MAPREDUCE FONKSİYONLARI YAZILAN PROJE VERİLERİ

Bugün MapReduce ile ilgili teorik bilgilerimi iyice pekiştirdikten sonra uygulama kısmına geçtim.Elimdeki küçük örnek verilerle bir MapReduce fonksiyonu yazacaktım.MapReduce ‘in temel programlama dili Java olduğundan o şekilde yazmaya karar verdim.

Öncelikle elimizdeki verilere bakalım.Elimde bir reklam yönetim sistemi vardı ve bu sistemin loglarını inceleyeceğim.Log dosyası aşağıda görüldüğü gibidir :

1293868800864,319248,1,flickr.com,12

1293868801728,625828,1,npr.org,19

1293868802592,522177,2,wikipedia.org,16

1293868803456,535052,2,cnn.com,20

1293868804320,287430,2,sfgate.com,2

1293868805184,616809,2,sfgate.com,1

1293868806048,704032,1,nytimes.com,7

1293868806912,631825,2,amazon.com,11

1293868807776,610228,2,npr.org,6

1293868808640,454108,2,twitter.com,18

1293868809504,723726,1,foxnews.com,9

1293868810368,777690,2,cnn.com,2

1293868811232,441293,2,nytimes.com,3

1293868812096,938131,1,bbc.co.uk,18

1293868812960,931279,1,flickr.com,1

1293868813824,651914,1,yahoo.com,20

1293868814688,101535,2,facebook.com,5

1293868815552,909907,1,facebook.com,3

1293868816416,154380,2,amazon.com,1

1293868817280,718757,2,npr.org,13

Bu dosyayı açıklamak gerekirse ; Veriler delimiter fonksiyonu ile ayrıştırılmıştır.İlk alan Unix tipinde zamanı , ikinci alan kullanıcı kimliğini (id) , üçüncü alan aksiyonu yani 1 tıklama 2 görüntülüme gibi , dördüncü alan hangi domain üzerinde yayınlandığını ve son olarak dördüncü alan ise hangi reklam kampanyasına ait olduğunu göstermektedir.

İlk olarak konsol ekranımıza girip hadoop’u başlatıyoruz.Bununla birlikte HDFS ve YARN’ı da çalıştırıyoruz.Daha sonra kodlarımızı yazmaya başladık.Ben geliştirme ortamı olarak Eclipse kullandım.Ancak daha farklı programlar kullanılabilir.

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.conf.Configured;

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.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

public class AdManagement extends Configured implements Tool

{

Static class CustomMapper extends Mapper<Object,Text,IntWritable,IntWritable>

@Override

Public void map(Object key,Text record,Context context) throws IOException

{

String[] datas=record.toString().split(“,”);

String actionStr=datas[2];

String campaignStr=veriler[4];

try

{

int action=Integer.parseInt(actionStr);

int campaign=Integer.parseInt(campaignStr);

IntWritable mapOutKey=new IntWritable(campaign);

IntWritable mapOutValue=new IntWritable(action);

Context.write(mapOutKey,mapOutValue);

} catch(Exception e)

System.out.println(“An error occurred !!”);

e.printStackTrace();

}

}

}

public static class CustomReducer extends Reducer<IntWritable,IntWritable,Text>

{

public void reduce(IntWritable key,Iterable results,Context context) throws IOException,InterruptedException

{

int campaign=key.get();

int views=0;

int clicks=0;

for(IntWritable i : results)

{

int action=i.get();

if(action == 1)

views++;

else if(action == 2)

clicks++;

}

String stats=”**visualization” + views + “ clicks=” + clicks;**

context.write(new IntWritable(campaign), new Text(stats));

}

}

public static void main(String[] args) throws Exception

{

int res=ToolRunner.run(new Configuration(),new AdManagement(),args);

System.exit(res);

}

@Override

public int run(String[] args)throws Exception

{

İf(args.length != 2)

{

System.out.println(“Input and output paths must be entered”);

Return 1;

}

Path inputPath=new Path(args[0]);

Path outputPath=new Path(args[1]);

Configuration conf=getConf();

Job job=new Job(conf,”AdManagement”);

job.setJarByClass(AdManagement.class);

job.setMapperClass(CustomMapper.class);

job.setReducerClass(CustomReducer.class);

job.setMapOutputValueClass(IntWritable.class);

job.setMapOutputKeyClass(IntWritable.class);

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

TextInputFormat.setInputPaths(job, inputPath);

TextOutputFormat.setOutputPath(job, outputPath);

return job.waitForCompletion(true) ? 0 : 1 ;

}

}

Kısa kod açıklaması

|  |
| --- |
| Code: |
| static class CustomMapper extends Mapper<Object, Text, IntWritable, IntWritable> |

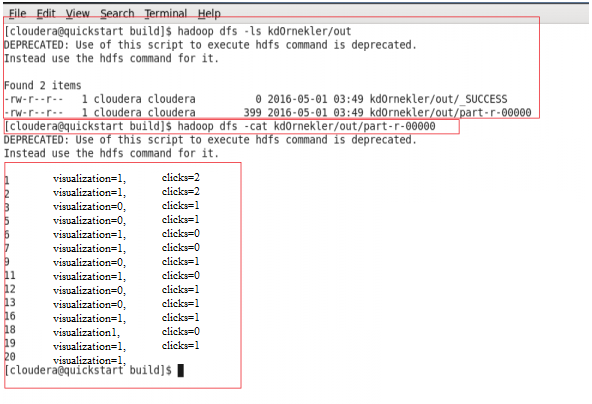
Hadoop çekirdeğinde bulunan Mapper objesini kullanarak kendi Mapper’ımızı yazıyoruz ve “map” metodunu override ediyoruz.  “,” işaretini split ederek bize lazım olan verileri önce string, daha sonra int türüne çeviriyoruz. Sonrada context objesine key-value olarak işletiyoruz.

Aynı şekilde Reduce fonksiyonunu yazdık.Yine çekirdek kütüphaneden implement ettiğimiz bir class yaratıyoruz. Daha sonra main metodunda bir alt metod kullanarak Hadoop’un MapReduce fonksiyonlarını çalıştırması için yönergeleri belirttim.

Hadoop tarafına gelirsek ; bu log dosyasını Hdfs içerisini atmamız lazım ki orada işleyebilelim.Öncelikle çalışmamızı içeri atabilmek için bir klasör oluşturdum.Klasörü oluşturduktan sonra log dosyamızı içine attım ve gelip gelmediğini kontrol ettim.Herşey yolundaydı. Java dosyasını build ettikten sonra bağımlılıklar için CLASSPATH envanterini kullanmamız gerekiyor.

Terminal girdisi :

$export CLASSPATH=/usr/lib/hadoop/client-0.20/\\*



Son Hali