks.*

CS 6240 Assignment 2 Section 1 **Priyanka Shenoy**

Pseudo Code-

1. No Combiner

map(key , csv_file_data)
{
- for each line in csv_file_data
    - Create word = station_id {mapper_key}
    - Split line into station_id, year, temperature_values
        if line has TMIN
            - isMinOrMax = false ; min_temp = current_min_temp
            - max-temp = 0.0
        if line has TMAX
            - isMinOrMax = true ; min_temp = current_max_temp
            - min_temp = 0.0
    - acc_dt_str = {min_temp, max_temp, isMinOrMax} {mapper_value}

- emit {mapper_key , mapper_value}
}

reduce(mapper_key, mapper_value)
{
- for each value in mapper_value
    - if mapper_value isMinOrMax==false
        sum_min += current_min_temp
        count_min ++
    - if mapper_value isMinOrMax== true
        sum_max += current_max_temp
        count_max ++
- find average
    avg_min = sum_min/count_min
    avg_max = sum_max/count_max
- emits (mapper_key +”” + avg_min +”” + avg_max)
}

CS 6240 Assignment 2 Section 1 **Priyanka Shenoy**

2. With Combiner

map(key , csv_file_data)
{
- for each line in csv_file_data
    - Create word = station_id {mapper_key}
    - Split line into station_id, year, temperature_values
        -if line has TMIN
            - acc_dt_str =(min_temp, max_temp=0.0, count_min=1, count_max=0) {mapper_value}
        if line has TMAX
            - acc_dt_str =(min_temp=0.0, max_temp, count_min=0, count_max=1) {mapper_value}

- emit {mapper_key , mapper_value}
}

combiner(mapper_key , mapper_value)
{
- for each value in mapper_value
    sum_min += current_min_temp
    sum_max += current_max_temp
    count_min += current_min_count
    count_max += current_max_count
- acc_dt_str =(sum_min, sum_max, count_min, count_max) {combiner_value}
- emit {mapper_key, combiner_value}
}

reduce(mapper_key, combiner_value OR mapper_value)
{
- for each value in mapper_value
    sum_min += current_min_temp
    sum_max += current_max_temp
    count_min += current_min_count
    count_max += current_max_count
- find average
    avg_min = if (count_min == 0) then “None” else sum_min/count_min
    avg_max = if (count_max == 0) then “None” else sum_max/count_max
- emits (mapper_key +”” + avg_min +”” + avg_max)
}

CS 6240 Assignment 2 Section 1 **Priyanka Shenoy**

3. In Mapper Combiner

setup(context)
{
- Create HashMap H
}

map(key , csv_file_data)
{
- for each line in csv_file_data
    - Create word = station_id {mapper_key}
    - Split line into station_id, year, temperature_values
        -if line has TMIN
            - check if H has station_id
                - H.updateMin (min_temp)
            -else
                - H.add (station_id, acc_dt_str = temp_min, 0.0, 1, 0)

        -if line has TMAX
            - check if H has station_id
                - H.updateMax (max_temp)
            -else
                - H.add (station_id, acc_dt_str = 0.0, temp_max, 0, 1)
}

cleanup(context)
{
- for each value in H
    - emit (H[key] , H[value])
}

CS 6240 Assignment 2 Section 1 **Priyanka Shenoy**

reducer(mapper_key, mapper_value)
{
- for each value in mapper_value
    sum_min += current_min_temp
    sum_max += current_max_temp
    count_min += current_min_count
    count_max += current_max_count
- find average
    avg_min = if (count_min == 0) then “None” else sum_min/count_min
    avg_max = if (count_max == 0) then “None” else sum_max/count_max
- emits (mapper_key +”” + avg_min +”” + avg_max)
}

CS 6240 Assignment 2 Section 1 **Priyanka Shenoy**

4. Secondary Sort

map(key , csv_file_data)
{
for each line in csv_file_data
- Split line into station_id, year, temperature_values
- Create composite_key = (station_id, year) {mapper_key}
-if line has TMIN
- acc_dt_str =(min_temp, max_temp=0.0, count_min=1, count_max=0) {mapper_value}
if line has TMAX
- acc_dt_str =(min_temp=0.0, max_temp, count_min=0, count_max=1) {mapper_value}

- emit {mapper_key , mapper_value}
}

partitioner(mapper_key, mapper_value)
{
- finds the hash code of mapper_key – station_id
- splits the mapper_key on basis on number of machines present
}

naturalComparator(composite_key key1 ,composite_key key2)
{
- sort by key1.station_id
- if key1.station_id == key2.station_id
- sort by key1.year
}

groupingComparator(composite_key key1 ,composite_key key2)
{
- sort by key1.year
}

combiner(mapper_key , mapper_value)
{
- for each value in mapper_value
sum_min += current_min_temp

sum_max += current_max_temp
count_min += current_min_count
count_max += current_max_count
- acc_dt_str =(sum_min, sum_max, count_min, count_max) {combiner_value}
- emit {mapper_key, combiner_value}
}

reducer(mapper_key, mapper_value or combiner_value)
{
- current_year = mapper_key.year
- for each value in mapper_value
if mapper_key.year == current_year
- sum_min += current_min_temp
- sum_max += current_max_temp
- count_min += current_min_count
- count_max += current_max_count
else
- sum_min = current_min_temp
- sum_max = current_max_temp
- count_min = current_min_count
- count_max= current_max_count

avg_min = if (count_min == 0) then “None” else sum_min/count_min
avg_max = if (count_max == 0) then “None” else sum_max/count_max

- emit (mapper_key+mapper_key.year+ avg_min + avg_max)
}

//Additional Notes
// While calling the reduce method, we sort the objects in the order of its key which in our case is a composite key of station_id and year.
// Output of the reduce call here would be, (station_id,year), values for each record
// The output format expected is such that we need to have all station_id listed for one year as one single list
// For the same we use grouping comparator which compares the objects ONLY on the basis of year so as to have all station_id displayed with single year