Assignment - 2

Satya Akhil Chowdary Kuchipudi

Sec - 01

Map-Reduce Pseudo Code :

```
class compositevaluewritable
        tmaxval, tmaxcount, tminval, tmincount
```

hw2.a.NoCombiner

```
map(offset B, line L)
        parts = split L on ","
        if (parts[2] equals "TMAX")
                emit(parts[0], compositevaluewritable(parts[3], 1, 0, 0))
        if (parts[2] equals "TMIN")
                emit(parts[0], compositevaluewritable(0, 0, parts[3], 1))

reduce(stationid id, [compositevaluewritable1, compositevaluewritable2,...])
        tmaxval, tmaxcount, tminval, tmincount = 0
        for each compositevaluewritable in iterablelist do
                tmaxval += compositevaluewritable.tmaxval
                tmaxcount += compositevaluewritable.tmaxcount
                tminval += compositevaluewritable.tminval
                tmincount += compositevaluewritable.tmincount

        tmaxavg = tmaxval/tmaxcount
        tminavg = tminval/tmincount
        result =  tminavg + tmaxavg
        emit(id, result)
```

---

hw2.b.Combiner

```
map(offset B, line L)
        parts = split L on ","
        if (parts[2] equals "TMAX")
                emit(parts[0], compositevaluewritable(parts[3], 1, 0, 0))
```

```
        if (parts[2] equals "TMIN")
                emit(parts[0], compositevaluewritable(0, 0, parts[3], 1))

combine(stationid id, [compositevaluewritable1, compositevaluewritable2,...])
        tmaxval, tmaxcount, tminval, tmincount = 0
        for each compositevaluewritable in iterablelist do
                tmaxval += compositevaluewritable.tmaxval
                tmaxcount += compositevaluewritable.tmaxcount
                tminval += compositevaluewritable.tminval
                tmincount += compositevaluewritable.tmincount
        emit(id, compositevaluewritable(tmaxval, tmaxcount, tminval, tmincount))

reduce(stationid id, [compositevaluewritable1, compositevaluewritable2,...])
        tmaxval, tmaxcount, tminval, tmincount = 0
        for each compositevaluewritable in iterablelist do
                tmaxval += compositevaluewritable.tmaxval
                tmaxcount += compositevaluewritable.tmaxcount
                tminval += compositevaluewritable.tminval
                tmincount += compositevaluewritable.tmincount

        tmaxavg = tmaxval/tmaxcount
        tminavg = tminval/tmincount
        result =  tminavg + tmaxavgj
        emit(id, result)
```

---

hw2.c.InMapperCombiner

```
setup()
        map = (stationid, compositevaluewritable)

map(offset B, line L)
        parts = split L on ","
        if (parts[2] equals "TMAX")
                if map has parts[0]
                        oldcompositevaluewritable = map get parts[0]
                        update oldcompositevaluewritable.tmaxval += parts[3]
                        update oldcompositevaluewritable.tmaxcount += 1
                else
                        map put (parts[0], compositevaluewritable(parts[3], 1, 0, 0))

        if (parts[2] equals "TMIN")
                if map has parts[0]
                        oldcompositevaluewritable = map get parts[0]
                        update oldcompositevaluewritable.tminval += parts[3]
```

```
                update oldcompositevaluewritable.tmincount += 1
            else
                map put (parts[0], compositevaluewritable(parts[3], 1, 0, 0))

cleanup()
    for each entry in map
        emit(entry.key, entry.value)

reduce(stationid id, [compositevaluewritable1, compositevaluewritable2,...])
    tmaxval, tmaxcount, tminval, tmincount = 0
    for each compositevaluewritable in iterablelist do
        tmaxval += compositevaluewritable.tmaxval
        tmaxcount += compositevaluewritable.tmaxcount
        tminval += compositevaluewritable.tminval
        tmincount += compositevaluewritable.tmincount

    tmaxavg = tmaxval/tmaxcount
    tminavg = tminval/tmincount
    result =  tminavg + tmaxavgj
    emit(id, result)
```

---

hw2.2.SecondarySort

```
class compositekeywritable
    stationid, year

class compositevaluewritable
    year, tmaxval, tmaxcount, tminval, tmincount

setup()
    map = (compositekeywritable, compositevaluewritable)

map(offset B, line L)
    parts = split L on ","
    if (parts[2] equals "TMAX")
        if map has parts[0]
            oldcompositevaluewritable = map get parts[0]
            update oldcompositevaluewritable.tmaxval += parts[3]
            update oldcompositevaluewritable.tmaxcount += 1
        else
            map.put(parts[0], compositevaluewritable(parts[3], 1, 0, 0))

    if (parts[2] equals "TMIN")
        if map has parts[0]
```

```
                    oldcompositevaluewritable = map get parts[0]
                    update oldcompositevaluewritable.tminval += parts[3]
                    update oldcompositevaluewritable.tmincount += 1
            else
                    map.put(parts[0], compositevaluewritable(0, 0, parts[3], 1))

cleanup()
        for each entry of map
                emit(entry.key, entry.value)

reduce(compositekeywritable key, [compositevaluewritable1, compositevaluewritable2,...])
        list = []
        tmaxval, tmaxcount, tminval, tmincount = 0
        year = null
        for each secondary_sort_data_writable in list do
                if year == null
                        set currentvaluewritable = compositevaluewritable
                        year = compositevaluewritable1.year
                else if year == compositevaluewritable.year
                        update currentvaluewritable.tmaxval += compositevaluewritable.tmaxval
                        update currentvaluewritable.tminval += compositevaluewritable.tminval
                        update currentvaluewritable.tmaxcount +=
compositevaluewritable.tmaxcount
                        update currentvaluewritable.tmincount +=
compositevaluewritable.tmincount
                else
                        list add calavg(currentvaluewritable)
                        set currentvaluewritable = compositevaluewritable
                        year = compositevaluewritable1.year

        # Calculation tmax avg and tmin avg for last year in this reduce task which
        # wont hit the loop
        list add calavg(currentvaluewritable)

        result = list to string
        emit(key.stationid, result)

calavg(compositevaluewritable val)
        tmaxavg, tminavg, result = null;
        if val.tmaxcount is 0
        else
                tmaxavg = val.tmaxval/val.tmaxcount
        if val.tmincount is 0
        else
                tminavg = val.tminval/val.tmincount
        result = tmaxavg + tminavg
```

```
key_comparator(compositekeywritable k1, compositekeywritable k2)
        result = compare(k1.stationid, k2.station_id)
        if result is 0
                result = compare(k1.year, k2.year)
        return result

group_comparator(compositekeywritable k1, compositekeywritable k2)
        return compare(k1.stationid, k2.stationid)
```

Explanation : Reduce function call will process records (compositevalueobjects) that are grouped together by stationid. The objects will appear in sorted order of year due to the sort by key comparator. Key Comparator class sorts all the objects by year, if the stationids are same.

Performance Comparison

No-Combiner

Run 1 : Running Time for No-Combiner (Iteration 01) - 62 sec

Run 2 : Running Time for No-Combiner (Iteration 02) - 66 sec

Combiner

Run 1 : Running Time for Combiner (Iteration 01) - 60 sec

Run 2 : Running Time for Combiner (Iteration 02) - 62 sec

In-Mapper Combiner

Run 1 : Running Time for In-Mapper Combiner (Iteration 01) - 58 sec

Run 2 : Running Time for In-Mapper Combiner (Iteration 02) - 58 sec

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

  Combiner is called in the program Combiner. Yes, it was called more than once per Map Task based on the spilled records from map output buffer. If we divide the spilled records when comparing no combiner and combiner, the value is ~39, which is more than 17 map tasks (> 1 per map task).

- What difference did the use of a Combiner make in Combiner compared to NoCombiner?

  Combiner aggregates the data locally on each Mapper machine before it is shuffled across the network to various reducers. This decreases a lot of load on each reducer due to less memory used and as well as bytes shuffled. No Combiner on other hand does not do any kind of aggregation before reducers. So reducers input records will be same as no. of map output records. If we are to compare syslog of both files, we can see that

  Reduce input records=223782 (Combiner)

  Reduce shuffle bytes=3427761 (Combiner)

  Reduce input records=8798241(No Combiner)

  Reduce shuffle bytes=47399057 (No Combiner)

  These attest to our understanding that no Combiner is not as efficient as Combiner.

- Was the local aggregation effective in InMapperComber compared to NoCombiner?

  Yes. Because the amount of bytes output from map is lesser due to pre aggregation in each map task while NoCombiner doesn't do any kind of pre-aggregation. And also InMapper Combiner does all the aggregation in-memory. So the amount of map output bytes written is more for No Combiner. If we are to look at the syslog files of both, we can see

  Map output records=8798241 (No Combiner)

  Map output bytes=146602330 (No Combiner)

  Map output records=223782 (In Mapper Combiner)

  Map output bytes=4413409 (In Mapper Combiner)

  These attest to our understanding that local aggregation is indeed effective.

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

  InMapperCombiner is better than Combiner.

  Combiner is a process that runs locally on each Mapper machine to pre-aggregate data before it is shuffled across the network to the various cluster Reducers.

  The in-mapper combiner takes this optimization a bit further: the aggregations do not even write to local disk: they occur in-memory in the Mapper itself.

  Map output bytes=146602330 (Combiner)

  Map output bytes=4413409 (In Mapper Combiner)

  These attest to our understanding that InMapperCombiner is indeed better.

- 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.

  Run Time for sequential version of the program with 1991.csv data = 15,792 sec
  If we compare, sequential with Mapreduce programs, MR programs seems to have more run time in general than Sequential. This is mainly because of no local disk operations and shuffling of bytes and also data latency between machines. Both versions of programs are accurate and do not have any kind of differences.