Microsoft SharePoint
Products and Technologies

Analyzing Microsoft SharePoint Products and Technologies Usage

Author:

Mike Wise

Date published:

January 2009

Summary:

This white paper will help administrators gather and analyze Microsoft SharePoint Products and Technologies usage and performance data.

Table of Contents

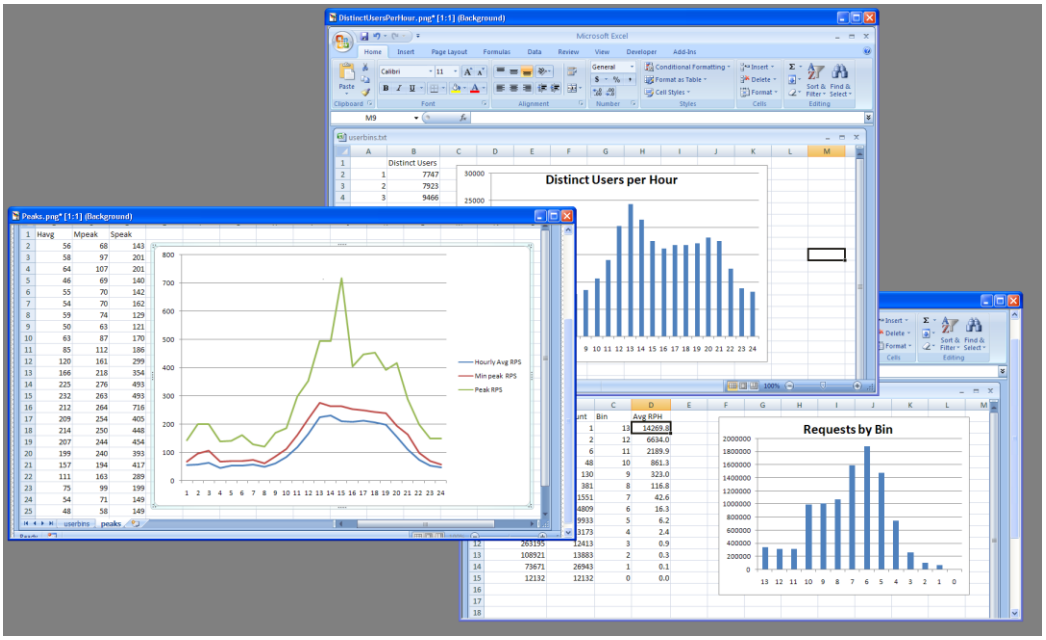
Introduction

Microsoft® SharePoint® Products and Technologies give an administrator few possibilities to look into what is actually happening on a farm; for example, how many users are active, how heavily they are using the system, what kind of requests are coming in, and from what kind of clients they originate. While this will be rectified in future versions, in fact much of this information is potentially available, but locked up in the IIS logs.

The question is, how do we extract this data? One of the easiest ways is to simply use Log Parser, an under-recognized but very powerful tool available free for download from Microsoft. The download link can be found at the end of this article. Be advised that Log Parser is not an officially supported product, thus any bugs or errors will not be handled by the Microsoft support channels. But this version has been around a while and it is quite stable.

Log Parser can read and write to a number of textual and binary formats, including all the IIS formats. In fact, it probably provides the best way to convert between these formats.

Log Parser allows the following types of data to be generated:



Installing and configuring Log Parser

Preparing the analysis machine

You probably will not want to install Log Parser on your server machine. Rather, you will want to install it on a client machine with adequate free space to hold the logs (calculate about 2 GB per server per day of log files you want to analyze).

Download Log Parser 2.2 and install it on your machine. See the References section at the end of this article for the link.

You may want to add the executable directory to your path environment variable to make Log Parser easier to use from the command prompt. The default path on a 64-bit client is C:\Program Files (x86)\Log Parser 2.2.

Preparing the server logs

Of course, the logs must be configured and collected, preferably from all active Web servers in the farm. While it is possible to analyze usage from a single Web server and draw conclusions about the entire farm, it can be somewhat misleading because it depends on how the load balancer is dividing up the requests. In fact, we can use the results of this analysis to draw conclusions about how “balanced” the load balancing actually is.

Log settings for IIS can be found in the IIS Log Manager running on your Web server. In this article, we assume that you have selected the W3C setting and have enabled at least the following fields:

Field	Column name
Date	date
Time	time
Client IP Address	c-ip
User Name	cs-username
Method	cs-method
URI Stem	cs-uri-stem
Protocol Status	sc-status
Protocol SubStatus	sc-substatus
Bytes Sent	sc-bytes
Bytes Received	cs-bytes
Time Taken	time-taken
User Agent	cs-user agent

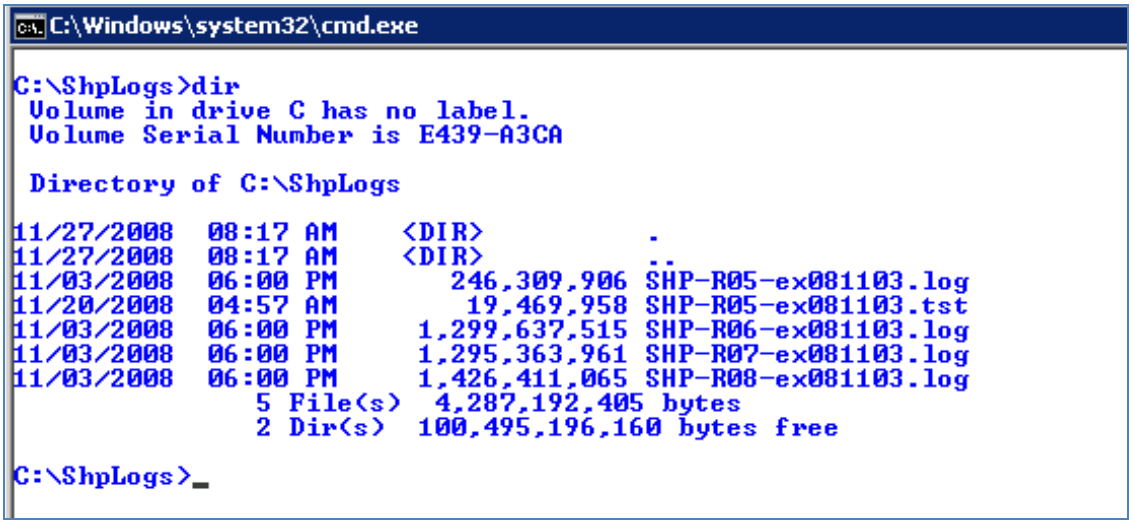
Of course, you might want to enable more of them. However, this will determine the size of the files that will take up disk space on the Web server, and impact processing time in the following. Large, heavily loaded Web servers can easily produce gigabytes of log files per day.

Once you have the logs, you will want to collect them together into a directory on a client computer somewhere for analysis. You will need to rename the files to reflect the name of the Web servers, because log files from the same date are likely to have the same name.

Log Parser queries

Enumerating records

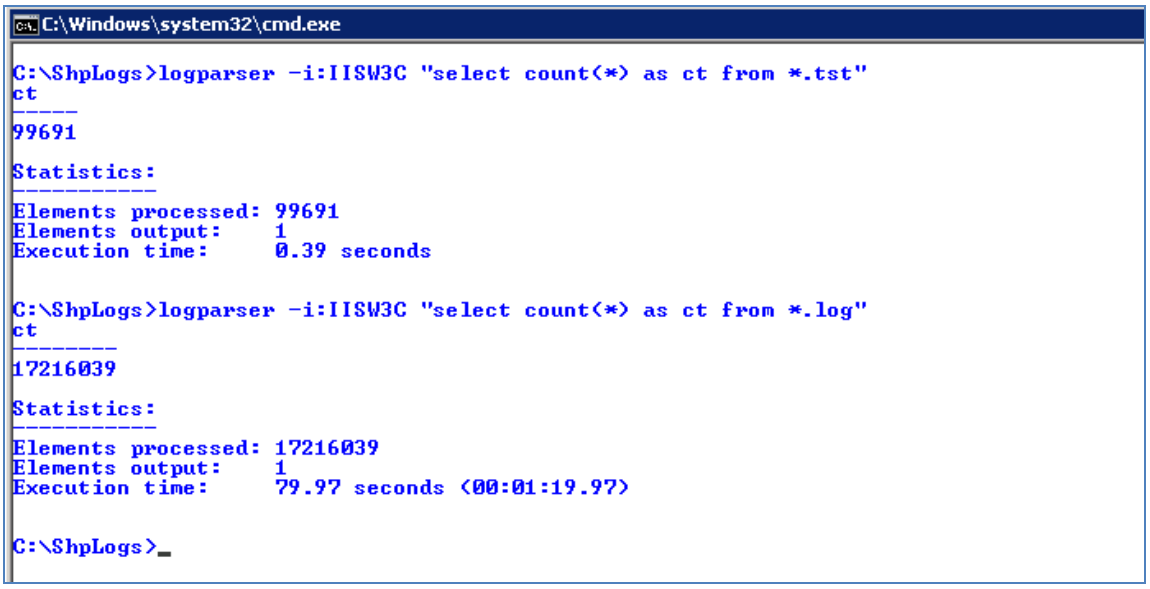
In the following example, we have renamed four log files\. We also created a .tst file (copying and deleting most of the lines using Notepad) out of a log file, and will use it to test our command before running said command on the all the large ones, which will take minutes to execute. Note that one of our log files is quite a bit smaller (245 MB instead of +1.2 GB) than the others. We don't know why this is yet, but we will use the following procedure to investigate.



So, let's get started by enumerating our records. First we try with the "tst" file, and when we see it works, we can scroll back with the up-arrow (command-line editing) and replace the "tst" with "log". We enter the command:

```
logparser -i:IISW3C "select count(*) as ct from *.log"
```

And we see this:



Note how much longer it takes, 79.9 seconds instead of less than 1. This is why we use the .tst file to try our commands out first and see if we like the results before proceeding to the .log files. However, from now on, we will only show the .log results.

So we see we have around 17 million records in one day. Hmm, seems like a lot. Let's see how successful they were by checking out the status codes.

```
logparser -i:IISW3C "select count(*) as ct,sc-status from *.log group by sc-
status"
```

This query yields the following (after we "pressed a key").

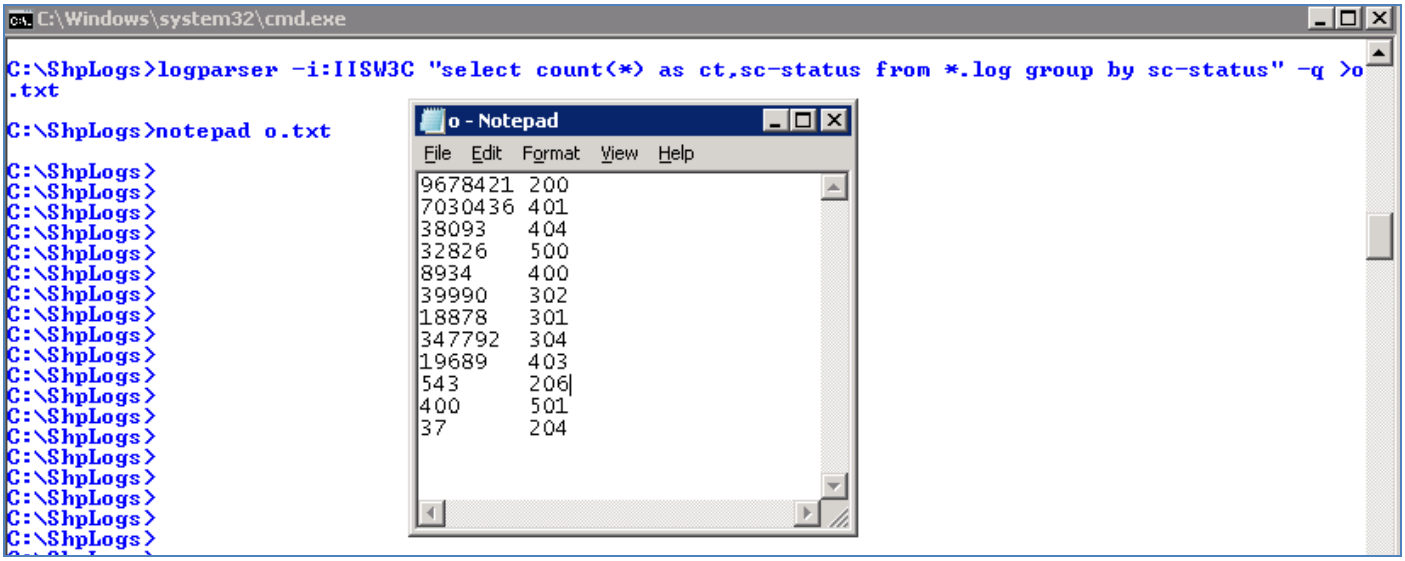
```
C:\Windows\system32\cmd.exe

C:\ShpLogs>logparser -i:IISW3C "select count(*) as ct,sc-status from *.log group by sc-status"
ct       sc-status
-------- ---------
9678421  200
7030436  401
38093    404
32826    500
8934     400
39990    302
18878    301
347792   304
19689    403
543      206
Press a key...
ct  sc-status
--- ---------
400 501
37  204

Statistics:
-----------
Elements processed: 17216039
Elements output:    12
Execution time:     142.43 seconds (00:02:22.43)
```

Took a lot longer this time (there was more to do). There is an **–q** option to make it output without a pause, but then you lose the headers too. An obvious ploy is to pipe the output to a file and then load it up into Notepad or Excel for further analysis, as in the following:

```
logparser -i:IISW3C "select count(*) as ct,sc-status from *.log group by sc-status" -q >o.txt
```



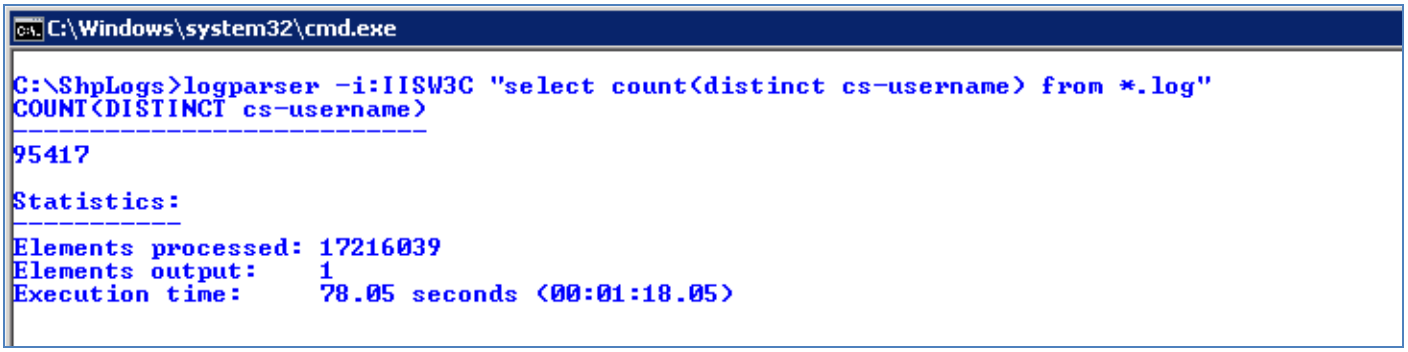
Now, back to the analysis. There are over seven million 401 authentication errors. Leaving aside the consideration as to whether or not this is excessive (it is a fact of life that SharePoint Products and Technologies do not keep a user cookie and re-authenticate every request), we see that essentially most requests are either 200s or 401s. There are some 304s (about 1.6 percent), but the relatively small number indicates that we are handling dependent (or static) requests rationally; our blob cache is working pretty well and client caching is working for the most part. If this number were larger, say 10 percent or above, then it would merit investigation.

Counting users

From now on, we will often not show screen shots of the command prompt (unless the result is short); we will just show the **logparser** command we are using and the results as a Notepad or Excel table.

Let's look at how many users we have logging in.

```
logparser -i:IISW3C "select count(distinct cs-username) from *.log" -q >o1.txt
```



95,417 different users used this server that day – quite a few, really.

Load balancing

Now we are curious as to how the load balancing is working and how user requests are distributed over the log files. For this we will need our first **two-stage query**. First we query by user and log file and store the results in a .csv file (it could be an IISW3C file too), and then we run a query on that file.

1. Run the following:

   ```
   logparser -i:IISW3C -o:CSV "select count(*) as ct,cs-username,logfilename
   from *.log group by cs-username,logfilename" >out.csv
   ```

   After about 80 seconds, this results in a 6.2 MB file in our directory. Now we query that to see how many users there are in each log file. This file has three columns: ct, cs-username, and logfilename. We now query it directly in order to discover how many unique users there are in each file.

2. Run the following:

   ```
   logparser -i:CSV "select sum(ct) as sum,count(*) as users,logfilename from
   out.csv group
   ```

   The output looks like:

   ```
   sum     users LogFilename
   ------- ----- --------------------------------
   1241755 5886  C:\ShpLogs\SHP-R05-ex081103.log
   4770368 37739 C:\ShpLogs\SHP-R06-ex081103.log
   5792908 38230 C:\ShpLogs\SHP-R07-ex081103.log
   5411008 38013 C:\ShpLogs\SHP-R08-ex081103.log
   ```

Because 5886+37739+38230+38013 = 119868, and this is bigger than the number of distinct users we saw before, we know that some users must be in more than one log. If you examine the out.csv file, you will note that the main user is the "-" user; however, this is mostly due to all the 401s, which we could eliminate by adding a **where** clause (**where sc-status<>401**).

The fact that some users are in more than one log file indicates that those users are being load balanced to more than one server. Which ones, and does it occur frequently? The following query will show us:

```
logparser -i:CSV "select sum(ct) as ct1,count(*) as logcount,cs-username from
out.csv group by cs-username order by logcount,ct1 desc"
```

Note the "**order by logcount,ct1 desc**". This will order the records so that the ones with the biggest **logcount** come first, with ct1 breaking the tie. And **desc** is short for "descending"; if it is omitted, the list will sort from small-to-large.

We don't output the users here in order to protect the user names. However, if we want to see how many there are in each class, we can do the following:

```
logparser -i:CSV -o:CSV "select sum(ct) as ct1,count(*) as logcount,cs-username
from out.csv group by cs-username order by logcount,ct1 desc" -q >cnt.csv
```

And then:

```
logparser -i:CSV "select count(*), logcount from cnt.csv group by logcount order
by logcount"
```

Output:

```
ct    logcount
----- --------
```

```
74228 1
18354 2
2412  3
424   4

Statistics:
-----------
Elements processed: 95418
Elements output:    4
Execution time:     0.12 seconds
```

The users who ended up on multiple Web servers are probably those whose IP address has changed during the day, maybe by changing computers, maybe due to a DHCP renewal. More interesting might be to investigate the file by IP address (using the **c-ip** field instead of the **cs-username**), but that is left as an exercise to the reader.

User type distribution

Returning to the .log files, we might now ask ourselves, “How were the top users distributed? “The following simple query answers that:

```
logparser -i:IISW3C "select top 20 count(*) as ct,cs-username as user from *.log
group by user order by ct desc"
```

Maybe we want to see what kinds of users we have (heavy, light, and so forth). Try this series of commands:

1. Get the users and their frequencies into a .csv file (note that we exclude the requests with 401 status codes here to eliminate the “dash” user with their seven million requests):

   ```
   logparser -i:IISW3C -o:CSV "select count(*) as ct,TO_INT(LOG(count(*))) as
   bin,cs-username from *.log where sc-status<>401 group by cs-username order
   by ct desc" -q >userfreq.csv
   ```

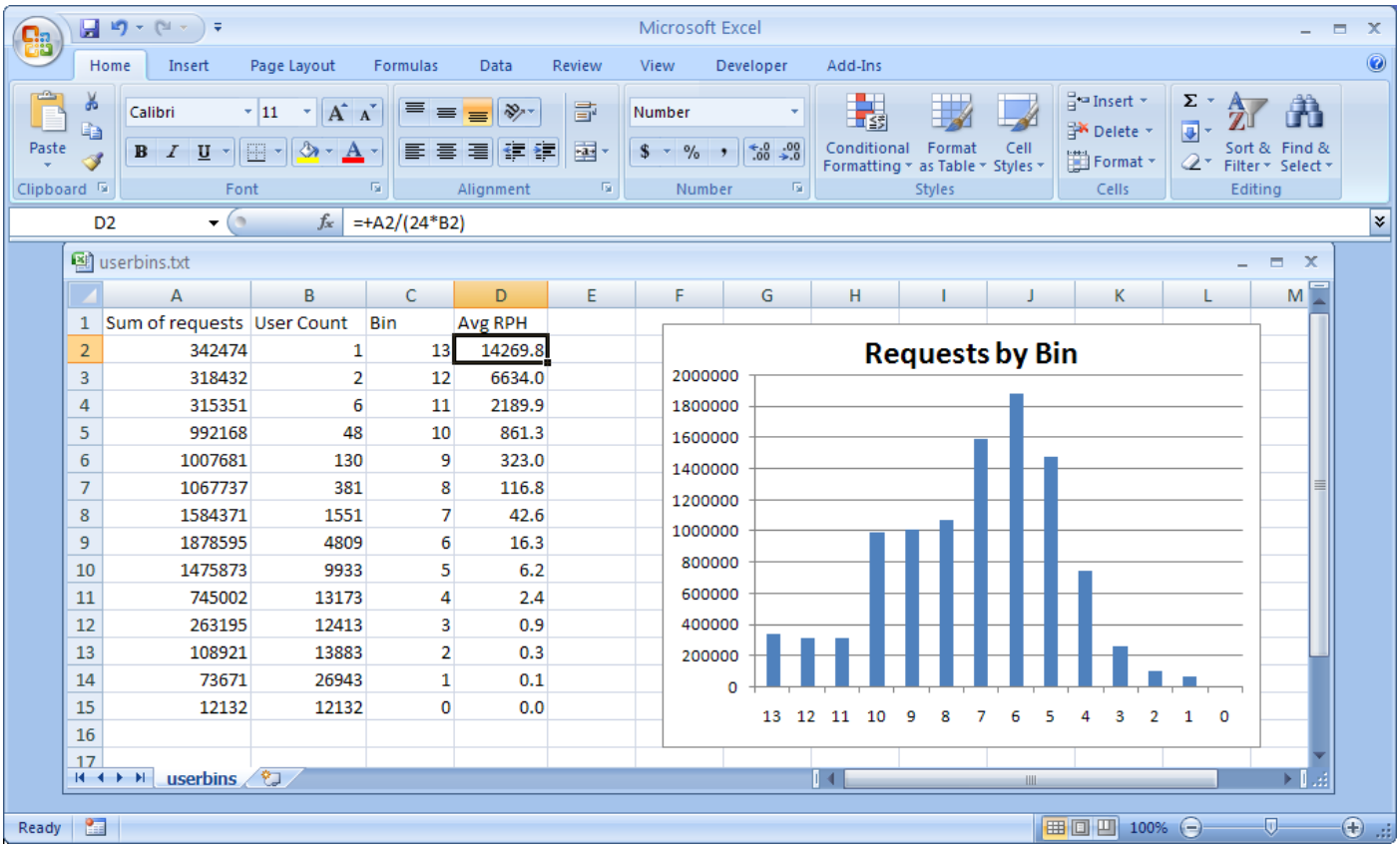
2. Now we can view them by bin. The following command:

   ```
   logparser -i:CSV "select sum(ct),count(*),bin from userfreq.csv group by
   bin order by bin desc" -q
   ```

   yields the following table:

```
342474    1       13
318432    2       12
315351    6       11
992168    48      10
1007681   130      9
1067737   381      8
1584371   1551     7
1878595   4809     6
1475873   9933     5
745002    13173    4
263195    12413    3
108921    13883    2
73671     26943    1
12132     12132    0
```

We import this into Excel, add a column for **Avg RPH** (by dividing the first column by 24 and then by the second column).



We see that our typical Extreme, Heavy, Typical and Light users correspond roughly to bins 6, 7, and 8. It seems that real user behavior seems to span a much wider range, namely buckets 10 to 4 inclusive, which ranges from about 2 to 800 RPH.

While this particular analysis includes dependent requests, we can do the same without them, the results are much the same, users show activity over a much wider range than the current server usage model indicates.

In this case, we “binned” (put the users into buckets) on the natural logarithm of the user's RPH. If you would rather bin on factors of two, then you will want to divide **LOG** by **LOG(2.0)**, as in:

```
TO_INT(DIV(LOG(count(*)),0.69315))
```

Similarly, if you want to bin on powers of 10, use **LOG(10.0)**, as in:

```
TO_INT(DIV(LOG(count(*)),2.3025))
```

Request (RPS) distribution over time

Now we would like to see how usage is varying by time. To do this comfortably, we really need to have the data in a more readable form, so now we combine all our data into one big .csv file. We also crack out the seconds, minutes and hours in a way that is more easily accessible. Because the query is long, we put it into a file (load.txt) and use a command-line parameter in Log Parser to access it:

Our query (in load.txt):

```
select EXTRACT_FILENAME(LogFilename),LogRow,

date, time, cs-method, cs-uri-stem, cs-username, c-ip, cs(User-Agent), cs-host,
sc-status, sc-substatus, sc-bytes, cs-bytes, time-taken,

add(
   add(
      mul(3600,to_int(to_string(to_localtime(to_timestamp(date,time)),'hh'))),
      mul(60,to_int(to_string(to_localtime(to_timestamp(date,time)),'mm')))
   ),
   to_int(to_string(to_localtime(to_timestamp(date,time)),'ss'))
) as secs,
```

```
to_int(to_string(to_localtime(to_timestamp(date,time)),'yy')) as yy,
to_int(to_string(to_localtime(to_timestamp(date,time)),'MM')) as mo,
to_int(to_string(to_localtime(to_timestamp(date,time)),'dd')) as dd,

to_int(to_string(to_localtime(to_timestamp(date,time)),'hh')) as hh,
to_int(to_string(to_localtime(to_timestamp(date,time)),'mm')) as mi,
to_int(to_string(to_localtime(to_timestamp(date,time)),'ss')) as ss,

to_lowercase(EXTRACT_PATH(cs-uri-stem)) as fpath,
to_lowercase(EXTRACT_FILENAME(cs-uri-stem)) as fname,
to_lowercase(EXTRACT_EXTENSION(cs-uri-stem)) as fext

from *.log

where sc-status<>401
```

And the LogParser invocation we use on it:

```
logparser -i:IISW3C file:load.txt -o:csv -q >bigo.csv
```

After a few minutes this yields a 3.5 GB file, “bigo.csv”. We will use this file in the future for a number of things, but first we reduce it further to a distribution by seconds:

```
logparser -i:CSV –o:CSV "select count(*) as ct,secs,max(ss) as ss,max(mi) as
mi,max(hh) as hh from bigo.csv group by secs order by secs"  -q >secsdist.csv

logparser -i:CSV -o:CSV "select count(*) as ct,div(secs,60) as minu,max(ss) as
ss,max(mi) as mi,max(hh) as hh from bigo.csv group by minu order by minu"  -q
>mindist.csv
```

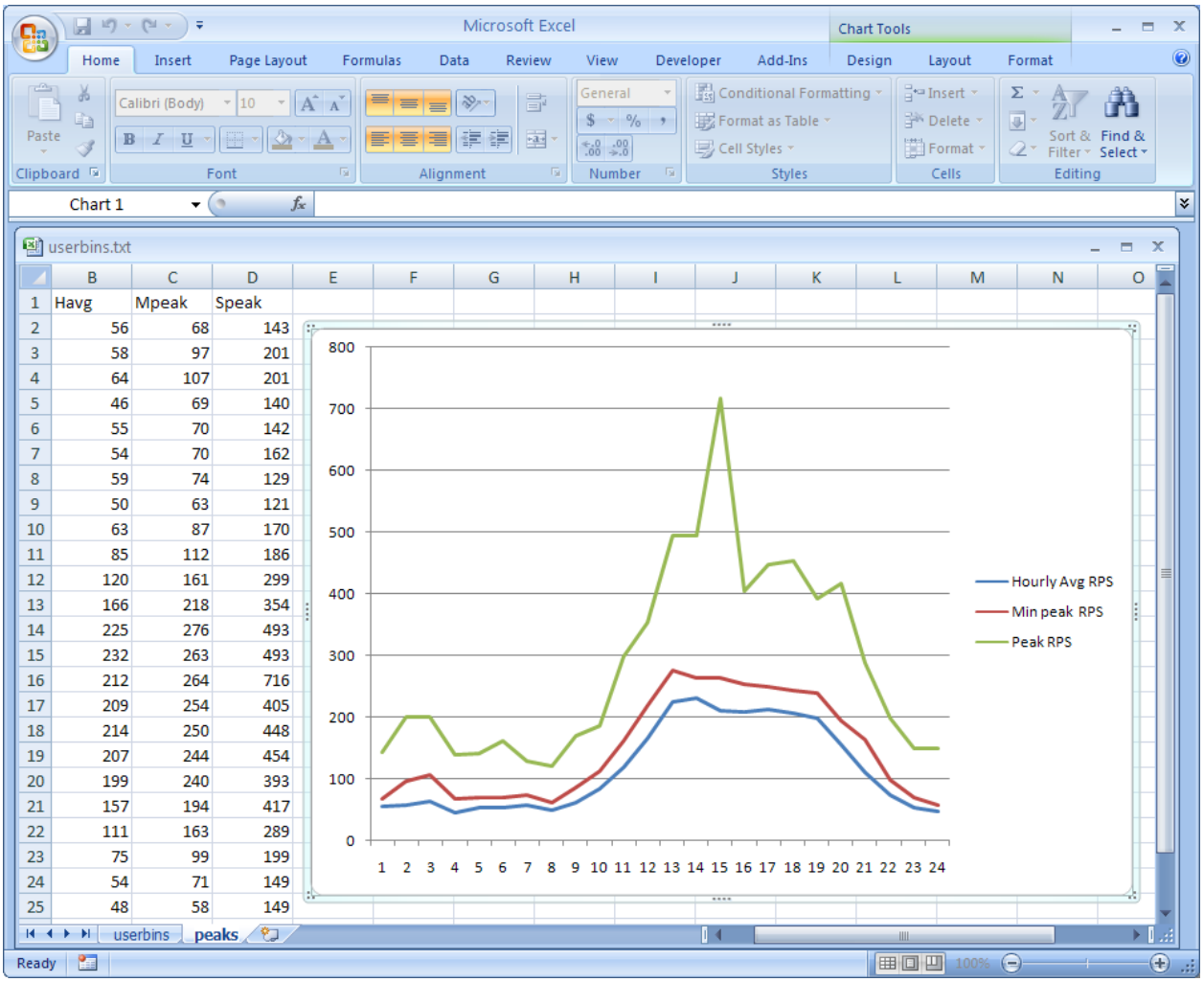
Now it is a simple matter to extract the peaks we want. First we want the hourly average RPS:

```
logparser –i:CSV "select hh,avg(ct) from mindist.csv group by hh order by hh"

logparser –i:CSV "select hh,max(div(ct,60)) from mindist.csv group by hh order by
hh"

logparser –i:CSV "select hh,max(ct) from secdist.csv group by hh order by hh"
```

Collecting these results together and plotting them in Excel yields:



Note that if we wanted to have only user operations (a measure used in planning that normally only includes user-initiated requests), then we would want to filter out all the static and dependent requests first (.gif, .png, .bmp, .js, .css, and .axd). The easiest way to do this would be to change the **where** clause to:

```
….
where sc-status<>401 and fext<>'gif' and fext<>'png' and fext<>'bmp'
                                          and fext<>'js' and fext<>'css' and
fext<>'axd'
```

However, below we show that the ratio in this case is about 3-1 for browsers, so we can just divide the above numbers by three for an approximate value.

Distinct users over time

We start out with a new .csv file, one containing the user distribution:

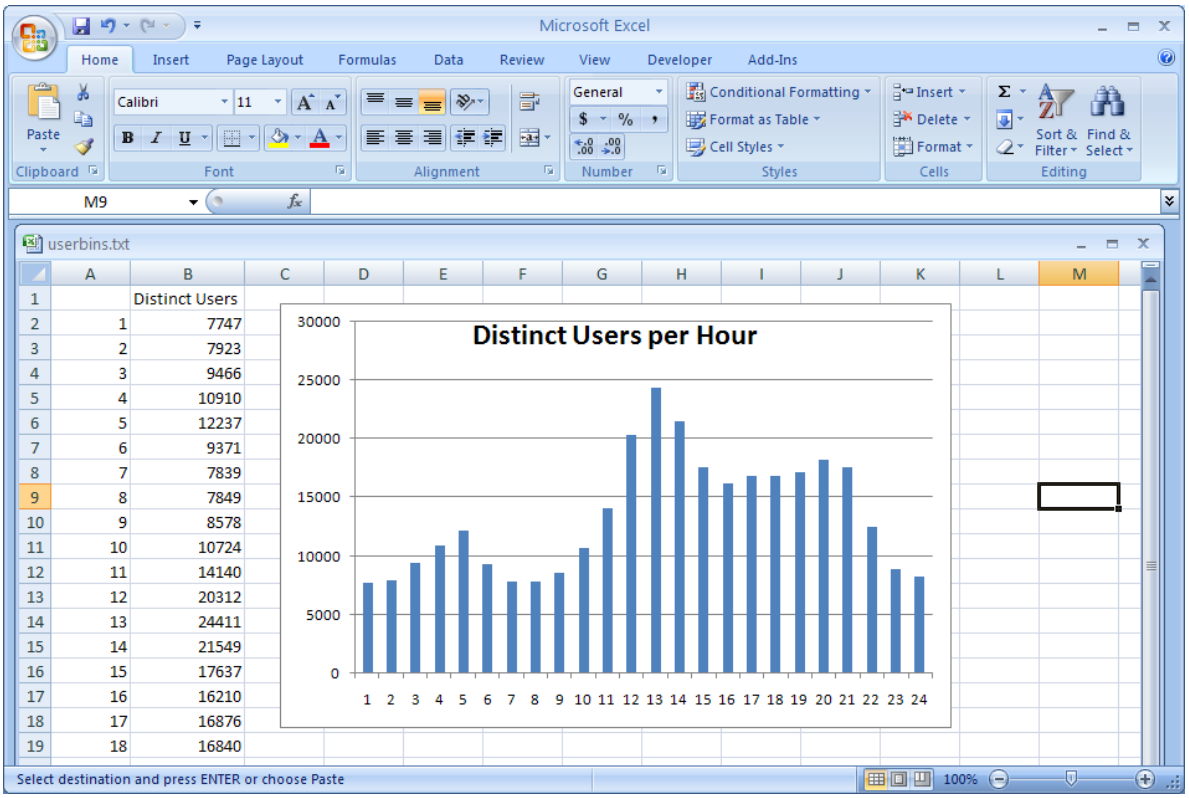
```
logparser -i:CSV -o:CSV "select count(*) as ct,cs-username,secs,max(ss) as
ss,max(mi) as mi,max(hh) as hh from bigo.csv group by secs,cs-username order by
secs,cs-username"  -q >userdist.csv
```

Then we consolidate by hour:

```
logparser -i:CSV -o:CSV "select hh,cs-username,sum(ct) as req from userdist.csv
group by hh,cs-username order by hh,cs-username"

logparser -i:CSV "select hh,count(*) from userhhdist.csv group by hh" -q
```

This reports distinct users by hour:



	A	B
1		Distinct Users
2	1	7747
3	2	7923
4	3	9466
5	4	10910
6	5	12237
7	6	9371
8	7	7839
9	8	7849
10	9	8578
11	10	10724
12	11	14140
13	12	20312
14	13	24411
15	14	21549
16	15	17637
17	16	16210
18	17	16876
19	18	16840

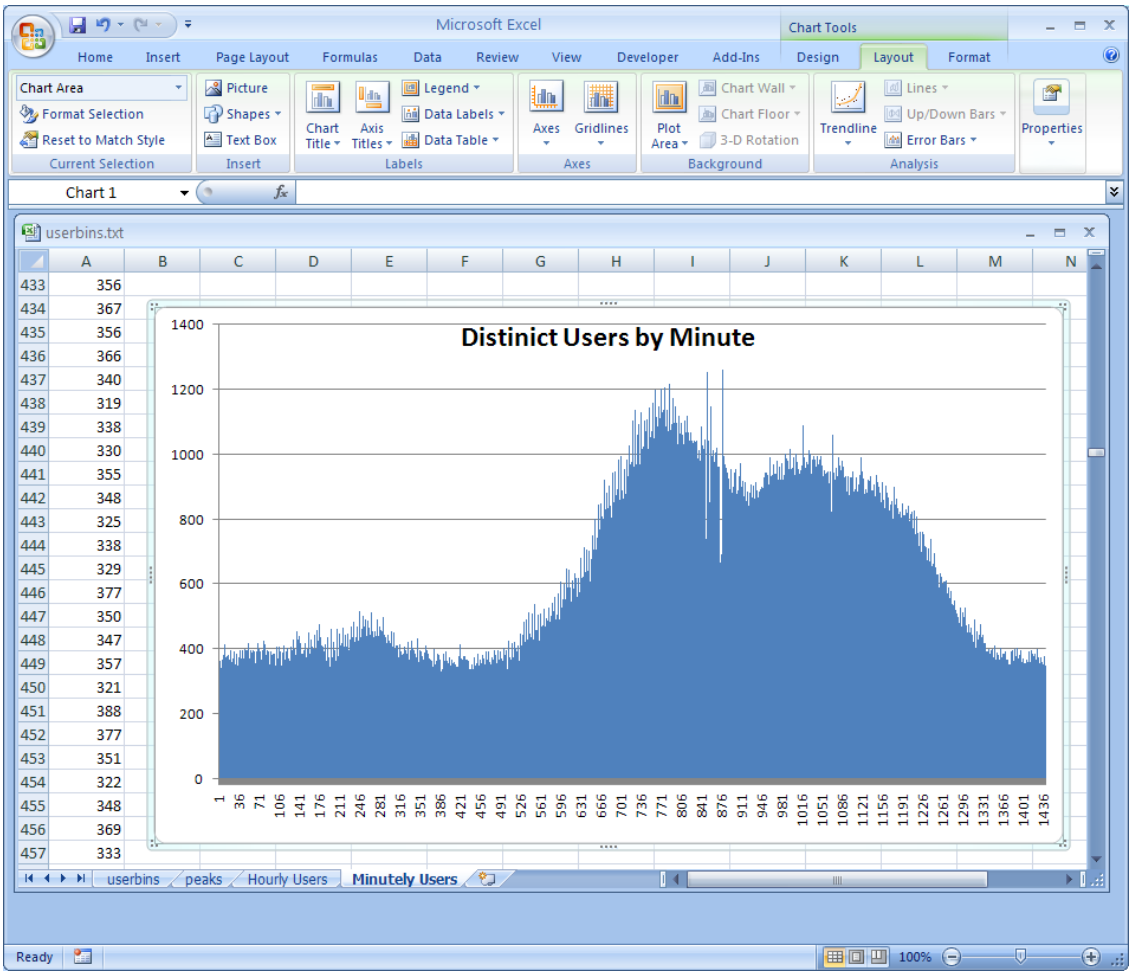
If we are interested in the minute, then we have to consolidate by minute:

```
logparser -i:CSV -o:CSV "select div(secs,60) as minu,cs-username,sum(ct) as req
from userdist.csv group by minu,cs-username order by minu,cs-username"
>usermidst.csv
```

and then:

```
logparser -i:CSV "select count(*) from usermidist.csv group by minu"
```

This of course gets us a much more detailed graph:



User agent distribution

The user agent is the program that was used to access the Web server. For example, it might be a browser, an Office client, or a program like Microsoft® SharePoint® Designer. The user agent is, in principle, identified by a string that the browser delivers with every request. However, this logic is clouded because sometimes applications make their behavior dependent on that string, and the agents have to pretend to be another agent (by supplying a different string) in order to get something to work.

Unfortunately, different libraries used by the same program might use different user agents. Thus, for example, when an Office client downloads a file, it uses more than one user agent string, some coming from the main program, and others coming from the WebDAV library. Nonetheless, agents are a useful way to categorize usage. However, as we will see, it is a bit complicated.

We start with:

```
logparser -i:IISW3C "select count(*) as ct,cs(user-agent) from *.log group by
cs(user-agent) order by ct desc"
```

This yields a messy output (note that we have truncated some of the lines with "…"):

```
ct      cs(User-Agent)
------- ------------------------------------------------------------------------
-------------------------
3080309 Microsoft+Office+Existence+Discovery
2158719 Microsoft-WebDAV-MiniRedir/6.0.6001
1154159 Microsoft-WebDAV-MiniRedir/6.0.6000
780160  MSFrontPage/12.0
651234
Mozilla/4.0+(compatible;+MSIE+7.0;+Windows+NT+6.0;+SLCC1;+.NET+CLR+2.0.50727.NET+
CLR+3.0.04506;++1.1…
```

```
581479
Mozilla/4.0+(compatible;+MSIE+6.0;+MS+Web+Services+Client+Protocol+2.0.50727.1434
)
487835
Microsoft+Office/12.0+(Windows+NT+6.0;+Microsoft+Office+OneNote+12.0.6320;+Pro)
340456
Mozilla/4.0+(compatible;+MSIE+6.0;+MS+Web+Services+Client+Protocol+2.0.50727.832)
322007
Mozilla/4.0+(compatible;+MSIE+6.0;+MS+Web+Services+Client+Protocol+2.0.50727.1433
)
259481
Mozilla/4.0+(compatible;+MSIE+7.0;+Windows+NT+6.0;+SLCC1;+.NET+CLR+2.0.50727;+.NE
CLR+3.0.04506;+.NET…
Press a key...
ct     cs(User-Agent)
------ ------------------------------------------------------------------------
-----------------------
200837
Microsoft+Office/12.0+(Windows+NT+6.0;+Microsoft+Office+Outlook+12.0.6320;+Pro)
164070 Microsoft-WebDAV-MiniRedir/5.1.2600
150287
Mozilla/4.0+(compatible;+MSIE+7.0;+Windows+NT+6.0;+SLCC1;+.NET+CLR+2.0.50727;+.NE
T+CLR+3.0.0450T+CLR+1.1.4…
Mozilla/4.0+(compatible;+MSIE+7.0;+Windows+NT+6.0;+SLCC1;+.NET+CLR+2.0.50727;+.NE
T+CLR+3.0.04501.1.4322;+In…
Mozilla/4.0+(compatible;+MSIE+7.0;+Windows+NT+6.0;+WOW64;+SLCC1;+.NET+CLR+2.0.507
27;+.NET+CLR+3ET+CLR+1.1.4….
Mozilla/4.0+(compatible;+MSIE+7.0;+Windows+NT+6.0;+SLCC1;+.NET+CLR+2.0.50727;+.NE
T+CLR+3.0.0450.1.4322;+In…
90803  Mozilla/4.0+(compatible;+MSIE+4.01;+Windows+NT;+MS+Search+5.0+Robot)
Press a key...
Task aborted by user.

Statistics:
-----------
Elements processed: 17216039
Elements output:    110
Execution time:     178.55 seconds (00:02:58.55)
```

The problem is that all the browser requests add extra parameters informing the application what libraries are installed and available on the browser client. Fortunately, Log Parser has a function that can parse strings nicely, **EXTRACT_TOKEN**. We use it as follows.

```
logparser -i:IISW3C "select count(*) as ct,EXTRACT_TOKEN(cs(user-agent),0,'/') as
agent from *.log group by agent order by ct desc"
```

Which yields the following more manageable result (headers and other irrelevancies deleted):

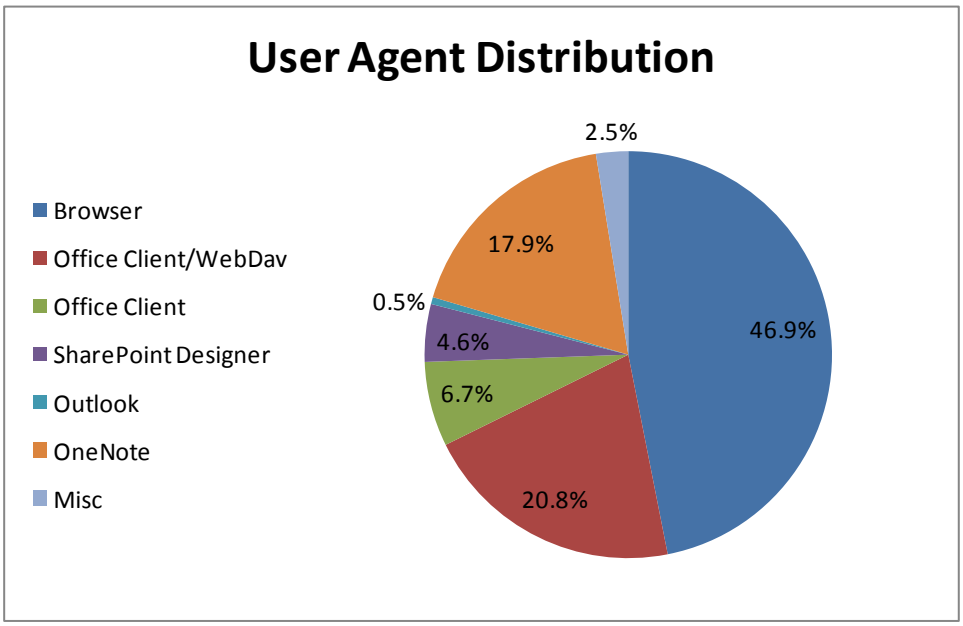
```
ct      agent
------- ---------------------------------------------------------------------
8073597 Mozilla
3580099 Microsoft-WebDAV-MiniRedir
3080309 Microsoft+Office+Existence+Discovery
1160285 Microsoft+Office
788682  MSFrontPage
101867  Windows-RSS-Platform
94667   OUTLOOK+STS
84177   -
67520   Microsoft+Office+Protocol+Discovery
55201   Microsoft+Data+Access+Internet+Publishing+Provider+Protocol+Discovery
```

```
37190 CFS
33519 non-browser
15407 GROOVE_WSS_SYNCH
12549 System+Center+Operations+Manager+2007+6.0.6278.0
10737 OutlookConnector
8887  SOAP+Toolkit+3.0
3519  MicrosoftExchangeServer-HttpClient
1806  NSPlayer
1558  Test+for+Web+Form+Existence
1144  MSIE+6.0
1039 MSOffice
611  MS-WebServices
492  RssBandit
223  InfoPathDA
208  Microsoft+Data+Access+Internet+Publishing+Provider+DAV
91   FDM+2.x
76   Windows-Media-Player
57   Opera
48   Internet+Explorer
45   DavClnt
42 Microsoft+Visio+MSIE
38 Microsoft+Data+Access+Internet+Publishing+Provider+Cache+Manager
34 Mindjet+MindManager
33 Java
32 VB+Project
30 http.exe
28 Sleipnir
27 Newzie+0.99.9+(www.newzie.com;News+Aggregator;+)
21 Widcomm+BtSendto+IE
20 contype
19 IDA
18 NewsGatorInbox
13 Microsoft+Windows+Network+Diagnostics
12 FeedDemon
12 JNLP
8  OfficeLive+Web+Service+Client+Protocol+1.5
6  Xenu+Link+Sleuth+1.2j
6  Feedreader+3.13+(Powered+by+Newsbrain)
6  Firefox
6  XML+Spy
3  veoh-Γ╕│Γ╕┤0+service+(NT+6.0;+IE+7.0.6001.18000;+en-US+Windows)
3  User-Agent
3  Microsoft-ATL-Native
2  WSDAPI
2  VCSoapClient
2  Jakarta+Commons-HttpClient
1  Outlook-Express
1  SlimBrowser
1  VB+OpenUrl

Statistics:
-----------
Elements processed: 17216039
Elements output:    59
Execution time:     86.40 seconds (00:01:26.40)
```

Of course this data comes from a developer shop (perhaps the biggest in the world – Microsoft), so we are likely to see all kinds of weird browser agent strings. But the majority will be the usual suspects.

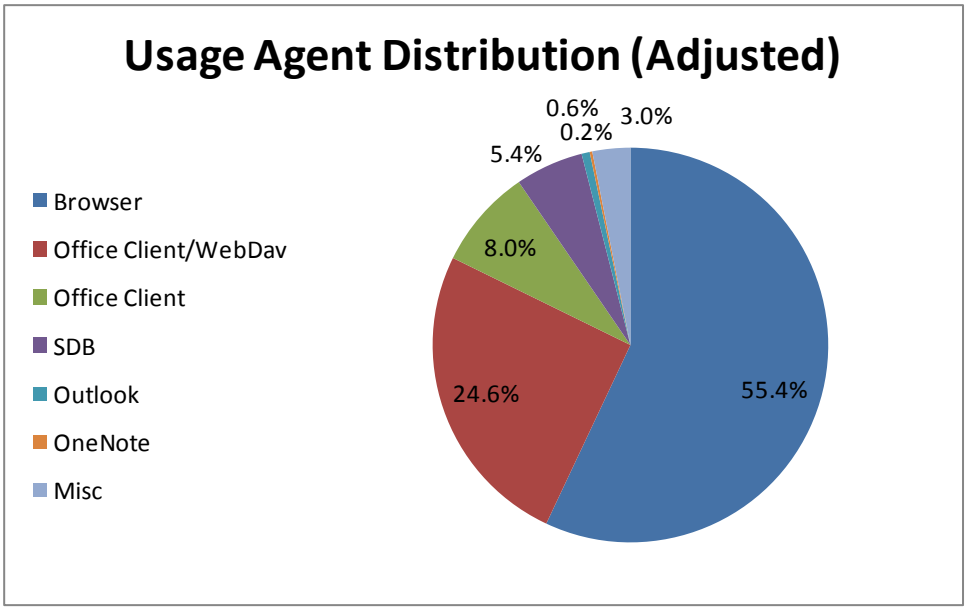
We copy the results into Excel for an analysis.



We note the following:

- The large number of **Microsoft+Office+Existence+Discovery** results is due to a OneNote issue that caused excessive polling. The actual number with correct behavior should not be significant and it would end up in Misc.
- **WebDAV** is mostly used by Office clients to manipulate files.
- Some of the SharePoint Designer calls are probably WebDAV calls initiated by an Office client because the latter uses some of the same protocol and libraries as SharePoint Designer.

So if we further simplify this, reducing the OneNote downloads and get the following distribution.



But let's look more closely.

Browser usage

We are interested in characterizing browser usage patterns now. First we get all the browser requests into a .csv file to reduce the total size and speed up the processing.

```
logparser -i:IISW3C "select * into ie.csv from *.log where EXTRACT_TOKEN(cs(user-agent),0,'/')='Mozilla' and sc-status<>401"
```

Output:

```
Statistics:
-----------
Elements processed: 17216039
Elements output:    5428305
Execution time:     177.60 seconds (00:02:57.60)
```

Note that we have dropped the 401s, and, as an aside, we can see that from the original 8073597 Mozilla requests, 264,000 were 401s, or 32.7 percent, which is almost a third.

Now we want to look at the extensions, and classify them as follows:

```
logparser -i:CSV "select count(*) as ct,TO_LOWERCASE(EXTRACT_EXTENSION(cs-uri-stem)) as ext from ie.csv group by ext order by ct desc"
```

Results:

```
ct      ext
------- ----
2162932 gif
704199  jpg
692126  aspx
607261  asmx
347486  css
245456  png
183874  js
100328
82836   axd
54753   one

Statistics:
-----------
Elements processed: 5428305
Elements output:    30
Execution time:     95.36 seconds (00:01:35.36)
```

Of course we are not interested in all of this data. Basically, we want the results shown in the table below. So we pipe the results into a text file (adding >ext.txt –q) to the end, import it into a spreadsheet and add up the numbers.

Extension	Class	Number	Percent
Total		5428305	
.aspx	Web pages	692114	12.7
.gif, .png, .bmp, .js, .css, .axd	Dependent or static request	3731271	68.7
.doc, .docx, .xls, .xlsx, .ppt, .pptx, .dotx, .txt, .zip, .vsd, .exe,. jpeg, .dll, .msg, .onetoc2, .xaml, .xlsm, .mp3	File downloads	126133	2.3
.asmx	Web Services	607261	11.1
blank	Redirects	100328	1.8
.htm, .html		73748	1.4

Also note we have a primary/dependent request ratio of 5.4, calculated by the ration of .aspx pages to the static request pages. The overall browser user operation to request ratio is about 3-1 or a bit less, depending on whether or not you include the .htm, .html, and blank requests into consideration.

Also interesting is the breakdown by file download:

Ext	Count	Pct
one	54753	39.5%
docx	15948	11.5%
xlsx	15831	11.4%
doc	11673	8.4%
pptx	9912	7.2%
dll	7392	5.3%
xls	4896	3.5%
pdf	4786	3.5%
ppt	3165	2.3%
msg	2121	1.5%
zip	1893	1.4%
vsd	1866	1.3%
onetoc2	997	0.7%
xaml	924	0.7%
xlsm	800	0.6%
txt	629	0.5%
mpp	266	0.2%
dotx	261	0.2%
exe	188	0.1%
mp3	78	0.1%
jpeg	66	0.0%
Total	138445	100.0%

Now let's look and see what .aspx pages are called:

```
logparser -i:CSV "select count(*) as ct,TO_LOWERCASE(EXTRACT_EXTENSION(cs-uri-stem)) as ext,TO_LOWERCASE(EXTRACT_FILENAME(cs-uri-stem)) as fname from ie.csv where ext='aspx' group by ext,fname order by ct desc"
```

This yields no less than 11051 different pages. Of course it has a long flat table. Looking at the top 20 we see the following:

Number	Percent	Page
192810	27.9%	listfeed.aspx
128122	18.5%	allitems.aspx
115473	16.7%	default.aspx
19100	2.8%	editform.aspx
12795	1.8%	upload.aspx
12270	1.8%	dispform.aspx
10522	1.5%	newform.aspx
8949	1.3%	excelrenderer.aspx
8341	1.2%	accessdenied.aspx
7404	1.1%	home.aspx
7147	1.0%	formserver.aspx
4835	0.7%	people.aspx
4761	0.7%	viewlsts.aspx
4499	0.7%	calendar.aspx
4026	0.6%	listedit.aspx
3525	0.5%	spsredirect.aspx
3458	0.5%	webfldr.aspx
3187	0.5%	aclinv.aspx
3023	0.4%	settings.aspx
2986	0.4%	filter.aspx
2793	0.4%	osssearchresults.aspx
132100	19.1%	Rest

Now we take a closer look at the ASMX Web service calls. What Web Services are being called?

```
logparser -i:CSV "select count(*) as ct,TO_LOWERCASE(EXTRACT_EXTENSION(cs-uri-
stem)) as ext,TO_LOWERCASE(EXTRACT_FILENAME(cs-uri-stem)) as fname from ie.csv
where ext='asmx' group by ext,fname order by ct desc"
```

And we get the following. These are probably asynchronous calls embedded in the .aspx page, and there are far fewer different kinds. Of course, we cannot see the actual name of the method invoked because it is inside the body of the request and not recorded in the IIS logs.

```
ct     ext  fname
------ ---- ----------------------
444600 asmx sitedata.asmx
113902 asmx lists.asmx
32722  asmx excelservice.asmx
6748   asmx search.asmx
3823   asmx webpartpages.asmx
3224   asmx publishingservice.asmx
1227   asmx sites.asmx
541    asmx usergroup.asmx
251    asmx dspsts.asmx
74     asmx copy.asmx
71     asmx webs.asmx
25     asmx businessdatacatalog.asmx
24     asmx userprofileservice.asmx
23     asmx spellcheck.asmx
2      asmx dws.asmx
2      asmx searchadmin.asmx
1      asmx documents.asmx
1      asmx views.asmx

Statistics:
-----------
Elements processed: 5428305
Elements output:    18
Execution time:     67.96 seconds (00:01:7.96)
```

Office client Web Service usage

Now we do the same thing for the Office client calls, that is, the non WebDAV calls. First we generate a .csv file to analyze.

```
logparser -i:IISW3C "select * into oc.csv from *.log where EXTRACT_TOKEN(cs(user-
agent),0,'/')='Microsoft+Office' and sc-status<>401"
```

Output:

```
Statistics:
-----------
Elements processed: 17216039
Elements output:    575916
Execution time:     87.64 seconds (00:01:27.64)
```

As before, we see that only 575916 of the original 1160285 requests were not 401, or about 50.2 percent.

Now, we break out the Web services with the following command:

```
logparser -i:CSV "select count(*) as ct,TO_LOWERCASE(EXTRACT_EXTENSION(cs-uri-
stem)) as ext,TO_LOWERCASE(EXTRACT_FILENAME(cs-uri-stem)) as fname from oc.csv
where ext='asmx' group by ext,fname order by ct desc"
```

Results:

```
ct     ext  fname
------ ---- -------------------------
443055 asmx lists.asmx
71901  asmx webs.asmx
```

```
45792   asmx workflow.asmx
1649    asmx socialdataservice.asmx
1331    asmx imaging.asmx
775     asmx dws.asmx
308     asmx publishedlinksservice.asmx
266     asmx views.asmx
129     asmx meetings.asmx
63      asmx alerts.asmx
42      asmx people.asmx
36      asmx slidelibrary.asmx
25      asmx versions.asmx
4       asmx webpartpages.asmx

Statistics:
-----------
Elements processed: 575916
Elements output:    14
Execution time:     6.37 seconds
```

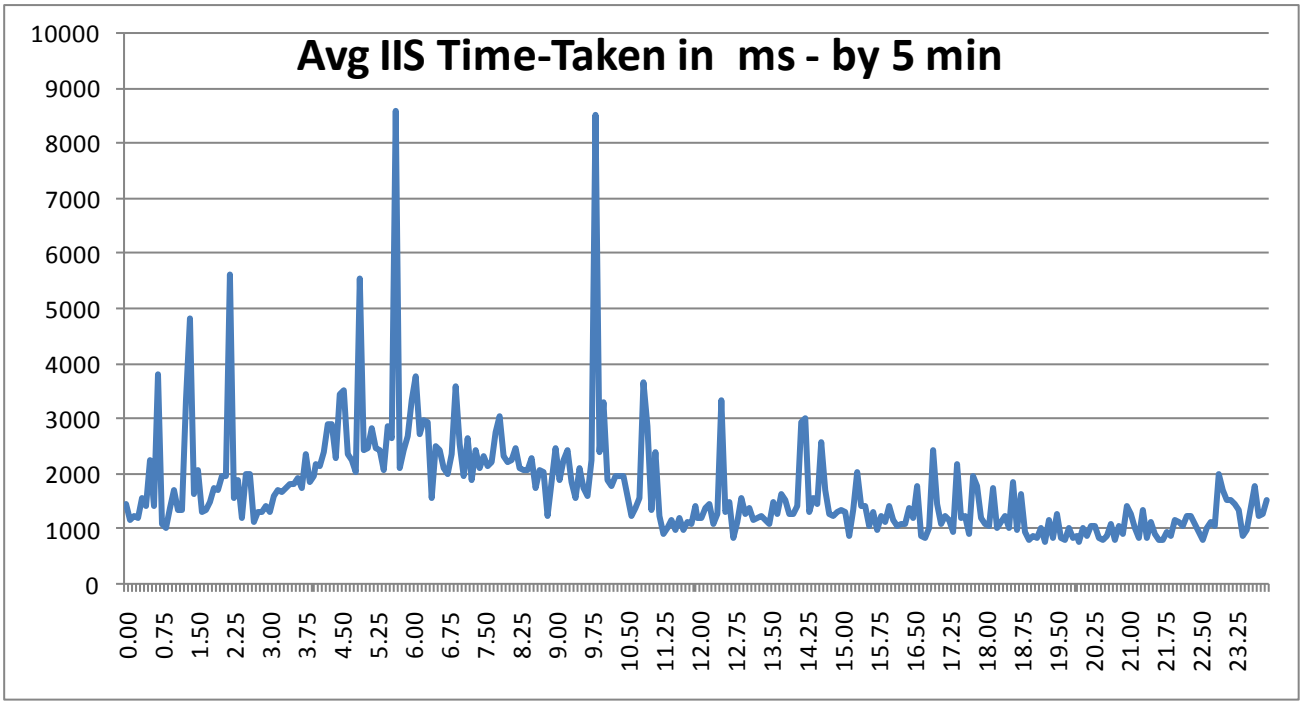
These are various Web Services called by the Office client.

Slow pages

Now we have a look at “slow” pages. We will look only at home pages (default.aspx pages) and look at the average “time-taken” field in IIS. This is the amount of time that IIS needed to render that page in html, including the necessary backend SQL calls. We use the bigo.csv file we generated earlier in this paper as a basis.

```
logparser -i:CSV -o:CSV "select avg(time-taken),div(secs,300) as
minu,max(mi),max(hh) from bigo.csv where TO_LOWERCASE(EXTRACT_FILENAME(cs-uri-
stem))='default.aspx' group by minu order by minu"  -q >homeavgmin.csv
```

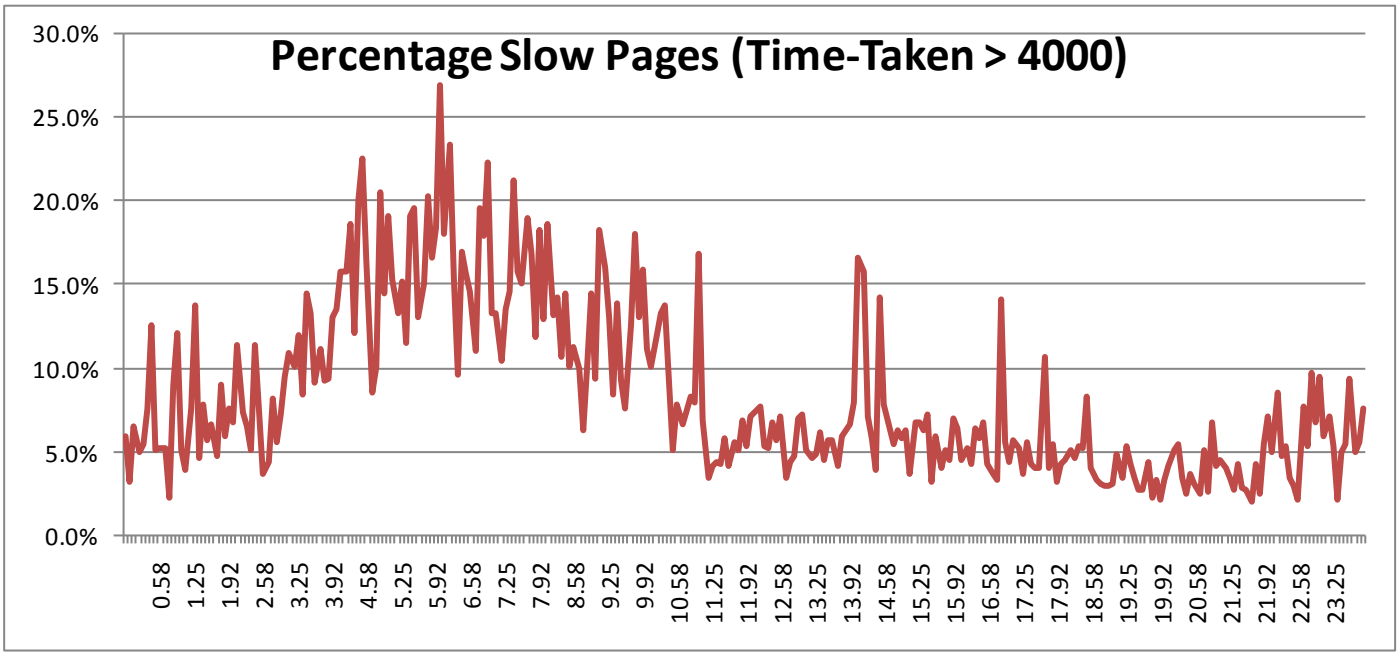
We take these values and plot them:



Now, this average does not really tell us a whole lot. We see that there are some times where the average page is twice as slow, but what percentage of users are actually experiencing slow pages? We examine this using two queries, getting two streams, one with the total number of samples, and one with the number of pages with values higher than 4000 ms (4 seconds):

```
logparser -i:CSV -o:CSV "select count(*) as ct,div(secs,300) as
minu,max(mi),max(hh) from bigo.csv where TO_LOWERCASE(EXTRACT_FILENAME(cs-uri-
stem))='default.aspx' group by minu order by minu"  -q >homepagect.csv

logparser -i:CSV -o:CSV "select count(*) as ct,div(secs,300) as
minu,max(mi),max(hh) from bigo.csv where TO_LOWERCASE(EXTRACT_FILENAME(cs-uri-
stem))='default.aspx' and time-taken>4000 group by minu order by minu"  -q
>slowpagect.csv
```



So we see at certain times of the day a large percentage of our users do experience slow page load times, particularly from 4 until 10 in the morning.

Note that merging the two streams was not an entirely trivial task in excel since some minute values in the slow page count could be missing. To avoid this you could use larger intervals (say 600 seconds), which would also give a smoother plot.

Alternatively, you could use an Excel lookup function. To generate our graph, we used the Excel **VLOOKUP** function to find the right value, and the **ISNA** function to turn missing values into zero. There may be an easier way, perhaps involving a modified Log Parser query that keeps the streams together in the first place.

Importing logs into SQL Server

Of course there are times when you may need to read the data into SQL Server or even another database. Log Parser can do that too. It is actually extremely easy to do, and the following command will read our test log into a table called ShpLogTest in the LogTest database.

```
logparser -i:IISW3C "select * into ShpLogTest from *.tst" -o:sql -server:Seadra -
database:LogTest -driver:"SQL Server" -createTable:ON


Statistics:
-----------
Elements processed: 99691
Elements output:    99691
Execution time:     54.69 seconds
```

This has the disadvantage that you lose all the useful Log Parser functions that are occasionally superior to what SQL offers. Fortunately, we can remedy that and have the best of both worlds.

The following input file and Log Parser invocation does just that, adding some extra columns to ease queries by date parts, file name, file extension, or file path, and excluding 401s.

Note: The current user is being used implicitly in the connection. Also, the time is being converted to the local time. This only works correctly if the local time on the analysis machine is the same as the servers where the log files originated from.

```
logparser -i:IISW3C file:loadShp1103.txt -o:sql -server:Seadra -database:LogTest
-driver:"SQL Server" -createTable:ON
```

The loadShp1103.txt file:

```
select EXTRACT_FILENAME(LogFilename),LogRow,

date, time, cs-method, cs-uri-stem, cs-username, c-ip, cs(User-Agent), cs-host,
sc-status, sc-substatus, sc-bytes, cs-bytes, time-taken,

add(
   add(
      mul(3600,to_int(to_string(to_localtime(to_timestamp(date,time)),'hh'))),
      mul(60,to_int(to_string(to_localtime(to_timestamp(date,time)),'mm')))
   ),
   to_int(to_string(to_localtime(to_timestamp(date,time)),'ss'))
) as secs,

to_int(to_string(to_localtime(to_timestamp(date,time)),'yy')) as yy,
to_int(to_string(to_localtime(to_timestamp(date,time)),'MM')) as mo,
to_int(to_string(to_localtime(to_timestamp(date,time)),'dd')) as dd,

to_int(to_string(to_localtime(to_timestamp(date,time)),'hh')) as hh,
to_int(to_string(to_localtime(to_timestamp(date,time)),'mm')) as mi,
to_int(to_string(to_localtime(to_timestamp(date,time)),'ss')) as ss,

to_lowercase(EXTRACT_PATH(cs-uri-stem)) as fpath,
to_lowercase(EXTRACT_FILENAME(cs-uri-stem)) as fname,
to_lowercase(EXTRACT_EXTENSION(cs-uri-stem)) as fext

into ShpLogTable

from *.log

where sc-status<>401
```

References

- Log Parser 2.2 (http://go.microsoft.com/fwlink/?LinkId=139171)