Implementasi BFS dalam Model Pemrograman Map Reduce

Ibrohim Kholilul Islam - 13513090[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]ibrohim@s.itb.ac.id

***Abstract*— MapReduce *framework* pada awalnya merupakan kerangka yang di kembangkan oleh Google, namun kemudian berkembang menjadi *framework* yang umum digunakan dalam analisis dengan skala besar. BFS dengan sebuah graf dapat diimplementasi dalam *famework* ini terutama dalam pemrosesan graf yang sangat besar.**

***Keywords*—MapReduce, Hadoop, BFS.**

I. PENGANTAR

Dalam dunia dengan data yang sangat besar, dengan orde 2^{40} tera byte hingga 2^{50} peta byte, membutuhkan sebuah model pemrograman yang handal dan mudah digunakan, sebuah paradigma yang dapat memproses data yang sangat banyak.

MapReduce *framework* pada awalnya merupakan kerangka yang di kembangkan oleh Google, namun kemudian berkembang menjadi *framework* yang umum digunakan dalam analisis dengan skala besar. Model ini kemudian menginisiasi perangkat lunak *open source*, Hadoop yang kemudian banyak digunakan oleh perusahaan-perusahaan besar seperti Yahoo!, Facebook, Adobe dan IBM.

Sebelum tahun 1980 kesalahan perangkat keras merupakan faktor utama yang mempengaruhi sistem yang handal. Sampai pada akhirnya teknologi perkembangan perangkat keras seperti IC (*Integrated Circuits*) membuat kemajuan yang sangat pesat. Teknologi tersebut mengurangi radiasi panas dan faktor-faktor lain yang mempengaruhi kinerja elektonik. Sehingga pada dekade ini kesalahan perangkat keras tidak lagi menjadi faktor yang cukup berpengaruh.

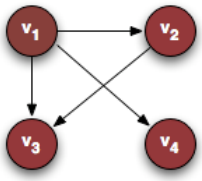
Pada sistem yang cukup besar seperti Facebook, atau Google, keadaan down-time seperti pada saat upgrade sistem, kesalahan pendinginan, ataupun matinya aliran listrik dapat ditanggulangi dengan cara sistim terdistribusi. Sehingga, jika pada suatu saat sebuah perangkat mengalami kesalahan, terdapat perangkat lain yang dapat menggantikan fungsi perangkat tersebut untuk menyelesaikan tugasnya.

II. BFS

A. GRAF

DEFINISI. *Sebuah graf G adalah pair (V,E) dari himpunan simpul dan himpunan sisi. V adalah himpunan simpul, yang ditulis {v_1, v_2, ..., v_n} dan E adalah himpunan terurut pasangan simpul yang disebut sisi.*

Dengan definisi tersebut, berikut adalah salah satu contoh graf:



Gambar 1 Ilustrasi Graf (sumber: http://20bits.com/article/graph-theory-part-i-introduction)

Graf dengan tipe seperti ini adalah graf berarah karena beberapa simpul hanya mengarah dari suatu simpul ke simpul lain. Dari definisi awal V = {v1, v2, v3, v4} dan E = {(v1, v2), (v1, v3), (v1, v4), (v2, v3)}.

B.TRAVERSAL

Breadth-first search (BFS) adalah salah satu cara untuk mendapatkan semua simpul yang dapat dicapai dari suatu simpul sumber. Dalam BFS, setiap simpul memiliki *state* yakni:

1. color[u] = *White* – keadaan “belum dikunjungi”,
2. color [u] = *Gray* – keadaan “dikunjungi namun belum dieksplorasi penuh”,
3. color [u] = *Black* - keadaan “telah dieksplorasi".

Berikut adalah algoritma umum graf deengan BFS:

```
BFS(V, E, s)
1.    for each u in V − {s}
2.       do color[u] ← WHITE
3.          d[u] ← infinity
4.          π[u] ← NIL
5.    color[s] ← GRAY
6.    d[s] ← 0
7.    π[s] ← NIL
8.    Q ← {}
9.    ENQUEUE(Q, s)
10    while Q is non-empty
11.      do u ← DEQUEUE(Q)
12.         for each v adjacent to u
13.               do if color[v] ← WHITE
14.                     then  color[v] ← GRAY
15.                           d[v] ← d[u] + 1
16.                           π[v] ← u
17.                           ENQUEUE(Q, v)
18.         DEQUEUE(Q)
19.      color[u] ← BLACK
```

III. MAP REDUCE

3.1 PARADIGMA FUNGSIONAL

DEFINISI. *Fungsi merupakan relasi biner yang memetakan dari daerah asal (domain) ke daerah hasil (codomain). Jika f merupakan fungsi yang memetakan dari himpunan A ke B maka dapat dituliskan sebagai*

$$f : A \rightarrow B$$

yang artinya f memetakan dari daerah asal A ke daerah hasil B.

Dari sebuah bahasan dalam dunia matematika inilah yang kemudian berkembang menjadi sebuah paradigma pemrograman, yakni paradigma fungsional.

Umumnya sebuah komputer didesain untuk melakukan langkah-langkah secara berurutan (control flow) berdasarkan model von Neumann. Sehingga perhitungan D = B^2-4*A*C dengan C=1, A=1, dan B=-2 akan dilakukan sebagai berikut:

```
1  C ← 1
2  A ← 1
3  B ← -2
4  T1 ← A*C
5  T2 ← 4*T1
6  T3 ← B^3
7  D ← T3-T2
```

Sedangkan pada model dataflow, proses perhitungan akan dilakukan berdasarkan ketersediaan data untuk diproses, alur kerja akan dilakukan sebagai berikut:

```
1  C ← 1;        A ← 1;        B ← -2
2  T1 ← A*C;     T3 ← B^3
3  T2 ← 4*T1
4  D ← T3-T2
```

Sehingga paradigma ini secara alami merupakan paradigma yang sangat cocok untuk pemrosesan secara paralel. Dari dasar itulah model pemrograman MapReduce dikembangkan.

Pada bahasa LISP, fungsi map memiliki parameter berupa sebuah fungsi dan himpunan nilai. Kemudian fungsi parameter tersebut dipakai untuk memroses himpunan tersebut. Contoh:

```
(map 'length '(() (a) (ab) (abc)))
```

Yakni memroses semua anggota himpunan dengan fungsi length, sehingga hasil dari map adalah:

```
(0 1 2 3)
```

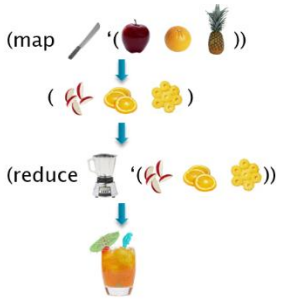
Fungsi reduce adalah fungsi yang menerima fungsi biner dan himpunan nilai. Fungsi ini akan menggabungkan semua nilai menggunakan fungsi biner tersebut. Misalkan dengan menggunakan fungsi jumlah untuk himpunan (0 1 2 3) :

```
(reduce #'+ '(0 1 2 3))
```

Maka hasilnya adalah:

```
6
```

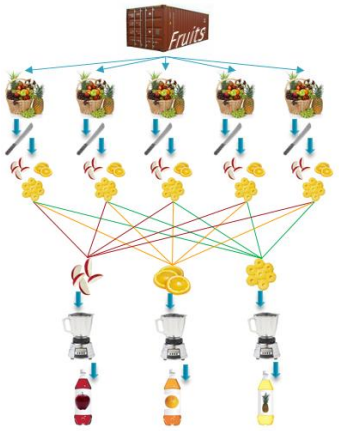
Map reduce pada awalnya merupakan model yang berawal dari paradigma fungsional itu yang secara alamiah merupakan paradigma yang sangat cocok dengan pemrosesan paralel. MapReduce pada dasarnya merupakan proses fungsi map dan reduce *seperti yang telah dibahas sebelumnya. Sebagai contoh, pada pembuatan jus.*



Gambar 2Ilustrasi MapReduce (sumber: http://salsahpc.indiana.edu)

Proses map dilakukan untuk memroses infromasi masukan menjadi sebuah informasi berupa tuple <kunci, nilai'> berdasarkan masukan. Keluaran dari proses map ini kemudian dilakukan proses pengurutan. Kemudian dilakukan proses reduce sesuai dengan kunci menjadi sebuah nilai.

Pada MapReduce, proses pembuatan jus ini dapat dibuat menjadi dua buah proses. Proses pertama yakni proses map yang dilakukan secara paralel. Setelah proses map tersebut dilakukan, untuk sementara data keluaran akan diseimpan sampai akhirnya dilakukan proses reduce. Seperti yang dilakukan pada gambar 2.



Gambar 3 Ilustrasi MapReduce (sumber: http://salsahpc.indiana.edu)

3.2 FRAMEWORK MAPREDUCE

MapReduce adalah framework yang digunakan untuk pemrosesan paralel. Programmer diberikan API sehingga tidak berurusan kembali dengan hal-hal mengenai

paralelisasi, remote execution, load balancing, ataupun fault tolerance.

Sebagai contoh penggunaan MapReduce, misalkan pada penghitungan kada pada sekumpulan dokumen yang sangat besar. Map di definisikan sebagai fungsi yang membaca dokumen dan mengembalikan nilai mengembalikan tuple <kata, 1>. Data intermediet ini di diurutkan oleh MapReduce untuk kemudian dilakukan reduce. Kemudian map reduce menjumlahkan setiap value dari tuple yang dihasilkan map. Sehingga berikut adalah kode yang dihasilkan:

```
map(String key, String value):
// key: document name,  value: document contents
for each word w in value:
  EmitIntermediate(w, "1");

reduce(String key, Iterator values):
// key: a word;  values: a list of counts
int result = 0;
for each v in values:
  result += ParseInt(v);
Emit(AsString(result));
```

III. BFS PADA MAPREDUCE

Dengan mendefinisikan simpul sebagai

```
ID    EDGES|DISTANCE_FROM_SOURCE|COLOR|
```

Dengan EDGES sebagai lis simpul yang terhubung dengan simpul ini. Dan DISTANCE_FROM_SOURCE di diset dengan Integer.MAX_VALUE sebagai penanda belum diketahui. Misalkan pada suatu graf BFS dimulai dari simpul #1, sehingga graf tersebut jika direpresentasikan dalam string menjadi:

```
1       2,5|0|GRAY|
2       1,3,4,5|Integer.MAX_VALUE|WHITE|
3       2,4|Integer.MAX_VALUE|WHITE|
4       2,3,5|Integer.MAX_VALUE|WHITE|
5       1,2,4|Integer.MAX_VALUE|WHITE|
```

Dengan menggunakan BFS, iterasi pertama yang akan muncul adalah:

```
1       2,5,|0|BLACK
2       1,3,4,5,|1|GRAY
3       2,4,|Integer.MAX_VALUE|WHITE
4       2,3,5,|Integer.MAX_VALUE|WHITE
5       1,2,4,|1|GRAY
```

Selanjutnya,

```
1       2,5,|0|BLACK
2       1,3,4,5,|1|BLACK
3       2,4,|2|GRAY
4       2,3,5,|2|GRAY
5       1,2,4,|1|BLACK
```

Kemudian pada akhirnya:

```
1       2,5,|0|BLACK
2       1,3,4,5,|1|BLACK
3       2,4,|2|BLACK
4       2,3,5,|2|BLACK
5       1,2,4,|1|BLACK
```

Dalam Java fungsi Map didefinisikan sebagai berikut:

```
1.  map(LongWritable key, Text value):
2.        Node node = Node(value.toString());
3.
4.        if (node.getColor() == Node.Color.GRAY)
5.          for (int v : node.getEdges())
6.            Node vnode = Node(v);
7.            vnode.setDistance(node.getDistance() + 1);
8.            vnode.setColor(Node.Color.GRAY);
9.            EmitIntermediate(vnode.getId(), vnode.getL
    ine());
10.         node.setColor(Node.Color.BLACK);
11.
12.       EmitIntermediate(new IntWritable(node.getId())
    , node.getLine())
13.
```

Kemudin fungsi Reduce didefinisikan sebagai berikut:

```
1.  reduce(IntWritable key, Iterator<Text> values):
2.    List<Integer> edges = null;
3.    distance = Integer.MAX_VALUE;
4.    Node.Color color = Node.Color.WHITE;
5.
6.    while (values.hasNext())
7.      value = values.next();
8.      Node u =  Node(key.get()+value.toString());
9.      if (u.getEdges().size() > 0)
10.       edges = u.getEdges();
11.     if (u.getDistance() < distance)
12.       distance = u.getDistance();
13.     if (u.getColor().ordinal() > color.ordinal())
14.       color = u.getColor();
15.   Node n = Node(key.get());
16.   n.setDistance(distance);
17.   n.setEdges(edges);
18.   n.setColor(color);
19.   Emit(key, n.getLine());
```

Dalam Framework Hadoop, implementasi skema tersebut adalah sebagai berikut:

```
1.  package org.apache.hadoop.examples;
2.
3.  import java.io.IOException;
4.  import java.util.Iterator;
5.  import java.util.List;
6.
7.  import org.apache.commons.logging.Log;
8.  import org.apache.commons.logging.LogFactory;
9.  import org.apache.hadoop.conf.Configuration;
10. import org.apache.hadoop.conf.Configured;
11. import org.apache.hadoop.fs.Path;
12. import org.apache.hadoop.io.IntWritable;
13. import org.apache.hadoop.io.LongWritable;
14. import org.apache.hadoop.io.Text;
15. import org.apache.hadoop.mapred.*;
16. import org.apache.hadoop.util.Tool;
17. import org.apache.hadoop.util.ToolRunner;
18.
19. public class GraphSearch extends Configured implements Tool {
20.
21.   public static final Log LOG = LogFactory.getLog("org.apache.hadoop.examples.GraphSearch");
22.     public static class MapClass extends MapReduceBase implements
23.       Mapper<LongWritable, Text, IntWritable, Text> {
24.
25.     public void map(LongWritable key, Text value, OutputCollector<IntWritable, Text> output,
26.         Reporter reporter) throws IOException {
27.
28.       Node node = new Node(value.toString());
29.       if (node.getColor() == Node.Color.GRAY) {
30.         for (int v : node.getEdges()) {
31.           Node vnode = new Node(v);
32.           vnode.setDistance(node.getDistance() + 1);
33.           vnode.setColor(Node.Color.GRAY);
34.           output.collect(new IntWritable(vnode.getId()), vnode.getLine());
35.         }
36.         node.setColor(Node.Color.BLACK);
37.       }
38.       output.collect(new IntWritable(node.getId()), node.getLine())
39.
40.     }
41.   }
42.
43. public static class Reduce extends MapReduceBase implements
44.       Reducer<IntWritable, Text, IntWritable, Text> {
45.
46.     public void reduce(IntWritable key, Iterator<Text> values,
47.         OutputCollector<IntWritable, Text> output, Reporter reporter) throws IOException {
48.
49.       List<Integer> edges = null;
50.       int distance = Integer.MAX_VALUE;
51.       Node.Color color = Node.Color.WHITE;
52.
53.       while (values.hasNext()) {
54.         Text value = values.next();
55.
56.         Node u = new Node(key.get() + "\t" + value.toString());
57.
58.         if (u.getEdges().size() > 0) {
59.           edges = u.getEdges();
60.         }
61.
62.         if (u.getDistance() < distance) {
63.           distance = u.getDistance();
64.         }
```

```
65.
66.           if (u.getColor().ordinal() > color.ordinal()) {
67.             color = u.getColor();
68.           }
69.         }
70.         Node n = new Node(key.get());
71.         n.setDistance(distance);
72.         n.setEdges(edges);
73.         n.setColor(color);
74.         output.collect(key, new Text(n.getLine()));
75.       }
76.     }
77.
78.     static int printUsage() {
79.       System.out.println("graphsearch [-m <num mappers>] [-r <num reducers>]");
80.       ToolRunner.printGenericCommandUsage(System.out);
81.       return -1;
82.     }
83.
84.     private JobConf getJobConf(String[] args) {
85.       JobConf conf = new JobConf(getConf(), GraphSearch.class);
86.       conf.setJobName("graphsearch");
87.       conf.setOutputKeyClass(IntWritable.class);
88.       conf.setOutputValueClass(Text.class);
89.       conf.setMapperClass(MapClass.class);
90.       conf.setReducerClass(Reduce.class);
91.       for (int i = 0; i < args.length; ++i) {
92.         if ("-m".equals(args[i])) {
93.           conf.setNumMapTasks(Integer.parseInt(args[++i]));
94.         } else if ("-r".equals(args[i])) {
95.           conf.setNumReduceTasks(Integer.parseInt(args[++i]));
96.         }
97.       }
98.       return conf;
99.     }
100.    public int run(String[] args) throws Exception {
101.      int iterationCount = 0;
102.      while (keepGoing(iterationCount)) {
103.        String input;
104.        if (iterationCount == 0)
105.          input = "input-graph";
106.        else
107.          input = "output-graph-" + iterationCount;
108.        String output = "output-graph-" + (iterationCount + 1);
109.        JobConf conf = getJobConf(args);
110.        FileInputFormat.setInputPaths(conf, new Path(input));
111.        FileOutputFormat.setOutputPath(conf, new Path(output));
112.        RunningJob job = JobClient.runJob(conf);
113.        iterationCount++;
114.      }
115.      return 0;
116.    }
117.    private boolean keepGoing(int iterationCount) {
118.      if(iterationCount >= 4) {
119.        return false;
120.      }
121.      return true;
122.    }
123.    public static void main(String[] args) throws Exception {
124.      int res = ToolRunner.run(new Configuration(), new GraphSearch(), args);
125.      System.exit(res);
126.    }
127.
128. }
```

Sumber: http://www.johnandcailin.com/blog/cailin/breadth-first-graph-search-using-iterative-map-reduce-algorithm

REFERENCES

[1] Ibrohim Kholilul Islam, "MapReduce – Aplikasi Fungsi dalam Pemrosesan Paralel," 2014.

[2] Jeffrey Dean and Sanjay Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters*. Google, Inc. 2004. (diakses pada 7 Desember 2014) http://research.google.com/archive/mapreduce-osdi04.pdf.

[3] Kenneth P. Birman, *Reliable Distributed System: Technologies, Web Services, and Applications*. Cornell University, Ithaca, NY: Departement of Computer Science, 2010, pp. 237-240.

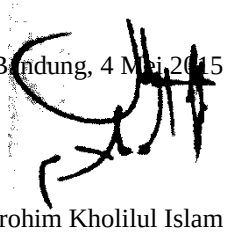
[4] Peter Henderson, *Functional Programming: Application and Implementation*. London: Prentice-Hall 1980, pp.242-246

[5] https://www.cs.rutgers.edu/~pxk/417/notes/content/mapreduce.html

[6] http://www.johnandcailin.com/blog/cailin/breadth-first-graph-search-using-iterative-map-reduce-algorithm

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.



Bandung, 4 Mei 2015

Ibrohim Kholilul Islam
13513090