Group 9 Report

102062132 林育輝 102062315 洪若恒 102062336 楊峻瑋

我們這組選擇的主題是實作 BFR，是根據以下的方法實作：

Processing the "Memory-Load" of points (1):

- **1)** Find those points that are "**sufficiently close**" to a cluster centroid and add those points to that cluster and the **DS**
  - These points are so close to the centroid that they can be summarized and then discarded
- **2)** Use any main-memory clustering algorithm to cluster the remaining points and the old **RS**
  - Clusters go to the **CS**; outlying points to the **RS**

Processing the "Memory-Load" of points (2):

- **3) DS set:** Adjust statistics of the clusters to account for the new points
  - Add ***N***s, ***SUM***s, ***SUMSQ***s
- **4)** Consider merging compressed sets in the **CS**
- **5)** If this is the last round, merge all compressed sets in the **CS** and all **RS** points into their nearest cluster

我們利用三組 MapReduce：

✓ **Initial_DS_Mapper、Initial_DS_Reducer：負責處理 initial data。**

當第一組 memory-load data 來時，Mapper 會在 cleanup 時隨機取三個點作為一開始 DS 的 centroid。

```
public void cleanup(Context context) throws IOException, InterruptedException
{
    Random ran = new Random();

    for(int i = 0; i < NUM; i++) //randomly pick centroids
    {
        int index = ran.nextInt(x_list.size());   ///??????????????
        String out = x_list.get(index) + " " + y_list.get(index);

        double sumsq_x = (Double.parseDouble(x_list.get(index)))*(Double.parseDouble(x_list.get(index)));
        double sumsq_y = (Double.parseDouble(y_list.get(index)))*(Double.parseDouble(y_list.get(index)));
        String summarize = out + " " +String.valueOf(sumsq_x)+ " " +String.valueOf(sumsq_y);
        x_list.remove(index);
        y_list.remove(index);
        context.write(new Text(String.valueOf(1)), new Text(summarize));
    }
    for(int i = 0; i < x_list.size(); i++)
    {
        String out = x_list.get(i) + " " + y_list.get(i);
        context.write(new Text(String.valueOf(0)), new Text(out));
    }
}
```

Reducer 會把所有第一次收到的 data 拿來和 centroid 計算 mahalanobis distance，若點的 distance 小於 threshold 就把他加進離它最近的 cluster，並 update 該 cluster 的 N、SUM、SUMSQ。

```
//add points one by one
for(int j = 0; j < x_list.size(); j++)
{

    if(cluster_flag[j] == true)
    {
        double[] coord = new double[2];
        coord[0] = x_list.get(j);
        coord[1] = y_list.get(j);

        //get mahalanobis distance
        double mahalanobis_dist = 0.0;
        for(int k = 0; k < 2; k++) // get mahalanobis distance of all dimensions k
        {
            double sigma = sumsq[k]/output_N[i] - Math.pow(sum[k]/output_N[i], 2);
            sigma = Math.abs(sigma);
            sigma = Math.sqrt(sigma);
            if(output_N[i]==1)
            {
                sigma = 1.0;
            }
            mahalanobis_dist += Math.pow( (coord[k]-cen[k])/sigma, 2 );
        }
        mahalanobis_dist = Math.sqrt(mahalanobis_dist);
        if(mahalanobis_dist < thresold)
        {
            output_N[i] += 1;
            output_sum[0][i] += coord[0];
            output_sum[1][i] += coord[1];
            output_sumsq[0][i] += Math.pow( coord[0], 2 );
            output_sumsq[1][i] += Math.pow( coord[1], 2 );
            cluster_flag[j] = false;
            String out = String.valueOf(i) + " "+ String.valueOf(coord[0]) + " " + String.valueOf(coord[1]);
            context.write( new Text("C"), new Text(out) );
        }
    }
}
```

- ✓ **DS_Mapper、DS_Reducer：負責處理 initial 之後來的 data**

在處理完 initial memory-load data 後，會 recursively 的處理接下來每一塊 memory-load 的檔案。Mapper 會將上一輪的 data 和新讀到的點分開，讓 Reducer 能分別處理。

```
public void setup(Context context) throws IOException, InterruptedException
{
    URI[] files = context.getCacheFiles();
    URI uri = files[0];
    BufferedReader reader = new BufferedReader(new FileReader((new File(uri.getFragment()).toString())));
    String line;
    int count = 0;
    while(reader.ready()){
        line = reader.readLine().toString();
        context.write(new Text("new"), new Text(line));
    }
}
public void map(Object key, Text value, Context context) throws IOException, InterruptedException
{
    String line = value.toString();
    context.write(new Text("old"), new Text(line));
}
```

Reducer 負責的工作有兩個：首先是先把新來的點根據 mahalanobis distance 加入離它最近且距離小於 threshold 的 cluster。接著再把剩下的點加入 RS，等待我們用 Kmeans 處理。我們會從 RS 中挑選互相距離離最遠的點作為 Kmeans 的 centroid。

```
//deal with old points : R、DS_Cluster、C、COUNT、KMeansCen
else
{
    for(Text val : values)
    {
        String[] input_div = val.toString().split("\\s+");
        //Deals with Initial RS
        if(input_div[0].equals("R"))
        {
            // add garbage for kmeans
            KmeansP_xlist.add(Double.parseDouble(input_div[1]));
            KmeansP_ylist.add(Double.parseDouble(input_div[2]));
            context.write(new Text("R"), new Text( input_div[1] + " " + input_div[2] ) );
        }
        else if( input_div[0].equals("C") )
        {
            context.write(null, val);
        }
        else if(input_div[0].equals("DS_Cluster"))
        {
            int index = Integer.parseInt(input_div[1]);
            //add centroid parameters into cluster info(initialization)
            centroid.add(val.toString());
            output_N[index] = Double.parseDouble(input_div[2]);
            output_sum[0][index] = Double.parseDouble(input_div[3]);
            output_sum[1][index] = Double.parseDouble(input_div[4]);
            output_sumsq[0][index] = Double.parseDouble(input_div[5]);
            output_sumsq[1][index] = Double.parseDouble(input_div[6]);
        }
        else if(input_div[0].equals("COUNT"))
        {
            index_count = Integer.parseInt(input_div[1]);
        }
        else{
            context.write(null, val);
        }
    }
}
```

- ✓ **KmeansMapper、KmeansReducer：處理 Kmeans。**

  每次處理完一塊 memory-load data，我們就用 Kmeans 去處理 RS 中的點。

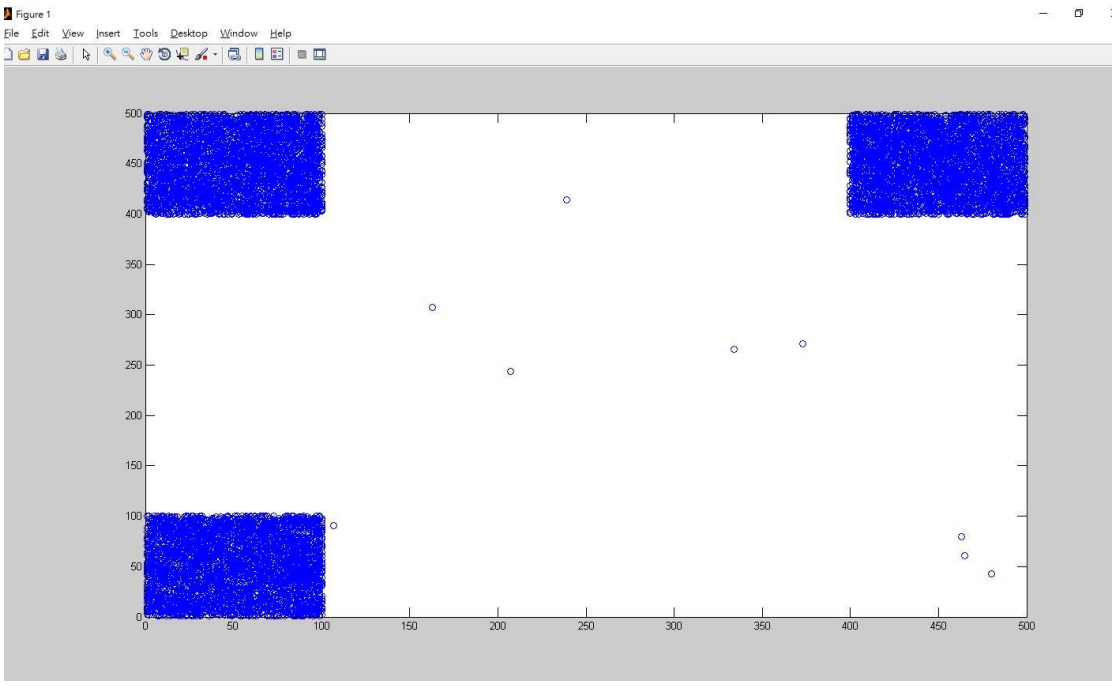
```
// k means
// job.addCacheFile(new URI("BFR/round_"+ Integer.toString(i+1) +  "#kmeans" ));
int iter = 10;
for(int j = 0; j < iter; j++){
    job.cleanupProgress();
    job = Job.getInstance();
    job.setJarByClass(BFR.class);
    job.setMapperClass(KmeansMapper.class);
    job.setReducerClass(KmeansReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    FileInputFormat.addInputPath(job, new Path("output/KMeans_"+ Integer.toString(i) +Integer.toString(j) +"/part-r-00000")) ;
    //FileInputFormat.addInputPath(job, new Path( otherArgs[0] )) ;
    if(j == iter - 1)
        FileOutputFormat.setOutputPath(job, new Path("output/round_" +  Integer.toString(i+1)));
    else
        FileOutputFormat.setOutputPath(job, new Path("output/KMeans_"+ Integer.toString(i) + Integer.toString(j+1)));
    job.waitForCompletion(true);
}
```

```java
public void cleanup(Context context) throws IOException, InterruptedException
{
    for(int i = 0; i < RS_xlist.size(); i++)
    {
        double cost = Double.MAX_VALUE;
        double kmeans_threshold = 100.0;
        int cluster_num = -1;
        for(int j = 0; j < KMCen_xlist.size(); j++)
        {
            double sum = 0.0;
            sum += Math.pow( KMCen_xlist.get(j) - RS_xlist.get(i), 2 );
            sum += Math.pow( KMCen_ylist.get(j) - RS_ylist.get(i), 2 );
            sum = Math.sqrt(sum);
            if(sum < cost && sum < kmeans_threshold)
            {
                cluster_num = j;
                cost = sum;
            }
        }
        if(cluster_num == -1)
        {
            context.write(new Text("R"), new Text(String.valueOf(RS_xlist.get(i)) + " " + String.valueOf(RS_ylist.get(i)) ));
        }
        else
        {
            context.write(new Text("CS"), new Text(String.valueOf(KMCen_index.get(cluster_num)) + " " + String.valueOf(RS_xlist.get(i)) + " " + String.valueOf(RS_ylist.get(i)) ));
        }
    }
    for(int i = 0; i < KMCen_index.size(); i++){
        context.write(new Text("KmeansCen"), new Text(String.valueOf(KMCen_index.get(i)) + " " + String.valueOf(KMCen_xlist.get(i)) + " " + String.valueOf(KMCen_ylist.get(i)) ));
    }
}
```

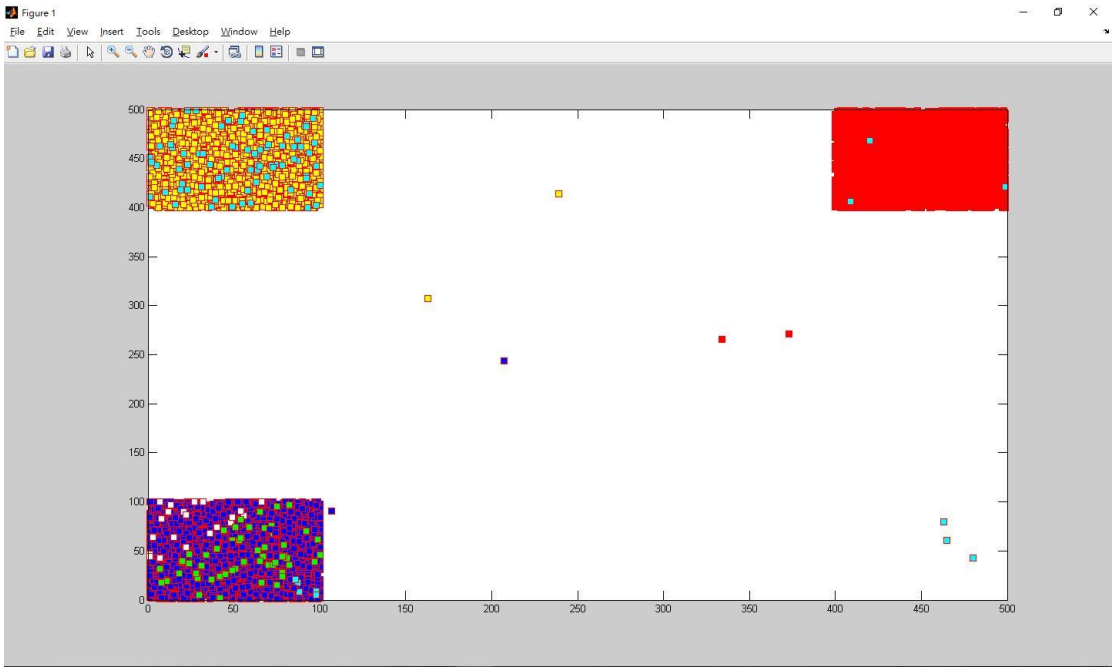
為了展現 BFR 的特性，我們設計的 data 測資會有四筆，每一筆代表一個 memory-load 的資料，而我們的 algorithm 會按照順序將這些測資中的點讀入後進行 clustering。

首先，我們先將第一筆測資讀入 Initial MapReduce 中，並且從這筆測資找尋 centroid 後開始進行 clustering。接下來的測資會讀入 DS MapReduce，並且依照第一次所找出的 centroid 來進行 clustering，若此點與某個 centroid 間的 mahalanobis distance 小於我們所定義的 threshold，便將其納入其 DS 下。而沒被分到的 RS 會進行 Kmeans 來繼續分群，盡量將靠近的 RS 集合成 CS，做完後再讀入下一筆資料，不斷進行 iteration。

我們的測資如圖：



進行我們的 BFR 後結果如下：



其中黃色、紅色、紫色分別代表為一組 DS，因為我們為了方便展示就沒有將其真的丟棄，其餘顏色則為 RS 或 CS。

而結果文字檔部分內容如下：

```
CS  7 60.0 438.0
CS  6 36.0 444.0
CS  8 85.0 449.0
CS  7 49.0 401.0
CS  8 87.0 469.0
C   2 91.0 78.0
C   2 49.0 57.0
C   2 71.0 72.0
C   2 85.0 12.0
C   2 6.0 94.0
C   2 56.0 25.0
C   2 207.0 244.0
C   0 163.0 307.0
C   1 373.0 271.0
C   1 417.0 441.0
C   2 107.0 91.0
C   1 334.0 266.0
C   0 239.0 414.0
DS_Cluster  0 3926.0 200515.0 1763698.0 1.3543047E7 7.95609004E8
DS_Cluster  1 3977.0 1782889.0 1785624.0 8.02517687E8 8.05057076E8
DS_Cluster  2 3918.0 200146.0 198350.0 1.3585724E7 1.3327344E7
COUNT   12
C   2 41.0 94.0
C   2 28.0 83.0
C   2 63.0 6.0
C   2 57.0 100.0
KmeansCen   4 499.0 421.0
KmeansCen   2 98.0 5.0
KmeansCen   1 7.0 100.0
KmeansCen   7 98.0 402.0
KmeansCen   0 74.0 38.0
KmeansCen   6 7.0 463.0
KmeansCen   9 480.0 43.0
```

前面會標示出這個點是屬於哪個類型，C 為被 Clustering 的 DS，CS 為 Compression set，KmeansCen 則為 Kmeans 的 Centroid，這些點的後面皆有他們

那一群所屬的 index，以及那些點的座標。DS_Cluster 為我們將 DS clustering 後所維持的 index、sum、sumsq 以及 N。count 是我們用來計算目前已經有多少個 Kmeans 的 centroid。