Using the R Software for Log File Analysis

MIHALIS TSOUKALOS

Mihalis Tsoukalos is a UNIX system administrator who also knows programming, databases, and mathematics. You can reach him at http://www.mtsoukalos.eu/ and @mactsouk (Twitter). mactsouk@gmail.com

In this article, I am hoping I can inspire you to use R for analyzing log files. Although I selected Apache log files to analyze, you can get your data from anywhere you can imagine including Squid log files and uptime, sar, or vmstat output.

In case you are not familiar with R [1], it is a GNU project based on S, which is a statistics-specific language and environment developed at the famous AT&T Bell Labs. I am using R primarily because it can handle huge amounts of data. I am not so sure that Microsoft Excel, for example, could process 1 million lines of data.

You may find other software that can automatically analyze Apache log files and create reports, but it is easier to generate your own customized reports by using R. And, as you are going to see later, R can deal with log files pretty easily.

The first time you try a new R command, it is better to use the R interactive shell, but keep in mind that all R commands can be put into an R script and be executed from the UNIX shell or as a cron job.

I use RStudio [2] for interacting with the R shell but you can also run R on its own. Nevertheless, RStudio is free, and my advice is to use it as it simplifies your interaction with R.

Advantages

Although R is not easy to learn, it has many advantages, including the fact that it can be used in UNIX scripts, its large number of existing packages (CRAN) [3], its outstanding graphical capabilities, its ability to process lots of data, its advanced statistical capabilities, its ability to connect to a database, and that it is a powerful programming language.

You do not have to use everything at once, but having software with so many capabilities is very convenient.

The Problem

Processing log files that contain Web server data can be a very demanding job, so I wanted a solution that is powerful, customizable, efficient, and expandable. As I am already familiar with R and comfortable with statistics, I decided to use R to do the job.

The format of my Apache log file is called "myformat" and is defined inside Apache's main configuration file (/etc/apache2/apache2.conf):

```
$ grep myformat /etc/apache2/apache2.conf
LogFormat "%h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\" %D"        myformat
```

A sample entry looks as follows:

```
66.249.74.239 - - [09/Feb/2014:20:21:34 +0200] "GET  /ten-things-love-about-programming HTTP/1.1" 200
7588 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)" 1575345
```

Using the R Software for Log File Analysis

The "myformat" definition is a non-standard Apache LogFormat: it is the "combined" standard LogFormat with the addition of the %D field at the end. The %D field makes Apache record "the time taken to serve the request" in microseconds. If you are using a standard LogFormat, some of the presented examples will not work for you but you can still use another column like the eighth column that records the size of the request in bytes.

Getting the Data

Before being able to use the log file data, you must first import it into R. The good thing is that R can parse log files without requiring any additional work from you. Reading a log file named log.log is as simple as executing the following R command:

```
> LOGS = read.table("log.log", sep=" ", header=F)
```

In case your log file is not regular, you may get output that is similar to the following warning messages:

```
1: In scan(file, what, nmax, sep, dec, quote, skip, nlines, na.strings,  :
  EOF within quoted string
2:In scan(file, what, nmax, sep, dec, quote, skip, nlines, na.strings,  :
  number of items read is not a multiple of the number of columns
```

Apache log files are regular and always have the correct number of columns. If your log file is not regular, the `read.table()` function will not work; a Perl or Python script could solve such problems but you should have to decide what to do with the records with the missing fields.

I used a small log file (about 1500 lines) for the purposes of this article, but R is capable of processing huge log files without problems provided that you have a relatively modern computer.

After executing the `read.table()` command, the LOGS variable holds all data from the log.log file. The `head(LOGS)` command will show you the first few lines of the LOGS variable to get a better idea of how your data is stored by R. As there is no header line with the name of each column inside log.log, R automatically named columns as V1, V2, etc. If you want to process the data of the V10 column, you can use the LOGS$V10 variable.

I can see all the available column names by running the following command:

```
> names(LOGS)
 [1] "V1"  "V2"  "V3"  "V4"  "V5"  "V6"  "V7"  "V8"  "V9"  "V10" "V11"
```

Note that if your log file entries have a different format than mine, you may need to change some of the forthcoming R commands to match your format.

The `attach()` command creates new variables after the names of the columns of the LOGS variable, and it is used for convenience.

```
> attach(LOGS)
```

From now on, I can use V1 instead of LOGS$V1 to refer to the first column of the LOGS variable, V2 for the second column, etc.

The most important thing to consider when trying to analyze data is deciding which data you are going to analyze. I will use the following sets:

1. Column V11, which is the last field of each line in the log file.
2. Column V7, which is the Apache status code.
3. Column V4, which contains both the date and the time of each request. It will be used for counting the total number of requests per week day and per hour.
4. Columns V7, V8 (the size in bytes), and V11 will be used for having a general overview of the Web site traffic.

Analyzing the Data

Getting the data into R is not that difficult, but analyzing the data is a totally different story! This section will show some basic yet powerful R commands.

The single most useful command you can run on a data set with numeric values is the `summary()` command. The following statistical definitions will help you better understand the output of the `summary()` command:

- Min.: This is the minimum value of the whole data set.

- Median: It is an element that divides the data set into two subsets (left subset and right subset) with the same number of elements. If the data set has an odd number of elements, then the Median is part of the data set. On the other side, if the data set has an even number of elements, then the Median is the mean value of the two center elements of the data set.

- 1st Qu.: The 1st Quartile (q1) is a value that does not necessarily belong to the data set, with the property that at most 25% of the data set values are smaller than q1 and at most 75% of the data set values are bigger than q1. You can also consider it as the Median value of the left half subset of the sorted data set.

- Mean: This is the mean value of the data set (the sum of all values divided by the number of the items in the data set).

- 3rd Qu.: The 3rd Quartile (q3) is a value, not necessarily belonging to the data set, with the property that at most 75% of the data set values are smaller than q3 and at most 25% of the data set values are bigger than q3. Similarly to the 1st Quartile, you can consider the 3rd Quartile as the Median of the right half subset of the sorted data set.

- Max.: This is the maximum value found in the data set.

Note that there are many ways to determine Quartiles, so if you try another statistical package, you may get slightly different results.

I definitely wanted statistics about response time values, so I ran the `summary()` command on the V11 column:

```
> summary(V11)
   Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
    474       787     12760    507800    905200   9101000
```

The output shows that the 25% of requests are served extremely fast (the value of the 1st Quartile is very small). The other good thing is that the value of the Median is also small. It does make sense for some requests to take longer since the Web site serves images and also has an Administrator Interface which asks the MySQL database to run complex queries that take longer to execute than regular user visits.

I also wanted to know the total number of status codes, so I ran the following command using the V7 column:

```
> table(LOGS[,7])

  200   302   304   403   404   500
  943     2   162    71    12     8
```

The output shows that most requests are served without problems so, generally speaking, the Web site works well. Having a large number of 404 and 403 status codes is not a good thing.

Visualizing the Data

I ran the following command which creates a very simple bar plot for the status code data (Figure 1):

```
> barplot(table(LOGS[,7]))
```

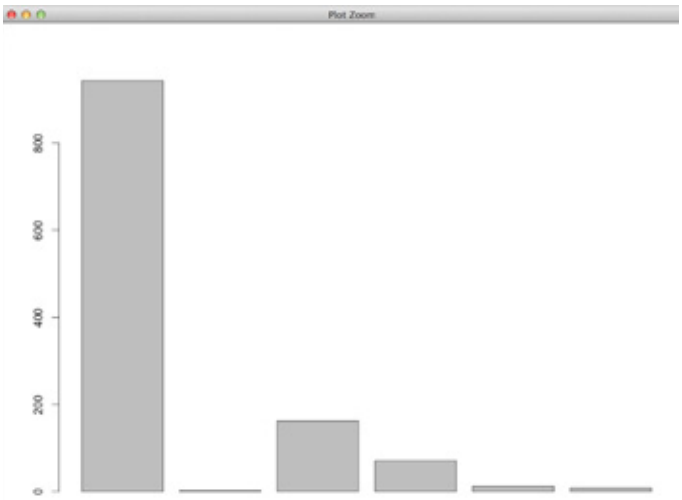

Figure 1: Creating a bar plot for the status code data

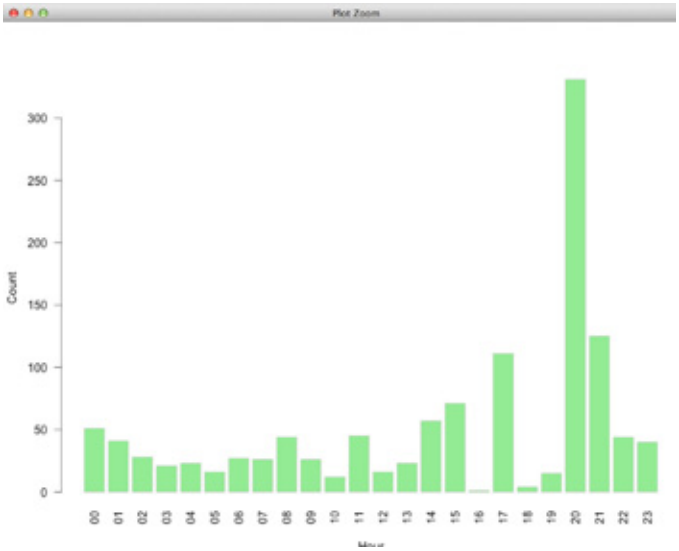

Figure 2: Traffic per hour

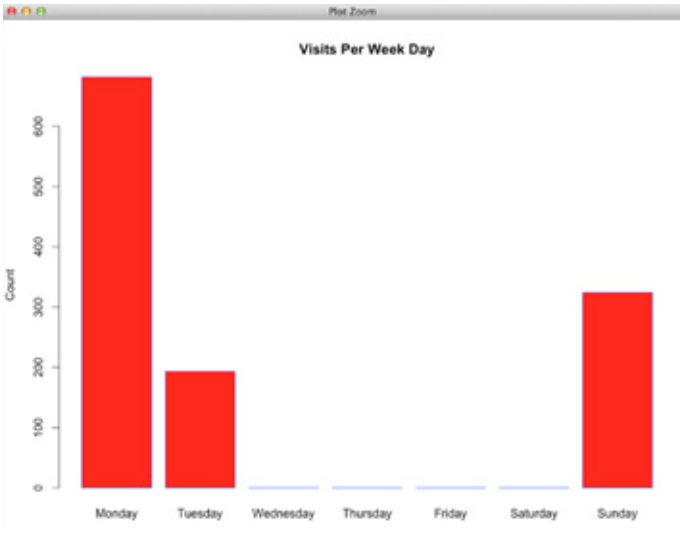

Figure 3: Traffic per weekday

If you want R to save the bar plot to an image which is 1024 x 1024 pixels, you should run the following R commands instead:

```
> png('test.png', width=1024, height=1024)
> barplot(table(LOGS[,7]))
> dev.off()
```

Next, I wanted to visualize the number of requests per week day and per hour of the day. Getting the date and time from the V4 column is a little tricky but it can be done using the following command:

```
> newV4 <- strptime(V4 , format('[%d/%b/%Y:%H:%M:%S'))
```

Now, I could use the new variable (newV4) to create plots that showed the traffic per hour (Figure 2) and per weekday (Figure 3) as follows:

Using the R Software for Log File Analysis

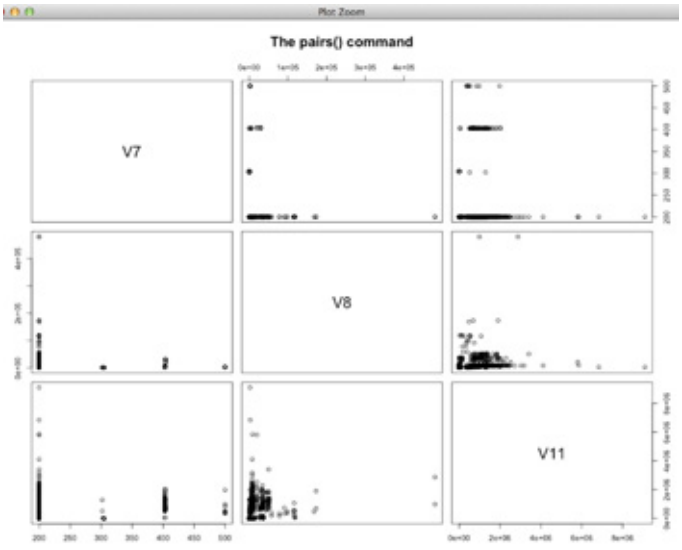

Figure 4: The pairs() command

```
> day = format(newV4, "%A")
> head(day)
[1] "Sunday" "Sunday" "Sunday" "Sunday" "Sunday" "Sunday"
> hours = format(newV4, "%H")
> head(hours)
[1] "19" "19" "19" "19" "19" "19"
barplot( table(factor(day, levels=c("Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday", "Sunday"))), xlab="Day", ylab="Count",
col="red", border="blue", main='Visits Per Weekday')
> barplot(table(hours), xlab="Hour", ylab="Count", col="lightgreen",
las=2, border="gray")
```

The `pairs()` command is especially useful since it gives a general overview of the data. I will only use three columns, but you can use as many columns as you want. Figure 4 is generated using the following commands.

```
> tempLOGS <- LOGS
> tempLOGS[1:6] <- list(NULL)
> names(tempLOGS)
[1] "V7"  "V8"  "V9"  "V10" "V11"
> tempLOGS$V9 <- NULL
> tempLOGS$V10 <- NULL
> names(tempLOGS)
[1] "V7"  "V8"  "V11"
> pairs(tempLOGS, main="The pairs() command")
```

The `tempLOGS <- LOGS` command creates a copy of the LOGS variable into the tempLOGS variable. The `tempLOGS$V9 <- NULL` command empties the V9 column of the tempLOGS table, which practically erases the V9 column of the tempLOGS variable. The `tempLOGS[1:6] <- list(NULL)` command empties the first six columns of the table, which practically deletes them from the tempLOGS variable.

Outliers in the graphical output is a sign of unusual or hostile behavior and should be further examined.

Using R for Detecting Security Threats

I can also check for security threats using the R language. As my log file is from a Drupal site, I want to monitor the "GET /?q=node/add HTTP/1.1", "GET /?q=user/register HTTP/1.1", "GET /?q=node/add HTTP/1.0", and "GET /?q=user/register HTTP/1.0" requests that indicate hack attempts. To do so, I ran the following R commands that use the V4 and V6 columns:

```
> HACK = subset(LOGS, V6 %in% c("GET /?q=node/add HTTP/1.1", "GET
/?q=user/register HTTP/1.1", "GET /?q=node/add HTTP/1.0", "GET
/?q=user/register HTTP/1.0" ))
> names(HACK)
[1] "V1"  "V2"  "V3"  "V4"  "V5"  "V6"  "V7"  "V8"  "V9"  "V10" "V11"
> HACK[1:3] <- list(NULL)
> names(HACK)
[1] "V4"  "V5"  "V6"  "V7"  "V8"  "V9"  "V10" "V11"
> HACK$V5 <- NULL
> names(HACK)
[1] "V4"  "V6"  "V7"  "V8"  "V9"  "V10" "V11"
> HACK[3:5] <- list(NULL)
> names(HACK)
[1] "V4"  "V6"  "V10" "V11"
> HACK[3:4] <- list(NULL)
> names(HACK)
[1] "V4" "V6"
> head(HACK)
                    V4                          V6
253 [09/Feb/2014:20:47:47     GET /?q=node/add HTTP/1.0
254 [09/Feb/2014:20:47:48 GET /?q=user/register HTTP/1.0
257 [09/Feb/2014:20:50:44     GET /?q=node/add HTTP/1.0
258 [09/Feb/2014:20:50:50 GET /?q=user/register HTTP/1.0
271 [09/Feb/2014:21:41:00     GET /?q=node/add HTTP/1.0
272 [09/Feb/2014:21:41:01 GET /?q=user/register HTTP/1.0

> summary(HACK)
                 V4                          V6
 [10/Feb/2014:17:59:58: 2   GET /?q=node/add HTTP/1.0    :41
 [11/Feb/2014:04:16:12: 2   GET /?q=user/register HTTP/1.0:41
 [09/Feb/2014:20:47:47: 1   GET /?q=user/register HTTP/1.1:10
 [09/Feb/2014:20:47:48: 1   GET /?q=node/add HTTP/1.1    : 7
 [09/Feb/2014:20:50:44: 1   GET / HTTP/1.0               : 0
 [09/Feb/2014:20:50:50: 1   GET / HTTP/1.1               : 0
 (Other)           :91   (Other)                  : 0

> newV4 <- strptime(HACK$V4 , format('[%d/%b/%Y:%H:%M:%S'))
> day = format(newV4, "%A")
> barplot( table(factor(day, levels=c("Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday", "Sunday"))), xlab="Day", ylab="Count",
col="red", border="blue", main="Hack Attempts!")
```

Using the R Software for Log File Analysis

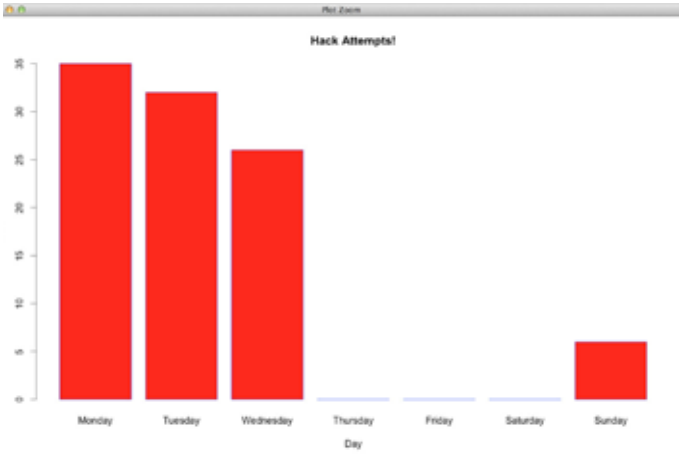

Figure 5: Counting hacking attempts

The produced bar plot can be seen in Figure 5.

Occasionally, if there are many hacking attempts, I may search a little more and find the IP—using column V1— that had the highest number of hacking attempts in order to add it to my firewall's deny list.

R offers many more packages that can help you analyze and visualize your data, but you have to decide what is best for your data and your needs and use it. Experimentation is a key part of the process that I also happen to find very interesting!

Conclusion

I think that you should definitely learn an advanced statistics package such as R because it will make you see your data in a totally different way. It will also help you create graphical output that is really unique and different from all those Excel charts.

I hope this article will be the beginning of your journey to R!

References

[1] R Project home page: http://www.r-project.org/.

[2] RStudio IDE: http://www.rstudio.com/.

[3] CRAN: http://cran.r-project.org/.