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());
```