<https://jaxenter.com/hadoop-mapreduce-tutorial-maximize-mapreduce-104471.html>

**Problem statement:**

For this article, let’s pretend that we are running an online advertising company. We run advertising campaigns for clients (like Pepsi, Sony) and the ads are displayed on popular websites such as news sites (CNN, Fox) and social media sites (Facebook). To track how well an advertising campaign is doing, we keep track of the ads we serve and ads that users click.

**Scenario**

Here is the sequence of events:

1. We serve the ad to the user
2. If the ad appears on users browser, aka user saw the ad. We track this event as VIEWED\_EVENT
3. If user clicks on the ad, we track this event as CLICKED\_EVENT

These events are logged as plain text log files on our web servers. These logs can get rather large, into hundreds of millions (100,000,000) per day. We are talking about hundreds of gigabytes or even terabytes of data. There is lot of interesting information and trends that can be gleaned by analyzing this massive amount of data, and Hadoop is a good choice to analyze it. For this article, we will calculate the campaign effectiveness by measuring the ***clicks vs. views ratio.***

**Input Log files format and description:**

Log Files: The log files are in the following format:

times- tamp, user\_id, view/click, domain, campaign\_id.

E.g: 1262332801728, 899523, 1, npr.org, 19

◾timestamp : unix time stamp in milliseconds

◾user\_id : each user has a unique id

◾action\_id : 1=view, 2=click

◾domain : which domain the ad was served

◾campaign\_id: identifies the campaign the ad was part of

**Sample log file**

◾1293868800864,319248,1,flickr.com,12

◾1293868801728,625828,1,npr.org,19

◾1293868802592,522177,2,wikipedia.org,16

◾1293868803456,535052,2,cnn.com,20

**Code Walkthrough: Mapper**

The first thing to note is how our mapper class is defined:

static class MyMapper extends Mapper<Object, Text, IntWritable, IntWritable>

Explanation of the 4 parameters:

First two parameters are - Input Key, Input Value. That is “Object” and “Text”.

* Input Key is the line number in the log file. Since we don’t care about it, we are just casting as Object. Input value is the actual line in the log file, a java string data type, the equivalent MapReduce class is *Text.*
* Output Key is the IntWriteable and Output value is IntWritable. Mapper outputs are Campaign\_id and Action\_Id (which is a View or Click)

The mapper function is quite straightforward. We split the line into tokens. Extract ‘campaign’ and ‘action’ fields. Wrap these two as IntWritables and write them out.

Code walkthrough: Reducer

Let’s start with Reducer class definition:

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

* *input key/value : Int- Writable/IntWritable*: These class types match the output of Mapper (it is very important that Ouput datatypes of mapper class should be same of input datatypes of Reducer class
* and the output is *Int- Writable/Text*

Let’s look at the reduce function:

public void reduce(IntWritable key, Iterable<IntWritable> results, Context context)

Key/value : *IntWritable/List* *of IntWritables* for every campaign, we are getting all actions for that campaign as an iterable list. We are iterating through *action\_ids* . And calculating ‘views’ and ‘clicks’. Once we are done calculating, we write out the results. This is possible because all actions for a campaign are grouped and sent to one reducer.

Mapper code:

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 CampaignMR1 {

{

static class MyMapper extends Mapper<Object, Text, IntWritable, IntWritable>

{

@Override

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

{

//timestamp, userid, action id, domain, campaign id

String[] tokens = record.toString().split(",") ;

String actionStr = tokens[2];

string campaignStr = tokens[4];

try

{

int action = Integer.parseInt(actionStr);

int campaign = Integer.parseInt(campaignStr);

IntWritable mapOutkey = new IntWritable(campaign);

IntWritable mapOutValue = nefw IntWritable(action);

context.write(mapOutkey, mapOutValue);

}

catch (Exception ex){

System.out.println("Excetipn:");

e.printStackTrace();

}

}

}

}

**Reducer code:**

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

{

public void reduce(IntWritable key, Iterable<IntWritable> 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 = "views=" + views + ", clicks=" + clicks;

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

}

}