CS6240 – Parallel Data Processing with Map-Reduce

SECTION2 HW 2 AKASH KADAM

Program Execution Run time comparison

Program	Execution Time (sec) Run1	Execution Time (sec) Run2
No Combiner	92	86
Combiner	84	86
In Mapper	88	82

Flow of Records data comparison throughout the map reduce task

Program	Map In	Map Out	Combine In	Combine Out	Reduce In	Reduce Out
In Mapper	30865324	223782	0	0	223782	14135
Combiner	30865324	8798241	8798241	447564	447564	14135
No Combiner	30865324	8798241	0	0	8798241	14135

No Combiner Pseudo code

```
map(key, value){
        emit(stationID, record)
}

reduce(stationID, records){
        for(r in records){
                if(r contains tmax record){
                        tmaxSum += r.temp;
                        tmaxC++;
                }
                else{
                        tminSum += r.temp;
                        tminC++;
                }
        avgTmax = tmaxSum/tmaxC;
        avgTmin = tminSum/tminC;

        result = avgTmin, avgTmax;

emit(key, result)
}
```

Combiner Pseudo Code

```
map(key, value){
        Text record = tvalue + ttype + tcount;
        emit(stationID, record)
}

combine(stationID, records){
        for(r in records){
                if(r contains tmax record){
                        tmaxSum += r.temp;
                        tmaxC++;
                }
                else{
                        tminSum += r.temp;
                        tminC++;
                }
        }
emit(stationID, new Text(tmaxSum + "TMAX" + tmaxC));
emit(stationID, new Text(tminSum + "TMIN" + tminC));
}


reduce(stationID, records){
        for(r in records){
                if(r contains tmax record){
                        tmaxSum += r.temp;
                        tmaxC++;
                }
                else{
                        tminSum += r.temp;
                        tminC++;
                }
        }
        avgTmax = tmaxSum/tmaxC;
        avgTmin = tminSum/tminC;

        result = avgTmin, avgTmax;

emit(key, result)
}
```

In Mapper Pseudo Code

```
map(key, value) {
        setup(){
                tmap = new HashMap<String, ComputeSum>();
        }

        map(){
                tmap.put(stationID, ComputeSum)
        }

        cleanup(){
                String result = tmaxS + tmaxC + tminS + tminC;
                emit(stationID, result)
}

reduce(key, values){

        for(v in values){
                tmaxSum += v.tmaxS;
                tmaxCount+= v.tmaxC;
                tminSum += v.tminS;
                tminCount += v.tminC;
        }

        avgTmax = tmaxSum / tmaxCount;
        avgTmin = tminSum / tminCount;

emit(stationID, (avgTmin, avgTmax))
}
```

Secondary Sort Pseudo Code

```
class IdAndYear{
        String stationID;
        Long year;
}

class ComputeSum{
        String ttype;
        Double ttemp;
        Double tcount;
// computes the ttempSum and tcountSum based on ttype
}

map(key, value){
        setup(){
                        // IdAnYear handles the customKey generation
                        tmap = new HashMap<IdAndYear, ComputeSum>();
        }
        map(){
                tmap.put(IdAndYear, ComputeSum)
                // ComputeSum computes the tmaxSum, tmaxCount, tminSum and tminCount
        }

        cleanup(){
                String result = tmaxS + tmaxC + tminS + tminC;
                emit((stationID, year), result);
        }
} // end map

keyComparator(stationID, year){
        // compare the stationID first
        // if stationID are equal sort in ascending order of the year
}

groupingComparator(stationID, temperature){
        // consider only the stationID for comparison and return when equal
}

reduce(compositeKey, records){
        // for each record in records compute the total tmax, tmin and counts
        // take the average and emit the output.
        // maintain a prevyear and curryear counter and soon as the counters
        // don't match I stop computing the avg for the prevyear and emit the op
emit (stationID, (year, tmin, tmax))
}
```

Was the Combiner called? Was it called more than once per map task?
Ans) Yes the combiner was called during both the runs and I conducted an experiment by keeping a counter for each combiner call and map call. The counter value of combiner was less than the map call counter which clearly suggests that combiner function may or may not be called for every map function task.

What difference did the combiner make in Combiner as compared to NoCombiner?
Ans) The use of Combiner made a huge difference as the number of input records for reduce phase were lowered thus decreasing the load on the reduce phase. Following are the details

a) with combiner

Map output records=8798241
Combine input records=8798241
Combine output records=447564
Reduce input records=447564

b) with no combiner

Map output records=8798241
Reduce input records=8798241

Was the local aggregation effective in InMapperComb compared to NoCombiner?
Ans) The local aggregation reduced the load on the reduce phase in the InMapperComb as compared to NoCombiner. Map output records=223782 as compared to 8798241 with NoCombiner.

Which one is better Combiner or InMapperComb?
Ans) In my opinion if you want to reduce the load on the reduce phase I would prefer InMapperComb as it is guaranteed that the load will be reduced. Whereas in the case of Combiner it is not guaranteed that the Combiner will get executed 100% or on exactly what split of records from the map phase will the Combiner work.

Comparison with the Sequential Execution program?
Ans) The run time for the sequential execution from HW1 on the 1991.csv file took 15 seconds as compared to 85 seconds which the map reduce version took. The output result was accurate but time taken was very less in the sequential version. The reason is the input data is very small and I think map reduce version will always work fast if the input data is very large as compared to the execution of the sequential version on large data.