Monitor Your iRule Performance with iControl and Google-O-Meter Charts

Joe Pruitt, 2010-15-12

iRule performance has long been a question our users have asked about. We get questions all the time about how to determine what impact a specific iRule has on the normal operation of the BIG-IP's they are running on. The question is a hard one to answer as we rarely see two iRules that are the same and the mere fact that you can code your iRule however you want, makes it very hard to provide info on the impact they would have in a generic manner.

A while back we wrote a tech tip on optimizing iRules titled "Evaluating iRule Performance" in which we illustrate how to enable timing metrics on an iRule to help report the number of CPU cycles the iRule is using up during it's processing. We make these metrics available in the BIG-IP CLI and GUI as well as from the iControl API. This makes it easy to integrate them into external applications such as the iRule Editor.

I'm always thinking of ways to visualize the metrics we can get from iControl and I got to thinking about the article I wrote a while back on Building an iControl PowerShell Monitoring Dashboard with Google Charts and looked again at the Google Charts API to see if anything new had cropped up recently.

For those that don't know about Google Charts, Google has made available an awesome, in my opinion, API to enable the dynamic creation of charts to include with your own web applications. At the time of this article, they offer Bar charts, Box charts, Candlestick charts, Compound charts, Dynamic icons, Formulas, GraphViz charts, Line charts, Map charts, Pie charts, QR codes, Radar charts, Scatter charts, Venn charts, and, the one I decided to use in this article, Google-O-Meter charts.

iRule Metrics

By using the "timing on" command within an iRule, several metrics can be retrieved from the LocalLB.Rule.get_statistics() method. These values are:

- STATISTIC_RULE_MINIMUM_CYCLES - The minimum number of clock cycles spent during an execution of the rule event.
- STATISTIC_RULE_AVERAGE_CYCLES - The average number of clock cycles spent during an execution of the rule event.
- STATISTIC_RULE_MAXIMUM_CYCLES - The maximum number of clock cycles spent during an execution of the rule event.
- STATISTIC_RULE_ABORTS - The number of aborts due to TCL errors.
- STATISTIC_RULE_FAILURES - The number of times the iRule has failed for any reason.
- STATISTIC_RULE_TOTAL_EXECUTIONS - The number of times the iRule event has been executed.

These metrics are on an event by event basis so that means we can get detailed results for each of the events (HTTP_REQUEST, STREAM_MATCHED, etc) used within a given iRule. From these metrics, along with the speed of the CPU on the target platform, one can calculate metrics such as

- CPU Cycles/Request
- Runtime (ms)/Request
- Percent CPU Usage/Request

Google-O-Meter charts

Google-O-Meter charts are in the form of a gauge and a simple one looks like the image on the right. It provides a minimum and maximum for the gauge along with a value for the "arms". You have full control over the colors and labels so I figured this would be a great way to represent the various metrics we can retrieve from iRules.

With the metrics returned from the iControl API, we could build a dashboard with these charts that illustrated in real-time how the iRule was performing.

The Code

I will follow the model in my previous dashboard example by building an engine in PowerShell that does the data collection and processing. It will then use the automation features in Internet Explorer to dynamically create a browser instance and feed it with the output from the engine. The engine will continue processing in a loop and feed the updated statistics to the browser allowing for a dynamically refreshing dashboard.

The script will take the following parameters

- BIGIP – The address of your BIGIP.
- User – The admin user for your BIGIP.
- Pass – The admin password for your BIGIP.
- VirtualServer – An optional virtual server name to filter the results on.
- iRule – An optional iRule name to filter the results on.
- Metric – The metric you would like reported (“CYCLES”, “RUNTIME”, “CPUUSAGE”, “ALL”)
- Debug – If set to $true, debug information is displayed to the console during the script operation.
- Interval – The frequency in seconds between polls. The default value is 10 (seconds)

The main logic for the engine is a while loop in which data is collected with the Get-Data function and pumped to the instance of Internet Explorer with the Refresh-Browser function. The script then sleeps during the specified sleep interval and then repeats the procedure.

```
 1: function Run-Dashboard()
 2: {
 3:   while($true)
 4:   {
 5:     Write-Host "Requesting data..."
 6:     $file_data = Get-Data;
 7:
 8:     Refresh-Browser $file_data;
 9:     Start-Sleep $script:INTERVAL;
10:   }
11: }
```

Retrieving Data

The Get-Data function is really the heart of this application. This function requests the list of iRules assigned as resources to the specified Virtual Server(s) as well as the metrics for all the iRules on the system. I chose to make a single call here to reduce the processing speed of the function. I could have filtered the results and only queried the metrics or the iRules associated with Virtuals but took the easy road with the get_all_statistics() method. I’ll leave it an exercise for the reader to make this change.

The script then loops through the virtual servers and builds a table report for each one. The format of the table each row being an iRule and each column representing the events that are used within that iRule. I wrote some code to calculate all of the types of events in all iRules for the given virtual server so that we could have a nice layout.

The presentation (HTML) for the dashboard is stored in the $page_data variable and returned to the calling code.

```
 1: function Get-Data()
 2: {
 3:   $now = [DateTime]::Now;
 4:
 5:   $page_data = "<html><head><title>$($script:TITLE)</title>
 6:   <style type='text/css'>
 7:     body,td,th { font-family: Tahoma; font-size: 10pt; }
 8:     .datatable { background-color: #C0C0C0; }
 9:     .colheader { background-color: cyan; }
10:     .rowheader { background-color: yellow; }
11:   </style>
12:   </head>
13:   <center><h1>iRule Monitor</h1><br/><h2>$now</h2>";
14:
15:   $vslist = [string[]]$(Get-VirtualServerList);
16:
17:   $VirtualServerRuleAofA = (Get-F5.iControl).LocalLBVirtualServer.get_rule( $vslist );
18:   $RuleStatistics = (Get-F5.iControl).LocalLBRule.get_all_statistics();
19:
20:   # loop through all the virtual servers
```

```
21:   for($i=0; $i -lt $VirtualServerRuleAofA.Length; $i++)
22:   {
23:     $vs = $vslist[$i];
24:
25:     # rules for current vs
26:     $VirtualServerRuleA = $VirtualServerRuleAofA[$i];
27:
28:     # rule stats for current vs
29:     $RuleStatisticEntryA = Filter-RuleStatistics $VirtualServerRuleA $RuleStatistics;
30:     if ( $RuleStatisticEntryA -ne $null )
31:     {
32:       # Build out a column list...
33:       $columns = @{};
34:       foreach($RuleStatisticEntry in $RuleStatisticEntryA)
35:       {
36:         if ( ($columns.count -eq 0) -or ($null -eq $columns[$RuleStatisticEntry.event_name]) )
37:         {
38:           $columns.Add($RuleStatisticEntry.event_name, 0);
39:         }
40:       }
41:       $scolumns = $columns.GetEnumerator() | Sort-Object Name;
42:
43:       # Build out result graphs
44:
45:       Write-DebugMessage "VIRTUAL SERVER $vs";
46:
47:       $page_data += "<table border='1' class='datatable'>"
48:       $page_data += "<tr><th colspan='$($columns.Count + 1)'>Virtual Server '$vs'</th></tr>";
49:
50:       $page_data += "<tr><th>iRule</th>"
51:       foreach($key in $scolumns)
52:       {
53:         $page_data += "<th class='colheader'>$($key.Name)</th>";
54:       }
55:       $page_data += "</tr>";
56:
57:       # loop through all rules for current  virtual server
58:       foreach($VirtualServerRule in $VirtualServerRuleA)
59:       {
60:         if ( ($null -eq $script:IRULE) -or ($VirtualServerRule.rule_name -eq $script:IRULE) )
61:         {
62:           $page_data += "<tr>";
63:           $page_data += "<td class='rowheader'>$($VirtualServerRule.rule_name)</td>";
64:
65:           foreach($key in $scolumns)
66:           {
67:             $aborts = $null;
68:             $avg_cycles = $null;
69:             $failures = $null;
70:             $max_cycles = $null;
71:             $min_cycles = $null;
72:             $total_executions = $null;
73:
74:             foreach($RuleStatisticEntry in $RuleStatisticEntryA)
75:             {
76:               if ( ($RuleStatisticEntry.rule_name -eq $VirtualServerRule.rule_name) -and
77:                    ($RuleStatisticEntry.event_name -eq $key.Name) )
78:               {
79:                 Write-DebugMessage "Searching for '$($RuleStatisticEntry.rule_name), $($RuleStatisticEntry.event_name)...";
80:                 # Found a match for the row and column
81:                 $min_cycles = Extract-Statistic $RuleStatisticEntry.statistics "STATISTIC_RULE_MINIMUM_CYCLES";
82:                 $avg_cycles = Extract-Statistic $RuleStatisticEntry.statistics "STATISTIC_RULE_AVERAGE_CYCLES";
83:                 $max_cycles = Extract-Statistic $RuleStatisticEntry.statistics "STATISTIC_RULE_MAXIMUM_CYCLES";
84:                 $aborts = Extract-Statistic $RuleStatisticEntry.statistics "STATISTIC_RULE_ABORTS";
85:                 $failures = Extract-Statistic $RuleStatisticEntry.statistics "STATISTIC_RULE_FAILURES";
86:                 $total_executions = Extract-Statistic $RuleStatisticEntry.statistics "STATISTIC_RULE_TOTAL_EXECUTIONS";
87:                 break;
88:               }
89:             }
90:
91:             $min = 0;
92:             $data_value = $null;
93:             $max = 5000;
94:
95:             $speedMhz = [Convert]::ToDouble($script:CPUSPEED);
96:             $speed = [Convert]::ToUInt64(1000000 * $speedMhz);
97:
98:             $cycles_min = $min_cycles;
99:             $cycles_avg = $avg_cycles;
100:            $cycles_max = $max_cycles;
101:
102:            $runtime_min = [Convert]::ToDouble($min_cycles) * (1000.0/$speed);
103:            $runtime_avg = [Convert]::ToDouble($avg_cycles) * (1000.0/$speed);
104:            $runtime_max = [Convert]::ToDouble($max_cycles) * (1000.0/$speed);
105:
106:            $cpu_min = 0;
```

```
            #$min = 100.0 * ([Convert]::ToDouble($min_cycles) / [Convert]::ToDouble($speed));
            $cpu_avg = 100.0 * ([Convert]::ToDouble($avg_cycles) / [Convert]::ToDouble($speed));
            #$max = 100.0 * ([Convert]::ToDouble($max_cycles) / [Convert]::ToDouble($speed));
            $cpu_max = 100;

            if ( 0 -eq $data_value ) { $data_value = $null; }
            if ( $null -ne $data_value ) { $data_value = $data_value.ToString("0.0000"); }

            $cycles_chart = Get-Chart $cycles_min $cycles_avg $cycles_max "CYCLES";
            $runtime_chart = Get-Chart $runtime_min $runtime_avg $runtime_max "RUNTIME" "0.00000";
            $cpu_chart = Get-Chart $cpu_min $cpu_avg $cpu_max "CPUUSAGE" "0.00000";
            $success_chart = Get-Chart 0 ($aborts + $failures) $total_executions "ERRORS";

            Write-DebugMessage $charturl;

            $page_data += "<td>";

            switch($script:METRIC)
            {
              "CYCLES" {
                $page_data += "$cycles_chart";
                $page_data += $success_chart;
              }
              "RUNTIME" {
                $page_data += "$runtime_chart";
                $page_data += $success_chart;
              }
              "CPUUSAGE" {
                $page_data += "$cpu_chart";
                $page_data += $success_chart;
              }
              "ALL" {
                $page_data += $cycles_chart;
                $page_data += $runtime_chart;
                $page_data += $cpu_chart;
                $page_data += $success_chart;
              }
              default {
                $page_data += "$cycles_chart";
              }
            }
            $page_data += "</td>";
          }
          $page_data += "</tr>";
        }
      }
      $page_data += "</table><p>"
    }
  }

  $page_data += "</center>";

  return $page_data;
}
```

Generating The Chart

I wrote a little helper function to generate the image reference for the Google-O-Meter chart. It takes as input the min value for the chart, the value for the arrow, the max value for the chart, the type of chart (for the label) as well as an optional data format specifier (for those pesky small double values).

```
function Get-Chart()
{
  param($min, $val, $max, $type, $fmt);

  $chart = "";
  Write-DebugMessage "Querying chart for '$min', '$val', '$max', '$type'...";

  if ( ($null -ne $val) -and ($null -ne $fmt) )
  {
    $val = $val.ToString($fmt);
  }

  $suffix = "";
  switch($type)
  {
    "CYCLES" { $suffix = ""; }
    "RUNTIME" { $suffix = " ms."; }
    "CPUUSAGE" { $suffix = "%"; }
  }

  $prefix = Get-ChartPrefix;
  $charturl = "http://${prefix}chart.apis.google.com/chart?cht=gom";

```

```
  $charturl += "&chs=$($script:CHARTSIZE)";
  $charturl += "&chco=00FF00,FF0000";
  $charturl += "&chds=$min,$max";
  $charturl += "&chd=t:$val";
  $charturl += "&chxt=x,y";
  $charturl += "&chxl=0:|$val$suffix|1:|0||$max";
  $charturl += "&chf=c,lg,45,FFE7C6,0,76A4FB,0.75";

  Write-DebugMessage "returning chart: $charturl...";

  $chart += "<img src='$charturl'/>";
  switch($type)
  {
    "CYCLES" {
      $chart += "<br/><center>CPU Cycles/Request</center>";
    }
    "RUNTIME" {
      $chart += "<br/><center>Runtime (ms)/Request</center>";
    }
    "CPUUSAGE" {
      $chart += "<br/><center>Percent CPU Usage/Request</center>";
    }
    "ERRORS" {
      $chart += "<br/><center>Total Errors</center>";
    }
  }

  $chart += "<br/>";

  return $chart;
}
```

Feeding Results To The Browser

The Refresh-Browser function will feed the data to the browser. It first checks to see if a browser instance has been created and, if not, creates it. It then updates the browsers Document.DocumentElement.lastChild.InnerHTML property to dynamically update the contents of the page. This causes the page to refresh itself.

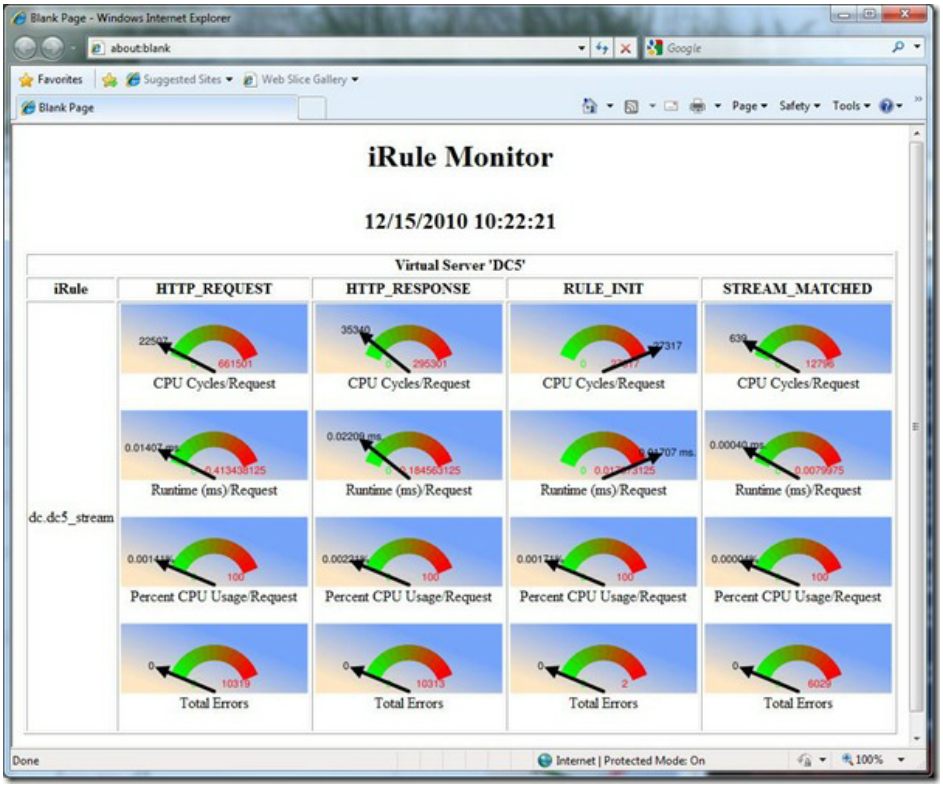
```
function Refresh-Browser()
{
  param($file_data);

  if ( $null -eq $script:BROWSER )
  {
    Write-DebugMessage "Creating new Browser"
    $script:BROWSER = New-Object -com InternetExplorer.Application;
    $script:BROWSER.Navigate2("About:blank");
    $script:BROWSER.Visible = $true;
    $script:BROWSER.TheaterMode = $script:THEATER;
  }
  $docBody = $script:BROWSER.Document.DocumentElement.lastChild;
  $docBody.InnerHTML = $file_data;
}
```

The Results

The following is how the application looks when run with the following parameters. I've specified to monitor the virtual server 'DC5' and only the 'dc.dc5_stream' iRule. It will present graphs for all the available metrics.

```
PS> .\PsiControlDashboard.ps1 -BIGIP bigip -User user -Pass pass -VirtualServer DC5 -iRule "dc.dc5_stream" -Metric "ALL"
```



Get The Code

The full PowerShell script is available in the iControl CodeShare under PsIruleDashboard.

Related Articles on DevCentral

- Creating An iControl PowerShell Monitoring Dashboard With Google ...
- PS iControl Dashboard Question - DevCentral - F5 DevCentral ...
- v.10 - BIG-IP Dashboard > DevCentral > F5 DevCentral > Tech Tips
- Getting Started wth iControl and AJAX > DevCentral > F5 DevCentral ...
- What is iControl? > DevCentral > F5 DevCentral > Tech Tips
- iControl Development with BIG-IP LTM VE – Part 1 > DevCentral > F5 ...
- iControl Quickstart Guide for .NET > DevCentral > F5 DevCentral ...
- Self-Contained BIG-IP Provisioning with iRules and pyControl ...
- DevCentral Top5 11/7/2008
- Java iControl Objects - LTM Pool Member > DevCentral > F5 ...
- DevCentral > F5 DevCentral > Tech Tips > 2009 Archive
- Quick iRule Redirection > DevCentral > F5 DevCentral > Tech Tips
- F5 DevCentral > Tech Tips > iControl
- F5 DevCentral > Tech Tips > iRules

Technorati Tags: iControl, Dashboard, iRules, timing, Google, Charts, Joe Pruitt

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