Hands-on MapReduce

Daya Adianto

(credit : Samuel Louvan, Remmy Augusta Menzata Zen)

Tujuan: Memahami konsep dasar implementasi program sederhana menggunakan paradigma MapReduce dalam bahasa Java dan Python

Pada tutorial sebelumnya kita sudah menjalankan contoh program Word Count yang ada pada Apache Hadoop. Pada tutorial ini kita akan membuat sendiri program Word Count dengan menggunakan MapReduce pada Java dan/atau Python.

Bagi Anda yang biasa *programming* dengan bahasa Java, silakan ikuti latihan ini dengan mengerjakan contoh MapReduce Java. Apabila Anda belum pernah *programming* atau biasa *programming* dengan bahasa selain Java, silakan coba kerjakan contoh MapReduce Python.

Kode MapReduce Java

- Struktur Program MapReduce dalam Java terdiri dari 3 bagian yaitu **class Mapper, class Reducer, dan method main**. Kerangka kodenya kurang lebih sebagai berikut:

Catatan: Apabila Anda menggunakan IDE (Eclipse, IntelliJ) di komputer pribadi, silakan unduh terlebih dahulu library Java Apache Hadoop 2.7.3 di https://mvnrepository.com/artifact/org.apache.hadoop dan tambahkan berkas-berkas JAR-nya ke classpath supaya IDE tidak menampilkan pesan-pesan compile-error.

```
import java.io.IOException;
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;
```

```
public class WordCount {
     public static class MyMapper {

     }
     public static class MyReducer {

     }

     public static void main(String[] args) {
     }
}
```

Membuat Mapper

- Untuk membuat kelas MyMapper menjadi Mapper pada proses MapReduce maka kelas tersebut perlu mengextends kelas Mapper. Ingat bahwa proses Map menerima dan mengeluarkan output. Kelas Mapper menerima 4 argumen Generics yaitu Mapper.
- Pada proses WordCount kita menerima masukan text dan mengubah masukan menjadi <kata, 1>
- Maka kelas MyMapper harus diubah menjadi:
  `public static class MyMapper extends Mapper<Object, Text, Text, IntWriteable>`
  `Text` merupakan tipe data String pada Hadoop, IntWriteable adalah tipe data int pada Hadoop
- Selanjutnya kita perlu membuat method map yang menerima parameter K1, V1, dan Context. Context merupakan tempat menyimpan hasil. Buat method map di dalam class MyMapper sebagai berikut

```
public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
     StringTokenizer itr = new
     StringTokenizer(value.toString());
     IntWritable one = new IntWritable(1);
     Text word = new Text();
     while (itr.hasMoreTokens()) {
          word.set(itr.nextToken());
          context.write(word, one);
     }
}
```

Method tersebut membaca input dan memisahkan berdasarkan whitespace dan memasukkannya ke konteks menjadi <kata, 1>

Membuat Reducer

- Ingat bahwa reducer menerima <K1, List<V1>> dan mengubahnya menjadi <K1, V2>.
- Kelas Reducer menerima 4 argumen Generics yaitu Reducer<K1,V1,K2,V2>. Pada kasus wordcount, key merupakan string dan value merupakan angka, reducer akan menjumlahkan list untuk setiap key.

- Maka kelas MyReducer harus diubah menjadi:

```
public static class MyReducer extends Reducer<Text, IntWriteable, Text, IntWritable>
```

- Selanjutnya buat method reduce yang menerima K1, List<V1> dan Context. Buat method reduce di dalam class MyReducer sebagai berikut:

```
public void reduce(Text key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
     int sum = 0;
     IntWritable result = new IntWritable();
     for (IntWritable val : values) {
          sum += val.get();
     }
     result.set(sum);
     context.write(key, result);
}
```

Lengkapi method **main** sebagai berikut :

```
public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();
      Job job = Job.getInstance(conf, "word count");
      job.setJarByClass(WordCount.class);
      job.setMapperClass(MyMapper.class);
      job.setCombinerClass(MyReducer.class);
      job.setReducerClass(MyReducer.class);
      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(IntWritable.class);
      FileInputFormat.addInputPath(job, new Path(args[0]));
```

```
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Compile and Running

- Siapkan dulu file input, file input harus disimpan di HDFS. Kita akan menggunakan file test/test.txt yang Anda masukkan pada hands on sebelumnya. Untuk mengecek apakah file tersebut masih ada gunakan perintah

  `hdfs dfs -cat test/test.txt`

- Jika belum ada silakan buat file Java seperti pada hands-on sebelumnya.
  - Catatan: Kompilasi berkas Java harus dilakukan di server hands-on!
- Compile dengan cara:
  `hadoop com.sun.tools.javac.Main WordCount.java`
- Selanjutnya semua class-class tersebut akan dibuat ke dalam jar dengan cara:
  `jar cf wc.jar WordCount*.class`
- File input ada di test/test.txt misalkan kita ingin menaruh direktori output di outputwc maka jalan perintah berikut untuk menjalankan:
  `hadoop jar wc.jar WordCount test/test.txt outputwc`
- Untuk melihat hasil gunakan perintah
  `hdfs dfs -cat outputwc/part-r-00000`

LATIHAN

1. Copy file **WordCount.java** menjadi **WordCountFilter1.java** dengan menggunakan perintah cp **WordCount.java WordCountFilter1.java**. Lalu ubah class tersebut agar kata yang muncul pada output hanya kata yang jumlahnya

lebih dari 1. Hint: Jangan lupa memperbaiki nama variabel dan class yang di copy.

2. Copy file **WordCount.java** menjadi **WordCountFilter2.java.** Lalu ubah class tersebut agar kata yang diproses hanya kata yang panjang katanya lebih dari 3. Hint: Untuk mendapatkan panjang suatu text dapat dilakukan dengan mengubah text tersebut menjadi string dan menghitung panjangnya. (`text.toString().length()`)

3. Salah satu cara lain untuk mengolah teks adalah dengan menghitung **Bigram**. Contoh program WordCount adalah menghitung Unigram, karena kita hanya menghitung 1 kata. Bigram menghitung kemunculan setiap 2 kata yang berdampingan. Andaikan terdapat teks berikut: "Ibu Budi pergi ke pasar"

   Unigram / WordCount akan menghasilkan:
   Ibu, 1
   Budi, 1
   pergi, 1
   ke, 1
   pasar, 1

   Sedangkan Bigram akan menghasilkan:
   Ibu Budi, 1
   Budi pergi, 1
   pergi ke, 1
   ke pasar, 1

   Buatlah class **BigramCount** yang melakukan hal tersebut.

Kode MapReduce Python

Pada direktori home Anda, buatlah dua file **mapper.py** dan **reducer.py**

mapper.py

Sama seperti pada kode Java yang sudah kita buat sebelumnya mapper.py akan berisi implementasi dari fungsi mapper. Kodenya sebagai berikut :

```
#!/usr/bin/python
import sys

for line in sys.stdin:
    line = line.strip()
    words = line.split()

    for word in words:
        print '%s\t%s' % (word, 1)
```

reducer.py

Kode dari fungsi reducer adalah sebagai berikut:

```
#!/usr/bin/python

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)

    try:
        count = int(count)
    except ValueError:
        continue

    if current_word == word:
        current_count += count
    else:
        if current_word:
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word

if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

Eksekusi Program

1. Anda bisa menguji program di lokal sistem Anda sebelum menggunakan Hadoop. Contoh tampilan (sesuaikan dengan filepath Anda):

```
bd@master:~/samuel/python$ echo "Samuel Louvan Samuel Louvan" |  /home/bd/samuel/python/mapper.py
Samuel  1
Louvan  1
Samuel  1
Louvan  1
bd@master:~/samuel/python$ echo "Samuel Louvan Samuel Louvan" |  /home/bd/samuel/python/mapper.py | sort -k1,1 | /home/bd/samu
el/python/reducer.py
Louvan  2
Samuel  2
```

2. Untuk mengeksekusi program menggunakan Hadoop, Anda bisa menggunakan perintah dengan format berikut:
   hadoop jar /opt/hadoop-2.7.3/share/hadoop/tools/lib/hadoop-streaming-2.7.3.jar -mapper **PATH_KE_FILE_MAPPER_ANDA** -reducer **PATH_KE_FILE_REDUCER_ANDA** -input **NAMA_FILE_INPUT_ANDA** -output **NAMA_FILE_OUTPUT**

   Contoh:

```
bd@master:/opt/hadoop-2.7.3$ hadoop jar /opt/hadoop-2.7.3/share/hadoop/tools/lib/hadoop-streaming
-2.7.3.jar -mapper /home/bd/samuel/python/mapper.py -reducer /home/bd/samuel/python/reducer.py -i
nput test.txt -output test_output
```

   Pastikan file mapper dan reducer Anda memiliki permission untuk dieksekusi. Kalau tidak, Anda perlu melakukan **chmod +x <nama file>** sebelum mengeksekusi perintah di atas.

3. Anda bisa melihat hasil seperti biasa menggunakan **hdfs dfs -cat <NAMA_DIREKTORI_OUTPUT>**

LATIHAN:

1. Coba untuk melakukan implementasi Latihan sebelumnya (dalam Java) dengan menggunakan Python.
2. Copy file **temperature.txt** dari direktori:
   /home/bd/samuel/weather/Hadoop-MapReduce/NCDC
   File ini berisi data dari sensor cuaca National Climatic Data Center,
   Implementasikan kode MapReduce yang **mencari nilai temperatur maksimum untuk setiap tahunnya.**

   Untuk setiap baris, data yang perlu Anda ekstrak adalah tahun, suhu dan quality Code. Tahun berada di karakter ke 16 sampai 19

Suhu berada di karakter 88 sampai 92
Quality code berada di karakter 93

Misal untuk **line 1** dari file tersebut:
Tahun adalah 1902
Suhu dan quality code : -00941 (suhu -94, quality code 1)

Apabila suhu berisi angka **9999** maka suhu tersebut tidak valid dan tidak boleh diproses. Suhu yang valid juga harus memiliki quality code 0 atau 1 atau 4 atau 5 atau 9.