Contextual Advertising using Apache Hive and Amazon EMR

An internet advertising company operates a data warehouse using Hive and Amazon Elastic MapReduce. This company runs machines in Amazon EC2 that serve advertising impressions and redirect clicks to the advertised sites. The machines running in Amazon EC2 store each impression and click in log files pushed to Amazon S3.

Submitted By: Richard@AWS
AWS Products Used: Amazon Elastic MapReduce
Created On: September 25, 2009

Storing Logs on Amazon S3

The ad serving machines produce two types of log files: impression logs and click logs. Every time we display an advertisement to a customer, we add an entry to the impression log. Every time a customer clicks on an advertisement, we add an entry to the click log.

Every five minutes the ad serving machines push a log file containing the latest set of logs to Amazon S3. This allows us to produce timely analyses of the logs. See the following article on monitoring the health of ad serving programs: http://aws.amazon.com/articles/2854.

The ad server machines push their impression logs into Amazon S3. For example:

```
s3://elasticmapreduce/samples/hive-ads/tables/impressions/
  dt=2009-04-13-08-05/ec2-12-64-12-12.amazon.com-2009-04-13-08-05.log
```

We put the log data in the elasticmapreduce bucket and include it in a subdirectory called tables/impressions. The impressions directory contains additional directories named such that we can access the data as a partitioned table within Hive. The naming syntax is [Partition column]=[Partition value]. For example: dt=2009-04-13-05.

Launching a Development Job Flow

Our first task is to combine the click and impression logs into a single table that specifies if there was a click for a specific ad and information about that click.

Before we create a table, let's start an interactive job flow so that we can enter our Hive commands one at a time to test that they work. After we verify the Hive commands, we can use them to create a script, store the script on Amazon S3, and create a job flow that executes the script.

There are two ways to start an interactive job flow. You can either use the Amazon Elastic MapReduce command line interface available at http://aws.amazon.com/developertools/2264 or you can use the AWS Management Console available at http://console.aws.amazon.com.

To run an interactive Hive session, you need an Amazon EC2 key pair so you can ssh into the master node. If you don't have an EC2 key pair, you need to create one using the AWS Management Console.

1. Select the EC2 tab.
2. Select Key Pairs on the left navigation pane.
3. Click "Create KeyPair."
4. Save your secret key PEM file somewhere, you'll need it later.

To start an interactive Hive session using the AWS Management Console

1. Select the Elastic MapReduce tab.
2. Select the "US East" Region.
3. Click "Create New Job Flow."
4. Choose a descriptive name for your job flow, for example, "Hive Ads Tutorial -- Interactive."
5. Select "Hive Program" and click "Continue".
6. Select "Interactive Hive Session" and click "Continue".
7. In the "Type of Instance" list, select "Large (m1.large)".
8. Specify an EC2 key pair in the "Advanced Options" and click "Continue".
9. Accept the default selection to "Proceed with no Bootstrap Actions" and click "Continue".
10. Click "Create Job Flow" to complete the wizard and launch your Hive interactive job flow.
11. On the job flows page, wait until the job flow enters the "WAITING" state and then click "Refresh."
12. Select the job flow and find the DNS name of the master node in the detail pane in the bottom half of the screen.
13. Save the master DNS name. You'll use it to ssh to the master node.

Alternatively, you can use the command line client to start an interactive job flow. Make sure you specify your EC2 keypair name in your credentials.json file, as described in the README that comes with the command line client. In the command line client, you use the create command to start a Hive interactive job flow.

```
 $ ./elastic-mapreduce --create --alive --hive-interactive --name "Hive Job Flow"  -
-instance-type m1.large --availability-zone us-east-1a
 Created job flow [JobFlowID]
```

This job flow takes a few minutes to transition from the STARTING to the WAITING states. You can monitor the progress of the job flow using the list command.

```
  $ ./elastic-mapreduce --list
  [JobFlowID]     STARTING   Hive Cluster
     PENDING        Setup Hive
```

After the job flow starts and successfully executes the "Setup Hive" step you should see a result similar to the following.

```
  $ ./elastic-mapreduce --list
  [JobFlowID]     WAITING   ec2-67-202-12-120.compute-1.amazonaws.com  Hive Cluster
     COMPLETED      Setup Hive
```

Now that you have a running job flow you can ssh to the master node using the PEM file that you downloaded when you created your Amazon EC2 key pair.

```
  $ ssh -i ~/my-keypair-private-key.pem hadoop@ec2-67-202-12-120.compute-1.amazonaw
s.com
```

If you're using the command line client and you have specified the location of your PEM file in your credentials.json file using a key-pair-file setting, you can ssh to the master node using the command line client with

```
  $ ./elastic-mapreduce --ssh --jobflow [JobFlowID]
```

Running a Hive Session on the Master Node

Once you connect to the master node via ssh you run Hive with the following command. (Replace mybucket with your own Amazon S3 bucket.)

```
hadoop@domU-12-31-39-07-D2-14:~$ hive \
  -d SAMPLE=s3://elasticmapreduce/samples/hive-ads \
  -d DAY=2009-04-13 -d HOUR=08 \
  -d NEXT_DAY=2009-04-13 -d NEXT_HOUR=09 \
  -d OUTPUT=s3://mybucket/samples/output
hive>
```

This command introduces some variables that we'll use shortly in our Hive statements. Later in this tutorial when we run the Hive statements from a script stored in Amazon S3, we'll pass the variables into the job flow. For now, it is convenient to have these variables available as we create statements in interactive mode.

Let's see if Hive has any tables defined.

```
hive> show tables;
OK
Time taken: 3.51 seconds
hive>
```

Hive reports that there are no tables defined. So let's create a table from the impression and click logs that are stored in Amazon S3.

Declaring Tables in Amazon S3

We need to use a custom Serde (Serializer-deserializer) to read the impressions and clicks data, which is stored in JSON format. Serdes enables Hive to read data stored in a custom format. Our Serde is stored in a JAR file located in Amazon S3 and we tell Hive about it via the following statement.

```
ADD JAR ${SAMPLE}/libs/jsonserde.jar ;
```

Notice that we're using the variable ${SAMPLE} defined when we invoked the Hive interpreter in the previous section. Unlike the syntax for other Hive statements, the JAR location supplied in the ADD statement is not a quoted string.

Now that our Serde is defined, we can tell Hive about our clicks and impressions data by creating an external table.

```
CREATE EXTERNAL TABLE impressions (
  requestBeginTime string, adId string, impressionId string, referrer string,
  userAgent string, userCookie string, ip string
)
PARTITIONED BY (dt string)
ROW FORMAT
  serde 'com.amazon.elasticmapreduce.JsonSerde'
  with serdeproperties ( 'paths'='requestBeginTime, adId, impressionId, referrer,
userAgent, userCookie, ip' )
LOCATION '${SAMPLE}/tables/impressions' ;
```

The data for this table resides in Amazon S3. Creating the table is a quick operation because we're just telling Hive about the existence of the data, not copying it. When we query this table Hive will read the table using Hadoop.

AWS PRODUCTS

Amazon CloudFront
Amazon EC2
Amazon Flexible Payments Service
Amazon SimpleDB
Amazon SQS
Amazon S3
AWS Elastic Beanstalk
Amazon SES

TECHNOLOGY

Java
Windows
.NET
Ruby
Python
PHP
Mobile

RELATED LINKS

Release Notes
Developer Tools
Documentation
Public Data Sets
Security Center
Videos & Webinars
Amazon Machine Images (AMIs)

The table is partitioned based on time. As yet, Hive doesn't know which partitions exist in the table. We can tell Hive about the existence of a single partition using the following statement.

```
ALTER TABLE impressions ADD PARTITION (dt='2009-04-13-08-05') ;
```

If we were to query the table at this point the results would contain data from just this partition. We can instruct Hive to recover all partitions by inspecting the data stored in Amazon S3 using the RECOVER PARTITIONS statement.

Note: After Hive 0.13.1 RECOVER PARTITIONS is deprecated. The same capability is supported using MSCK REPAIR table_name . For more information, see https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL .

```
ALTER TABLE impressions RECOVER PARTITIONS ;
```

We follow the same process to recover clicks.

```
CREATE EXTERNAL TABLE clicks (
  impressionId string
)
PARTITIONED BY (dt string)
ROW FORMAT
  SERDE 'com.amazon.elasticmapreduce.JsonSerde'
  WITH SERDEPROPERTIES ( 'paths'='impressionId' )
LOCATION '${SAMPLE}/tables/clicks' ;

ALTER TABLE clicks RECOVER PARTITIONS ;
```

AWS PRODUCTS

Amazon CloudFront
Amazon EC2
Amazon Flexible Payments Service
Amazon SimpleDB
Amazon SQS
Amazon S3
AWS Elastic Beanstalk
Amazon SES

TECHNOLOGY

Java
Windows
.NET
Ruby
Python
PHP
Mobile

RELATED LINKS

Release Notes
Developer Tools
Documentation
Public Data Sets
Security Center
Videos & Webinars
Amazon Machine Images (AMIs)

Combining the Clicks and Impressions Tables

We want to combine the clicks and impressions tables so that we have a record of whether or not each impression resulted in a click. We'd like this data stored in Amazon S3 so that it can be used as input to other job flows.

```
CREATE EXTERNAL TABLE joined_impressions (
  requestBeginTime string, adId string, impressionId string, referrer string,
    userAgent string, userCookie string, ip string, clicked Boolean
  )
  PARTITIONED BY (day string, hour string)
  STORED AS SEQUENCEFILE
  LOCATION '${OUTPUT}/joined_impressions'
;
```

This table is partitioned as well. An advantage of partitioning tables stored in Amazon S3 is that if Hive needs only some of the partitions to answer the query then only the data from these partitions will be downloaded from Amazon S3.

The joined_impressions table is stored in SEQUENCEFILE format, which is a native Hadoop file format that is more compressed and has better performance than JSON files.

Next, we create some temporary tables in the job flow's local HDFS partition to store intermediate impression and click data.

```
CREATE TABLE tmp_impressions (
  requestBeginTime string, adId string, impressionId string, referrer string,
  userAgent string, userCookie string, ip string
)
```

```
STORED AS SEQUENCEFILE;
```

We insert data from the impressions table for the time duration we're interested in. Note that because the impressions table is partitioned only the relevant partitions will be read.

```
INSERT OVERWRITE TABLE tmp_impressions
  SELECT
    from_unixtime(cast((cast(i.requestBeginTime as bigint) / 1000) as int)) reques
tBeginTime,
    i.adId, i.impressionId, i.referrer, i.userAgent, i.userCookie, i.ip
  FROM
    impressions i
  WHERE
    i.dt >= '${DAY}-${HOUR}-00' and i.dt < '${NEXT_DAY}-${NEXT_HOUR}-00'
;
```

The start of the time period is DAY-HOUR and the end of the period is NEXT_DAY-NEXT_HOUR. NEXT_DAY is the day of the next time period. It differs from ${DAY} only when we're processing the last hour of a day. In this case the time period ends on the next day.

For clicks, we extend the period of time over which we join by 20 minutes. Meaning we accept a click that occurred up to 20 minutes after the impression.

```
CREATE TABLE tmp_clicks (
  impressionId string
) STORED AS SEQUENCEFILE;
```

```
INSERT OVERWRITE TABLE tmp_clicks
  SELECT
    impressionId
  FROM
    clicks c
  WHERE
    c.dt >= '${DAY}-${HOUR}-00' AND c.dt < '${NEXT_DAY}-${NEXT_HOUR}-20'
;
```

Now we combine the impressions and clicks tables using a left outer join. This way any impressions that did not result in a click are preserved. This join also enables us to search for clicks that occurred after the time period. The query also excludes any clicks that did not originate from an impression in the selected time period.

```
INSERT OVERWRITE TABLE joined_impressions PARTITION (day='${DAY}', hour='${HOUR}')
SELECT
  i.requestBeginTime, i.adId, i.impressionId, i.referrer, i.userAgent, i.userCooki
e,
  i.ip, (c.impressionId is not null) clicked
FROM
  tmp_impressions i LEFT OUTER JOIN tmp_clicks c ON i.impressionId = c.impressionI
d
;
```

Because the joined_impressions table is located in Amazon S3 this data is now available for other job flows to use.

AWS PRODUCTS

Amazon CloudFront
Amazon EC2
Amazon Flexible Payments Service
Amazon SimpleDB
Amazon SQS
Amazon S3
AWS Elastic Beanstalk
Amazon SES

TECHNOLOGY

Java
Windows
.NET
Ruby
Python
PHP
Mobile

RELATED LINKS

Release Notes
Developer Tools
Documentation
Public Data Sets
Security Center
Videos & Webinars
Amazon Machine Images (AMIs)

AWS PRODUCTS

Amazon CloudFront
Amazon EC2
Amazon Flexible Payments Service
Amazon SimpleDB
Amazon SQS
Amazon S3
AWS Elastic Beanstalk
Amazon SES

TECHNOLOGY

Java
Windows
.NET
Ruby
Python
PHP
Mobile

RELATED LINKS

Release Notes
Developer Tools
Documentation
Public Data Sets
Security Center
Videos & Webinars
Amazon Machine Images (AMIs)

Terminate an Interactive Session

At this point in the tutorial if you'd like to take a break you can terminate your job flow either using the AWS Management Console by selecting the job flow and then pressing "Terminate Job Flow," or by using the --terminate command in the command line client.

```
$ ./elastic-mapreduce --terminate [JobFlowID]
```

We will return to an interactive session in subsequent sections so you can choose to leave your job flow running. Because the job flow is interactive it will not shut down until you terminate it.

Running in Script Mode

Let's collect all of the Hive statements developed so far in this tutorial and place them in Amazon S3 in a file called

```
s3://elasticmapreduce/samples/hive-ads/libs/join-clicks-to-impressions.q
```

Now we can spawn a job flow to join clicks to impressions for a particular time period using (Replace mybucket in OUTPUT with your bucket):

```
$ SAMPLE=s3://elasticmapreduce/samples/hive-ads
$ OUTPUT=s3://mybucket/samples/output
$ ./elastic-mapreduce --create --name "Join Clicks" \
    --hive-script --arg $SAMPLE/libs/join-clicks-to-impressions.q \
    --args -d,SAMPLE=$SAMPLE \
    --args -d,DAY=2009-04-13,-d,HOUR=08 \
    --args -d,NEXT_DAY=2009-04-13,-d,NEXT_HOUR=09 \
    --args -d,INPUT=$SAMPLE/tables \
    --args -d,OUTPUT=$OUTPUT \
    --args -d,LIB=$SAMPLE/libs
```

To run these job flows regularly, every hour one would use a workflow or task scheduling system.

Contextual Advertising Model

In the previous section, we created a regular process to extract clicks and impressions data from log files and to join that data in a table called joined_impressions.

Lets us now consider the task of experimenting with a new algorithm that implements contextual advertising. In this scenario, we want to create a simple, statistically inspired model for ad serving.

Given an advertising context consisting of user agent, user IP, and page URL, we'd like to predict which of our available advertisements is most likely to result in a click.

Let's say that an advertising context consists of a number of features that are true. For example, a feature could be the user agent containing the keyword Mozilla or that the IP address began with the prefix 23.12.

We'd like to estimate the probability of a click given the context.

```
P[click|context]
```

One heuristic for doing this is the following formula.

```
product_{f in context} Pr[click|f=true]
```

AWS PRODUCTS

Amazon CloudFront
Amazon EC2
Amazon Flexible Payments Service
Amazon SimpleDB
Amazon SQS
Amazon S3
AWS Elastic Beanstalk
Amazon SES

TECHNOLOGY

Java
Windows
.NET
Ruby
Python
PHP
Mobile

RELATED LINKS

Release Notes
Developer Tools
Documentation
Public Data Sets
Security Center
Videos & Webinars
Amazon Machine Images (AMIs)

This heuristic multiplies the probability of a click for each feature that is true in the advertising context. If we take the negative log of this formula, we get the following formula.

```
- sum_{f in context} log ( count[click,f=true] / count[f=true] )
```

Because the log of zero is -inf we want to exclude from the sum any features for which the click through probability is zero. For these cases, we insert a minimum value of 0.0001.

Declaring External Tables in the Interactive Job Flow

For this part of the tutorial we're running again in interactive mode. If you terminated the interactive job flow you created earlier you'll have to start another one. Otherwise you can continue on using the job flow you started earlier.

Start hive again with the following

```
hadoop@domU-12-31-39-07-D2-14:~$ hive \
  -d SAMPLE=s3://elasticmapreduce/samples/hive-ads
```

Our first task is to declare again the joined_impressions table and to recover partitions.

Note: After Hive 0.13.1 RECOVER PARTITIONS is deprecated. The same capability is supported using MSCK REPAIR table_name . For more information, see https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL .

```
CREATE EXTERNAL TABLE IF NOT EXISTS joined_impressions (
 request_begin_time string, ad_id string, impression_id string,
 page string, user_agent string, user_cookie string, ip_address string,
 clicked boolean
)
PARTITIONED BY (day STRING, hour STRING)
STORED AS SEQUENCEFILE
LOCATION '${SAMPLE}/tables/joined_impressions';
```

```
ALTER TABLE joined_impressions RECOVER PARTITIONS;
```

Let's check that the partitions are in order.

```
SHOW PARTITIONS joined_impressions;
```

Producing the Feature Matrix

We need to do some transformation on our impression data to produce Boolean features. For user agent, we would like to extract keywords. For IP addresses we'd like to take only the top two bytes. For page URLs, we'd like to convert them to lower case. In this section, we'll examine each of these in turn.

User Agent

Every time you visit a website your browser identifies itself with a user agent. The user agent contains information about the browser and machine the customer is using to view the ad, for example "Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10.5; en-US; rv:1.9.1) Gecko/20090624 Firefox/3.5".

An easy way to convert the user agent string into a sequence of keywords is to use a python script. As we'll see shortly, we can call this script directly from within a Hive statement.

```
#!/usr/bin/python

import sys
import re

for line in sys.stdin:
  user_agent, ad, clicked = line.strip().split('\t')
  components = re.split('[;/,\(\) ]', user_agent)
  for component in components:
    if len(component) != 0:
       print '\t'.join([component, ad, clicked])
```

This script reads table rows passed to sys.stdin one line at a time. Each line is tab separated and has three columns: user_agent, ad, and clicked. The script outputs one record per keyword found in the user agent field.

The output of this script is a table with columns: keyword, ad, and clicked. The script outputs multiple records if a keyword occurs more than once in an impression. Possible improvements to the script include removing duplicate keywords and sharpening the recognition of keywords.

To call this script from within a Hive, we issue a MAP statement.

```
MAP
 joined_impressions.user_agent, joined_impressions.ad_id,
 joined_impressions.clicked
USING
 '${SAMPLE}/libs/split_user_agent.py' AS
 feature, ad_id, clicked
FROM
  joined_impressions
LIMIT 10;
```

The columns user_agent, ad_id, and clicks from the joined_impressions table are input to the script and the result is a table with the columns feature, ad_id, and clicked.

The output of the statement is displayed on the console so we limit the number of lines output to ten. We can see from the output that the keywords contain spaces and are not lower cased. To normalize the output we apply the user defined functions trim and lower and we prefix each keyword by 'ua:' so these features can be mixed with other features.

```
  SELECT concat('ua:', trim(lower(temp.feature))) as feature, temp.ad_id, temp.click
ed
  FROM (
    MAP joined_impressions.user_agent, joined_impressions.ad_id, joined_impressions.
clicked
    USING '${SAMPLE}/libs/split_user_agent.py' as feature, ad_id, clicked
    FROM joined_impressions
  ) temp
  LIMIT 10;
```

IP Address

To normalize the IP address, we extract the first two octets of the IP address. Using a regex makes this easy. The regex below says start matching at the beginning of the field, find one to three digits followed by a period, and then one to three more digits and capture that pattern. The regexp_extract UDF takes the string to match, the regex to use, and then the capturing group to return. In this case, we want the first captured group.

AWS PRODUCTS

Amazon CloudFront
Amazon EC2
Amazon Flexible Payments Service
Amazon SimpleDB
Amazon SQS
Amazon S3
AWS Elastic Beanstalk
Amazon SES

TECHNOLOGY

Java
Windows
.NET
Ruby
Python
PHP
Mobile

RELATED LINKS

Release Notes
Developer Tools
Documentation
Public Data Sets
Security Center
Videos & Webinars
Amazon Machine Images (AMIs)

AWS PRODUCTS

Amazon CloudFront

Amazon EC2

Amazon Flexible Payments Service

Amazon SimpleDB

Amazon SQS

Amazon S3

AWS Elastic Beanstalk

Amazon SES

TECHNOLOGY

Java

Windows

.NET

Ruby

Python

PHP

Mobile

RELATED LINKS

Release Notes

Developer Tools

Documentation

Public Data Sets

Security Center

Videos & Webinars

Amazon Machine Images (AMIs)

```
SELECT
  concat('ip:', regexp_extract(ip_address, '^([0-9]{1,3}\.[0-9]{1,3}).*', 1)) AS
    feature, ad_id, clicked
FROM
  joined_impressions
LIMIT 10;
```

URL

To extract a feature from the URL of the page on which the advertisement displays, we make the URLs all lowercase and add "page:" to the beginning.

```
SELECT concat('page:', lower(page)) as feature, ad_id, clicked
FROM joined_impressions
LIMIT 10;
```

Combining the Features

Now that we've written queries to normalize each of the feature types let's combine them into one table. We can do this using Hive's UNION operator. Keep in mind that all sub queries in the union must have the same number of columns that have the same, exact names.

```
  SELECT *
  FROM (
    SELECT concat('ua:', trim(lower(ua.feature))) as feature, ua.ad_id, ua.clicked
    FROM (
      MAP joined_impressions.user_agent, joined_impressions.ad_id, joined_impression
s.clicked
      USING '${SAMPLE}/libs/split_user_agent.py' as (feature STRING, ad_id STRING, c
licked BOOLEAN)
      FROM joined_impressions
    ) ua

   UNION ALL

   SELECT concat('ip:', regexp_extract(ip_address, '^([0-9]{1,3}\.[0-9]{1,3}).*',
1)) as feature, ad_id, clicked
   FROM joined_impressions

   UNION ALL

   SELECT concat('page:', lower(page)) as feature, ad_id, clicked
     FROM joined_impressions
   ) temp
   limit 50;
```

Note that we had to modify the user agent query slightly. Passing data through a mapper strips the columns of their types and returns them as strings. To merge with the other tables, we need to define clicked as a Boolean.

Index Table

Now that we've compiled a logical table of tuples (feature, ad_id, clicked), it is time to process these to form our heuristic table. Logically, this is a sparse matrix with the axes, features and ad_id. The value represents the percentage of times an ad was clicked. This percentage is represented by the following table.

```
CREATE TABLE feature_index (
```

```
    feature STRING,
    ad_id STRING,
    clicked_percent DOUBLE )
  STORED AS SEQUENCEFILE;
```

AWS PRODUCTS

Amazon CloudFront
Amazon EC2
Amazon Flexible Payments Service
Amazon SimpleDB
Amazon SQS
Amazon S3
AWS Elastic Beanstalk
Amazon SES

TECHNOLOGY

Java
Windows
.NET
Ruby
Python
PHP
Mobile

RELATED LINKS

Release Notes
Developer Tools
Documentation
Public Data Sets
Security Center
Videos & Webinars
Amazon Machine Images (AMIs)

Now, we extend the query from above:

```
  INSERT OVERWRITE TABLE feature_index
    SELECT
      temp.feature,
      temp.ad_id,
      sum(if(temp.clicked, 1, 0)) / cast(count(1) as DOUBLE) as clicked_percent
    FROM (
      SELECT concat('ua:', trim(lower(ua.feature))) as feature, ua.ad_id, ua.clicked
      FROM (
        MAP joined_impressions.user_agent, joined_impressions.ad_id, joined_impressi
ons.clicked
        USING '${SAMPLE}/libs/split_user_agent.py' as (feature STRING, ad_id STRING,
clicked BOOLEAN)
      FROM joined_impressions
    ) ua

    UNION ALL

    SELECT concat('ip:', regexp_extract(ip_address, '^([0-9]{1,3}\.[0-9]{1,3}).*',
1)) as feature, ad_id, clicked
    FROM joined_impressions

    UNION ALL

    SELECT concat('page:', lower(page)) as feature, ad_id, clicked
    FROM joined_impressions
  ) temp
  GROUP BY temp.feature, temp.ad_id;
```

There are a few new aspects to our Hive statement. The first is the GROUP BY at the end of the query. We group by feature and ad_id because these are the keys of our output.

To find the percentage, we need to find the total number of rows in the grouping and the number of rows in which clicked is true. The count is easy; we just use the standard SQL function, count. However, this returns an integer and we want a double for division, so we use the the cast function

```
  cast(count(clicked = 'true') as DOUBLE)
```

To calculate the number of impressions for each feature which resulted in a click, we use the conditional function "if". The function "if" takes three parameters: the conditional, the value to return when true, and the value to return when false. In our case, we want to return 1 when true and 0 when false and then sum these values.

```
  sum(if(clicked = 'true', 1, 0))
```

Finally, we divide the number where clicked is true by the total count to obtain Pr[click|feature].

Applying the Heuristic

Now that we have our heuristic table we can try a few sample tests to see how it performs for the features 'us:safari' and 'ua:chrome'.

```
SELECT
  ad_id, -sum(log(if(0.0001 > clicked_percent, 0.0001, clicked_percent))) AS value
FROM
  feature_index
WHERE
  feature = 'ua:safari' OR feature = 'ua:chrome'
GROUP BY
  ad_id
ORDER BY
  value ASC
LIMIT 100
;
```

The result is advertisements ordered by a heuristic estimate of the chance of a click. At this point, we could look up the advertisements and see, perhaps, a predominance of advertisements for Apple products.

At this point if your interactive hive job flow is still running, don't forget to terminate it.

Summary

In this tutorial, we've seen how to develop a job flow to process impression and click logs uploaded to S3 by web server machines. The result of this job flow is a table in Amazon S3 that was used by an analyst to develop and test a model for contextual advertising. The Hive statements collected by the analyst could be used within a job flow to generate a model file. The analyst could upload the file to Amazon S3 and thus make it available to adserver machines to serve ads contextually.

Additional Hive Resources

For additional information about Hive and Amazon Elastic MapReduce, go to http://aws.amazon.com/articles/2857.