Analyzing Performance Metrics for File Virtualization

Derrick Schommer, 2011-29-09

Trying to analyze your file storage from an RPC-level can be a challenge, but the fruits of your labor can be tasty if you put in the work (sorry, went Apple picking last weekend and it's still on my mind.) It is time to bust out your storage solutions command reference and begin searching for "show screens" for finding statistics counts for the CIFS and NFS RPCs. File Virtualization can be very beneficial for complex environments, but understanding your environment goes beyond simply knowing how big your files are and how many files you have on disks in your data center.

Let's take some time to draw out some theories as to how we can analyze the statistics on our file storage devices using their text-based statistical screens. Our two big examples, as some may expect, are Network Appliance and EMC and, while their statistics are similar, we're going to focus mainly on Network Appliance storage but the results can be applied to many devices on the market.

We'll capture both NFS and CIFS RPC's so that we can understand their impact on the environment (including the absence of statistics because no statistics will still be important information). The two commands we'll focus on are called "cifs stat" and "nfs stat" which, as you may guess, show statistics for CIFS and NFS respectively. While the output is extensive, here is a small example of a the CIFS information from this output:

```
cifs stat
```

```
...
```

```
find_first2 2044150854  10%
```

```
find_next2 61012764  0%
```

```
query_fs_info 37863253 0%
```

```
query_path_info 7097072943 35%
```

```
set_path_info 0  0%
```

```
query_file_info 984065779 5%
```

```
set_file_info 150199113 1%
```

```
create_dir2 0  0%
```

```
...
```

These statistics can, often times, be super-intimidating because they are often extremely large and, alone, extremely limited in use. For instance, what does *7097072943* really mean for the "query_path_info" CIFS RPC? Of course, you can clear the statistics and start watching them closer from here on out, or you can utilize these large numbers as your initial "baseline" and continue to monitor the statistics over a set interval. If you've not figured it out already, that's what we're going to do, save clearing statistics for the newbies, we're going hardcore! (okay, some administrators may not want you to do this or not give you access to clear their statistics, rightly so.)

The first step I suggest would be to pick a language that is easily accessible to your environment for scripting; you may choose TCL because you are familiar with it and on a Unix/Linux/OSX system, like myself, or Visual Basic Script (VBS) for those that require Microsoft Windows in their environment. One key factor in this scripting language is the ability to utilize tools like RSH, SSH or Telnet in order to access the remote storage device to issue the required commands. For our VBS solutions, we've made use of putty/plink with to authenticate into a remote system. With TCL, the expect package has most of the tools to spawn off ssh/rsh and other shell activity.

Using TCL as a base example, I've designed an interface in which a variable $style represents SSH/TELNET or RSH:

```
switch -- $style {
 "rsh" {
   set useRSH true
  }

  # Login via TELNET
  "telnet" {
   spawn telnet "$host"
   handleTELNET $user $password $prompt
  }

  # Login via SSH
 "ssh" {
  spawn ssh "$user@$host"
  handleSSH $password $prompt
 }
}
```

The above logic breaks out the correct login handler and then "handles" the negotiations of authentication before continuing. In the case of RSH, it's extremely beautiful in its lack of authentication (don't tell the security guys, but this is the easiest method of scripting!) While we don't show the boring handler functions here, these functions do the heavy lifting of awaiting password prompts, supplying the credentials and awaiting an initial prompt.

For Network Appliance, I encourage the use of RSH because it doesn't count as an authenticated user so, in the case in which an administrator is already logged in, it won't boot you offline when too many sessions have connected. If you're consistently booted off the system you'll have a very difficult time getting timed interval statistics.

Once you've successfully logged in you can poll for statistics. For sanity, I always first poll for the version of the software followed by the statistics and keep a time stamp and counter for the iteration so that we can graph the trends over time. A high level example may look like:

```
set stamp [clock format $seconds -format "%D %H:%M:%S"]
```

```
puts $fd "INDEX: $index"
```

```
puts $fd "TIME: $stamp\n"
```

```
puts $fd [sendCMD $prompt "version"]
```

```
puts $fd [sendCMD $prompt "cifs stat" ]
```

```
puts $fd [sendCMD $prompt "nfs stat" ]
```

Here we write to our log file (the file handle $fd, which must be opened before logging) and write an index counter (starting at zero or one and incrementing each iteration) along with the current time stamp and then get the version and statistics information. Our little function "sendCMD" does the *send* and *expect* function calls to ship the information to the filer and return the results. These results are stored in our log file and look very much like the command we issued on the filer.

Toss this logic around a loop and sleep for some set interval (I like a 5-minute sleep timer) and re-run it again with a new $index and timestamp against the storage filer for a few days or a week and you'll end up with a great trend of important RPC information fully logged and ready to parse!

How Do I Parse This Epic RPC Data!?

With the tricky use of regular expressions or your favorite text-extraction mechanism, you'll want to pull out each RPC and its associated value so you can examine it closer. Let's use the attribute retrieval RPC again:

```
query_path_info  7097072943  35%
```

This can be stored so that we record "query_path_info" and "7097072943" using your favorite string parsing mechanism, we don't need the 35% to really get the meat of the RPC. Continue to record that information for every interval in our log output for trending and graphing, as long as you feel it worth, but I suggest you measure that time in days not hours.

Let's pretend this was the only RPC in our output in order to avoid any confusion. Assuming *query_path_info* was recorded in our little log file six times (only 6-intervals, but it's just an example), we'd have values of ever incrementing counts like:

```
query_path_info  7097072943
```

```
query_path_info  7097278634
```

```
query_path_info  7097497868
```

```
query_path_info  7097667437
```

```
query_path_info  7097902576
```

```
query_path_info  7098298014
```

Assuming we've logged this information every 5-minutes and we have a total of six iterations of the statistic in our log file, we can start building some trending. First, lets normalize this data and assume the first entry is our "baseline" value as we said above, that means we set this to zero and find the incremental differences in the data, we also drop off the very last statistical value because it doesn't have any more statistics to build a difference from (you can't subtract from nothing!)

So, let's do some 2nd grade math here with some 4rd grade numbers:

```
RPC   Value  (This RPC - Last RPC)
```

~~query_path_info 0~~

```
query_path_info  205691  (7097278634 - 7097072943)
```

```
query_path_info  219234  (7097497868 – 7097278634)
```

```
query_path_info  169569  (7097667437 – 7097497868)
```

```
query_path_info  235139  (7097902576 – 7097667437)
```

```
query_path_info  395438  (7098298014 – 7097902576)
```

~~query_path_info 7098298014 (no future RPC's)~~

This example reduces the big intimidating numbers down to something more manageable; this represents the total number of ***query_path_info*** RPC calls that have occurred between the 5-minutes our script sat sleeping at the wheel—the difference from the first interval to the second interval. To build these, we took the differences of the RPC that occurred at time ***n*** from the RPC that occurred at time ***n+5-minutes*** to figure out how many RPC's occurred between the two intervals. You can run your subtraction in the opposite order if you use absolute values or, in other means, drop that negative sign (e.g. 7097072943 - 7097278634 = abs(-205691)).

Let's reduce them even further by calculating the ***query_path_info*** RPC's "per second" over that five minute interval. Given the constant that 5-minutes comes down to 300-seconds (5 x 60) we can break those out with the following mathematical results:

```
RPC   Per/Second#  (Value / 5-minutes)
```

```
query_path_info  686/second (205691 / 300)
```

```
query_path_info  731/second (219234 / 300)
```

```
query_path_info  565/second (169569 / 300)
```

```
query_path_info  784/second (235139 / 300)
```

```
query_path_info  1,318/second (395438 / 300)
```

Of course, nothing is perfect; we've given up some statistical value by rounding up a bit. For example, we report 686 *query_path_info* RPC's a second in our first result even though the true result is 685.63667 for simplicity—yes we're lying. We round up because an environment being sized for File Virtualization might as well take into consideration the worst case so that the correct hardware can be implemented with room for growth.

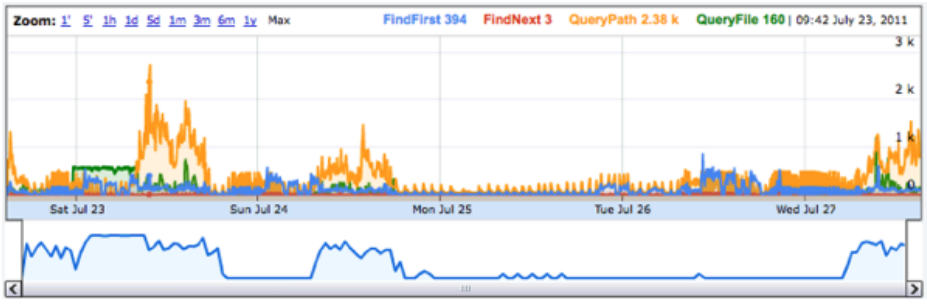
If we design these simple mathematical equations into a little web user-interface using PHP or Python, you can build off some free tools like Google Charts to lay out your RPC counts into some pretty neat graphs. Using an Annotated Timeline chart, assuming you're also recording that timestamp we talked about above, you can generate an HTML chart similar to (but abridge, we left out some wordy code just for the example):

```
function drawChart() {

 var data = new google.visualization.DataTable();

 data.addColumn('datetime', 'Date');

 data.addColumn('number', 'QueryPath');

 data.addRows(5);

 data.setValue( 0, 0, new Date( 2011, 06, 22, 15, 03 ) );

 data.setValue( 0, 1, 686);

 data.setValue( 1, 0, new Date( 2011, 06, 22, 15, 08 ) );

 data.setValue( 1, 1, 731);

 data.setValue( 2, 0, new Date( 2011, 06, 22, 15, 13 ) );

 data.setValue( 2, 1, 565);

 data.setValue( 3, 0, new Date( 2011, 06, 22, 15, 18 ) );

 data.setValue( 3, 1, 784);
```

```
data.setValue( 4, 0, new Date( 2011, 06, 22, 15, 23 ) );

data.setValue( 4, 1, 1318);


var chart = new google.visualization.AnnotatedTimeLine(

 document.getElementById('chart_cifs_qpi') );

chart.draw(data, {displayAnnotations: true} );

}
```

The google API will then render the text to look super awesome, much like their financial graphs in timeline form like:

This chart took the information we generated, simplified and reduced to a “per second” interval and develops a timeline trend for the given RPC. This is a simple example, but when implemented with all RPC’s you want to track, you can build a diagram more like:



This timeline takes into account (using the same mathematical equations) a mapping of the CIFS FindFirst, FindNext, QueryPath and QueryFile from the statistics within the ‘cifs stat’ calls, over a 5-minute interval for five days worth of time.

With this data, we can start to develop trends and understand what the clients within an environment are doing. In this larger example we can theorize a number of active file operations occurring on Saturday with the increase in QueryFile RPC’s, which a Microsoft Windows client typically does before opening a file for create/read/write activity. The weekend also shows a large amount of file system activity with QueryPath information and directory listings (indicated by the FindFirst calls). Lower FindNext RPC calls suggests that most CIFS directory listings can be returned in a single RPC request without needing to ‘next’ through for additional information. I also call into question the ethics of such a scenario having their folks work on Saturdays! In all reality, however, these are probably automated systems or system backups executing during off-peak hours to prevent heavy load during the workday.

Next time you have a chance to sit down, think about your poor storage devices and consider all the work they have to do throughout the day. Consider designing some tools to really understand how your data is being used. Understanding your data will better prepare you for the realization that you might need a file virtualization solution like the ARX to most effectively use your storage.

Once you've discovered just how your data looks on the network, you may also want to consider how your data looks on your file system and how to best virtualize your environment. Checkout our Data Manager tool for a completely different look at what you're platters and solid state are doing throughout the day.

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.
Corporate Headquarters
info@f5.com

F5 Networks
Asia-Pacific
apacinfo@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
emeainfo@f5.com

F5 Networks
Japan K.K.
f5j-info@f5.com

©2016 F5 Networks, Inc. All rights reserved. F5, F5 Networks, and the F5 logo are trademarks of F5 Networks, Inc. in the U.S. and in certain other countries. Other F5 trademarks are identified at f5.com. Any other products, services, or company names referenced herein may be trademarks of their respective owners with no endorsement or affiliation, express or implied, claimed by F5. CS04-00015 0113