**Laporan Tugas Hadoop**

**IF4031 Pengembangan Aplikasi Terdistribusi**

# **Halaman Judul**

**IMPLEMENTASI TRIANGLE COUNT DENGAN**

**HADOOP MAP-REDUCE**

oleh

13515011

13515057

Reinhard Benyamin L

Erick Wijaya



PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2018

# **Daftar Isi**

[**Halaman Judul**](#_os8s3qz12w82) **1**

[**Daftar Isi**](#_p6qatjidrqhw) **2**

[**Deskripsi Algoritma**](#_bko5sulcg9dk) **3**

[Mapper Preprocessing](#_i829d6sk76v0) 3

[Reducer Preprocessing](#_6zk1j0upp209) 3

[Mapper 1](#_klkso5c6rj93) 4

[Reducer 1](#_l3byijpqhacq) 4

[Mapper 2](#_tvqq3evbh7ir) 4

[Reducer 2](#_pno9qiikbpph) 4

[Mapper Final](#_4h0b6p9l9u6k) 5

[Reducer Final](#_g99jgzd1xnvx) 5

[**Hasil dan Waktu Eksekusi**](#_wyt7ym1qum2f) **6**

[Dataset yang digunakan](#_3ylrsbdck2fn) 6

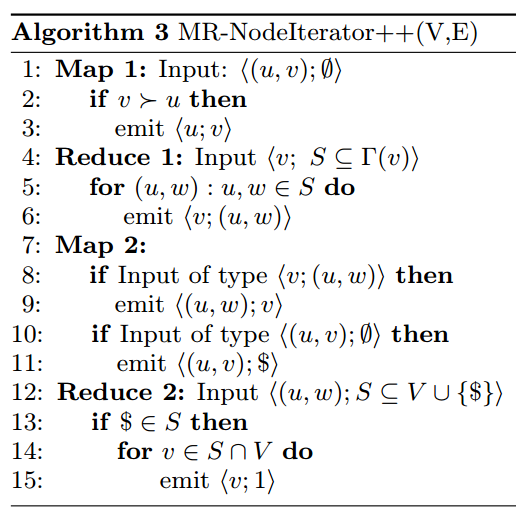
[Hasil eksekusi](#_9743btxsgxyy) 6

[Waktu eksekusi](#_5mwksreotnya) 6

[**Source Code**](#_54z0waiftoy8) **7**

# **Deskripsi Algoritma**

Algoritma yang dibuat kelompok kami menyerupai dengan algoritma pada MR-NodeIterator++, namun kami menambahkan job (Map dan Reduce) tambahan pada awal dan akhir dari algoritma tersebut. Berikut adalah proses algoritmanya.



## **Mapper Preprocessing**

Pada mapper pertama, mapper menerima values dari file input (twitter\_rv.net) untuk melakukan preprocessing. Preprocessing yang dilakukan adalah dengan menambah koma diantara id user dan id follower. Hasil penambahan koma menjadi key-out dari mapper sedangkan value-out yang digunakan adalah nilai Ø. Misalnya apabila ada input berupa <1 8> (<key value>), maka output dari mapper adalah <1,8 Ø> dan <8,1 Ø>. Pada hasil mapper ini bisa jadi akan terbentuk duplikasi, tetapi duplikasi akan ditangani pada reducer preprocessing.

## **Reducer Preprocessing**

Pada tahap reducer preprocessing, setiap duplikasi yang ada akan dieliminasi. Caranya cukup sederhana yaitu dengan melakukan emit nilai key-out dengan nilai key-in dan melakukan emit value-out dengan nilai value-in. Dengan cara tersebut, duplikasi misalnya <1,8 Ø> dan <1,8 Ø> akan otomatis direduksi menjadi <1,8 Ø> karena key-out memiliki nilai yang sama.

## **Mapper 1**

Mapper 1 mengikuti algoritma mapper 1 dari MR-NodeIterator++. Input yang diterima dari mapper 1 memiliki format <u,v Ø>. Pada tahap ini, nilai u dan v akan dibandingkan, apabila v>u maka mapper akan melakukan emit <u v>. Akan tetapi bila kondisi tersebut tidak terpenuhi, emit tidak akan dilakukan.

## **Reducer 1**

Algoritma dari reducer 1 juga mengikuti algoritma dari MR-NodeIterator++. Pada tahap ini akan dibangkitkan seluruh kombinasi value yang memiliki key yang sama. Apabila mapper 1 menghasilkan nilai <1 2>, <1 3>, dan <1 4>, maka reducer akan menerima masukan <1 [2,3,4]> lalu akan membangkitkan seluruh kombinasi dari [2,3,4]. Reducer akan melakukan emit nilai <1 [2,3]>, <1 [2,4]>, dan <1 [3,4]>.

## **Mapper 2**

Mapper 2 dapat menerima dua jenis format input, yaitu format <u,v Ø> atau format <v [u,w]>. Apabila mapper menerima format <u,v Ø>, mapper akan melakukan emit <u,v $>. Karakter spesial $ digunakan sebagai penanda dan diasumsikan tidak ada pada input. Bila format yang diterima adalah <v [u,w]>, mapper akan membalik key dan value sehingga hasil emit adalah <[u,w] v>.

## **Reducer 2**

Pada reducer 2, pertama dilakukan iterasi pada array value untuk mencari nilai $. Apabila ditemukan nilai $, artinya input tersebut memiliki triangle. Setiap key dari input tersebut akan diiterasi kemudian dilakukan emit <v 1> untuk menandakan bahwa key tersebut dihitung sebagai 1 triangle.

## **Mapper Final**

Mapper final memiliki fungsi untuk menyamakan semua key dari input sebelumnya. Format input dari mapper final adalah <v 1>. Mapper kemudian akan melakukan emit <total 1> untuk setiap nilai v. Misalnya apabila mapper menerima <1 1> <2 1> <3 1> <4 1>, mapper akan melakukan emit <total 1> sebanyak 4 kali. Duplikasi dari <total 1> akan digunakan pada reducer final untuk menghitung jumlah triangle.

## **Reducer Final**

Reducer final akan menerima input yang memiliki format <total [1,1,1,...]>. Untuk menghitung jumlah triangle, cukup perlu melakukan iterasi array values dan menjumlahkan seluruh nilai tersebut. Misalnya bila reducer menerima <total [1,1,1]>, reducer akan melakukan emit <Triangle 3>. Nilai 3 yang diperoleh merupakan jumlah triangle akhir yang diperoleh.

# **Hasil dan Waktu Eksekusi**

## **Dataset yang digunakan**

Karena server sister tidak dapat diakses dan data dari twitter\_rv.net terlalu besar, kelompok kamu melakukan eksperimen pada komputer lokal yang sudah diinstall hadoop serta menggunakan dataset dari Enron email network. Dataset yang digunakan dapat diakses melalui pranala <https://snap.stanford.edu/data/email-Enron.html>.

## **Hasil eksekusi**

Program kami menghasilkan jumlah (closed) triangle sebesar 137296.

## **Waktu eksekusi**

Waktu eksekusi program kami sebesar 2 menit 12 detik.

# **Source Code**

|  |
| --- |
| */\*\*  \* Tugas Pengembangan Aplikasi Terdistribusi Hadoop MapReduce  \*   \* 13515011 - Reinhard Benjamin Linardi  \* 13515057 - Erick Wijaya  \*/*  **import** java.io.IOException; **import** java.util.ArrayList; **import** java.util.StringTokenizer;  **import** org.apache.hadoop.conf.Configuration; **import** org.apache.hadoop.fs.Path; **import** org.apache.hadoop.io.IntWritable; **import** org.apache.hadoop.io.Text; **import** org.apache.hadoop.mapreduce.Job; **import** org.apache.hadoop.mapreduce.Mapper; **import** org.apache.hadoop.mapreduce.Reducer; **import** org.apache.hadoop.mapreduce.lib.input.FileInputFormat; **import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat; **import** org.apache.hadoop.util.GenericOptionsParser;  **public** **class** TriangleCount {   **public** **static** **final** Text EMPTY\_VAL = **new** Text(**"X"**);  **public** **static** **final** Text DOLLAR\_VAL = **new** Text(**"$"**);  **public** **static** **final** Text TRIANGLE = **new** Text(**"Triangle"**);  **public** **static** **final** IntWritable ONE = **new** IntWritable(1);   **public** **static** **class** MapperPrep **extends** Mapper<Object, Text, Text, Text> {    **private** Text keyOut = **new** Text();   **public** **void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {  StringTokenizer itr = **new** StringTokenizer(value.toString());  String first = itr.nextToken();  String second = itr.nextToken();   keyOut.set(first + **","** + second);  context.write(keyOut, EMPTY\_VAL);  keyOut.set(second + **","** + first);  context.write(keyOut, EMPTY\_VAL);  }  }   **public** **static** **class** ReducerPrep **extends** Reducer<Text, Text, Text, Text> {   **public** **void** reduce(Text key, Iterable<Text> values, Context context) **throws** IOException, InterruptedException {  context.write(key, EMPTY\_VAL);  }  }   **public** **static** **class** Mapper1 **extends** Mapper<Object, Text, IntWritable, IntWritable> {    **private** IntWritable keyOut = **new** IntWritable();  **private** IntWritable valOut = **new** IntWritable();   **public** **void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {  StringTokenizer itr = **new** StringTokenizer(value.toString());  String realKey = itr.nextToken();   itr = **new** StringTokenizer(realKey, **","**);  **int** u = Integer.parseInt(itr.nextToken());  **int** v = Integer.parseInt(itr.nextToken());   **if**(v > u) {  keyOut.set(u);  valOut.set(v);  context.write(keyOut, valOut);  }  }  }   **public** **static** **class** Reducer1 **extends** Reducer<IntWritable, IntWritable, Text, Text> {    **private** Text keyOut = **new** Text();  **private** Text valOut = **new** Text();   **private** ArrayList<Integer> list = **new** ArrayList<>();    **public** **void** reduce(IntWritable key, Iterable<IntWritable> values, Context context) **throws** IOException, InterruptedException {  list.clear();    **for**(IntWritable value : values) {  list.add(value.get());  }   **for**(**int** idx1 = 0; idx1 < list.size() - 1; idx1++) {  **for**(**int** idx2 = idx1 + 1; idx2 < list.size(); idx2++) {  keyOut.set(String.valueOf(key.get()));  valOut.set(String.valueOf(list.get(idx1)) + **","** + String.valueOf(list.get(idx2)));  context.write(keyOut, valOut);  }  }  }  }   **public** **static** **class** Mapper2 **extends** Mapper<Object, Text, Text, Text> {   **private** Text keyOut = **new** Text();  **private** Text valueOut = **new** Text();    **public** **void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {  StringTokenizer itr = **new** StringTokenizer(value.toString());   String realKey = itr.nextToken();  String realValue = itr.nextToken();   **if** (realValue.equals(EMPTY\_VAL.toString())) {  keyOut.set(realKey);  context.write(keyOut, DOLLAR\_VAL);  } **else** {  keyOut.set(realValue);  valueOut.set(realKey);  context.write(keyOut, valueOut);  }  }  }   **public** **static** **class** Reducer2 **extends** Reducer<Text, Text, IntWritable, IntWritable> {    **private** IntWritable keyOut = **new** IntWritable();   **public** **void** reduce(Text key, Iterable<Text> values, Context context)   **throws** IOException, InterruptedException {  **boolean** hasDollar = **false**;    **for**(Text val : values) {  **if** (val.toString().equals(DOLLAR\_VAL.toString())) {  hasDollar = **true**;  **break**;  }  }   **if**(hasDollar) {  **for** (Text val : values) {  **if** (!val.toString().equals(DOLLAR\_VAL.toString())) {  keyOut.set(Integer.parseInt(val.toString()));  context.write(keyOut, ONE);  }  }   }  }  }   **public** **static** **class** MapperFinal **extends** Mapper<Object, Text, Text, IntWritable> {    **public** **void** map(Object key, Text value, Context context) **throws** IOException, InterruptedException {  context.write(TRIANGLE, ONE);  }  }   **public** **static** **class** ReducerFinal **extends** Reducer<Text, IntWritable, Text, IntWritable> {   **private** **int** sum = 0;  **private** IntWritable valOut = **new** IntWritable();   **public** **void** reduce(Text key, Iterable<IntWritable> values, Context context) **throws** IOException, InterruptedException {  values.forEach(val -> {  sum += val.get();  });   valOut.set(sum);  context.write(TRIANGLE, valOut);  }  }   **public** **static** **void** main(String[] args) **throws** Exception {  Configuration conf = **new** Configuration();  String[] otherArgs = **new** GenericOptionsParser(conf, args).getRemainingArgs();   **if**(otherArgs.length < 2) {  System.err.println(**"Usage: TriangleCount <in> [<in>...] <out>"**);  System.exit(2);  }   Job jobPrep = Job.getInstance(conf, **"Master of Java - Preprocess"**);  jobPrep.setJarByClass(TriangleCount.class);  jobPrep.setMapperClass(MapperPrep.class);  jobPrep.setReducerClass(ReducerPrep.class);  jobPrep.setMapOutputKeyClass(Text.class);  jobPrep.setMapOutputValueClass(Text.class);  jobPrep.setOutputKeyClass(Text.class);  jobPrep.setOutputValueClass(Text.class);    **for**(**int** i = 0; i < otherArgs.length - 1; i++) {  FileInputFormat.addInputPath(jobPrep, **new** Path(otherArgs[i]));  }    FileOutputFormat.setOutputPath(jobPrep, **new** Path(otherArgs[otherArgs.length - 1], **"prep"**));  jobPrep.waitForCompletion(**true**);  *//System.exit(jobPrep.waitForCompletion(true) ? 0 : 1);*   Job job1 = Job.getInstance(conf, **"Master of Java - MapReduce 1"**);  job1.setJarByClass(TriangleCount.class);  job1.setMapperClass(Mapper1.class);  job1.setReducerClass(Reducer1.class);  job1.setMapOutputKeyClass(IntWritable.class);  job1.setMapOutputValueClass(IntWritable.class);  job1.setOutputKeyClass(Text.class);  job1.setOutputValueClass(Text.class);  FileInputFormat.addInputPath(job1, **new** Path(otherArgs[otherArgs.length - 1], **"prep"**));  FileOutputFormat.setOutputPath(job1, **new** Path(otherArgs[otherArgs.length - 1], **"mapred1"**));  job1.waitForCompletion(**true**);  *//System.exit(job1.waitForCompletion(true) ? 0 : 1);*   Job job2 = Job.getInstance(conf, **"Master of Java - MapReduce 2"**);  job2.setJarByClass(TriangleCount.class);  job2.setMapperClass(Mapper2.class);  job2.setReducerClass(Reducer2.class);  job2.setMapOutputKeyClass(Text.class);  job2.setMapOutputValueClass(Text.class);  job2.setOutputKeyClass(IntWritable.class);  job2.setOutputValueClass(IntWritable.class);  FileInputFormat.addInputPath(job2, **new** Path(otherArgs[otherArgs.length - 1], **"prep"**));  FileInputFormat.addInputPath(job2, **new** Path(otherArgs[otherArgs.length - 1], **"mapred1"**));  FileOutputFormat.setOutputPath(job2, **new** Path(otherArgs[otherArgs.length - 1], **"mapred2"**));  job2.waitForCompletion(**true**);  *//System.exit(job2.waitForCompletion(true) ? 0 : 1);*   Job jobFinal = Job.getInstance(conf, **"Master of Java - Final TriangleCount"**);  jobFinal.setJarByClass(TriangleCount.class);  jobFinal.setMapperClass(MapperFinal.class);  jobFinal.setReducerClass(ReducerFinal.class);  jobFinal.setMapOutputKeyClass(Text.class);  jobFinal.setMapOutputValueClass(IntWritable.class);  jobFinal.setOutputKeyClass(Text.class);  jobFinal.setOutputValueClass(IntWritable.class);  FileInputFormat.addInputPath(jobFinal, **new** Path(otherArgs[otherArgs.length - 1], **"mapred2"**));  FileOutputFormat.setOutputPath(jobFinal, **new** Path(otherArgs[otherArgs.length - 1], **"final"**));  System.exit(jobFinal.waitForCompletion(**true**) ? 0 : 1);  } } |