MAPREDUCE UYGULAMA

MAPREDUCE APPLICATION

I continued on MapReduce applications today.I wanted to write the code that found the words and numbers in a given file. I wrote the code in this application by searching day by day and reading various code samples. As you can see from the code, a map class (TokenizerMapper) descending from the Mapper class and a reduce (IntSumReducer) class descending from the reducer class are created so that the configuration and job classes are created in the main function and the program is ready to run with the necessary parameters and map and reduce classes Has become.

Bugün MapReduce uygulamalarına devam ettim.Bana verilen bir dosya içindeki kelimeleri ve sayıları bulan kodu yazmam istendi.Gün boyu araştırarak , çeşitli kod örnekleri okuyarak bu uygulamanında kodlarını yazdım. Koddan da görüleceğe üzere Mapper sınıfından türeyen bir map sınıfı (TokenizerMapper) ile reducer sınıfından türeyen bir reduce (IntSumReducer) sınıfı oluşturularak main fonksiyonu içinde konfigürasyon (configuration) ve iş (job) sınıflarının oluşturulup gerekli parametrelerin ve map ve reduce sınıflarının verilmesi ile program çalışmaya hazır hale geldi.

Program :

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 TokenizerMapper extends Mapper<Object,Text,Text,IntWritable>{

private final static IntWritable one=new IntWritable(1);

private Text Word=new Text();

public void map(Object key,Text value,Context context) throws IOException,InterruptedException{

StringTokenizer itr=new StringTokenizer(value.toString());

While(itr.hasMoreTokens()){

Word.set(itr.nextToken());

Context.write(Word,one);

}

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

{

private IntWritable result=new IntWritable();

public void reduce(Text key,Iterable<IntWritable> values,Context context) throws IOException,InterruptedException {

İnt sum=0;

for(IntWritable val : values) {

sum+=val.get();

}

result.set(sum);

context.write(key,result);

}

}

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(TokenizerMapper.class);

job.setCombinerClass(IntSumReducer.class);

job.setReducerClass(IntSumReducer.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);

}

}

Bu çalışmada optimizasyon algoritmaları incelenmiş ve MapReduce çalışma mantığı, yapısı ve örnek programları üzerinde durulmuştur. Optimizasyon algoritmalarının kullanımına değinilerek; giderek artan büyüklüklerdeki optimizasyon problemlerinin çözümünde dağıtık işlemenin önemine dikkat çekilmiştir. Dağıtık işleme kapasitesine sahip bir sistem olan MapReduce’e bir giriş yapılarak optimizasyon algoritmalarının MapReduce üzerinde çalıştırılması üzerinde durulmuştur.