CS6240 Section01 – HW2 Report
Submitted by: Rajkumar Murukeshan

Map Reduce Source Code:

Task 1:

a. **NoCombiner Pseudo-Code**

```
Class Station{
        Integer tmaxValue, tminValue, tminCount, tmaxCount;

        // define constructor
        Station(tmaxValue, tminValue, tminCount, tmaxCount){
                //set the class variables for the object
        }
}

map(Key key, Value val) {
        String line = val.toString();
        // parse the line for obtaining stationID, year, temperature type and its value
        if (type == "TMAX" || "TMIN"){
                // create Station object with tmax or tmin values;
        }
        emit(stationID, Station object);
}

reduce(Text StationId, [Station0, Station1, … Station-n]){
        TminSum = Summation of tminValue of Station i , where i = 0 to n;
        TmaxSum = Summation of tmaxValue of Station i , where i = 0 to n;
        TminCount = Summation of tminCount of Station i , where i = 0 to n;
        TmaxCount = Summation of tmaxCount of Station i , where i = 0 to n;
        MeanMinTemp = TminSum/TminCount;
        MeanMaxTemp = TmaxSum/TmaxCount;
        String result= MeanMinTemp + ","  + MeanMaxTemp;
        emit(StationId, result);
}
```

b. **Combiner Pseudo-Code**

```
Class Station{
        Integer tmaxValue, tminValue, tminCount, tmaxCount;
        // define constructor
        Station(tmaxValue, tminValue, tminCount, tmaxCount){
                //set the class variables for the object
        }
}

map(Key key, Value val) {
        String line = val.toString();
        // parse the line for obtaining stationID, year, temperature type and its value
        if (type == “TMAX” || “TMIN”){
                // create Station object with tmax or tmin values;
        }
emit(stationID, Station object);
}

Combiner(Text StationId, [Station0, Station1, … Station-n]){
        TminSum = Summation of tminValue of Station i , where i = 0 to n;
        TmaxSum = Summation of tmaxValue of Station i , where i = 0 to n;
        TminCount = Summation of tminCount of Station i , where i = 0 to n;
        TmaxCount = Summation of tmaxCount of Station i , where i = 0 to n;
        Station aggregatedStation = new Station(
        emit(StationId, result);
}

reducer(Text StationId, [Station0, Station1, … Station-n]){
        TminSum = Summation of tminValue of Station i , where i = 0 to n;
        TmaxSum = Summation of tmaxValue of Station i , where i = 0 to n;
        TminCount = Summation of tminCount of Station i , where i = 0 to n;
        TmaxCount = Summation of tmaxCount of Station i , where i = 0 to n;
        MeanMinTemp = TminSum/TminCount;
        MeanMaxTemp = TmaxSum/TmaxCount;
        String result= MeanMinTemp + “,”  + MeanMaxTemp;
        emit(StationId, result);
}
```

c. **InMapperComb**

```
Class Station{
        Integer tmaxValue, tminValue, tminCount, tmaxCount;

        // define constructor
        Station(tmaxValue, tminValue, tminCount, tmaxCount){
```

```
        //set the class variables for the object
    }
}

Mapper Task:

setup() {
    // create a inmap variable as below:
    HashMap<String, Station> inMapStations = new HashMap<>();
}

map(Key key, Value val) {
    String line = val.toString();
    // parse the line for obtaining stationID, year, temperature type and its value
    if (type == “TMAX” || “TMIN”){
        // create or update an entry in the HashMap inMapStations with key as
        // StationId and value as Station object
    }
}

cleanupForMapper(){
    for each entry in inMapStations {
        emit(entry.key, entry.value);
    }
}

reducer(Test StationId, [Station0, Station1, … Station-n]){
    TminSum = Summation of tminValue of Station i , where i = 0 to n;
    TmaxSum = Summation of tmaxValue of Station i , where i = 0 to n;
    TminCount = Summation of tminCount of Station i , where i = 0 to n;
    TmaxCount = Summation of tmaxCount of Station i , where i = 0 to n;
    MeanMinTemp = TminSum/TminCount;
    MeanMaxTemp = TmaxSum/TmaxCount;
    String result= MeanMinTemp + “,”  + MeanMaxTemp;
    emit(StationId, result);
}
```

Task 2: **Secondary Sort**

```
Class CompositeKey{

    String stationId, year;
    // define constructor
    Station (stationId, year){
        //set the class variables for the object
    }

    // override equals, hashcode and compareTo methods

}



Class Station{

    String year;
    Integer tmaxValue, tminValue, tminCount, tmaxCount;

    // define constructor
    Station(year, tmaxValue, tminValue, tminCount, tmaxCount){
        //set the class variables for the object
    }
}
```

Mapper Task:

```
setup() {
    // create a inmap variable as below:
    HashMap< CompositeKey, Station> inMapStations = new HashMap<>();
}

map(Key key, Value val) {
    String line = val.toString();
    // parse the line for obtaining stationID, year, temperature type and its value
    // create compositeKey cKey with stationId and year obtained after parsing
    if (type == “TMAX” || “TMIN”){
        // create a Station Object station with min and max values;
        // create or update an entry in the HashMap inMapStations with key as
        // Composite Key cKey and value as Station object station
    }
}

cleanupForMapper(){
    for each entry in inMapStations {
        emit(entry.key(), entry.value());
    }
}

GroupingComparator(CompositeKey ckey1, CompositeKey cKey2){
    return cKey1.stationId.compareTo(cKey2.stationId);
}

SortComparator(CompositeKey ckey1, CompositeKey cKey2) {
    int compare = cKey1.stationId.compareTo(cKey2.stationId;
    if(compare == 0) {
        compare = cKey1.year.compareTo(cKey2.year);
    }
    return compare;
}

reducer(CompositeKey cKey, [Value1, Value2, … Value-n]){
    Int tminSum, tmaxSum, tminCount, tmaxCount;
    int currYear = -1, prevYear;
    List<String> result = new ArrayList<>();
    for(Station st: Values){
        prevYear = currYear;
        currYear= st.year;
        if(currYear != prevYear && prevYear != -1){
            //calculate meanMin and meanMax and append it to result
            // reset tminSum, tMaxSum, tmincount and tmaxCount
        } else if (currYear != prevYear && prevYear = -1){
            set tminSum, tMaxSum, tmincount and tmaxCount
        } else if (currYear == prevYear) {
            aggregate tminSum, tMaxSum, tmincount and tmaxCount
         }
    }
    // calculate meanMin, meanMax and append it to the result;
    // convert list result to String and emit that as reducer output value
    String output= result.ToString();
    emit(StationId, output);
}
```

Performance Comparison

1) First Run

Running Time for NoCombiner Task= 70 seconds
Running Time for Combiner Task= 66 seconds
Running Time for InMapperComb Task= 66 seconds

2) Second Run

Running Time for NoCombiner Task= 71 seconds
Running Time for Combiner Task= 64 seconds
Running Time for InMapperComb Task= 64 seconds

Question and Answers

1) Was the Combiner called at all in program Combiner? Was it called more than once per Map task?
Answer: Yes. The Combiner was called in program Combiner.
Combine Input Records: 8798241
Combine Output Records: 223782

Yes. It is called more than once per Map Task. Number of Merged Map Outputs= 323

2) What difference did the use of a Combiner make in Combiner compared to NoCombiner?
Answer: Combiner reduced the traffic and data flow into the reducer. By Using Combiner, we see that number of Reduce input records = 223782 and without using combiner, number of Reduce input records = 8798241
Hence, there is a great reduction of processing of data in reducer and this creates less traffic to the reducers

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

Answer: Yes. Usage of local aggregation in InMapperComb was effective when compared to NoCombiner. This aggregates the map calls for each task in the mapper task itself.
Using NoCombiner: Map output records=8798241
Map output bytes=146602330

Using InMapCombiner: Map output records=223782
Map output bytes=4413409

4) Which one is better, Combiner or InMapperComb? Briefly justify your answer
Answer: InMapperComb is better than Combiner because the aggregations are performed in the local disk in InMapperComb which reduces the load time, traffic.

5) How do the running times and accuracy of these MapReduce programs compare to the sequential implementation of per-station mean temperature?
Answer: Accuracy of these Map-Reduce programs are same compared to the Sequential Implementation. However there are more number of I/O operations involved in MapReduce technique, hence it takes more time to execute in Map Reduce.