CS-6240 HW-2 Report

Ajay Baban Kauthale(ajayk@ccs.neu.edu)

Weather Data Source Code

Source code is in “HW02/2. Source Code” directory

Map-Reduce Algorithm

Pseudo Code for part 1

1. **NoCombiner:**

Following pseudo code assumes station model with isTMAX (Boolean) and tmp(Double) instance variables. And station model extends writable.

```
class StationModel implements Wirtable {
    Boolean isTMAX; // to check whether record is TMAX or TMIn
    Double tmp; // to record temperature

    // override readFields, write and toString methods
}

class NoCombiner {
    // create map-reduce job
    // set mapper as StationMapper class
    // set reducer as StationTemperatureReducer class
    // set all necessary configuration
    // run the map-reduce job
}

class StationMappper extends Mapper<Object, Text, Text, StationModel> {
    // in map method
    // get all lines from the input
    for each line in values.lines
        if line not contains “TMAX” or “TMIN”
            continue;
        // split the line into words
        If line is TMAX line
            set isTMAX = true
        // set temperature into TMAX value
        // create new station with above TMAX value
        Emit(station)
}

class StationTemperatureReducer extends Reducer<Text, StationModel, Text, NullWritable> {
    // in reduce method
    // initialize total TMAX, total TMIN and counts for both
    For each station in values
        If station isTMAX is true
            // add TMAX to total and increment TMAX count
        Else
            // add TMIN to total and increment TMIN count
    // calculate TMAX and TMIN by dividing total by count
    // write station data into context
}
```

All following pseudo codes assumes station model below

```
class StationModel implements Wirtable {
    Double TMAX;
    Double TMIN;
    Integer TMAXCnt;
    Integer TMINCnt;

    // override readFields, write and toString methods
}
```

2. **Combiner:**

```
class Combiner {
    // create map-reduce job
    // set mapper as StationMapper class
    // set reducer as StationTemperatureReducer class
    // set in mapper combiner TemperatureCombiner class
    // set all necessary configuration
    // run map-reduce job
}

class StationMappper extends Mapper<Object, Text, Text, StationModel> {
    // in map method
    // get all lines from the input
    for each line in values.lines
        if line not contains “TMAX” or “TMIN”
            continue;
        // split the line into words
        If line is TMAX line
            // set temperature into TMAX value
            // create new station with above TMAX value
        Else
            // set temperature into TMIN value
            // create new station with above TMIN value

        Emit(station)
}

class StationTemperatureReducer extends Reducer<Text, StationModel, Text, NullWritable> {
    // in reduce method
    // initialize total TMAX, total TMIN and counts for both
    For each station in values
        If station isTMAX is true
```

```
                // add TMAX to total and increment TMAX count
            Else
                // add TMIN to total and increment TMIN count
        // calculate TMAX and TMIN by dividing total by count
        // write station data into context
}
```

```
class TemperatureCombiner extends Reducer<Text, StationModel, Text, StationModel> {
        // in reduce method
        // initialize total TMAX, total TMIN and counts for both
        For each station in values
            If station isTMAX is true
                // add TMAX to total and increment TMAX count
            Else
                // add TMIN to total and increment TMIN count
        // calculate TMAX and TMIN by dividing total by count
        // write station data into context
}
```

3. **InMapperComb:**

```
class InMapperComb {
        // create map-reduce job
        // set mapper as StationMapper class
        // set reducer as StationTemperatureReducer class
        // set all necessary configuration
        // run map-reduce job
}
```

```
class StationMappper extends Mapper<Object, Text, Text, StationModel> {
        // declare a map for storing stations
        Map StationMap;

        // set up the StationMap empty hashmap using setup() method

        // in map method
        // get all lines from the input
        for each line in values.lines
            if line not contains "TMAX" or "TMIN"
                continue;
            // split the line into words
            If StationMap not contains current station
                // create new station and add it to map
            If line is TMAX line
                // set temperature into TMAX value
                // create new station with above TMAX value
            Else
                // set temperature into TMIN value
                // create new station with above TMIN value
                // Update the **StationMap**

        // cleanup the **StationMap** by emitting all aggregated data to context using cleanup() method
}

class StationTemperatureReducer extends Reducer<Text, StationModel, Text, NullWritable> {
        // in reduce method
        // initialize total TMAX, total TMIN and counts for both
        For each station in values
                If station isTMAX is true
                        // add TMAX to total and increment TMAX count
                Else
                        // add TMIN to total and increment TMIN count
        // calculate TMAX and TMIN by dividing total by count
        // write station data into context
}
```

Pseudo Code for part 2

Models:

Here I used 2 models, one is for station composite key and another for station data.

```
class StationKey implements WritableComparable<StationKey> extends Writable {
        String id; // station id
        Integer year; // year

        // override hashcode, equals, compareTo, readfields and write methods
}

class StationModel implements Writable {
        Double TMAX;
        Double TMIN;
        Integer TMAXCnt;
        Integer TMINCnt;

        // override readFields, write and toString methods
}
```

Comparators:

Here I used 2 comparators, one is group comparator and another one is key comparator

```
class StationComparator extends WritableComparator {
        // implement compare method for stationkey
}

Class StationKeyComparator extends WritableComparator {
        // implement compare method for stationkey data
}
```

SecondarySort:

```
class SecondarySort {
    // create map-reduce job
    // set mapper as StationMapper class
    // set reducer as StationTemperatureReducer class
    // set ouputkeyclass and outputvalueclass to StationKey and StationModel classes
    // set all necessary configuration
    // run map-reduce job
}

class StationMappper extends Mapper<Object, Text, Text, StationModel> {
    // declare a map for storing stations
    Map StationMap;

    // set up the StationMap empty hashmap using setup() method

    // in map method
    // get all lines from the input
    for each line in values.lines
        if line not contains “TMAX” or “TMIN”
            continue;
        // split the line into words
        If StationMap not contains current station
            // create new station and add it to map
        If line is TMAX line
            // set temperature into TMAX value
            // create new station with above TMAX value
        Else
            // set temperature into TMIN value
            // create new station with above TMIN value

        // Update the StationMap

    // cleanup the StationMap by emitting all aggregated data to context using cleanup() method
}

class StationTemperatureReducer extends Reducer<Text, StationModel, Text, NullWritable> {
    // in reduce method
    // initialize total TMAX, total TMIN and counts for both
    // initialize current year and last year
    For each station in values
        If current year != previous year
            // append the result and reinitialize total TMIN, total TMAX and counts
        // set total TMAX, total TMIN and counts from station values

    // for last year
    If current year == previous year
        // append the result

    // write result string into context
```

}

Record display order for part 2:

The reduce function will process records by the station values and create new output when we get matching year, otherwise the records is processed as subsequent one and appended to the result.

The records will be displayed in following format
<Station Id>, [(<Year> ,<TMIN> , <TMAX>)]

Spark Scala program:

I. I intended to use RDD API for part 1. Although, it can be achieved using DataFrames, RDD seems easier to me because I am familiar with Java and this interface is closer to Java . In part1t we are having multiple data points such as StationKey (Id, Year), TMAX, TMIN etc. but most focus we are having here is on StationKey and temperature.

II. Spark Scala program/ Commands for part 1:

Commands need to be used

1. case class for creating classes
2. textFile() for getting raw split data
3. map() for mapping the key-value pair
4. safeInt(), safeDouble(), listData() etc. handling various data types.
5. Data(..) for getting data.
6. hashCode() for getting hashcode
7. forEach() for iterating over records.
8. Collect().foreach(println) for writing the data.
9. reduceByKey() for reducing the temperature data.
10. groupByKey() for grouping the data by key.

Example:

Create composite StationKey using scala:

```
case class StationKey(Id:String, Year:Int)
```

Create key-value pairs using map:

```
val rawDataArray = sc.textFile(args(0)).map(line => line.split(","))
val temperatureData = rawDataArray.map(arr => createKeyValueTuple(arr))

 //supporting code
 def createKeyValueTuple(data: Array[String]) :(StationKey,List[String]) = {
    (createKey(data),listData(data))
  }

  def createKey(data: Array[String]): StationKey = {
   StationKey (safeInt(data(STATION_ID)), safeInt(data(YEAR)))
  }

def listData(data: Array[String]): List[String] = {
   List(data(TMAX), data(TMIN), data(TMAXCnt), data(TMINCnt))
  }
```

Partitioning using Scala:

```
class StationPartitioner(partitions: Int) extends Partitioner {
  require(partitions >= 0, s"Number of partitions ($partitions) cannot be negative.")

  override def numPartitions: Int = partitions

  override def getPartition(key: Any): Int = {
    val k = key.asInstanceOf[StationKey]
    k.Id.hashCode() % numPartitions
  }
 }
```

Combiner using Scala: (Using custom partioner)

```
object Combiner extends SparkJob {

 def runCombinerExample(args: Array[String]): Unit = {

   val sc = context("Combiner")
   val rawDataArray = sc.textFile(args(0)).map(line => line.split(","))
   val temperatureData = rawDataArray.map(arr => createKeyValueTuple(arr))

   val keyedDataSorted = temperatureData.repartitionAndSortWithinPartitions(new StationPartitioner (1))

   //only done locally for demo purposes, usually write out to HDFS
   keyedDataSorted.collect().foreach(println)
 }
}
```

III. Aggregate functions used by programs:

1. NoCombiner: should use reduceByKey function
2. Combiner: should use reduceByKey, combineByKey functions.
3. InMapperComb: aggregateByKey and aggregate function

IV. Spark Scala program/ Commands for part 2:

Create composite StationKey using scala:

```
case class StationKey(Id:String, Year:Int)
```

Create key-value pairs using map:

```
val rawDataArray = sc.textFile(args(0)).map(line => line.split(","))
val temperatureData = rawDataArray.map(arr => createKeyValueTuple(arr))

 //supporting code
 def createKeyValueTuple(data: Array[String]) :(StationKey,List[String]) = {
    (createKey(data),listData(data))
 }

 def createKey(data: Array[String]): StationKey = {
   StationKey (safeInt(data(STATION_ID)), safeInt(data(YEAR)))
 }
```

```
def listData(data: Array[String]): List[String] = {
    List(data(TMAX), data(TMIN), data(TMAXCnt), data(TMINCnt))
  }
```

Partitioning using Scala:

```
class StationPartitioner(partitions: Int) extends Partitioner {
    require(partitions >= 0, s"Number of partitions ($partitions) cannot be negative.")

    override def numPartitions: Int = partitions

    override def getPartition(key: Any): Int = {
      val k = key.asInstanceOf[StationKey]
      k.Id.hashCode() % numPartitions
    }
  }
```

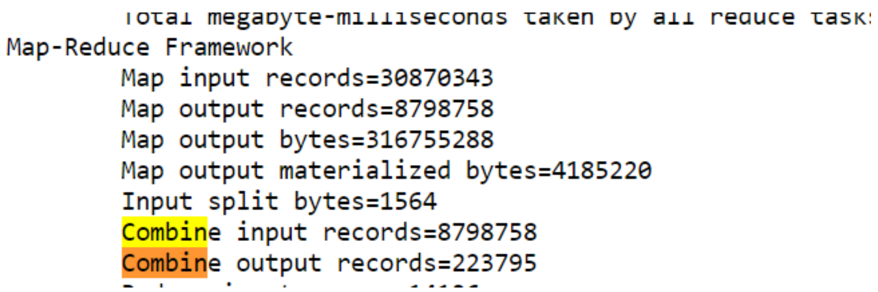
Secondary sort using Scala:

```
object SecondarySort extends SparkJob {

  def runSecondarySortExample(args: Array[String]): Unit = {

    val sc = context("SecondarySorting")
    val rawDataArray = sc.textFile(args(0)).map(line => line.split(","))
    val temperatureData = rawDataArray.map(arr => createKeyValueTuple(arr))

    val keyedDataSorted = temperatureData.repartitionAndSortWithinPartitions(new StationPartitioner (1))

    //only done locally for demo purposes, usually write out to HDFS
    keyedDataSorted.collect().foreach(println)
  }
}
```

Performance Comparison on EMR

Program	**Run 1**	**Run 2**
NoCombiner	1 minute 4 seconds	57 seconds
Combiner	56 seconds	59 seconds
InMapperComb	57 seconds	55 seconds

1. Was the Combiner called at all in program Combiner? Was it called more than once per Map task?
   No custom combiner is not called as I can see in the Syslog of the Combiner. Instead I can see two lines in Map-Reduce framework which are combining input and output



```
                Total megabyte-milliseconds taken by all reduce task
        Map-Reduce Framework
                Map input records=30870343
                Map output records=8798758
                Map output bytes=316755288
                Map output materialized bytes=4185220
                Input split bytes=1564
                Combine input records=8798758
                Combine output records=223795
```

As per map task it should be called more than once.

2. Was the local aggregation effective in InMapperComb compared to NoCombiner?
   As per the performance comparison above it seems effective since we are aggregating the data into HashMap in map phase with additional memory overhead.

Running time part 2:

Program	**Run 1**
SecondarySort	56 seconds