

BAB 1

ANALISIS DAN PERANCANGAN

Pada bab ini, penulis akan menjelaskan apa saja yang dilakukan dalam pengembangan *Agglomerative Hierarchical Clustering* untuk Spark. Pengembangan dilakukan untuk mencapai tujuan yaitu mendapatkan pola dari dataset yang diolah. Pola yang ingin didapatkan meliputi perhitungan rata-rata, nilai maksimum, nilai minimum dan nilai standar deviasi dari setiap atribut yang ada pada data. Selain itu, perlu didapatkan juga jumlah anggota pada setiap *cluster* yang dihasilkan dari algoritma *Agglomerative Hierarchical Clustering*.

1.1 Analisis Perangkat Lunak

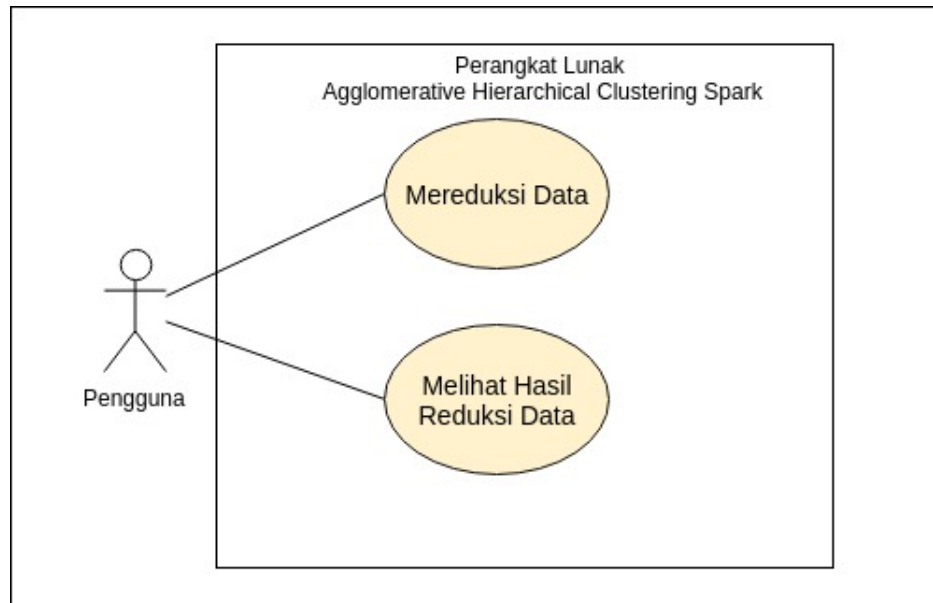
asdasd

1.2 Perancangan Perangkat Lunak

Pada bagian ini, akan dijelaskan perancangan perangkat lunak. Perancangan termasuk diagram *use case*, skenario, diagram kelas, dan rancangan antarmuka.

1.2.1 Diagram Use Case dan Skenario

Diagram *use case* merupakan sebuah pemodelan untuk perilaku dari perangkat lunak yang akan dibuat. Diagram *use case* digunakan untuk mengetahui fungsi apa saja yang ada dalam perangkat lunak. Fungsi-fungsi dari perangkat lunak akan dioperasikan oleh satu pengguna. Cara kerja dan perilaku dari perangkat lunak akan dijelaskan dalam bentuk diagram *use case*. Diagram *use case* dapat dilihat pada Gambar 1.1.



Gambar 1.1: Gambar diagram *use case* perangkat lunak *Agglomerative Hierarchical Clustering*

Berdasarkan gambar diagram *use case* diatas, berikut adalah skenario yang ada:

1. Nama *use case*: Mereduksi data

- Aktor: Pengguna
- Pre-kondisi: data yang akan diolah dimasukan kepada HDFS
- Pra -kondisi: hasil reduksi disimpan pada HDFS
- Deskripsi: Fitur untuk menjalankan program untuk mereduksi data
- Langkah-langkah:
 - (a) Pengguna mengisi JAR *path*, *input path*, dan *output path*
 - (b) Pengguna mengisi jumlah *executor* dan besar *executor memory*
 - (c) Pengguna mengisi jumlah partisi, batas maksimum objek, tipe metode, dan *cut-off distance*
 - (d) Pengguna menekan tombol *submit*
 - (e) Sitem melakukan pengolahan data dengan algoritma *Agglomerative Hierarchical Clustering* pada *cluster* Hadoop
 - (f) Sistem membuka tab baru untuk melihat tahap dan progres program
 - (g) Sistem menyimpan hasil reduksi pada HDFS

2. Nama *use case*: Melihat data

- Aktor: Pengguna
- Pre-kondisi: data yang akan ditampilkan sudah disimpan pada HDFS
- Pra -kondisi: menampilkan data yang ada pada HDFS
- Deskripsi: fitur untuk melihat data hasil reduksi
- Langkah-langkah:
 - (a) Pengguna mengisi *path* dimana data disimpan pada HDFS
 - (b) Sistem menampilkan data-data pada direktori tersebut

- SparkContext: kelas ini merupakan titik masuk untuk layanan-layan dari Apache Spark.

• DataReducer

DataReducer
<pre> + sc : SparkContext + numPar : int + maxObj : int + distanceType : int + cutOffDistance : int + inputPath : int + outputPath : int + DataReducer(sc : SparkContext, nPar : int, maxObject : int, distanceType : int, cutOffDistance : double, inputPath : String, outputPath : String) : + reduceData() : Unit - loadData() : RDD<String> - mapData() : RDD<Node> </pre>

Gambar 1.4: Gambar kelas DataReducer

Kelas DataReducer dirancang untuk memproses data. Proses reduksi secara parallel dilakukan pada kelas ini. Proses pemuatan dan penyimpanan data dilakukan pada kelas ini. Berdasarkan Gambar 1.4, berikut adalah penjelasan dari *methods* pada kelas DataReducer:

1. loadData: *method* untuk memuat data berdasarkan *input path* yang diberikan.
2. mapData: *method* untuk memisahkan atribut dan membungkus attribut dalam objek Node agar lebih mudah diproses. *Method* ini akan mengembalikan RDD bertipe Node.
3. reduceData: *Method* dimana proses reduksi data secara parallel terjadi. Data akan dipecah menjadi beberapa partisi pada method ini. Partisi ini akan diproses secara parallel. Method ini juga bertanggung jawab untuk meyimpan pola hasil reduksi kepada HDFS.

• Dendrogram

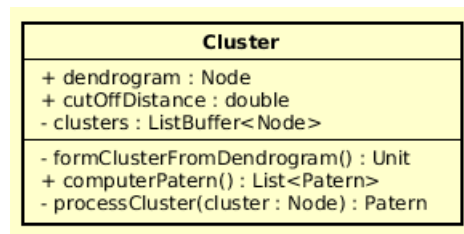
Dendrogram
<pre> - dendrogram : ArrayBuffer<Node> - nodeListCustom : ArrayBuffer<ListBuffer<Node>> - distanceMatrix : ArrayBuffer<ArrayBuffer<Node>> + nodeList : ListBuffer<Node> + distType : int + Dendrogram(nodeList : ListBuffer<Node>, distType : int) : Unit + getDendrogram() : Node + generateDendrogram() : Unit + formClusterBetweenNearestNeighbour() : Unit + recalculateMatrix() : Unit + findMinimumDistance() : Unit + calculateAverageLinkage() : Unit + calculateSingleLinkage() : Unit + calculateAverageLinkage() : Unit </pre>

Gambar 1.5: Gambar kelas Dendrogram

Kelas Dendrogram dirancang untuk memproses data dan membangun dendrogram sesuai algoritma *Agglomerative Hierarchical Clustering*. Berdasarkan Gambar 1.5, berikut adalah penjelasan *methods* pada kelas Dendrogram:

1. getDendrogram: *method* ini mengembalikan dendrogram.
2. generateDendrogram: *Method* untuk membangun dendrogram berdasarkan algoritma *Agglomerative Hierarchical Clustering*.
3. formClusterBetweenNearestNeighbour: *method* untuk menggabungkan *cluster* terdekat.
4. recalculateMatrix: *method* untuk menghitung ulang matriks jarak.
5. findMinimumDistance: *method* untuk mencari jarak minimum antara dua *cluster*.
6. calculateCentroidLinkage: *method* untuk mencari jarak antara centorid dua cluster.
7. calculateSingleLinkage: *method* untuk mencari jarak minimum antara dua cluster.
8. calculateCompleteLinkage: *method* untuk mencari jarak maksimum antara dua cluster.
9. calculateDistance: *method* untuk mencari jarak antara dua buah Node berdasarkan atributnya.

• Cluster

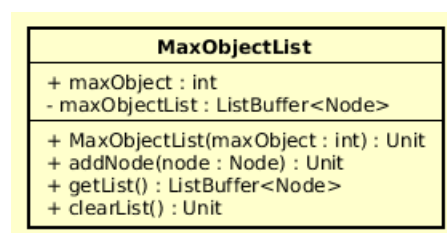


Gambar 1.6: Gambar kelas Cluster

Kelas Cluster dirancang untuk mengolah *cluster* untuk menghasilkan pola dengan memotong *cluster*. Berdasarkan Gambar 1.6, berikut adalah penjelasan *methods* pada kelas Cluster:

1. formClusterFromDendrogram: *method* ini bertugas untuk memotong *dendrogram* menjadi beberapa *cluster*.
2. computePatern: *method* untuk mengolah potongan-potongan *cluster* menjadi pola dengan memanggil *method* *processCluster*.
3. processCluster: *method* untuk memproses *cluster* dan membuat pola berdasarkan atribut-atribut pada *cluster*.

• MaxObjectList

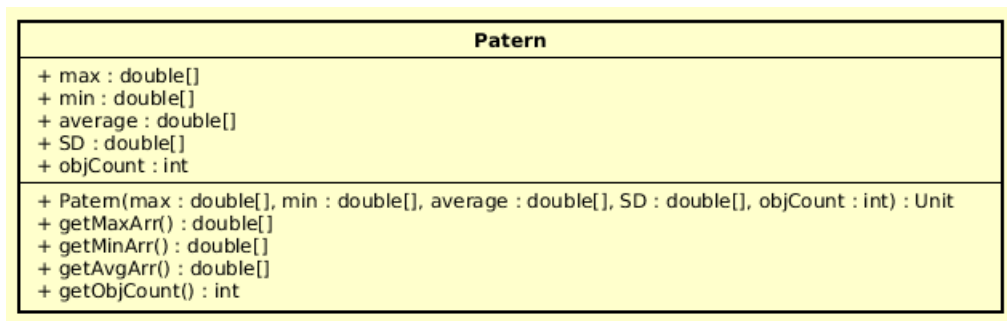


Gambar 1.7: Gambar kelas MaxObjectList

Kelas MaxObjectList dirancang untuk membatasi jumlah objek yang akan diolah pada kelas Dendrogram. Berdasarkan Gambar 1.6, berikut adalah penjelasan *methods* pada kelas MaxObjectList:

1. addNode: method untuk menambahkan Node pada pada *list*
2. getList: method ini mengembalikan *list* berisi objek Node.
3. clearList: method untuk mengosongkan *list*.

• Patern

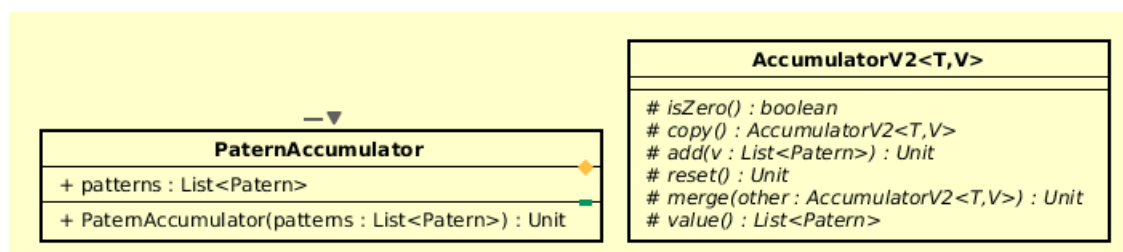


Gambar 1.8: Gambar kelas Patern

Kelas Patern dirancang untuk merepresentasikan pola pada *cluster*. Berdasarkan Gambar 1.6, berikut adalah penjelasan *methods* pada kelas MaxObjectList:

1. getMaxArr: *method* ini mengembalikan *array* berisi nilai maksimum dari setiap atribut.
2. getMinArr: *method* ini mengembalikan *array* berisi nilai minimum dari setiap atribut.
3. getAvgArr: *method* ini mengembalikan *array* berisi nilai rata-rata dari setiap atribut.
4. getSDArr: *method* ini mengembalikan *array* berisi nilai standar deviasi dari setiap atribut.
5. getObjCount: *method* ini mengembalikan jumlah objek.

• PaternAccumulator dan AccumulatorV2

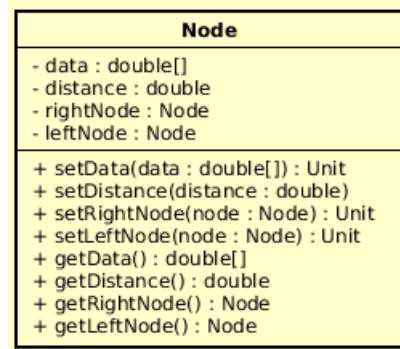


Gambar 1.9: Gambar kelas PaternAccumulator dan AccumulatorV2

Kelas PaternAccumulator dirancang untuk mengumpulkan pola-pola hasil reduksi. Kelas ini merupakan anak dari kelas abstrak *AccumulatorV2* yang setiap *method* harus di-*override* pada kelas anaknya. Berdasarkan Gambar 1.9, berikut adalah penjelasan *methods* pada kelas PaternAccumulator:

1. isZero: *method* untuk mengetahui apakah list *masih* kosong atau tidak.
2. copy: *method* untuk menduplikat objek PaternAccumulator.
3. add: *method* untuk menambahkan *list* berisi pola-pola.
4. merge: *method* untuk menggabungkan dua objek PaternAcumulator menjadi satu.

• Node



Gambar 1.10: Gambar kelas Node

Kelas Node digunakan untuk membentuk pohon yang merepresentasikan *dendrogram*. Selain itu kelas ini digunakan untuk merepresentasikan anggota pada *cluster*. Berdasarkan Gambar 1.10, berikut adalah penjelasan *methods* pada kelas Node:

1. setData: *method* untuk memasukan nilai-nilai atribut.
2. setDistance: *method* untuk megubah nilai jarak.
3. setRightNode: *method* untuk menambahkan anak kanan Node.
4. setLeftNode: *method* untuk menambahkan anak kiri Node.
5. getData: *method* ini mengembalikan nilai-nilai atribut.
6. getDistance: *method* ini mengembalikan jarak.
7. getRightNode: *method* ini mengebalikan anak belah kanan dari Node.
8. getLeftNode: *method* ini mengebalikan anak belah kiri dari Node.

1.2.3 Rancangan Antarmuka

DAFTAR REFERENSI

LAMPIRAN A

KODE PROGRAM

Listing A.1: MyCode.c

```

1 // This does not make algorithmic sense,
2 // but it shows off significant programming characters.
3
4 #include<stdio.h>
5
6 void myFunction( int input, float* output ) {
7     switch ( array[i] ) {
8         case 1: // This is silly code
9             if ( a >= 0 || b <= 3 && c != x )
10                 *output += 0.005 + 20050;
11             char = 'g';
12             b = 2^n + ~right_size - leftSize * MAX_SIZE;
13             c = (--aaa + &daa) / (bbb++ - ccc % 2 );
14             strcpy(a,"hello_$@?");
15         }
16         count = ~mask | 0x00FF00AA;
17     }
18 }
19
20 // Fonts for Displaying Program Code in LATEX
21 // Adrian P. Robson, nepsweb.co.uk
22 // 8 October 2012
23 // http://nepsweb.co.uk/docs/progfonts.pdf

```

Listing A.2: MyCode.java

```

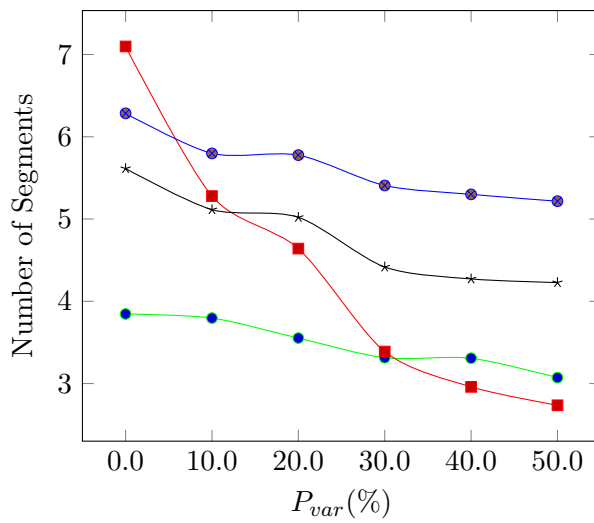
1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.HashSet;
4
5 //class for set of vertices close to furthest edge
6 public class MyFurSet {
7     protected int id; //id of the set
8     protected MyEdge FurthestEdge; //the furthest edge
9     protected HashSet<MyVertex> set; //set of vertices close to furthest edge
10    protected ArrayList<ArrayList<Integer>> ordered; //list of all vertices in the set for each trajectory
11    protected ArrayList<Integer> closeID; //store the ID of all vertices
12    protected ArrayList<Double> closeDist; //store the distance of all vertices
13    protected int totaltrj; //total trajectories in the set
14
15    /*
16     * Constructor
17     * @param id : id of the set
18     * @param totaltrj : total number of trajectories in the set
19     * @param FurthestEdge : the furthest edge
20     */
21    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
22        this.id = id;
23        this.totaltrj = totaltrj;
24        this.FurthestEdge = FurthestEdge;
25        set = new HashSet<MyVertex>();
26        ordered = new ArrayList<ArrayList<Integer>>();
27        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
28        closeID = new ArrayList<Integer>(totaltrj);
29        closeDist = new ArrayList<Double>(totaltrj);
30        for (int i = 0;i <totaltrj;i++) {
31            closeID.add(-1);
32            closeDist.add(Double.MAX_VALUE);
33        }
34    }
35
36 }

```


LAMPIRAN B

HASIL EKSPERIMEN

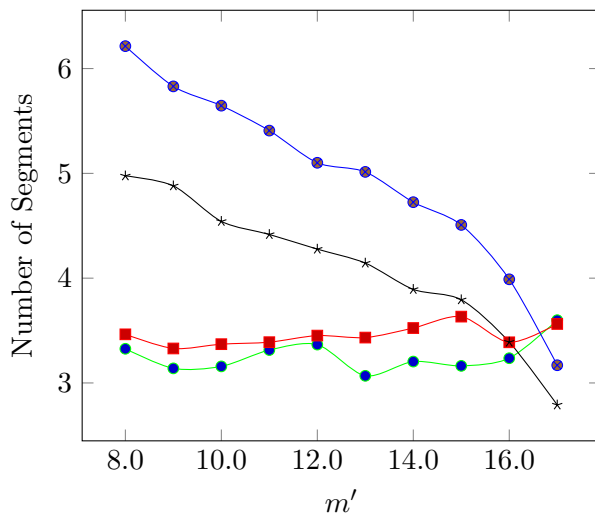
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4