CS6240 Parallel Data Processing Sec 01 Spring 2017

Harsha Jakkappanavar

Map-Reduce Algorithm

Pseudo code:

1. No Combiner:

```
// map (key1, value1)
map (offset B, line L)
    // B (key1): the byte offset of the line of text in the input file
    // L (value1): is a single line at that offset in document.
    // Extract temperature type, station id and temperature from the line and
    // initialize the variables.
    temperatureType, stationId, temperature;
    if the temperature type is either “TMAX” or “TMIN”
        // emit station id and accumulator datastructure.
        emit (stationId, {temperatureType, temperature});

// reducer (key2, value2)
reduce (stationId, [tA1, tA2. . . ])
    // stationId (key2): is id of the station.
    // [tA1, tA2. . .] (value2): is a list of
    // TemperatureAccumulator {temperatureType, temperature}
    //Initialize the output datastructure MeanTemperatureOutput
    // MeanTemperatureOutput holds two accumulators to count the
    // running sum and running count for TMIN and TMAX.
    for each tempAccumulator in [tA1, tA2. . .]
        if the temperatureType in tempAccumulator is TMIN
            // Update TMIN MeanAccumulator of the output datastructure with
            // temperature and count of 1
            MeanTemperatureOutput.updateTMinMeanAccumulator(temperature, 1);
        else
            // Update TMAX MeanAccumulator of the output datastructure with
            // temperature and count of 1
            MeanTemperatureOutput.updateTMaxMeanAccumulator(temperature, 1);

// MeanTemperatureOutput prints average for TMIN and TMAX
emit (stationId, MeanTemperatureOutput);
```

2. Cominer

```
// map (key1, value1)
map (offset B, line L)
    // B (key1): the byte offset of the line of text in the input file
    // L (value1): is a single line at that offset in document.
    // Extract temperature type, station id and temperature from the line and
    // initialize the variables.
    temperatureType, stationId, temperature;
    if the temperature type is either “TMAX” or “TMIN”
        // emit station id and accumulator datastructure.
        emit (stationId, {temperatureType, temperature, 1});

// reducer (key2, value2)
reduce (stationId, [tA1, tA2. . . ])
    // stationId (key2): is id of the station.
    // [tA1, tA2. . .] (value2): is a list of
    // TemperatureAccumulator {temperatureType, temperature, count}
    //Initialize the output datastructure MeanTemperatureOutput
    // MeanTemperatureOutput holds two accumulators to count the
    // running sum and running count for TMIN and TMAX.
    for each tempAccumulator in [tA1, tA2. . .]
        // Extract temperature and count from the accumulator datastructure
        temperature, count;
        if the temperatureType in tempAccumulator is TMIN
            // Update TMIN MeanAccumulator of the output datastructure with
            // temperature and count
            MeanTemperatureOutput
                .updateTMinMeanAccumulator(temperature, count);
        else
            // Update TMAX MeanAccumulator of the output datastructure with
            // temperature and count
            MeanTemperatureOutput
                .updateTMaxMeanAccumulator(temperature, count);

// MeanTemperatureOutput prints average for TMIN and TMAX
emit (stationId, MeanTemperatureOutput);
```

```
// combiner (key2, value2)
Combiner (stationId, [tA1, tA2. . .])
    // stationId(key2): is the id of the station.
    // [tA1, tA2. . .] (value2): is a list of
    // TemperatureAccumulator {temperatureType, temperature, count}
    // Initialize two accumulator datastructure, one for TMIN and one for TMAX
    tMinTempAccumulator, tMaxTempAccumulator;
    for each tempAccumulator in [tA1, tA2. . .]
        // Extract temperature and count from the accumulator datastructure
        temperature, count;
        if the temperatureType in tempAccumulator is TMIN
            update tMinTempAccumulator with temperature and count
        else
            update tMaxTempAccumulator with temperature and count

    emit (stationId, tMinTempAccumulator);
    emit (stationId, tMaxTempAccumulator);
```

3. InMapperCombiner

```
Mapper
// a HashMap h to synchronize accumulator data in the same map task.
// the hashmap stores the temperature data by station id, by temperature type.
setup ()
    // initialize the hashmap
    // HashMap: Map<StationId, Map<TemperatureType, TemperatureAccumulator>>
    temperatureAccumulatorMap = new hashmap();

// the map call reads each line and updates the map by station id, by temperature type
// with the corresponding temperature and count.
// map (key1, value1)
map (offset B, line L)
    // B (key1): the byte offset of the line of text in the input file
    // L (value1): is a single line at that offset in document.
    // Extract temperature type, station id and temperature from the line and
    // initialize the variables.
    temperatureType, stationId, temperature;

    if the temperature type is either “TMAX” or “TMIN”
        // fetch the value in the hashmap associated with this station id,
        // if this is empty initialize
        Map<StationId, TemperatureAccumulator>  temperatureTypeAccumulator =
                temperatureAccumulatorMap.get(stationId);
        // fetch the value in this hashmap associated with this temperatureType,
        // if this is empty initialize
        TemperatureAccumulator tempAccumulator =
                temperatureTypeAccumulator.get(temperatureType);

        // update the temperature and increment the count
        // for this accumulator datastructure.
        temperatureAccumulator.updateTemperature(temperature);
        temperatureAccumulator.updateCountSoFar(1);
        // update the hashmap with this data
        temperatureTypeAccumultor.put(temperatureType, temperatureAccumulator);
        temperatureAccumulatorMap.put(stationId, temperatureTypeAccumulator);
```

```
// prints the hashmap contents to the output
cleanup ()
    // iterate over the map contents and emit the output.
    for each
        Map<TemperatureType, TemperatureAccumulator> tempTypeAccumulator
            in temperatureAccumulatorMap
        for each TemperatureAccumulator in tempTypeAccumulator
            emit (stationId, temperatureAccumulator);


Reducer
// reducer (key2, value2)
reduce (stationId, [tA1, tA2. . . ])
    // stationId (key2): is id of the station.
    // [tA1, tA2. . .] (value2): is a list of
    // TemperatureAccumulator {temperatureType, temperature, count}
    //Initialize the output datastructure MeanTemperatureOutput
    // MeanTemperatureOutput holds two accumulators to count the
    // running sum and running count for TMIN and TMAX.
    for each tempAccumulator in [tA1, tA2. . .]
        // Extract temperature and count from the accumulator datastructure
        temperature, count;
        if the temperatureType in tempAccumulator is TMIN
            // Update TMIN MeanAccumulator of the output datastructure with
            // temperature and count
            MeanTemperatureOutput
                .updateTMinMeanAccumulator(temperature, count);
        else
            // Update TMAX MeanAccumulator of the output datastructure with
            // temperature and count
            MeanTemperatureOutput
                .updateTMaxMeanAccumulator(temperature, count);

// MeanTemperatureOutput prints average for TMIN and TMAX
emit (stationId, MeanTemperatureOutput);
```

4. Secondary sort

```
Mapper
// map (key1, value1)
map (offset B, line L)
    // B (key1): the byte offset of the line of text in the input file
    // L (value1): is a single line at that offset in document.
    // Extract temperature type, station id and temperature from the line and
    // initialize the variables.
    temperatureType, stationId, temperature;
    if the temperature type is either “TMAX” or “TMIN”
        // emit station id and accumulator datastructure.
        emit ({stationId, year}, {temperatureType, temperature});

Group Comparator
// sorts by station id, so that output of all the map call with same station id (in the key)
// goes to the same reduce call
groupComparator({stationId, year}, {stationId, year})
//      sorts the stations in ascending order, ignores year while sorting.
        sortByStationId();


Key Comparator
// sorts the records by stationId and then by year
keyComparator({stationId, year}, {stationId, year})
    // sorts the stationId in ascending order, for the colliding stationIds, the function
    // sorts by year in ascending order.
    sortByStationIdAndYear();

Partitioner
// partitions based on the hashcode of the stationId. The output of all the map call with
// the same station id (in the key) goes to the same reducer
    Partitioner({stationId, year}, {temperatureType, temperature})
        return hascode(stationId) % n // n is the number of partitions.
```

```
Reducer
// reducer (key2, value2)
reduce ({stationId, year}, [tA1, tA2. . . ])
    // {stationId, year} (key2): is a data structure,
    // to hold the station id and year as a key.
    // [tA1, tA2. . .] (value2): is a list of
    // TemperatureAccumulator {temperatureType, temperature}
    //Initialize the output datastructure MeanTemperatureOutput
    // MeanTemperatureOutput holds two accumulators to count the
    // running sum and running count for TMIN and TMAX.
    stationId = key.getStationId();
    year = key.getYear();
    StringBuilder sb = new StringBuilder(); // initialize a new string builder.
    for each tempAccumulator in [tA1, tA2. . .]
        // check if the year has changed
        if(key.getYear() not equal to year)
            // record the average TMIN and TMAX temperatures for this year in the
            // string builder and reinitialize the year and MeanTemperatureOutput
            sb.append(year, MeanTemperatureOutput)
            year = key.getYear();
            MeanTemperatureOutput = new MeanTemperatureOutput();
        if the temperatureType in tempAccumulator is TMIN
            // Update TMIN MeanAccumulator of the output datastructure with
            // temperature and count of 1
            MeanTemperatureOutput.updateTMinMeanAccumulator(temperature, 1);
        else
            // Update TMAX MeanAccumulator of the output datastructure with
            // temperature and count of 1
            MeanTemperatureOutput.updateTMaxMeanAccumulator(temperature, 1);

// emit the output from the string buffer.
emit (NullWritable, sb.toString());
```

The mapper parses the records by line and emits {stationId, year} as the intermediate key. We have defined a KeyComparator the sorts these intermediate keys by station id and year in ascending order. We have also defined a Partitioner the partitions the intermediate keys by station id, the partitioner makes sure that the keys from the same station id are going to the same reducer. We have also defined a Group Comparator, which groups the intermediate keys by station id and ignores the year part of the key. This makes sure that the keys having the same station id which are going to a reducer, with different year values are grouped and passed in the same reduce call.

Performance Comparison

Running times

1. No Combiner

Run 1 (in secs)	Run 2 (in secs)
70	76

2. Combiner

Run 1 (in secs)	Run 2 (in secs)
74	69

3. In Mapper Combiner

Run 1 (in secs)	Run 2 (in secs)
67	62

4. Secondary Sort: 85 secs

Questions:

1. Was the Combiner called at all in program Combiner? Was it called more than once per Map task?
   - The combiner was called and more than once, when I ran the Combiner program. Looking at the output logs it can be confirmed that the Combiner received "8798241" input records and gave "447566" output records. These values are zero when I run the NoCombiner program. It is difficult to identify if the combiner was called more than once for individual Map task, but in the observed case, the output records from the Mappers is equal to the input records at the Combiner, which implies that all the outputs from the Mapper were received at the Combiner.

2. What difference did the use of a Combiner make in Combiner compared to NoCombiner?
   - There is a slight time difference in the run times of Combiner and NoCombiner, Combiner running faster (Tabular data above). Also the Reduce shuffle bytes processed by the NoCombiner program "51730410" is considerably reduced in case

of Combiner program “4951902”, indicating that some of the data is processed and reduced in the Combiner program reducing the load at the Reducer.

3. Was the local aggregation effective in InMapperComb compared to NoCombiner?
   - Using local aggregation in InMapperCombiner decreased the running time of the program relative to NoCombiner. The Reduce shuffle bytes from the NoCombiner program “51730410” is reduced to “4955368” with the InMapperCombiner. The number of records received a the Reducer is reduced from “8798241” (with No Combiner) to “445204” (with InMapperCombiner) reducing the data traffic.

4. Which one is better, Combiner or InMapperComb? Briefly justify your answer.
   - Based on the running times observed InMapperCombiner is better that Combiner. The programmer has a control over the InMapperCombiner and the guarantee that the InMapperCombiner will run all the time, which is not possible to ensure in the Combiner program. As long as the data isn’t too big (such that the heap runs out of memory) it’s better to use InMapperCombiner.

5. How do the running times and accuracy of these MapReduce programs compare to 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.
   - I modified the sequential version of the code and the program took 32.74 seconds. Comparing the outputs of the two execution, I find that the outputs are same and correct. Various parameters like the size of the data being smaller and the additional task of mapping, reducing, data transfer traffic time and Hadoop overhead makes the MapReduce programs take more time. When dealing with larger data sets this time difference will reduce considerably.