Three Approaches to Data Analysis with Hadoop

A Dell Technical White Paper

Dave Jaffe, Ph.D.
Solution Architect
Dell Solution Centers

Executive Summary

This white paper demonstrates analysis of large datasets using three different tools that are part of the Hadoop ecosystem – MapReduce, Hive and Pig. The application used is a geographic and temporal analysis of Apache web logs. The problem is explained in depth and then solutions are shown for the three tools. Complete code is included in the Appendices, along with a description of the GeoWeb Apache Log Generator tool (available from http://github.com/DaveJaffe/BigDataDemos) as well as the R methods used to analyze and plot the results. Results are shown for all three tools with a 1TB set of log files and a 10TB set of log files.

November 2013

Table of Contents

Tables

Figures

1 Introduction

The Apache Hadoop ecosystem (see http://hadoop.apache.org) includes many open source utilities for analyzing large datasets. The choice of a particular tool depends on the needs of the analysis, the skillset of the user, and the tradeoff between development time and execution time. Three popular tools for analyzing data resident in the Hadoop Distributed File System (HDFS) include MapReduce, Hive and Pig. MapReduce requires a computer program (often written in Java) to read, analyze and output the data. Hive provides a SQL-like front end for those with a database background. Pig provides a high-level language for data processing that also enables the user to exploit the parallelism inherent in a Hadoop cluster. Hive and Pig generate MapReduce code to do the actual analysis.

In this paper the three approaches are contrasted using a popular use case for Hadoop: Apache web log analysis. While many enterprises are employing very sophisticated analysis algorithms to wring useful information from these logs (to deduce customer buying habits for example), the example shown here focuses on understanding where the web site's users are coming from and when.

The MapReduce, Hive and Pig programs shown here parse Apache web logs for the remote IP address (the IP address of the user's web browser) and time of day and then join the IP address to a table of web addresses to generate the country of origin and hour of the web access. They will work with any Apache web log files in the standard format.

The web logs analyzed in these tests were generated by a MapReduce program, the GeoWeb Apache Weblog Generator (available from author) that rapidly creates in HDFS synthetic Apache web logs with a realistic distribution of remote IP addresses and access times. Using these web logs as input, it is easy to check the output of the analysis programs for accuracy.

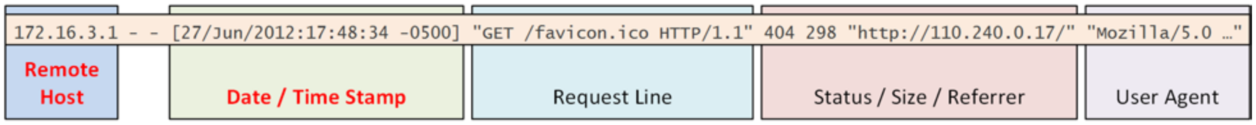
The problem is defined further in the next section, followed by sections on the MapReduce, Hive and Pig solutions, and then the results.

2 The Problem – Geographical Web Analysis

Detailed analysis of website logs is a common Big Data task. Depending on the type of website, these logs may contain information about customer shopping habits, social media networks, or web advertisement effectiveness. A typical web log is on the order of 100 bytes per click so large websites handling millions of simultaneous users can generate 100s of gigabytes or even terabytes of weblogs per day. To ferret out the nuggets of valuable information from this mass of data can require very sophisticated programs.

The solutions demonstrated in this paper tackle a simpler weblog analysis task: using the remote IP address and timestamp collected with each weblog to measure the amount of traffic coming to the website by country of origin on an hour-by-hour basis during the average day. The remote IP address is the first component of the standard Apache weblog and the hour may be extracted from the timestamp, which is the second component of most weblogs (see Figure 1). Our solutions need to extract these items, and look the IP address up in a table mapping IP addresses to host countries (for simplicity we will look at only the first two octets of the IP address and look them up in a table listing all the two-octet or Class B addresses that are used solely by a single country).

Figure 1. A Standard Apache Web Log and its Components


```
172.16.3.1 - - [27/Jun/2012:17:48:34 -0500] "GET /favicon.ico HTTP/1.1" 404 298 "http://110.240.0.17/" "Mozilla/5.0 …"
```

The data used in these tests was generated by a MapReduce program, the GeoWeb Apache Log Generator, available from the author (see Appendix 6). This program produces realistic sequential Apache web logs for a specified month, day, year and number of clicks per day. The remote hosts are distributed geographically among the top 20 Internet-using countries (see Table 1) and temporally so that each region is most active during their local evening hours (simulating a consumer or social web site), as shown in Table 2. The web site is assumed to be in the Central US time zone and each of the countries is assigned a single offset from that for simplicity.

Note that although the log format used by the Apache web server was used in these tests, the algorithms used in these solutions can easily be adapted to other formats.

Table 1. Top Twenty Internet-using Countries

Country	Percent	Country	Percent
China	31	Iran	3
US	13	Korea	3
India	7	Mexico	3
Brazil	5	Nigeria	3
Japan	5	Turkey	3
Germany	4	Italy	2
Russia	4	Phillipines	2
France	3	Pakistan	1
UK	3	Spain	1
Indonesia	3	Vietnam	1

Source: Wikipedia 2011 Data

Table 2. Hourly Distribution of Web Accesses in Local Time

Hour	Percent	Hour	Percent
00	4	12	3
01	3	13	3
02	1	14	2
03	1	15	2
04	1	16	3
05	1	17	4
06	2	18	6
07	2	19	8
08	2	20	12
09	2	21	12
10	2	22	12
11	2	23	10

3 The MapReduce Solution

The Hadoop MapReduce framework provides a flexible, resilient and efficient mechanism for distributed computing over large clusters of servers. The framework supports map tasks that read data stored in the Hadoop Distributed File System (HDFS) and emit key-value pairs, combine tasks that aggregate the values for each key being emitted by a mapper, and reduce tasks that process the values for each key. Writing a MapReduce program is the most general way to exploit the capabilities of this framework for data manipulation and analysis. See http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html for a good introduction to writing MapReduce code.

The design of the MapReduce program to solve the geographical web analysis is fairly straightforward: the Mapper will read log files from HDFS one line at a time, parse the first two octets of the remote IP address as well as the hour of the web access, look up the country corresponding to that IP address in a table, and emit a key composed of the country code and hour, with a value of 1. The Combiner and Reducer (actually the same program) will add up all the values per key and write 24 keys per country detected to HDFS, each with a value corresponding to the total number of hits coming from that country in that hour across the whole set of log files. The flow is shown in Figure 2.

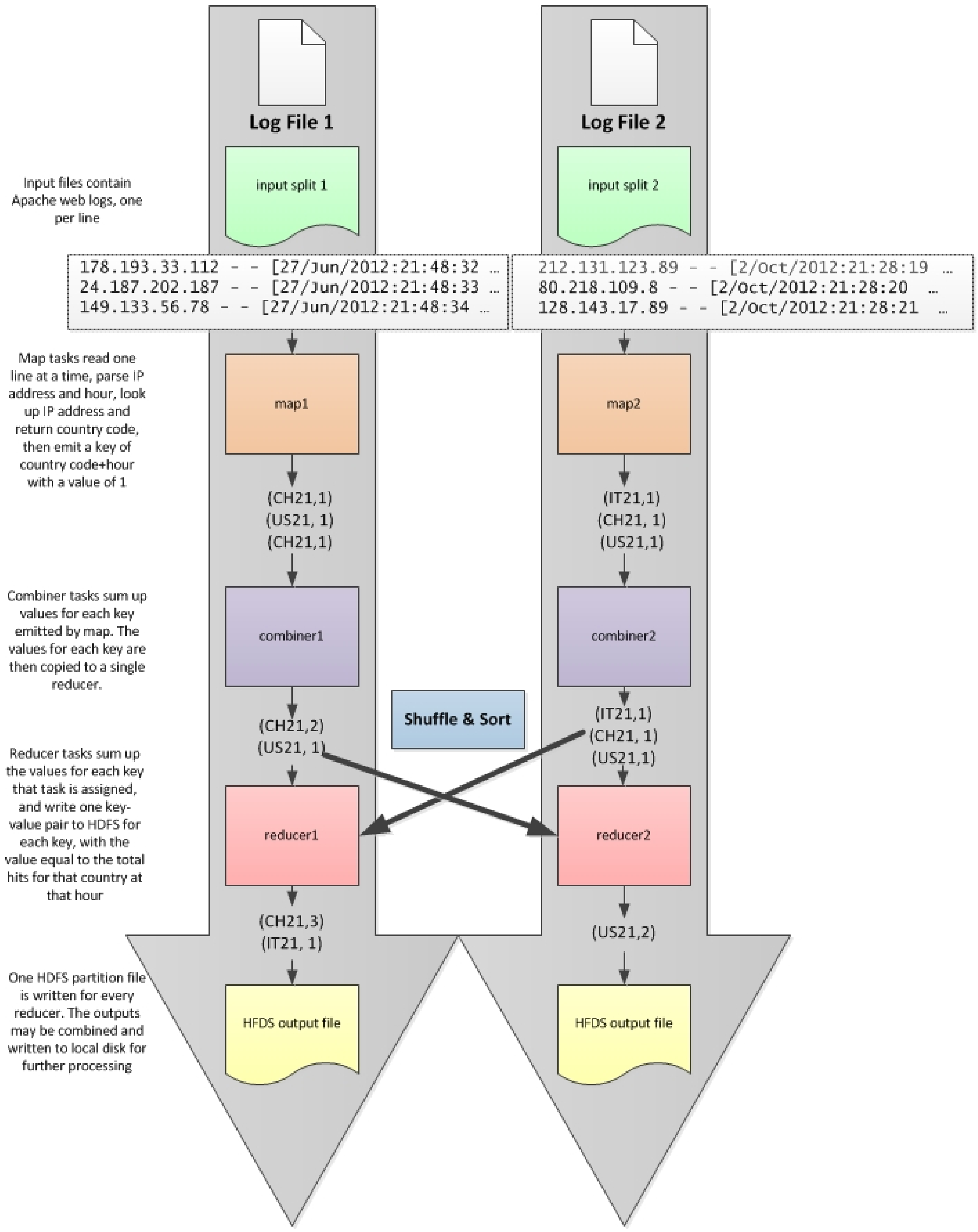
The table of Class B (first two octets) IP addresses was derived from GeoLite data created by MaxMind (http://www.maxmind.com). The GeoLite dataset of IP ranges and host country was unrolled and all Class B addresses used exclusively by a single country were listed along with the country code in a space-separated file, all_classbs.txt (see Table 3). This file is distributed to all mappers using the distributed cache feature of the MapReduce ToolRunner class. The file (first six lines) looks like:

```
1.3 CN
1.5 JP
1.6 IN
1.7 IN
1.8 CN
1.9 MY
```

The MapReduce program is implemented in three Java classes: the GeoWeb.java driver program, the GeoWebMapper.java mapper, and the SumReducer.java combiner and reducer (all code is included in the appendices).

The GeoWeb.java driver program is a standard ToolRunner-style MapReduce driver program. A Job object is created and then used to set input and output paths, as well as the Mapper, Combiner and Reducer classes. The final output key and value classes are set as Text and IntWritable, respectively. Finally, the ToolRunner run method is used to execute the job.

Figure 2. MapReduce Solution Flow Diagram (Simple Example)



Log File 1
Log File 2
Input files contain Apache web logs, one per line
input split 1
input split 2
Map tasks read one line at a time, parse IP address and hour, look up IP address and return country code, then emit a key of country code+hour with a value of 1
map1
map2
(CH21,1) (US21, 1) (CH21,1)
(IT21,1) (CH21, 1) (US21,1)
Combiner tasks sum up values for each key emitted by map. The values for each key are then copied to a single reducer.
combiner1
combiner2
Shuffle & Sort
(CH21,2) (US21, 1)
(IT21,1) (CH21, 1) (US21,1)
Reducer tasks sum up the values for each key that task is assigned, and write one key-value pair to HDFS for each key, with the value equal to the total hits for that country at that hour
reducer1
reducer2
(CH21,3) (IT21, 1)
(US21,2)
One HDFS partition file is written for every reducer. The outputs may be combined and written to local disk for further processing
HFDS output file
HFDS output file

```
178.193.33.112 - - [27/Jun/2012:21:48:32 …
24.187.202.187 - - [27/Jun/2012:21:48:33 …
149.133.56.78 - - [27/Jun/2012:21:48:34 …
```

```
212.131.123.89 - - [2/Oct/2012:21:28:19 …
80.218.109.8 - - [2/Oct/2012:21:28:20  …
128.143.17.89 - - [2/Oct/2012:21:28:21  …
```

Most of the action occurs in the GeoWebMapper.java mapper class. First a hash map, ip_locs, is created to contain the IP addresses and their associated country codes. Next, static objects for the output key and value are created to avoid the overhead of creating new objects for each emitted key-value pair. The code to read the all_classbs.txt file into the hash map is placed in the setup method so that this read will occur just once for each mapper created. The MapReduce framework hands out single web log lines to the map tasks in the form of key-value pairs where the key (not used) is the offset of that line in the file and the value is the line text. In the map task this text is parsed, with the first two octets of the remote IP address used to look up the corresponding country using the hash map's get method. Finally the country code and hour are joined in a string and emitted as the key along with value 1. Note that no assumptions can be made about the accuracy of the log format so that each step of the line parsing is checked. Any malformed log entry is simply discarded.

The combiner and reducer both use the SumReducer.java class. This is standard MapReduce code to add up all the values for a given key. When used in the Combiner, this adds all the 1 values emitted for each key emitted by a given mapper, generating a total number of hits per country and hour for the log files consumed by that mapper. The keys from each mapper are then shuffled and sorted and sent to the reducers, where the overall totals for each country-hour key are added and then written to HDFS. The use of the combiner to aggregate values for each key at the mapper significantly cuts down on network traffic during the shuffle and sort phase.

The three Java classes are compiled into a jar file, GeoWeb.jar. The first argument is the location in HDFS of the log files to be analyzed. The second argument is the location in HDFS where the final country code-hour/number of hits key-value pairs are to be written (this directory must not exist prior to running the program). The –files option is used to load the all_classbs.txt file into the distributed cache (if the file is not local then the full path must be included) and the –D option is used to set execution parameters such as number of reduce tasks. In these tests the following command was used to analyze 1TB of web log files previously generated by the GeoWeb Apache Log Generator program:

```
hadoop jar GeoWeb.jar GeoWeb -files all_classbs.txt -D \
  mapred.reduce.tasks=200 /user/test/weblogs/access_logs_1TB  \
  /user/test/weblogs/w1TB_mr_out
```

Since the synthetic web logs created for this test represent just the top 20 internet-using countries the output consists of 480 keys (20 countries x 24 hours), each associated with a value representing the total number of hits from that country during that hour. Note that since the same IP address data was used to generate the remote IP addresses as was used to analyze them, 100% of the log entries were successfully matched to a country in this test but in general the coverage will be lower.

The HDFS output may be collected, sorted, and stored in a file for further processing with:

```
hadoop fs -cat /user/test/weblogs/w1TB_mr_out/part* | sort > r_1TB_mr
```

In this test, the output file will have 480 lines of the form

```
BR00   2184017
BR01   2177154
etc.
```

with each line representing the country of origin and time of the web accesses, in this case hits coming from Brazil at midnight and 1:00 AM Central US time.

4 The Hive Solution

The Apache Hive tool (http://hive.apache.org), part of the Hadoop ecosystem, projects structure onto data stored in HDFS and provides HiveQL (HQL), a Structured Query Language (SQL) interface, to query that data. It is well suited for data analysts coming from a database background with SQL skills, who view data in terms of tables and joins. Hive creates a query plan that implements the HQL in a series of MapReduce programs, generates the code for these programs, and then executes the code.

The first step in a Hive program (see Appendix 4 for complete code) is to define the HDFS data in terms of SQL tables. For the GeoWeb problem both the web logs as well as the file mapping IP addresses to countries, all_classbs.txt, are turned into HQL tables (the all_classbs.txt file must be copied to HDFS first). The tables are created with the EXTERNAL keyword to point to the location in HDFS of the data, rather than using the Hive default location. For the web logs a serde (serializer-deserializer) provided by Hive, RegexSerDe is used to parse the data. This function requires the specification of an additional jar included in the Hive distribution, hive-contrib, using the ADD JAR command. Note that imparting the table structure on the HDFS data does not cause another copy of the data to be generated.

Once the data tables have been defined, the rest of the work is straightforward: the web log data is read in and parsed using the RegexSerDe and a temporary table containing the first two octets of the remote IP address and the hour of each web access is created. That temporary table is then joined to the table of Class B IP addresses to generate the country code for each row. The results are grouped by country code and hour, and the count of each combination is printed out.

To execute the code contained in file geoweb.q, the following command is run:

```
hive –f geoweb.q > hive1TB.out
```

Since the final SELECT statement of Hive outputs tab-separated values containing the country, hour and count, it is necessary to delete the first tab to put the data in a form similar to the output of the MapReduce program:

```
sed 's/\t//' hive1TB.out | sort > r_1TB_hive
```

When run against the same 1TB dataset of Apache web logs as used in the MapReduce program, the results are identical.

5 The Pig Solution

Apache Pig (http://pig.apache.org), is another data analysis tool in the Hadoop ecosystem. Pig provides a data flow language, Pig Latin, which enables the user to specify reads, joins, and other computations without the need to write a complete MapReduce program. Like Hive, Pig generates a sequence of MapReduce programs to implement the data analysis steps.

As in our Hive solution, our Pig solution (see Appendix 5) uses web log parsing functionality that must be loaded from a jar provided with the distribution, piggybank.jar. The web logs are loaded into a Pig data bag or relation called weblogs and Class B IP address data is loaded into one called classbs. At this point the solution works much like the other two: the web logs are parsed to pull out the first two octets of the remote IP address and the hour from the timestamp using the FOREACH weblog GENERATE command. The two items, or tuple, is saved in a data bag labeled "A". This relation is then joined with the classbs IP address information and stored in B. B is then grouped by the country code, hour tuple and the number in each group counted using another FOREACH/GENERATE command. The result is ordered by country code and hour and then stored back into HDFS.

Pig compiles the dataflow into MapReduce, resulting in a multi-pass MapReduce program. As shown in the code, for each of the key steps – JOIN, GROUP, ORDER – a PARALLEL option may be specified as a hint to MapReduce to determine the number of map or reduce tasks to deploy for that step.

To execute the code:

```
pig geoweb.pig
```

Since the output consists of tab-separated fields for the three items (country, hour and count), removing the first tab with sed will yield a result identical to that obtained from the MapReduce and Hive solutions:

```
hadoop fs -cat /user/test/weblogs/w1TB_pig_out/part* | sed 's/\t//' |
  sort > r_1TB_pig
```

The output from the Pig program is identical to those from MapReduce and Hive.

6 Results

The three solutions were tested on a 20-datanode Hadoop cluster in Dell's Round Rock, Texas, Solution Center using the Intel Distribution for Hadoop (IDH) version 2.4.1. The IDH namenodes and edgenode ran on PowerEdge R720 servers with dual 2.0 GHz 8-core Intel E5-2650 processors, 128 GB of memory and six 600 GB SAS disks in a RAID 10 configuration. The 20 datanodes were PowerEdge R720XD servers with the same processors, 64 GB of memory and 24 500 GB SATA disks, each in a RAID0 configuration. The total raw disk space on the cluster was over 200TB. The systems were connected through Dell Force10 S60 switches and 1GbE network interface cards. The implementation of IDH on Dell PowerEdge servers including generalized cluster configuration parameters is described in http://en.community.dell.com/techcenter/extras/m/white_papers/20412222.aspx.

A set of 366 Apache web log files, one for each day of 2012, was created by the GeoWeb Apache Weblog Generator tool and stored in HFDS. Each day consisted of 11,900,000 weblog entries. The total size occupied by the log files was 1TB. A second set of log files was created with 119,000,000 entries per day, for a total size of 10TB.

The MapReduce, Hive and Pig codes were run sequentially against the 1TB set of log files (see Table 3 for version information). The results, 480 lines representing the number of hits from each of the 20 countries modeled for each hour of the day, were identical for the three solutions. The total number of hits per country over the year were totaled and shown in Table 4 and Figure 3, along with the calculated percentage of overall use. That these percentages match the input distribution indicates that the log parsing and joining with the IP address table is working perfectly in all three solutions.

Table 3. Software Version Information

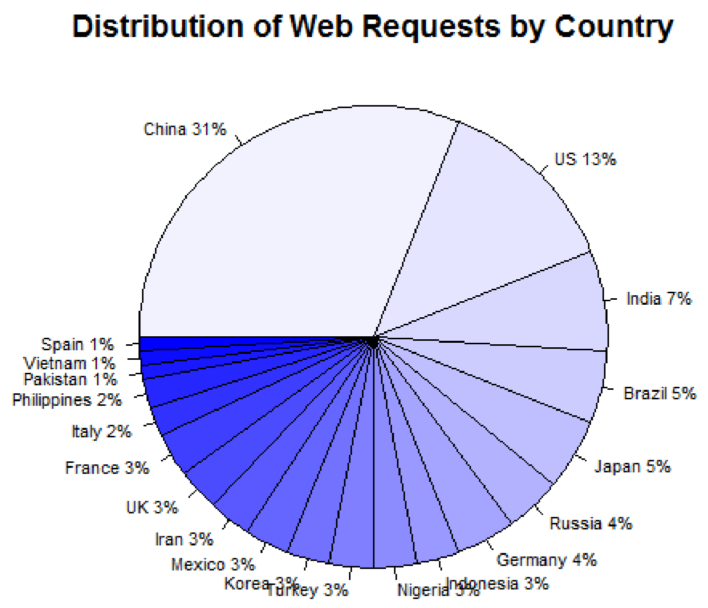
Hadoop	1.0.3
Hive	0.9.0
Pig	0.11.1

Table 4. Number of Hits by Country

Country	Number of Hits	%
China	1350141892	31
US	566091664	13
India	304873967	7
Brazil	217811610	5
Japan	217799723	5
Russia	174375623	4
Germany	174268918	4
Indonesia	130718174	3
Nigeria	130691403	3
Turkey	130645786	3

Country	Number of Hits	%
Korea	130638316	3
Mexico	130627379	3
Iran	130618729	3
UK	130607961	3
France	130586314	3
Italy	87128606	2
Philippines	87104881	2
Pakistan	43560124	1
Vietnam	43559447	1
Spain	43549483	1

Figure 3. Number of Hits by Country

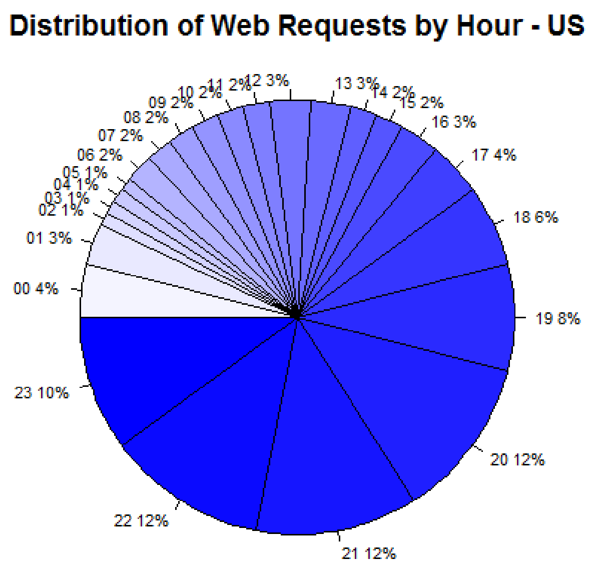


Next, looking at the total hits from a given country over the course of a 24-hour period, the sum over the year of web accesses per hour for the US is shown in Table 5 and Figure 4. Since the web log creator program assumed that the web site was a consumer-oriented enterprise located in the Central US time zone the hits are maximized in the evening hours (for simplicity the entire US is considered to be in the Central time zone). As in the case of the geographical distribution, the temporal distribution in the output results equals that used in the web log generator program, indicating that the parsing and processing of the time stamp in each web log is working correctly.

Table 5. Number of Hits per Hour for US

Hour	Number of Hits	%	Hour	Number of Hits	%
00	22635602	4	12	16989183	3
01	16990891	3	13	16990705	3
02	5661930	1	14	11329712	2
03	5663820	1	15	11331251	2
04	5652084	1	16	16965389	3
05	5646139	1	17	22641141	4
06	11330209	2	18	33946208	6
07	11329415	2	19	45278373	8
08	11321714	2	20	67946618	12
09	11314793	2	21	67942229	12
10	11311546	2	22	67949826	12
11	11332635	2	23	56590251	10

Figure 4. Number of Hits Per Hour for US



The three solutions were then run against the 10TB set of log files. The performance, in terms of elapsed time to analyze the log data is shown in Table 6. As might be expected the MapReduce program performs the best of the three solutions for both sets of log files since it is a single program explicitly written for the MapReduce framework. Hive and Pig, which generate multiple MapReduce programs to accomplish the same task, take longer. However, the difference narrows with the larger dataset size, indicating that the overhead of running multiple batch jobs in Hive and Pig becomes less of a factor for longer-running batch jobs. The scalability of all three solutions is excellent and is better for Hive and Pig than MapReduce. As shown in the last column it takes less than 10x the time to analyze a 10x larger dataset.

Table 6. Performance Results

Tool	Time to Analyze 1TB of web logs	Time Relative to MapReduce	Time to Analyze 10TB of web logs	Time Relative to MapReduce	Scaling 10TB vs 1TB workload
MapReduce	7 m 40 s	1x	69 m 54 s	1x	9.12x
Hive	9 m 34 s	1.25x	74 m 59 s	1.07x	7.4x
Pig	20 m 57 s	2.73x	183 m 54 s	2.63x	8.78x

These results illustrate the tradeoff between development time and execution time. Hive and Pig solutions are usually quicker to develop but take longer to run than MapReduce, with less of a disadvantage for larger workloads. All three approaches are excellent big data analysis solutions.

Appendix 1 GeoWeb.java

```
/*
GeoWeb.java: GeoWeb application driver

Last Updated 10/17/13

Dave Jaffe, Dell Solution Centers

Distributed under Creative Commons with Attribution by Dave Jaffe
(dave_jaffe@dell.com).  Provided as-is without any warranties or conditions.

Analyzes Apache web logs
* Determines remote location from IP address
* Reports number of clicks per country per hour

Syntax:
hadoop jar GeoWeb.jar GeoWeb -files all_classbs.txt <HDFS Apache web log
directory> <HDFS output directory>

Input: Apache web logs
Output: Number of clicks in logs per country per hour

File all_classbs.txt must exist in local directory
all_classbs.txt: a list of class B IP addresses for all countries from the GeoLite
dataset
  Only those class B's that come from a single country are used.
  Example line: 23.242 US

This product includes GeoLite data created by MaxMind, available from
http://www.maxmind.com

Example weblog line:
172.16.3.1 - - [27/Jun/2012:17:48:34 -0500] "GET /favicon.ico HTTP/1.1" 404 298 "-
" "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)"

Remote host/-/-/Date-timestamp/Request line/status/size/-/Referer/User agent

See http://httpd.apache.org/docs/current/logs.html
*/

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class GeoWeb extends Configured implements Tool
  {
  @Override
  public int run(String[] args) throws Exception
    {
    if (args.length != 2)
      {
      System.out.println("Usage: GeoWeb <input dir> <output dir>\n");
    return -1;
    }

  Job job = new Job(getConf());
  job.setJarByClass(GeoWeb.class);
  job.setJobName("GeoWeb");

  FileInputFormat.setInputPaths(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));

  job.setMapperClass(GeoWebMapper.class);
  job.setReducerClass(SumReducer.class);
  job.setCombinerClass(SumReducer.class);

  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);

  boolean success = job.waitForCompletion(true);
  return success ? 0 : 1;
  }

public static void main(String[] args) throws Exception
  {
  int exitCode = ToolRunner.run(new Configuration(), new GeoWeb(), args);
  System.exit(exitCode);
  }
} // End Class GeoWeb
```

Appendix 2 GeoWebMapper.java

```
/*
GeoWebMapper.java: mapper for GeoWeb application

Last Updated 10/17/13

Dave Jaffe, Dell Solution Centers

Distributed under Creative Commons with Attribution by Dave Jaffe
(dave_jaffe@dell.com).  Provided as-is without any warranties or conditions.

See documentation in GeoWeb.java driver program
*/

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashMap;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class GeoWebMapper extends Mapper<LongWritable, Text, Text, IntWritable>
  {
  HashMap<String,String> ip_locs = new HashMap<String, String>();
  private final static IntWritable one = new IntWritable(1);
  private Text textObject = new Text();

  @Override
  public void setup(Context context)
    throws IOException, InterruptedException
    // Read in file of class B IP addresses and corresponding country codes
    // (eg. 113.204 CN)
    // from distributed cache, store in hashmap
    {
    BufferedReader br;
    String classb_line;
    try
      {
      File f = new File("all_classbs.txt");
      br = new BufferedReader(new FileReader(f));
      while ((classb_line = br.readLine()) != null)
        {
        //System.out.println("classb line: " + classb_line);
        String[] fields = classb_line.toString().split(" ");
        if (fields.length == 2) ip_locs.put(fields[0], fields[1]);
        }
      }
    catch (IOException e)
      {e.printStackTrace();}
    //System.out.println("N= " + ip_locs.size());
    }

  @Override
  public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException
    {
    // Processes Apache logs of form:
    // 39.44.210.129 - - [01/Jan/2012:00:11:14 -0500] "GET /ds2/dsbrowse.php?\
    // browsetype=category HTTP/1.1" 200 2147 "http://72.8.133.189/"\
    // "Mozilla/5.0 (Windows; U; Windows NT 5.1; ja; rv:1.9.2.3) Firefox/3.6.3"

    String line = value.toString();
    String[] fields = line.split(" ");
    //if (fields.length <= 3) System.out.println(line);
    // Split log line by spaces, proceed if 4 or more fields found
    if (fields.length > 3)
      {
      String ip_address = fields[0];
      String time_stamp = fields[3];
      String[] octet = ip_address.split("\\.");
      if (octet.length > 1)
        {
        // Concatenate first two octets, use to look up ctry code in hashmap
        String ctry_code = ip_locs.get(octet[0] + "." + octet[1]);
        if (ctry_code != null)
          {
          // If timestamp is correctly formatted put out hour
          if (time_stamp.length() <15)
            {System.out.println("line= " + line + " time_stamp= " +
            time_stamp);return;}
          String hour = time_stamp.substring(13,15);
          if (hour != null)
            {
            // If everything is parsed correctly emit key=ctry_code+hour, value=1
            textObject.set(ctry_code + hour);
            context.write(textObject, one);
            } // End if (hour != null)
          } // End if (ctry_code != null)
        } // End if (octet.length > 1)
      } // End if (fields.length > 3)
    } // End Map
  } // End Class GeoWebMapper
```

Appendix 3 SumReducer.java

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

/*
SumReducer
*/
public class SumReducer extends Reducer<Text, IntWritable, Text, IntWritable>
  {
  @Override
  public void reduce(Text key, Iterable<IntWritable> values, Context context)
    throws IOException, InterruptedException
    {
    int wordCount = 0;
    for (IntWritable value : values)
      {
      wordCount += value.get();
      }
    context.write(key, new IntWritable(wordCount));
    }
  }
```

Appendix 4 The Hive Code – geoweb.q

```
-- geoweb.q: Hive commands to analyze Apache web logs for location and hour
-- Last Updated 11/1/13
-- Dave Jaffe, Dell Solution Centers
-- Distributed under Creative Commons with Attribution by Dave Jaffe
-- (dave_jaffe@dell.com).  Provided as-is without any warranties or conditions.

ADD JAR /usr/lib/hive/lib/hive-contrib-0.9.0-Intel.jar;

-- specify number of reduce tasks
SET mapred.reduce.tasks=200;

-- read web logs with Regex serde
-- (from https://cwiki.apache.org/confluence/display/Hive/
-- GettingStarted#GettingStarted-ApacheWeblogData)
DROP TABLE weblogs;
CREATE EXTERNAL TABLE weblogs (
  host STRING,   identity STRING,  user STRING,  time STRING,  request STRING,
 status STRING,   size STRING,  referer STRING,  agent STRING)
ROW FORMAT SERDE 'org.apache.hadoop.hive.contrib.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
  "input.regex" = "([^ ]*) ([^ ]*) ([^ ]*) (-|\\[[^\\]]*\\]) ([^ \
\"]*|\"[^\"]*\") (-|[0-9]*) (-|[0-9]*)(?: ([^ \"]*|\"[^\"]*\") ([^ \
\"]*|\"[^\"]*\"))?",
  "output.format.string" = "%1$s %2$s %3$s %4$s %5$s %6$s %7$s %8$s %9$s"
)
stored as textfile
-- Location of access logs to be analyzed
location '/user/test/weblogs/access_logs_1TB';

--read file containing all class B IP addresses and country (eg: 184.78 US)
DROP TABLE classbs;
CREATE EXTERNAL TABLE classbs (
  IP_B STRING,   ctry_code STRING)
row format delimited
fields terminated by ' '
stored as textfile
location '/user/test/weblogs/classbs';

-- Create temporary table with first two octets of host IP and hour
DROP TABLE t2;
CREATE TABLE t2 AS SELECT REGEXP_EXTRACT(host,"^([0-9]{1,3})\.([0-9]{1,3})",0) AS
host_ip_b, SUBSTRING(time,14,2) AS hour FROM weblogs;

-- Join temporary table to list of Class B addresses
SELECT classbs.ctry_code, t2.hour, COUNT(*) FROM t2 JOIN classbs ON (t2.host_ip_b
= classbs.ip_b) GROUP BY classbs.ctry_code, t2.hour;
```

Appendix 5 The Pig Code – geoweb.pig

```
-- geoweb.pig: Pig script to analyze Apache web logs, counting hits
-- by country and hour
-- Last Updated 11/1/13
-- Dave Jaffe, Dell Solution Centers
-- Distributed under Creative Commons with Attribution by Dave Jaffe
-- (dave_jaffe@dell.com).  Provided as-is without any warranties or conditions.


REGISTER /usr/lib/pig/lib/piggybank.jar

-- Load apache web logs from HDFS
weblog = LOAD '/user/test/weblogs/access_logs_1TB/*' USING \
org.apache.pig.piggybank.storage.apachelog.CombinedLogLoader AS (remoteAddr, \
remoteLogname, user, time, method, uri, proto, status, bytes, referer, userAgent);

-- Load file of Class B IP addresses - each line consists of IP address
-- and ctry, eg.  184.78 US
classbs = LOAD '/user/test/weblogs/classbs/all_classbs.txt' USING PigStorage(' ')
\ AS (ip_b:chararray, ctry_code:chararray);

-- Pull first two octets of IP address and hour from web log
A = FOREACH weblog GENERATE \
REGEX_EXTRACT((chararray)remoteAddr,'^([0-9]{1,3})\\.([0-9]{1,3})',0) \
AS host_ip_b, SUBSTRING((chararray) time,12,14) AS hour;

-- Join with table of IP addresses, group by country code and hour,
-- order and store result back to HDFS
B = JOIN A by host_ip_b, classbs by ip_b PARALLEL 200;
C = GROUP B BY (ctry_code, hour) PARALLEL 200;
D = FOREACH C GENERATE FLATTEN($0), COUNT($1) as count;
E = ORDER D BY ctry_code, hour PARALLEL 200;
STORE E into '/user/test/weblogs/w1TB_pig_out' USING PigStorage;
```

Appendix 6 The GeoWeb Apache Log Generator

The GeoWeb Apache Log Generator generates synthetic yet realistic Apache web logs with specified geographic and temporal distributions of web accesses. The program produces sequential web logs for a specified month, day, year and number of web accesses per day. Sessioning is simulated with the average number of clicks per user configurable in the code.

The program uses a table matching IP addresses to countries derived from GeoLite data from MaxMind (http://www.maxmind.com) to generate geographically realistic remote IP addresses in the logs. The GeoLite IP ranges were unrolled, and every Class B IP address (first two octets of IP address) that corresponds to a single country was included in the table. The remote IP addresses generated by the program are distributed geographically among the top 20 Internet-using countries according to their number of users. The distribution can be readily changed in the code if desired.

The Log Generator models a consumer or social web site operating in the US Central time zone so that usage from each country peaks during their local evening hours. For simplicity a single time zone is assigned to each country.

The web request (such as a GET command), referrer (referring web site if present), and user agent (web browser used) fields are randomly selected from input files containing lists of each item. The user can thus tailor the weblogs to their particular application. In the current code these fields are randomized but with some development the program could be used to generate meaningful log sequences for clickstream analysis.

Implemented as a Map-only MapReduce program, the program reads input files with lines containing the year, month, day and number of web accesses per day, and creates a file in the specified output HDFS directory with the name access_logs-yyyymmdd, containing that number of web accesses.

Because it is a MapReduce program the Log Generator is extremely scalable. The 1TB set of web logs used in this paper was generated in 10 minutes on the 20-datanode cluster described in the Results section.

To run the program, create the input files and copy to an HFDS directory, then place the Class B IP address file and the referrer, request and user agent files in the local directory and:

```
hadoop jar CreateWeblogs.jar CreateWeblogs –files \
  all_classbs.txt,referrers.txt,requests.txt,user_agents.txt \
  <HDFS input directory> <HDFS output directory>
```

Code is available from http://github.com/DaveJaffe/BigDataDemos.

Appendix 7 Analysis and Plotting with R

The R open source data analysis and plotting language (available from http://cran.r-project.org) was used to work up the data used in this paper.

The code to generate and plot the hits-by-country data:

```
> d<-read.delim("r_1TB_mr", header=FALSE, colClasses = "character")
> d3<-cbind(substr(d[,1], 1, 2), substr(d[,1], 3, 4), d[,2])   # Pull apart the
ctry code from hour
> m<-matrix(as.numeric(d3[,3]),24,20)
> rownames(m) <- d3[1:24,2]
> colnames(m) <- d3[1+24*0:19,1]
> format(sum(m), sci=F) # Overall total number of web accesses
[1] "4355400000"
> 11900000*366  # Equal to the number of accesses in the input access logs
[1] 4355400000
> pct<-sort((colSums(m)/sum(m)),decreasing=TRUE)   # Sum by country and sort
> pct<-round(100.0*pct)
> pct
CH US IN BR JP RU DE ID NG TR KR MX IR GB FR IT PH PK VN ES
31 13  7  5  5  4  4  3  3  3  3  3  3  3  3  2  2  1  1  1
> lbls<-
c("China","US","India","Brazil","Japan","Russia","Germany","Indonesia","Nigeria","
Turkey","Korea","Mexico","Iran","UK","France","Italy","Philippines","Pakistan","Vi
etnam","Spain")
> lbls <- paste(lbls, pct) # add percents to labels
> lbls <- paste(lbls,"%",sep="") # add % to labels
> pal <- colorRampPalette(c("white","blue"))
> pie(pct, lbls, main="Distribution of Web Requests by Country", radius=1,
col=pal(21)[2:21], cex=0.7, clockwise=TRUE, init.angle=180)
```

The code to generate and plot the hits-by-hour for the US:

```
> hour_pct<-round(100*m[,'US']/sum(m[,'US']))
> hour_lbls<-d3[1:24,2]
> hour_lbls <- paste(hour_lbls, hour_pct) # add percents to labels
> hour_lbls <- paste(hour_lbls,"%",sep="") # add % to labels
> pal <- colorRampPalette(c("white","blue"))
> pie(hour_pct, hour_lbls, main="Distribution of Web Requests by Hour - US",
radius=1, col=pal(25)[2:25], cex=0.7, clockwise=TRUE, init.angle=180)
```