CS. 6240-02 HW2 Vikas Janardhanan

For each of the three programs in 1 above, write compact pseudo-code. Look at the online modules and your lecture notes for examples. Remember, pseudo-code captures the essence of the algorithm and avoids wordy syntax.

a)

NoCombiner:

1. Create StationTempInfo having below flags:
   isMaxRecord -> Indicates if min/max record
   temp -> contains the temperature value
2. Map(line from file)

```
stationDetails <- Splitandgetfields(line)
stationID <- stationDetails[0]
If stationDetails[2] = “TMIN”:
        emit(stationID,(false,stationDetails[3])
else if stationDetails[2] = “TMAX”
        emit(stationID,(false,stationDetails[3])
```

3. reduce(stationID,List<StationTempInfo>)

```
Initialize tMaxAvg,tMinAvg,tMaxCnt,tMinCnt as 0
For each stationInfo in List<StationTempInfo>:
        If stationInfo.isMaxRecord:
                tMaxAvg+=stationInfo.temp
                tMaxCnt+=1
        else:
                tMinAvg+=stationInfo.temp
                tMinCnt+=1
result <- stationID+”, “+tMaxAvg/tMaxCnt+”, “+tMinAvg/tMinCnt
emit(null,result)
```

b)Combiner:

1. Create StationTempInfo having below flags:
   maxtemp -> Aggregate sum of max temperature
   mintemp -> Aggregate sum of min temperature
   maxcnt - > count of max temperature records encountered
   mincnt -> count of min temperature records encountered.
2. Map(line from file)

stationDetails <- Splitandgetfields(line)
stationID <- stationDetails[0]
If stationDetails[2] = “TMIN”:
emit(stationID,(0,stationDetails[3],0,1)
else if stationDetails[2] = “TMAX”
emit(stationID,(stationDetails[3],0,1,0)

3. Combiner(stationID,List<StationTempInfo>)
   Initialize combinedStationInfo of type StationTempInfo
   For each stationInfo in List<StationTempInfo>:
   combinedStationInfo.maxtemp+=stationInfo.maxtemp
   combinedStationInfo.mintemp+=stationInfo.mintemp
   combinedStationInfo.maxcnt+=stationInfo.maxcnt
   combinedStationInfo.mincnt+=stationInfo.mincnt
   emit(stationID,combinedStationInfo)
4. reduce(stationID,List<StationTempInfo>)
   Initialize tMaxAvg,tMinAvg,tMaxCnt,tMinCnt as 0
   For each stationInfo in List<StationTempInfo>:
   If stationInfo.isMaxRecord:
   tMaxAvg+=stationInfo.temp
   tMaxCnt+=1
   else:
   tMinAvg+=stationInfo.temp
   tMinCnt+=1
   result <- stationID+”, “+tMaxAvg/tMaxCnt+”, “+tMinAvg/tMinCnt
   emit(null,result)

c)InMapperComb:

1. Create StationTempInfo having below flags:
   maxtemp -> Aggregate sum of max temperature
   mintemp -> Aggregate sum of min temperature
   maxcnt - > count of max temperature records encountered
   mincnt -> count of min temperature records encountered.
2. Mapper class:
   Initialize stationtempInfos hashmap in setup() function
   Map(line from file)
   stationDetails <- Splitandgetfields(line)

stationID <- stationDetails[0]
If stationDetails[2] = “TMIN”:
If stationID is present in stationtempInfos:
Update mincnt and mintemp with the value found in hashmap
Else:
Add new entry into hashmap with mincnt and mintemp value.
else if stationDetails[2] = “TMAX”
If stationID is present in stationtempInfos:
Update maxcnt and maxtemp with value found in hashmap
Else:
Add new entry into hashmap with maxcnt and maxtemp value found
cleanup()
foreach stationID in stationtempInfos:
emit(stationID,stationtempInfos[stationID])

3. reduce(stationID,List<StationTempInfo>)
Initialize tMaxAvg,tMinAvg,tMaxCnt,tMinCnt as 0
For each stationInfo in List<StationTempInfo>:
If stationInfo.isMaxRecord:
tMaxAvg+=stationInfo.temp
tMaxCnt+=1
else:
tMinAvg+=stationInfo.temp
tMinCnt+=1
result <- stationID+”, “+tMaxAvg/tMaxCnt+”, “+tMinAvg/tMinCnt
emit(null,result)

d) SecondarySort

1. Create StationTempInfo class having below flags:
maxtemp -> Aggregate sum of max temperature
mintemp -> Aggregate sum of min temperature
maxcnt - > count of max temperature records encountered
mincnt -> count of min temperature records encountered

year -> year when the temperature was recorded

2. StationYearKey class with below fields:
   Year -> year when temperature was recorded
   stationID->stationID associated
   compareTo()
       Compares first by stationID then by year
3. Map(line from file)
       stationDetails <- Splitandgetfields(line)
       stationID <- stationDetails[0]
       year <- extractfromfield1
       If stationDetails[2] = “TMIN”:
           emit((stationID,year),(0,stationDetails[3],0,1,year)
       else if stationDetails[2] = “TMAX”
           emit((stationID,year),(stationDetails[3],0,1,0,year)
4. getPartition()
       Send to reducer based on hashing on stationID
5. Combiner(stationID,List<StationTempInfo>)
   Initialize combinedStationInfo of type StationTempInfo
   For each stationInfo in List<StationTempInfo>:
       combinedStationInfo.maxtemp+=stationInfo.maxtemp
       combinedStationInfo.mintemp+=stationInfo.mintemp
       combinedStationInfo.maxcnt+=stationInfo.maxcnt
       combinedStationInfo.mincnt+=stationInfo.mincnt
   emit(stationID,combinedStationInfo)
   6.KeyComparator()
       Compare by stationID and then if same then compare by year
   7.GroupingComparator()
       Compare only by stationID so that irrespective of the year, all the same stationID records goes in the same reduce call.
   8. reduce(StationYearKey,List<stationTempInfos>)
       cur_year=null
       Initialize tMinAvg,tMaxAvg,tMinCnt,tMaxCnt
       result=StationYearKey.stationID+”, [“
       for each stationTempInfo in stationTempInfos:
           if cur_year!= null and stationTempInfo.year!=cur_year
               Calculate average for cur_year and write to result
               Reset the aggregate variables

Aggregate current records mincnt,maxcnt,mintemp,maxtemp
into tMinCnt,tMaxCnt,tMinTemp,tMaxTemp
Emit(null,result)

Run all three programs from 1 above in Elastic MapReduce (EMR) on the unzipped climate data from 1991.csv, using six m4.large machines (1 master, 5 workers). Report the running time of each program execution. (Find out how to get the running time from a log file. It does not have to be down to a tenth of a second.) Repeat the time measurements one more time for each program, each time starting the program from scratch. Report all 3 programs * 2 independent runs = 6 running times you measured. (12 points)

** Running time is measured based on the log messages "Running job" and "Job completed successfully"

NoCombiner

Run 1:
2017-02-10 20:29:57,798 INFO org.apache.hadoop.mapreduce.Job (main): Running job: job_1486758450215_0001
2017-02-10 20:31:18,610 INFO org.apache.hadoop.mapreduce.Job (main): Job job_1486758450215_0001 completed successfully
Total Time:1 minute 9s

Run 2:
2017-02-10 20:43:38,213 INFO org.apache.hadoop.mapreduce.Job (main): Running job: job_1486758450215_0003
2017-02-10 20:44:45,683 INFO org.apache.hadoop.mapreduce.Job (main): Job job_1486758450215_0003 completed successfully
Total Time:1 minute 7s

Combiner

Run 1:
2017-02-10 19:29:57,461 INFO org.apache.hadoop.mapreduce.Job (main): Running job: job_1486754861975_0001
2017-02-10 19:31:11,000 INFO org.apache.hadoop.mapreduce.Job (main): Job job_1486754861975_0001 completed successfully

Total Time: 1 minute 14s

Run 2:
2017-02-10 20:11:59,391 INFO org.apache.hadoop.mapreduce.Job (main): Running job: job_1486757370112_0001
2017-02-10 20:13:09,937 INFO org.apache.hadoop.mapreduce.Job (main): Job job_1486757370112_0001 completed successfully
Total Time: 1 minute 10s

InMapperComb
Run 1:
2017-02-10 21:21:30,111 INFO org.apache.hadoop.mapreduce.Job (main): Running job: job_1486761205102_0002
2017-02-10 21:22:36,527 INFO org.apache.hadoop.mapreduce.Job (main): Job job_1486761205102_0002 completed successfully
Total Time: 1 minute 6s

Run 2:
2017-02-10 21:30:11,610 INFO org.apache.hadoop.mapreduce.Job (main): Running job: job_1486761205102_0003
2017-02-10 21:31:17,008 INFO org.apache.hadoop.mapreduce.Job (main): Job job_1486761205102_0003 completed successfully
Total Time: 1 minute 6s

CS. 6240-02 HW2 Vikas Janardhanan

Look at the syslog file. It tells you about the number of records and bytes moved around in the system. Try to find out what these numbers actually mean, focusing on interesting ones such as Map input records, Map output bytes, Combine input records, Reduce input records, Reduce input groups, Combine output records, Map output records. Based on this information, explain as much as you can the following, backing up your answer with facts/numbers from the log files: (4 points each)

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

- Yes the combiner in the program as evident from:
  Combine input records=8798241
  Combine output records=223783
  Since there was an almost one fourth reduction in the number of records passed to reducer we can infer that combiner was called more than once per map task.

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

- The number of input records to reducer was decreased by around one fourth in the combiner program as compared to nocombiner because of the use of combiner. As a result of this aggregation of the same key data at the combiner level, the amount of data traffic sent across the network is reduced by a lot. It also reduces the amount of work done at the reducer level.
  Combiner:
  Map output records=8798241
  Combine input records=8798241
  Combine output records=223783
  Reduce input records=223783

  NoCombiner:
  Map output records=8798241
  Combine input records=0
  Combine output records=0
  Reduce input records=8798241

CS. 6240-02 HW2 Vikas Janardhanan

Was the local aggregation effective in InMapperComb compared to NoCombiner?

Based on the syslog information, we can see that local aggregation in InMapperComb was effective:

- Faster runtime observed in InMapperComb than NoCombiner
  InMapperComb:
  2017-02-10 21:21:30,111 INFO org.apache.hadoop.mapreduce.Job (main): Running job: job_1486761205102_0002
  2017-02-10 21:22:36,527 INFO org.apache.hadoop.mapreduce.Job (main): Job job_1486761205102_0002 completed successfully
  Total Time: 1 minute 6s
  NoCombiner:
  2017-02-10 20:29:57,798 INFO org.apache.hadoop.mapreduce.Job (main): Running job: job_1486758450215_0001
  2017-02-10 20:31:18,610 INFO org.apache.hadoop.mapreduce.Job (main): Job job_1486758450215_0001 completed successfully
  Total Time:1 minute 9s

- The number of records sent to reducer is reduced by around one-fourth in InMapperComb as compared to NoCombiner. This results in reduced network traffic as well.
  InMapperComb:
  Map input records=30868726
  Map output records=223783
  Reduce input records=223783
  NoCombiner:
  Map input records=30868726
  Map output records=8798241
  Reduce input records=8798241

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

- InMapperComb has advantages over Combiner with respect to fact that we can be sure that inmappercomb would get executed whereas we don't have control over the execution of Combiner.

Also, InMapperComb aggregates data immediately as it is produced by map call as opposed to Combiner which aggregates after the intermediate data is generated.
But InMapperComb has disadvantage that if the number of unique keys are large, the amount of aggregation happening at mapper would be less and the amount of memory used by hash map would be quite considerable and might not scale well.
However, in our program, the number of unique stationIDs are not that large. So we don't face the disadvantage of using inMemoryComb.
Hence, InMemoryComb is a better choice for the problem we have. It is also supported by the fact that InMemoryComb produced the least running time.

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

- Running time: 11019.2 ms
  The output produced by both seems to be accurate and same.

Run the program from part 2 (secondary sort) above in Elastic MapReduce (EMR), using six m4.large machines (1 master, 5 workers). Report its running time.

- 2017-02-10 23:23:33,294 INFO org.apache.hadoop.mapreduce.Job (main): Running job: job_1486768859864_0001
  2017-02-10 23:24:23,905 INFO org.apache.hadoop.mapreduce.Job (main): Job job_1486768859864_0001 completed successfully
  Total Runtime: 50s