More Next Blog» shan2cog@gmail.com Dashboard Sign Out

I guys, This blog have been move to new address hadoopgyaan.tk so, please from now onwards new case studies have been posts in the given website.Keep visiting my new website for latests case studies.

TECHNOVA Technical Institute Hadoop Installation Gyaan Hadoop Gyaan Hive UDF's Gyaan Python Gyaan About Me

THURSDAY, 30 JUNE 2016

Apache Hadoop : Facebook Likes Analyzer MapReduce Case Study

Facebook Likes Analyzer example

It is a small java codes that shows how to use Hadoop to analyze facebook data i.e likes. It tells the most frequent likers in the friend circle.

Check out above case study in below weblink:

Facebook Likes Analyzer MapReduce Case Study

I hope this tutorial will surely help you. If you have any questions or problems please let me know.

Happy Hadooping with Patrick..

Posted by Hadoop Gyaan at 09:36:00

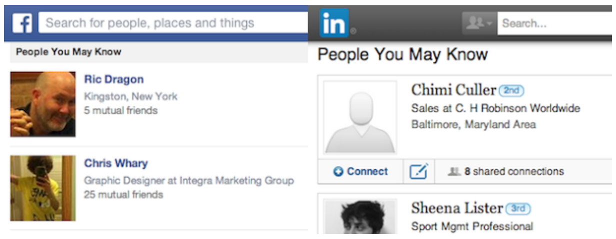
Reactions: funny (0) interesting (0) cool (0)

No comments:

Labels: Hadoop, Mapreduce

WEDNESDAY, 29 JUNE 2016

Apache Hadoop : “People You May Know” Social Network Friendship Recommendation MapReduce Case Study



“People You May Know” Social Network Friendship Recommendation example

The best friendship recommendations often come from friends. The key idea is that if two people have a lot of mutual friends, but they

LIKE US ON FACEBOOK

TWITTER

Follow @HadoopGyaan

SEARCH THIS BLOG

Search

HADOO GYAAN ARCHIVE

June (24)
July (8)
August (11)
September (2)

FOLLOW BY EMAIL

Email address... Submit

are not friends, then the system should recommend them to be connected to each other.

Let's assume that the friendships are undirected: if A is a friend of B then B is also a friend of A. This is the most common friendship system used in Facebook, Google+, Linkedin, and several social networks.

The relationships between user and user can be understood easier in the graph.

0	1,2,3
1	0,2,3,4,5
2	0,1,4
3	0,1,4
4	1,2,3
5	1,6
6	5

In the graph, you can see user 0 is not friends of user 4, and 5, but user 0 and user 4 have mutual friends 1, 2, and 3; user 0 and user 5 have mutual friend 1. As a result, we would like to recommend user 4 and 5 as friends of user 0.

The output recommended friends will be given in the following format. <Recommended friend to USER(# of mutual friends: [the id of mutual friend, ...]),…>. The output result is sorted according to the number of mutual friends, and can be verified from the graph.

0	4 (3: [3, 1, 2]),5 (1: [1])
1	6 (1: [5])
2	3 (3: [1, 4, 0]),5 (1: [1])
3	2 (3: [4, 0, 1]),5 (1: [1])
4	0 (3: [2, 3, 1]),5 (1: [1])
5	0 (1: [1]),2 (1: [1]),3 (1: [1]),4 (1: [1])
6	1 (1: [5])

Now, let's fit this problem into single MapReduce job. User 0 has friends, 1, 2, and 3; as a result, the pair of <1, 2>, <2, 1>, <2, 3>, <3, 2>, <1, 3>, and <3, 1> have mutual friend of user 0. As a result, we can emit <key, value> = <1, r=2; m=0>, <2, r=1; m=0>, <2, r=3; m=0>…, where r means recommended friend, and m means mutual friend. We can aggregate the result in the reduce phase, and calculate how many mutual friends they have between a user and recommended user. However, this approach will cause a problem. What if user A and the recommended user B are already friends? In order to overcome this problem, we can add another attribute is Friend into the emitted value, and we just don't recommend the friend if we know they are already friends in the reduce phase. In the following implementation, m = -1 is used when they are already friends instead of using extra field.

Mapper Class

Reducer Class

Driver Class

```
import java.util.*;
import java.io.*;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.util.Tool;
import org.apache.log4j.Logger;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
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 Recommender {
public static class FriendMapper extends Mapper<LongWritable, Text, IntWritable, Text> {
IntWritable userID = new IntWritable();
Text friends = new Text();
HashMap<String,String> hash = new HashMap<String,String>();
@Override
public void map(LongWritable key, Text value, Context context)
throws IOException, InterruptedException {
// Program now begins reading input dataset line by line
String split_1[] = value.toString().split("\t");
String split_2[]=null;
userID.set(Integer.parseInt(split_1[0]));
if(split_1.length==1){friends.set("null");context.write(userID,friends);}
else{
split_2 = split_1[1].split(",");
//Write user id as key and each friendID,-1 as value. This shows that user and those id's are friends already and shouldnt be recommended.
}
if(split_2!=null)
{
for(int i=0;i<split_2.length;i++)
{
if(split_2[i] != null)
{
friends.set(split_2[i] + ",-1");
context.write(userID,friends);
//    hash.put(split_2[i],split_2[i]);
}
}
//Now iterate over the hashmap to map each friend combo. This time use +1 since we do not know whether or not they are friends.
/*   for(String hashkey : hash.keySet()){

      for(String hashValue : hash.values()){

       if((hashkey.equals(hashValue))==false)
       {
        userID.set(Integer.parseInt(hashkey));
        friends.set(hashValue + ",1");
        context.write(userID, friends);
       }
      }
     }*/
for(int i=0;i<split_2.length;i++){
for(int j=0;j<split_2.length;j++){
if(split_2[i] != split_2[j]){
userID.set(Integer.parseInt(split_2[i]));
friends.set(split_2[j]+",1");
context.write(userID, friends);
}
}
}
}
}
}
public static class FriendReducer extends Reducer<IntWritable, Text, IntWritable, Text> {
HashMap<Integer,Integer> hash = new HashMap<Integer,Integer>();
StringBuilder recommendedList = new StringBuilder();
LinkedList<Integer> friendId = new LinkedList<Integer>();
LinkedList<Integer> comFriendCount = new LinkedList<Integer>();
Text result = new Text();
Text currentVal = new Text();
int count,flag;
int temp,temp1,temp2;
public void reduce(IntWritable key, Iterable<Text> values, Context context)
throws IOException, InterruptedException {
for(Text value: values){
currentVal = value;
if(currentVal.toString().equals("null") == false){
flag = Integer.parseInt(currentVal.toString().split(",")[1]);
if(hash.containsKey(Integer.parseInt(currentVal.toString().split(",")[0]))){
if(flag==1){
if(hash.get(Integer.parseInt(currentVal.toString().split(",")[0])) != 0){
temp=hash.get(Integer.parseInt(currentVal.toString().split(",")[0])) + 1;
hash.put(Integer.parseInt(currentVal.toString().split(",")[0]),temp);
}
}
else{
hash.put((Integer.parseInt(currentVal.toString().split(",")[0])),0);
}
}
else{
if(flag==1){
hash.put(Integer.parseInt(currentVal.toString().split(",")[0]), 1);
}
else{
hash.put((Integer.parseInt(currentVal.toString().split(",")[0])),0);
}
}
}
else {
result.set("\t");
context.write(key,result);
}
}
for(Map.Entry<Integer,Integer> entry : hash.entrySet() ){
if(entry.getValue()!=0){
friendId.add(entry.getKey());
comFriendCount.add(entry.getValue());
}
}
for(int i=0;i<comFriendCount.size();i++){
for(int j=i+1;j<comFriendCount.size();j++){
if(comFriendCount.get(i)<comFriendCount.get(j)){
temp1=friendId.get(j);
friendId.set(j, friendId.get(i));
friendId.set(i,temp1);
temp2=comFriendCount.get(j);
comFriendCount.set(j, comFriendCount.get(i));
comFriendCount.set(i,temp2);
}
else{
if(comFriendCount.get(i)==comFriendCount.get(j) && friendId.get(i)>friendId.get(j)){
temp1=friendId.get(j);
friendId.set(j, friendId.get(i));
friendId.set(i,temp1);
}
}
}
}
if(friendId.size()>0)
{
recommendedList.append("\t").append(friendId.get(0).toString());
for(int k=1;k<Math.min(10,friendId.size());k++){
recommendedList = recommendedList.append(",").append(friendId.get(k).toString());
}
result.set(recommendedList.toString());
hash.clear();
friendId.clear();
comFriendCount.clear();
```

```
recommendedList.setLength(0);
context.write(key, result);
}
}
}
public static void main(String[] args) throws Exception{
/*int res  = ToolRunner .run( new Recommender(), args);
       System .exit(res);*/
Configuration myconf = new Configuration();
Job job  = Job .getInstance(myconf, " recommender ");
job.setJarByClass(Recommender.class);
FileInputFormat.addInputPath(job,  new Path(args[ 0]));
FileOutputFormat.setOutputPath(job,  new Path(args[ 1]));
job.setMapperClass( FriendMapper.class);
job.setReducerClass( FriendReducer.class);
job.setOutputKeyClass( IntWritable .class);
job.setOutputValueClass( Text .class);
System.exit(job.waitForCompletion(true)  ? 0 :1);
}
}
```

Downloads :

People You May Know Sample Text

I hope this tutorial will surely help you. If you have any questions or problems please let me know.

Happy Hadooping with Patrick..

Posted by Hadoop Gyaan at 12:11:00

Reactions: funny (0) interesting (0) cool (0)

1 comment:

Labels: Hadoop, Mapreduce

Newer Posts Home Older Posts

Subscribe to: Posts (Atom)

Powered by Blogger.