**DATABASE MANAGEMENT SYSTEMS**

Submitted by,

MOHIT MORARKA (861130666)

PRASHANTH PONNADI (861135032)

**1> Collaboration Details**

|  |  |
| --- | --- |
| **MOHIT** | **PRASHANTH** |
| Mapper code | Parsing the input file to get weather attributes |
| Reducer code | Multiple reducers |
| Tree structure for multiple map reduce rounds | Output the final skyline results from last map reducer. |
| Removing the dominated points from each layer using max and min comparison | Data structure for holding the weather attributes and getter setter methods. |
| Experimental results | Experimental results |

**2> Description of the source code**

**1)** **Parsing the input file to get our weather attributes**-

We first create a weather object to hold our weather attributes.

Then we parse the input records to get the attributes of our weather object.

We require the following attributes from the file-

/\*\*

\* The file assumes a starting point of 1

\* StationID: 1-6 Int

\* Year: 15-18 Int

\* Moda: 19-22 Double

\* Temp: 25-30 Double

\* Dewp: 36-41 Double

\* SLP: 47-52 Double

\* Max: 103-108 Double

\* STP: 58-63 Double

\* WDSP: 79-83 Double

\* MXSPD: 89-93 Double

\* Gust: 96-100 Double

\* Min: 111-116 Double

\*/

We use regular expressions to remove the empty spaces from the line. We also remove the \* characters from the line to get clean weather attributes.

We store these attributes in a weather object and then push this object into our data structure.

**2) Mapper Pseudo Code-**

Input for our mapper is the weather data

Output

Key is the modulus of our weather object

long mod = newKey % Integer.parseInt(context.getConfiguration().get("mod"));

Here the newKey is a just a key to generate unique key which is incremented for each line as we read the weather attributes.

Value is the weather object which contains all the weather attributes which we have mentioned above

**I have added a snapshot of my Mapper code here-**

****

**3) Reducer Pseudo Code-**

Input for our reducer is the weather data which was output by the mapper and it gets divided between the multiple reducers based on their key.

Output

Key is the modulus of our weather object

long mod = newKey % Integer.parseInt(context.getConfiguration().get("mod"));

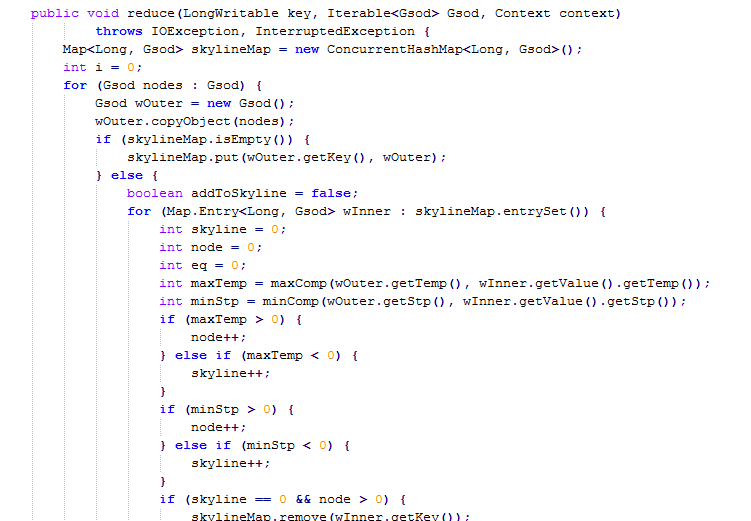
Value

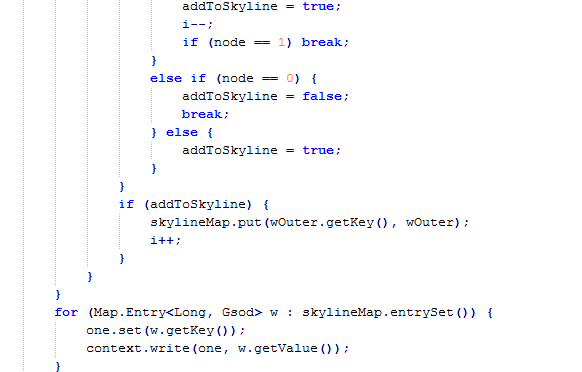
We remove all the dominated points from one layer of the tree in a single reducer round and output it which becomes the input for our next map reduce round.

In the reducer we are doing the comparison for our maximum and minimum attributes and for every weather object in that tree layer with the points which are already there in the skyline.

We check whether a point is dominated by a point in the skyline, If it does we remove that point otherwise we push that object into our Skyline Map.

We take two variables here one for the skyline and one for the weather object. We do the maximum and the minimum attribute comparisons and keep on incrementing and decrementing it based on the max and min comparisons and then check for the final value of these variables to determine whether the point dominates or not.

****

****

**4) Tree Structure for Multiple Map Reduce rounds –**

We specify an upper bound and a lower bound which determines how many rounds of Map reduce we should have. Number of rounds is obtained by dividing the upper bound with the lower bound after every map reduces round.

We have a tree structure dependent on these upper and lower bounds which determine how many layers we require in the tree. We can set these bounds depending upon the amount of input data we have, For instance for a small data set we might not require a large number of tree layers. So by providing these upper and lower bound values we can dynamically assign the number of layers we require in our tree structure

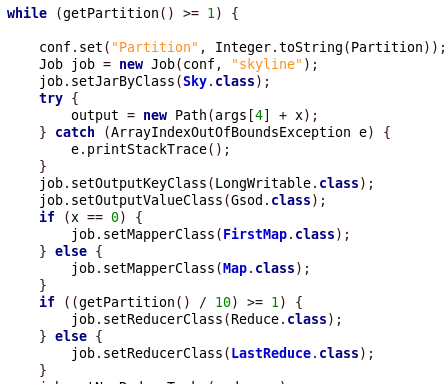
During each layer which is a separate map reduce round we are eliminating the records so that the weather data keeps on shrinking.

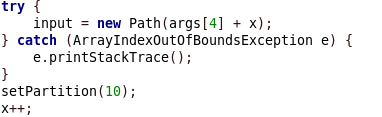
After every map reduce round output of our reducer becomes the input our next map reduce round which ensures that we are removing the non-dominated points at each layer of our tree.

Eventually at the end we have the root of the tree which is again a Map reduce round which will output only the non-dominated values residing on the skyline.

Multiple map reduce rounds ensure us to have a performance efficient skyline computation which would not be the case if we have a single round of map reduce.

I have added a snapshot describing how we create multiple map reduce rounds and how the output for one map reduce round becomes the input for our next map reduce round.

****

****

**5) Multiple reducers –**

We have used multiple reducers in order to improve the efficiency of our skyline computation.

We specify the number of reducers in our job configuration using setNumReducerTasks.

job.setNumReduceTasks(reducers);

In each map reduce round we divide the output of Mapper into multiple reducers based on the key.

**6) Data Structure for Holding the Weather data –**

We have used a data structure for holding the weather attributes, In our design we have named it as Gsod.

Inside the mapper it reads the input data file and extracts the nine attribute values needed for our skyline computation and push it to our Gsod data structure.

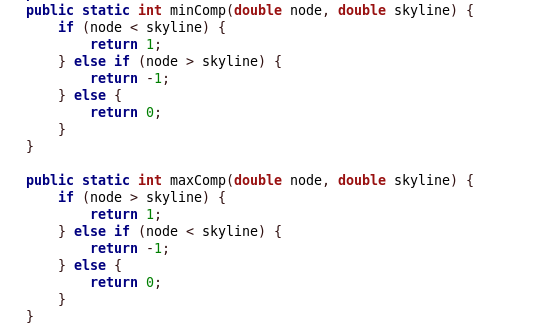
Gsod contains the getter and setter methods for updating and retrieving the attribute values for our weather attributes.

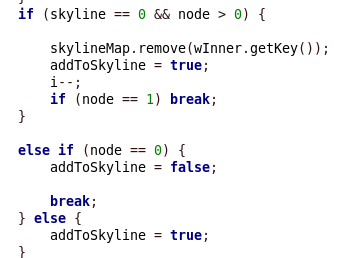
**7) Pruning the dominated points -**

At every layer we do the max and the minimum comparisons and keep on pruning the data points which are already in the skyline.

This is more of a naïve comparison between the node and the skyline points.

I have added snapshot for the comparison done for determining whether a point belongs to the skyline





**8> Experimental Results**

**For our experiments we have made used of a single mapper and multiple reducers.**

**We have used 1, 2 and 4 reducers to ensure scalability.**

**Number of Reducers -1**

**We have calculated these values for 3 runs-**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Run** | **Number of Reducers** | **Total Mapper Time** | **Total Reducer Time** | **Total Time** |
| 1 | 1 | 5840 | 87792 | 93632 |
| 2 | 1 | 5932 | 89298 | 95230 |
| 3 | 1 | 5496 | 88558 | 94054 |

**Average Running Time: 94305.3**

**Number of Reducers -2**

**We have calculated these values for 3 runs-**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Run** | **Number of Reducers** | **Total Mapper Time** | **Total Reducer Time** | **Total Time** |
| 1 | 2 | 7867 | 75759 | 83626 |
| 2 | 2 | 7623 | 75621 | 83244 |
| 3 | 2 | 7931 | 75624 | 83555 |

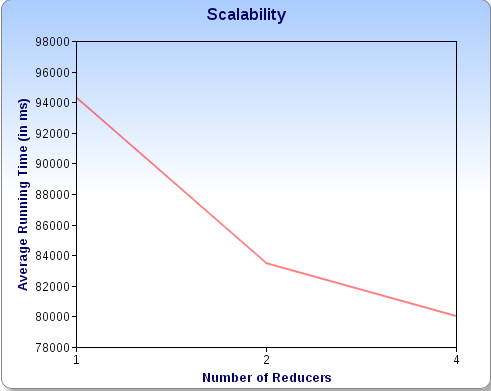
**Average Running Time: 83475.7**

**Number of Reducers - 4**

**We have calculated these values for 3 runs-**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Run** | **Number of Reducers** | **Total Mapper Time** | **Total Reducer Time** | **Total Time** |
| 1 | 4 | 12294 | 69187 | 81481 |
| 2 | 4 | 12383 | 67158 | 79541 |
| 3 | 4 | 12274 | 66763 | 79037 |

**Average Running Time: 80019.66**



**Discussion-**

As we can see from the graph that when we are increasing the number of reducers the total running time for mappers and reducers is decreasing.

As we are scaling up the number of reducers multiple reducers can work in parallel and thus increase the efficiency of our skyline computation.

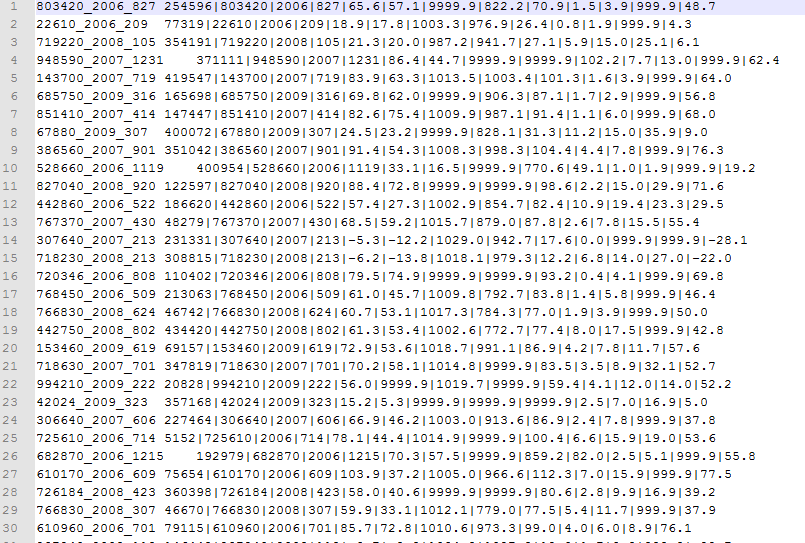
**Skyline Result-**

I have attached one result file which contains all the skyline points.

For the 11 million data set we have a skyline of 31,860 points.

I have added a snapshot of our result here-

Result is in the form objectID = concat( STN, “\_” , YEAR,”\_”, MODA)



**Instructions on how to reproduce the results-**

1. First create an input directory folder in your Hadoop file system.

Hadoop fs –mkdir <input-dir>

1. Copy the input data file which I have attached “gsod\_aggregated.out” and copy it to your input directory folder.
2. I have attached one executable jar file named “Skyline-1.0-SNAPSHOT.jar”
3. Command line parameters to be passed are number of reducers, input directory and output directory.

hadoop jar Skyline-1.0-SNAPSHOT.jar edu.cs236.skyline.Skyline <no\_reducers> <Input-dir> <Output-Dir>

number of reducers can be 1, 2 or 4.

edu.cs236.skyline.Skyline is the class name.

Eg- hadoop jar Skyline-1.0-SNAPSHOT.jar edu.cs236.skyline.Skyline 4 Input Output

1. I have also attached all the source files and the result file file which contains all the skyline points created in the final result round by removing all the dominated points at all the levels of the tree.
2. This project was done on maven and I have also attached the pom.xml file.
3. In case we have 4 levels of the tree our final reducer result would be stored in Output directory appended by a variable x .We will have 4 Output directories Output0, Output1, Output2, Output3. Final skyline result would be stored in Output3/part-r-00000
4. All the remaining folders Output0, Output1 and Output2 would contain intermediate results of map reduce rounds.