**Load Runner 9.5**

Trainer: Venkat

Mail ID: gv2225@gmail.com

Mobile #: 9052126622

1. Previous Version of LoadRunner – 9.0, 9.10
2. V9.10 was in market for only 5 months because of the below issue -
3. Problem in LR 9.1 version is – LR VUGen always restarts for every 5 mins.

**New features in 9.5 –**

1. It supports for Vista operating system. (9.0 does not support it)
2. It provides protocol advisor to identify the type of the protocol supports for the application.
3. It provides the randomized functions to select a random value from an array.

lr\_paramarr\_random()

lr\_paramarr\_idx()

lr\_paramarr\_len()

**Components in LoadRunner –**

1. **Virtual User Generator (VUGen) –**

It is used to generate and enhance the Vuser script for each scenario.

1. **Controller –**

Design and execute the multi user’s test runs (Runs can be smoke/load/stress/endurance. etc)

1. **Analysis (Analyzer)-**

It is used to analyze the test execution results after every test run.

1. **Load Generator –**

It generates the amount of Vusers to run on the server.

1. **Agent Process –**

It establishes the communication between the Load Generator and Load Controller.

**Virtual User Generator (VUGen) –**

In this component, we will generate the script based on the supported protocol.

**Protocol Advisor** – Use Protocol Advisor to identify the supported protocol in our business process.

1. Start Web Server from Start>All Programs>LoadRunner>Samples>Web>Start Web Server.

Login ID/Password – jojo/bean for HP Web Tours site.

If application is using –

1. Javascript for client scripting – use Web (HTTP/HTML) protocol
2. AJAX for client scripting – use AJAX (Click and Script) protocol

Other factors for considering the protocol are –

1. Licensing costs (Cost of AJAX is more, 10 times more than the Web protocol)
2. Environment costs - Machine Configuration Costs (Load Generators) 1 GB - 500 Users example

**Let us suppose, 1 GB of RAM is available in a system –**

**Then using that system, only 100 Vusers can be generated when AJAX protocol is used and 500 Vusers can be generated when Web Protocol is used.**

If AJAX Protocol is used - 10 MB RAM space is used for 1 Vuser

If Web protocol is used - 2 MB RAM space is used for 1 Vuser.

**Recording Modes –**

There are two types of recording modes that are supported in Web Protocol. Recording modes are different for each protocol.

|  |  |  |
| --- | --- | --- |
| **Sl. No** | **HTML based recording (Mode is HTML)** | **URL based recording (Mode is HTTP)** |
| 1 | It generates the web Vuser functions for each user action on the web page. | It generates separate web Vuser functions for each non-html resources (ex: Images) on the web page. |
| 2 | It takes less time to execute the user actions as it captures less lines of code. | It takes more time to execute the user actions as it captures multiple web\_url () functions for all non-html resources. |
| 3 | It identifies the recording mode as HTML. | It identifies the recording mode as HTTP. |
| 4 | Functions in HTML –  web\_url (),  web\_submit\_data (),  web\_submit\_form (),  web\_image (),  web\_link() | Functions in HTTP –  web\_url (),  web\_submit\_data (),  web\_concurrent\_start (),  web\_concurrent\_end () |

**Correlation –**

It is the concept of handling the data which is generated by the server at run time.

There are two types of correlation methods that supports in load runner.

1. **Manual Correlation:**

Handle the server generated data by following the below steps without help of the automation features in the tool.

1. ***Identify the server generated value in script.***
   * Generate the two identical business scripts.
   * Go to Tools menu and select ‘Compare with Script’ option.
   * Open another script for the comparison.
   * The two scripts comparison will display under WDiff utility.
   * The differentiated values in the two scripts will be highlighted.
   * Identify the correlated value from the comparison.
2. ***Capture the new set of data which is generated by the server.***
3. Copy the identified correlated value
4. Go to Tree view and find out the copied value in server response of previous request.
5. Identify the left (LB) and right boundaries (RB) of the correlated value.
6. Identify the Ordinal value (ORD)
7. Insert the correlation function with all the arguments as below -

Web\_reg\_save\_param(“ParameterName”, “LB=”, “RB=”, “ORD=”,LAST);

// LB – Left Boundary, RB – Right Boundary

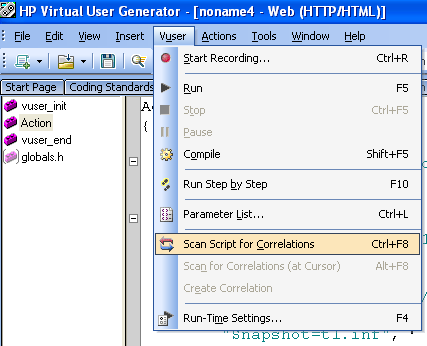
Ex: web\_reg\_save\_param("SessionID","LB=<input type=hidden name=userSession value=", "RB=>", "ORD=1", LAST);

1. ***Save it into a variable from the captured list.***
2. By default, it saves the value into the given parameter whenever the ORD is specific number.
3. We need to save the value into a separate variable whenever the ORD=ALL
4. To see that data is properly captured in a session variable, we can run the script by placing return 0; after the function where session value is generated. Before running the script, we have to set the following run-time settings – General Log🡪Select Extended Log radio button🡪Check Parameter Substitution checkbox.
5. ***Replace the identified original value with the variable at all the places in the script.***
6. Replace the variable at all the places of identified original value in the script.
7. **Automatic Correlation.**

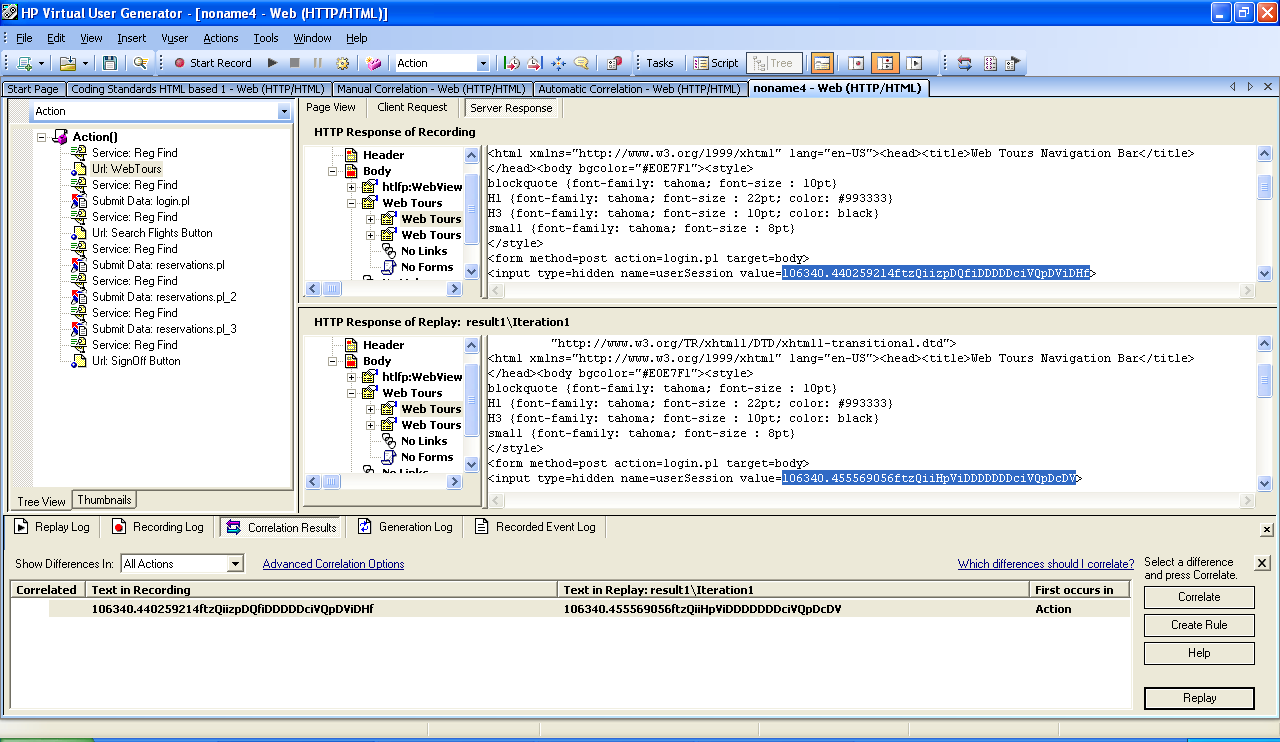
It is the concept of handling the correlated data with the help of tool functionalities in the script.

There are two types of automatic correlation –

1. **Scan Scripts for Correlations:**
   1. Generate script with end to end user actions.
   2. Run the script once and select ‘Scan Script for Correlations’ under Vuser menu option.



* 1. It shows the differentiate values between the recording and replay runs in Correlation Results.



* 1. Select the value to be correlated and Click Correlate button in the Correlation Results.
  2. It will automatically insert the correlated function with all the arguments.

// [WCSPARAM WCSParam\_Diff1 44 106340.440259214ftzQiizpDQfiDDDDDciVQpDViDHf] Parameter {WCSParam\_Diff1} created by Correlation Studio

web\_reg\_save\_param( "WCSParam\_Diff1",

"LB=userSession value=",

"RB=>",

"Ord=1",

"IgnoreRedirections=Yes",

"Search=Body",

"RelFrameId=1.2.1",

LAST );

1. **Using Correlation Rule:**
   1. Enable the correlation in recording options.
   2. Create the correlation rule with all the arguments – Left Boundary (LB), Right Boundary (RB) and ORD values.

// [WCSPARAM WCSParam\_Diff1 44 106340.440259214ftzQiizpDQfiDDDDDciVQpDViDHf] Parameter {WCSParam\_Diff1} created by Correlation Studio

web\_reg\_save\_param( "WCSParam\_Diff1",

"LB=userSession value=",

"RB=>",

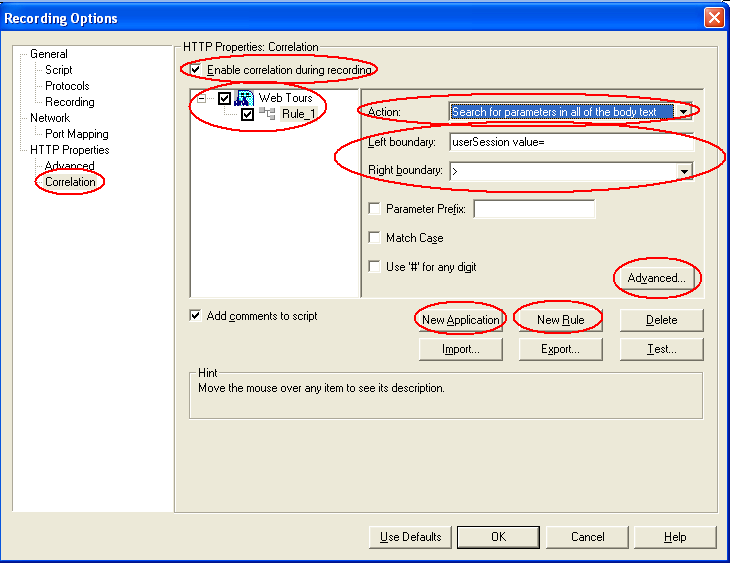
"Ord=1",

"IgnoreRedirections=Yes",

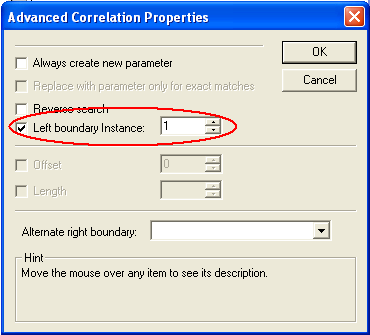
"Search=Body",

"RelFrameId=1.2.1",

LAST );



**Note:** Left Boundary Instance option (ORD value is set here) is enabled in Advanced Correlation Properties only after selecting ‘Search for parameters in links and form actions’ option in Action dropdown.



* 1. Generate the script with all end to end user actions.
  2. It will automatically insert the correlation function with all the arguments during the script recording.

**Correlation Function:** web\_reg\_save\_param()

**Arguments:**

1. **ParameterName**: It saves the corresponding correlated value.
2. **LB**: It is a left boundary of the correlated value.
3. **RB**: It is a right boundary of the correlated value.
4. **ORD**: It is the ordinal value of the correlated value. It defines the specific position or occurrence between the given left and right boundaries.

**ORD = 1**: It captures the first position of the value between the given left and right boundaries.

**ORD = 2**: It captures the second position of the value between the given left and right boundaries.

**ORD = 3**: It captures the third position of the value between the given left and right boundaries.

**ORD = all**: It captures all occurrences of the values between the given left and right boundaries.

**Correlation Function Additional Arguments:**

**LB/IC – Ignore Cases**

web\_reg\_save\_param("ViewState",

"LB=name=\"\_\_VIEWSTATE\" id=\"\_\_VIEWSTATE\" value=\"",

"RB=\" />",

"ORD=1",

“Search=Body”,

“RelFrameId=1.2.1”,

LAST);

1. **Search=Body**: Define the Search criteria to capture the correlated value (whether to search in the body or header. Default option is *Body*)
2. **RelFrameId**: Define the Frame Id to capture the correlated data from the web page.

**Note:** To get the *RelFrameId,* replay the script in a Full Log mode and find out the *RelFrameId* in a replay log. The user should know the Left Boundary to find the *RelFrameId* in Replay Log.

* + Search for the Left Boundary value in Replay Log and scroll up slowly to find the *RelFrameId* in the Replay Log.

**Handle the Dynamic Boundary values:**

Whenever you observe that the boundary values are changing dynamically, follow the below approaches to handle the dynamic boundary values –

1. *Text Flags:* Use the below two types of *Text Flags* to handle the dynamic boundary values.
2. *DIG:* When you observe the dynamic boundary characters are digits, use *DIG* (Digit) text flag and replace with ‘*#’* symbol at each place of dynamic digits.

Example:

//<option value="1">1</option>

Left Boundary = *<option value="1">*

Right Boundary = *</option>*

//<option value="11">11</option>

Left Boundary = *<option value="11">*

Right Boundary = *</option>*

//<option value="1999">1999</option>

Left Boundary = *<option value="1999">*

Right Boundary = *</option>*

In the above example, I would like to capture the multiple values using ORD=ALL. But, I observed that one digit is dynamically changing in the given left boundary. So, I have written the correlation function as below –

//<option value="1">1</option>

//LB/DIG - Left Boundary Digit

// # represents digits from 0 to 9

web\_reg\_save\_param("MonthSingleDigit8",

"LB/DIG=<option value=\"#\">",

"RB=</option>",

"ORD=ALL",

LAST);

//<option value="11">11</option>

web\_reg\_save\_param("MonthDoubleDigit8",

"LB/DIG=<option value=\"##\">",

"RB=</option>",

"ORD=ALL",

LAST);

//<option value="1999">1999</option>

web\_reg\_save\_param("YearDigit8",

"LB/DIG=<option value=\"####\">",

"RB=</option>",

"ORD=ALL",

LAST);

1. *ALNUM:* When you observe that the dynamic boundary characters are combination of alphanumeric, use *ALNUM* (ALphaNUMeric) text flag and replace with ‘*^’* symbol at each place of dynamic character.

Example:

//<option value="cgf1">cgf1</option>

Left Boundary = *<option value="cgf1">*

Right Boundary = *</option>*

In the above example, I observed *‘cgf1’* are the dynamic characters in the left boundary. So, I use the correlation function as below –

web\_reg\_save\_param("AlphaNumeric",

"LB/ALNUM=<option value=\"^^^^\">",

"RB=</option>",

"ORD=ALL",

LAST);

1. *IC:* This is used to Ignore the Cases.
2. *Using SaveLen and SaveOffSet:* Handle the dynamic boundaries using *SaveLen* and *SaveOffSet.*

*SaveLen:* It saves the given length of the characters into the correlated parameter.

*SaveOffSet:* It ignores the given length of the characters from the beginning of captured value.

Example:

// Without using LB/DIG text flag

//<option value="1">1</option>

1"> 🡪 it defines the number of characters to be ignored using *SaveOffSet.*

1 🡪 it defines the number of characters to be saved into the parameter using *SaveLen*.

//LB - Left Boundary

// SaveLen = 1, means 1 character needs to be saved.

// SaveOffSet = 3, means 3 characters are ignored before saving the characters using SaveLen

web\_reg\_save\_param("MonthSingleDigit8",

"LB/DIG=<option value=\"",

"RB=</option>",

"ORD=ALL",

"SaveLen=1",

"SaveOffSet=3",

LAST);

**Dynamic Item Data:**

In one of my application, I observed that the item data fields are dynamically changing.

* I compared the dynamic fields in the two scripts that are recorded for the same work flow.

When I recorded the script for the first time, one product contained only one item in the script. When I recorded the script for the second time with different product (by selecting different product), it contained 3 item fields in the script. So, to handle these two situations, I used web\_custom\_request function. (I observed the dynamic fields between the two scripts that are recorded between the same work flows)

* Build a Custom body with all the constant or static fields in the Submit\_Data (script)
* Capture the number of dynamic item fields using the correlation function with ORD=all.
* Iterate the loop based on the parameter array count (Dynamic Item Fields Count).
* Construct a custom body with all the static fields and dynamic fields.
* Use web\_custom\_request() function in the place of web\_submit\_data()

Example:

//name="\_\_VIEWSTATE" id="\_\_VIEWSTATE" value="

web\_reg\_save\_param("ViewState",

"LB=name=\"\_\_VIEWSTATE\" id=\"\_\_VIEWSTATE\" value=\"",

"RB=\" />",

"ORD=1",

LAST);

//name="\_\_EVENTVALIDATION" id="\_\_EVENTVALIDATION" value="

web\_reg\_save\_param("EventValidation",

"LB=name=\"\_\_EVENTVALIDATION\" id=\"\_\_EVENTVALIDATION\" value=\"",

"RB=\" />",

"ORD=1",

LAST);

//<input name="ctl00$ContentPlaceHolder1$gridSubproducts$ctl02$txtQty" type=

web\_reg\_save\_param("Items",

"LB=<input name=\"ctl00$ContentPlaceHolder1$gridSubproducts$",

"RB=$txtQty\" type=",

"ORD=all",

LAST);

lr\_save\_string(lr\_eval\_string("\_\_EVENTTARGET=&\_\_EVENTARGUMENT=&\_\_VIEWSTATE={ViewState}&\_\_VIEWSTATEENCRYPTED=&\_\_EVENTVALIDATION={EventValidation}&ctl00$ContentPlaceHolder1$ImgRegister.x=79&ctl00$ContentPlaceHolder1$ImgRegister.y=8"),"CustomBody");

a = lr\_paramarr\_len("Items");

for(i = 1; i<=a;i++)

{

temp = lr\_paramarr\_idx("Items",i);

lr\_save\_string(temp,"NewItem");

lr\_save\_string(lr\_eval\_string("{CustomBody}&ctl00$ContentPlaceHolder1$gridSubproducts${NewItem}$txtQty=2"),"CustomBody");

}

web\_custom\_request("Subproductview.aspx\_2",

"URL=http://192.168.1.43/onlineshopping/Subproductview.aspx?Kitchen&SubId=89",

//"Method=POST",

"body={CustomBody}",

LAST);

**Randomized Functions:**

To select a random value from the available array list, use the below types of random functions whenever ORD=all.

1. **lr\_paramarr\_random() :**

It selects one random value from the available array range.

Example: In my application. City list box contains list of values. So, I capture all the values using ORD=all. So, to select one random value from the correlated parameter array, I use this function.

1. **lr\_paramarr\_idx() :**

It selects the specific index from the list of array values.

1. **lr\_paramarr\_len() :**

It calculates the length of the given parameter array.

**Note: The above all array functions are available from LR 9.5. So, in earlier versions of Load Runner, we use the below function to satisfy the random behavior. (To achieve the random behavior)**

**In the below example –**

**Temp is a variable**

**City is a correlated parameter**

**i = rand()%20+1; // i holds the random number**

**sprint(Temp,”City\_%d”,i); // this is the method to get the random value from an array in previous versions of LR 9.5**

**Len = atoi(lr\_eval\_string(“city\_count”)); // this is the method to find the array length in previous versions of LR 9.5)**

**Parameterization –**

To submit the different set of user inputs, use Parameterization.

Combination of data that can be used while parameterization –

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Select Next Row:** | **Sequential** | **Random** | **Unique** |
| **Update Value On:** | ***Description*** | *The new parameter value will be updated sequentially.* | *The new parameter value will be selected randomly from the available list of values.* | *To use unique parameter value every time in the execution where the data cannot be reused.* |
| **Each Iteration** | *It will update a new parameter value for every iteration of the script execution.* | **Each Vuser accesses the file for each iteration and selects sequential value from file for each iteration.** | **Each Vuser accesses the file for each iteration and selects random value from file for each iteration.** | **Each Vuser accesses the file for each iteration and selects unique value from file for each iteration. (The selected value from file should be unique for all the Vusers in each iteration)** |
| **Each Occurrence** | *It will update a new ‘parameter value’ for every occurrence of the parameter file within the same iteration.* |  |  |  |
| **Once** | *It will update the same parameter value for all the iterations.* | **Each Vuser accesses the file once and selects sequential value from file once.** | **Each Vuser accesses the file once and selects random value from file once.** | **Each Vuser accesses the file once and selects unique value from file once. (The selected value from file should be unique for all the Vusers in different iterations)** |

This option is only used in case of single Vuser, For multi Vusers, this is not preferred.

**Sequential:**

The new parameter value will be updated sequentially.

**Random:**

The new parameter value will be selected randomly from the available list of values.

**Unique:**

To use unique parameter value every time in the execution where the data cannot be reused.

**Controller –**

In this component, we will design and execute the multi user test scenarios.

1. Upload all the scripts into the controller.
2. Define the load distribution for each script. (Controller will design the scenario and the script will be executed taking the help of Load Generator which is used to generate load (Vusers)).
3. Assign the Load Generator to generate the Vusers on server.
4. To use any machine as a load generator, Agent Process should be installed and running on that machine.
5. Load Generator can be connected either from Windows or UNIX.
6. In Web Protocol – Recommended memory for 1 Vuser is 2MB of RAM i.e. 1 Vuser requires 2MB of RAM. So, it is always better to allocate 50% of RAM for Vusers and remaining 50% for System Resources.

The recommended memory for each Vuser in Web (HTTP/HTML) protocol is 2MB of RAM.

The recommended memory for each Vuser in Citrix protocol is 10MB of RAM.

|  |  |
| --- | --- |
| **System RAM size** | **Recommended Vusers** |
| 1GB RAM | 250 Vusers |
| 2GB RAM | 500 Vusers |
| 4 GB RAM | 1000 Vusers |

1. We can add more than one load generators (multiple load generators) using the below two methods –
   * 1. Using Comma (,) Separator
     2. By re-browsing (uploading) the same script multiple times.
2. Define the ramp up.
3. *Simultaneous* – All the Vusers will be initialized on the server at the same time (simultaneously)
4. *Slow Ramp Up –* It initializes each Vuser on server with a given interval time.

NOTE: Always it is recommended to use slow ramp up.

1. Define the Steady state or duration.
2. *Run Until Completion –* It runs the Vuser on server until completing the given number of iterations.

Note: To design the scenario, using *Run Until Completion* we need to define the fixed number of iterations in run time settings.

1. *Run for (Specified Time) –* Define the time duration to run the Vuser on server.

Note: There is no need to define the iterations in run time settings because it will automatically iterate with possible number if iterations in the given time duration.

1. *Run indefinitely* (This option is available when only when we select *‘Basic Schedule’* Run Mode)
2. Define the Ramp down.
3. *Simultaneous*
4. *Slow Ramp Down*

Note: Always Simultaneous Ramp down is recommended to down the Vusers after the given duration.

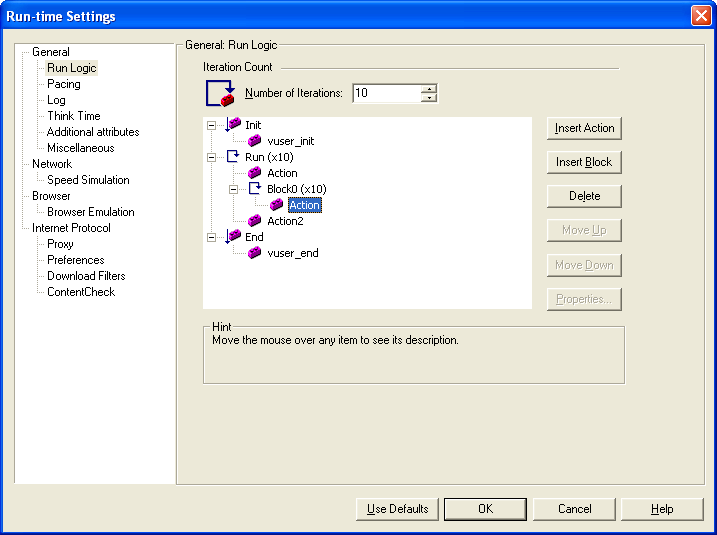
1. Select the Schedule type as below.
2. *Schedule By Scenario:* Design the schedule which is common for all the scripts.
3. *Schedule By Group:* Design the schedule for each script individually.
4. Define the Run-mode as below –
5. *Real-world schedule:* It supports to design the script with multiple ramp ups, multiple steady states and multiple ramp downs.
6. *Basic schedule:* It supports to design only with single ramp up, single steady state and single ramp down.
7. Define the standard run time settings as below –

***General Settings –***

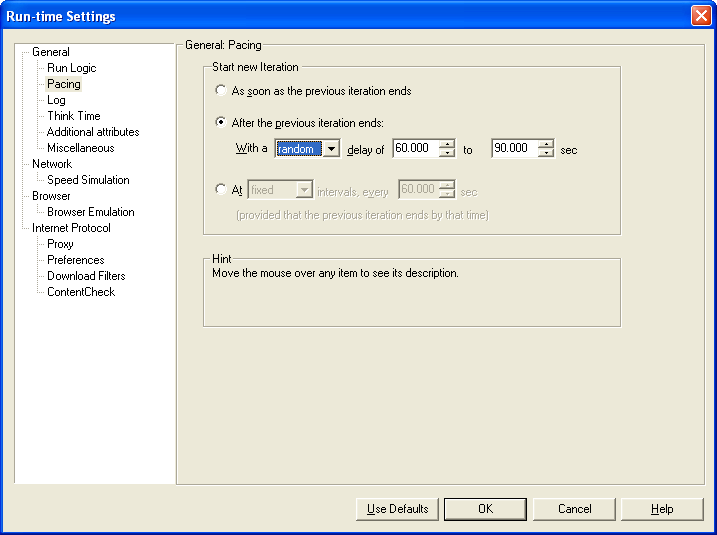
1. *Run Logic:* Configure the number of iterations to repeat the same user action by each Vuser.
2. *Main Iterations:* It repeats the user actions with the given number of times.

Note:Iterations are applicable only for action methods. (Actions where the scripting is done)

1. *Block Iterations:* Define the Block iterations to repeat each user action with different number of times.



1. *Pacing:* It is a user waiting time between the iteration executions.
2. *As soon as the previous iteration ends*
3. *After the previous iteration ends – with a ‘fixed/random’ delay of – secs.*
4. *At ‘fixed/random’ intervals, every – secs.*

**

1. *Log:* It captures the behavior of the script execution.

*Log Options:*

1. *Send Messages only when an error occurs*
2. *Always Send Messages.*

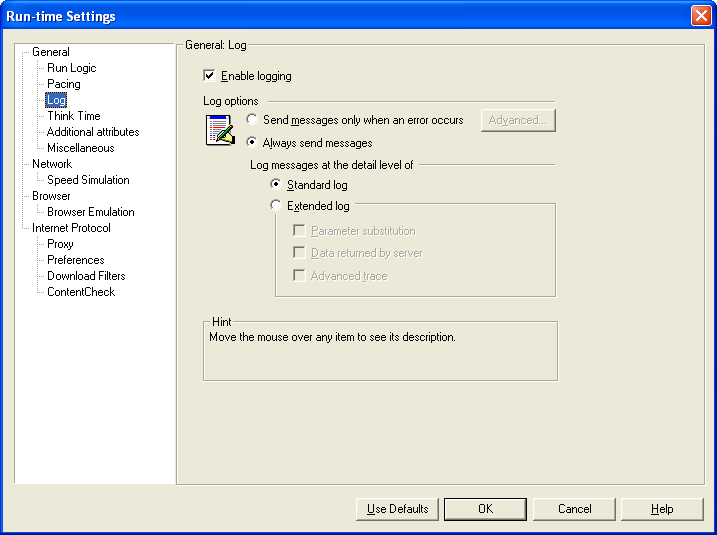
*Note:*

1. It is always recommended to use *‘Send Messages only when an error occurs’* during the load test execution.

2. If we place start and end transactions at the beginning and the end of the script to capture the complete Script response time, we get *‘wasted time’* with the end iteration statement in the log. This wasted time is because of the user waiting at the end the script to end the script end transaction.

*Logging levels –*

1. *Standard Log –*
2. *Extended Log –*
   1. *Parameter Substitution*
   2. *Data returned by server*
   3. *Advanced trace.*

**

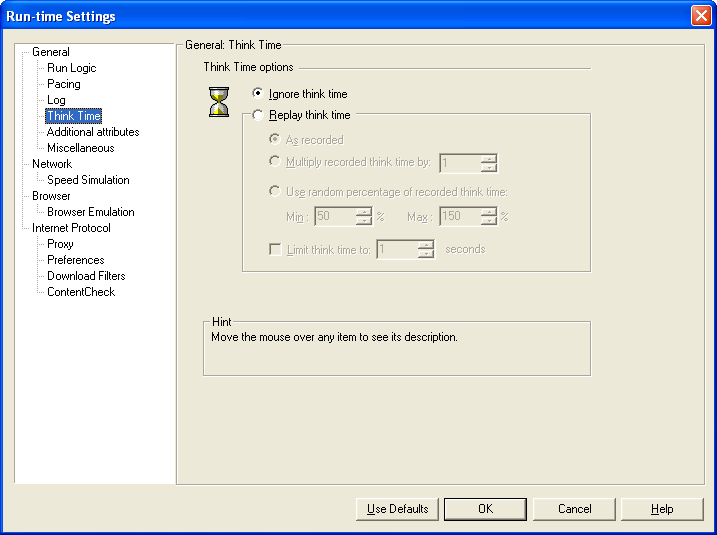
1. *Think Time: It is a user waiting time between the transaction executions.*

*Think Time Options:*

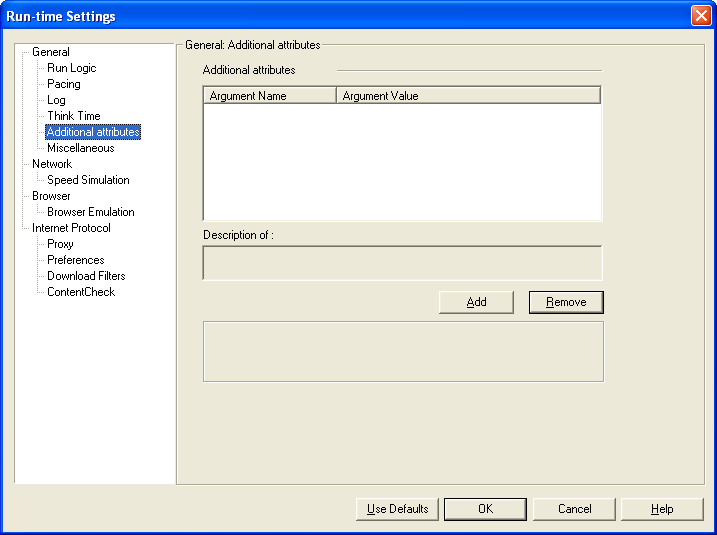
1. *Ignore Think Time*
2. *Replay Think Time*
3. *As Recorded*
4. *Multiply recorded think time by: --*
5. *Use random percentage of recorder think time:*

*Min: -- % Max: -- %*

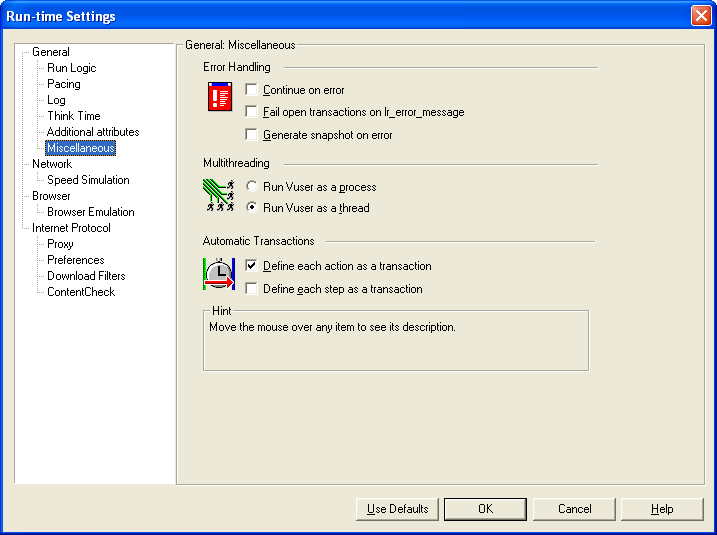
1. *Limit think time to – seconds.*

**

1. *Additional Attributes:*



1. Miscellaneous:



*Error Handling:*

1. Continue On Error
2. Fail open transactions on lr\_error\_message
3. Generate snapshot on error

Note: Always we need to check/select *Generate Snapshot on error* to capture error pages at runtime.

*Multi Threading:*

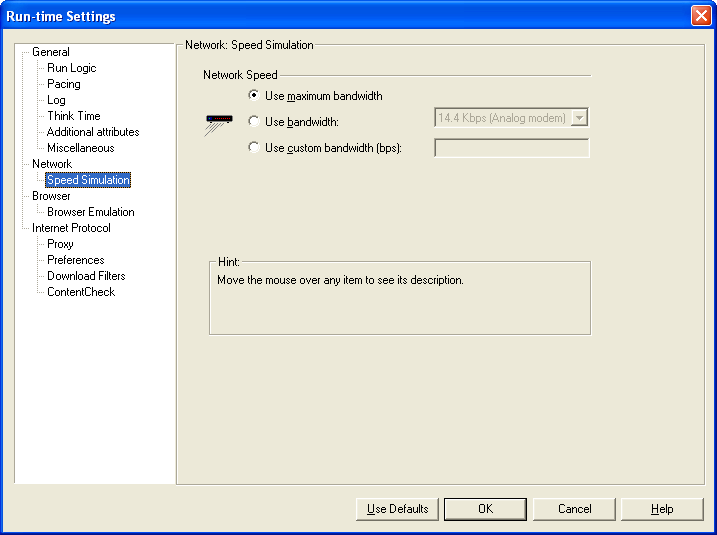
1. *Run Vuser as a process*
2. *For each Vuser it creates a separate driver program in the load generator.*
3. *It occupies more system resources in load generator as it created a separate driver program. (For each Vuser mdrv.exe)*
4. *Run Vuser as a thread*
5. *For all the Vusers, it creates a single driver program in the load generator.*
6. *It occupies less system resources as it is using a single driver program for all the Vusers (mdrv.exe)*

*Automatic Transaction:*

1. *Define each action as a transaction (Always disable this)*
2. *Define each step as a transaction*

***Network Settings –***

1. *Speed Simulation:*

**

In this we can select type of the bandwidth to use between load generator and server.

1. Use maximum bandwidth
2. Use bandwidth
3. Use custom bandwidth (bps):

Always recommended to select *Use maximum bandwidth* only.

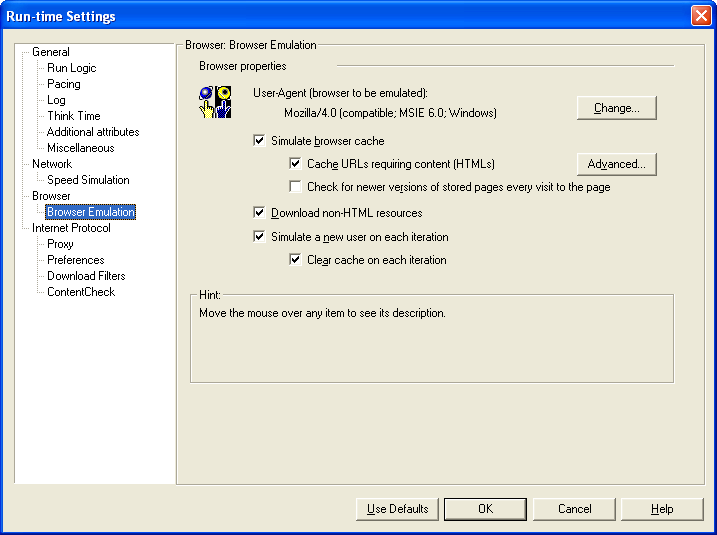
***Browser Settings –***

1. *Browser Emulation:*

In this we can do all the browser level enhancements.

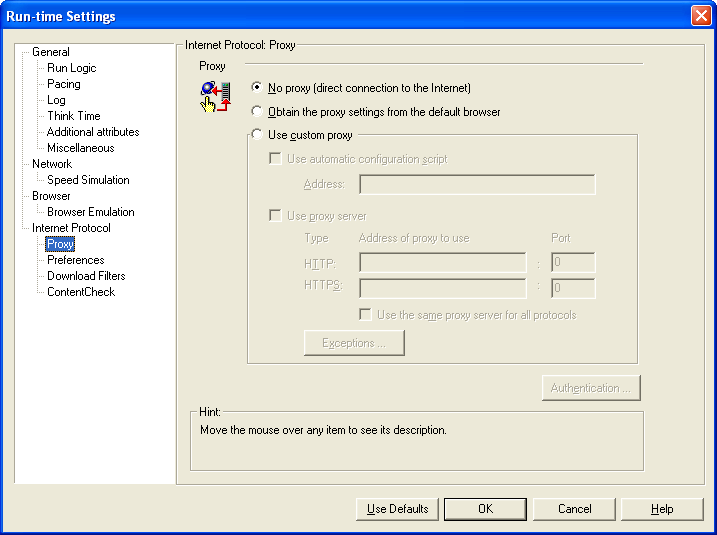
1. Simulate Browser Cache
2. Simulate a new user on each iteration

Note: Whenever we don’t want to reuse the browser cache, Enable S*imulate a new user on each iteration* and by clearing the existing cache on each iteration.

**

***Internet Protocol Settings –***

1. *Proxy:*

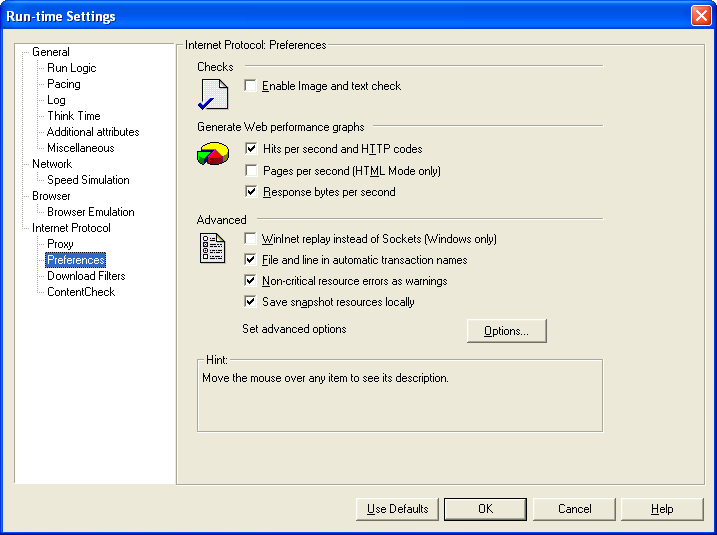
**

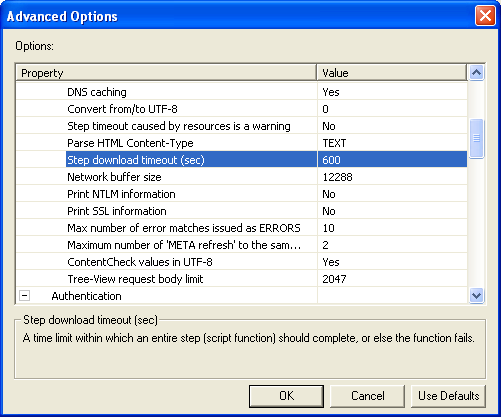
It is one type of middleware server to validate the requests and responses from one network to another network.

1. No Proxy (direct connection to the internet)
2. Obtain the proxy settings from the default browser
3. Use Custom Proxy

Note:

1. For the LAN (Load Generators and Servers are residing on the same network) based performance testing, it is recommended to use *No Proxy.*
2. For WAN based OR Internet based (Load Generators and Servers are residing on different network) Performance Testing, it is recommended to use *Obtain the proxy settings from the default browser.*
3. *Preferences:*

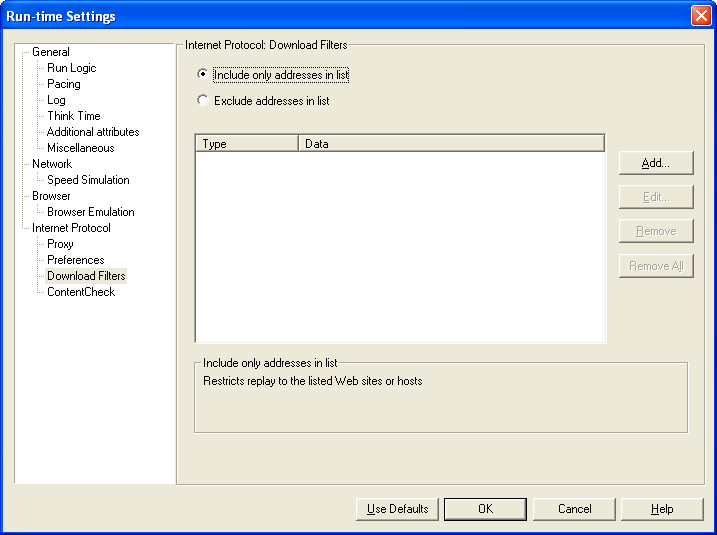
**

**

*Step download timeout (sec):* It is a waiting time by the Vuser to execute each Vuser request.

Note: By default, *Step download timeout* is 120 seconds. So, Increase the timeout seconds to 900 seconds.

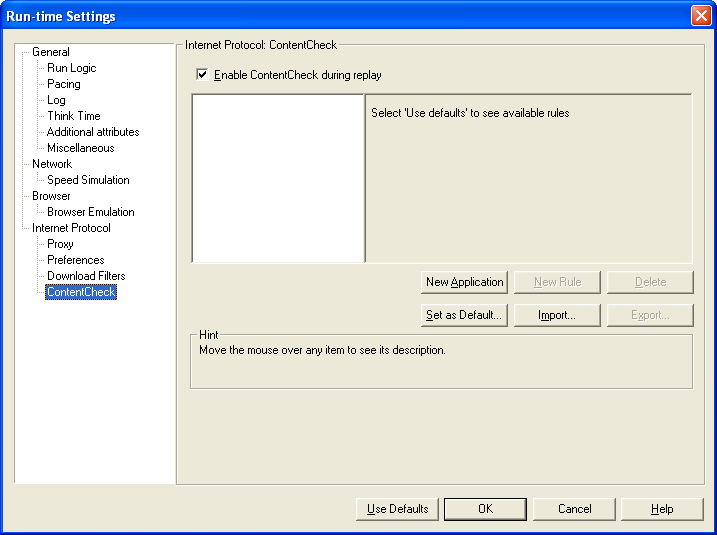
1. *Download filters:*

**

To avoid unnecessary URLs in the execution, use *download filters.*

Note: It is a replacement for commenting the script to avoid unnecessary URLs in the execution.

1. *Control Check*

**

This is also called *Global Check.* It is a global verification text checkpoint to validate the text on  
 all the web pages.

1. Save the results into the results directory (Results menu > Results Settings > Set Results Directory).

Results Naming Convention – <ApplicationName>\_<NoOfUsers>\_<TypeOfTest>\_<ExecutionDate>

Example: AB\_100Users\_LoadTest\_05092011

1. Enable *Auto Collate Results* in Results menu.

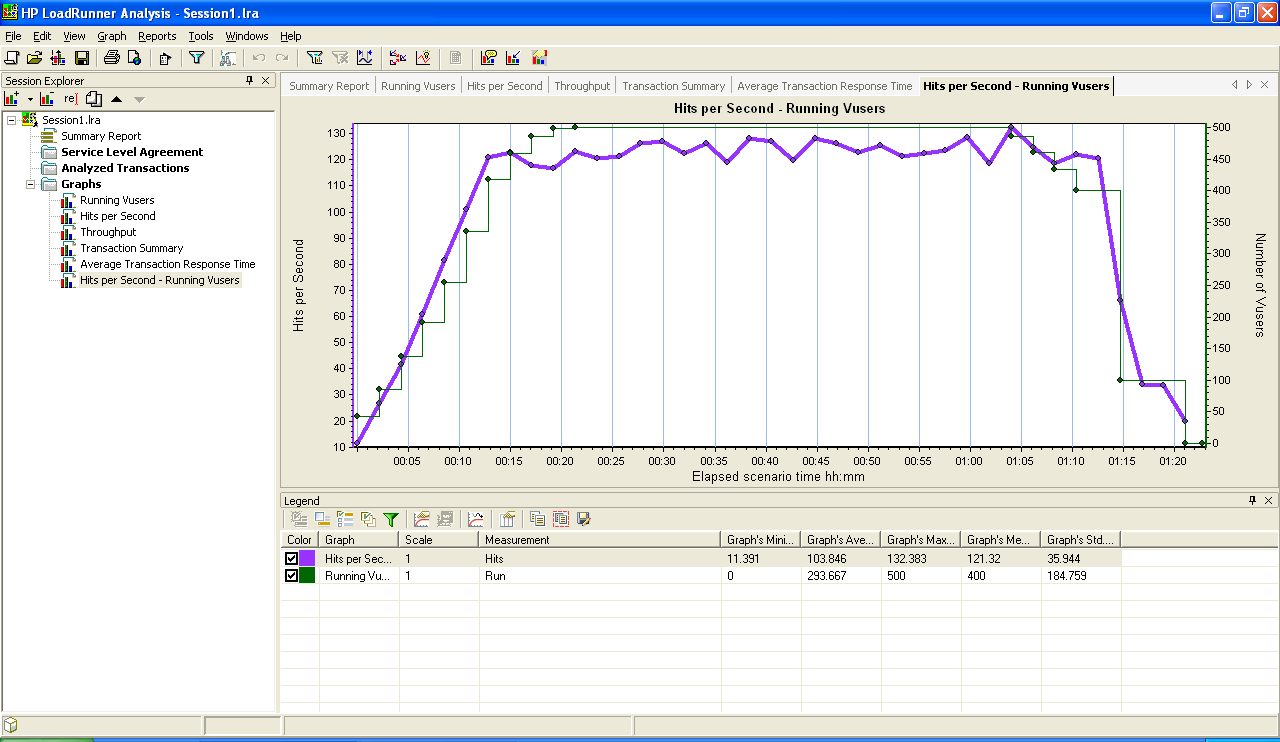
At runtime Vuser logs are stored at Load Generator and after the execution the logs should be placed into the results directory in the Controller. So, we need to always enable *Auto Collate Results* option in Results menu.

**NOTE:** If we place start and end transactions at the beginning and the end of the script to capture the complete Script response time, we get *‘wasted time’* with the end iteration statement in the log. This wasted time is because of the user waiting at the end the script to end the script end transaction.

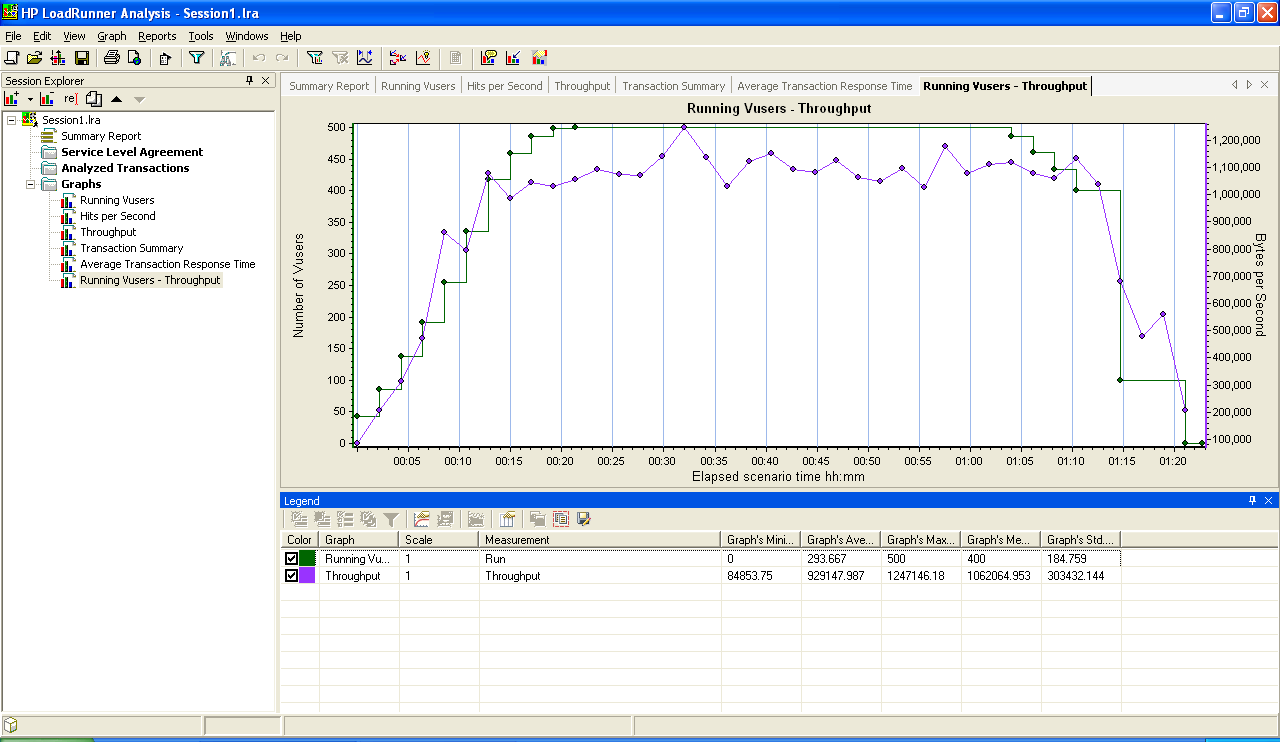
**Analysis –**

Analyze the below types of performance test results to identify the bottle necks of the application.

1. Client side metrics
2. Server side metrics
3. Client side Metrics Analysis: Analyze the below types of Client side metrics to identify the performance bottlenecks.
   1. *Response time*: Analyze each transaction response time with the given SLA.  
      (<=5sec in Arabian bank project SLA)
   2. *90th percent*: It is a highest transaction response time for 90% of passed transactions.
      * + 1. Generate the raw data on avg. transaction response time graph
          2. Select the transaction name for the analysis
          3. Sort all transaction response time in ascending order
          4. Calculate the 90th percent transaction count from the corresponding passed transactions.
          5. Find the 90th percent value at calculated 90% transaction count
   3. *Hits per second:* It measures the no. of requests made by the client to web server per second in the execution.   
      Observation: Hits per second is direct proportional with number of Vusers. When the load increases, hits also will increase and when the load decreases hits also will decrease.

**

* 1. Throughput: it shows the amount of data transferred from client to web server and web server to client.



Note: byte/sec to Mbps conversion:

1 byte = 8 bits

So X byte/sec = x\*8 bits/sec = (x\*8)/1024 Kpbs = (x\*8)/(1024)^2 Mbps

Example 933370.799 byte/sec = 7.121 Mbps

Note: Bytes/Sec conversion to Kbps OR Mbps

Kbps is Kilo bits per second

Mbps is Mega bits per second

1 Byte = 8 bits

So X bytes/sec conversion into kbps is –

Y = (X\*8)/1024

X bytes/sec conversion into Mbps is –

Z = Y/1024

Example: 933370.799 Bytes/Sec

(933370.799\*8)/1024 Kbps = 7291.9593671875 Kbps

(933370.799\*8)/(1024\*1024) Mbps = 7291.9593671875/1075 = 7.121 Mbps

*Throughput SLA:* The maximum bandwidth utilization can be 80% of total bandwidth.

Observation: I observed the average bandwidth utilization is 7.12 Mbps which is normal in the execution.

* 1. *Transactions per Second (TPS):*

Analyze average transactions per second rate with the expected number of transactions.

Note: For enhancing the TPS rate, we can customize the Think time and Pacing time in the scenario design.

1. Expected Transaction rate can be per sec/ per hour / per day / per week / per year
2. To increase TPS, we have to decrease the Think time and Pacing time.
3. To decrease TPS, we have to increase the Think time and Pacing time.

**Vuser Log Analysis:**

1. Analyze each Vuser log file to find out the root cause of user failed in the execution.
2. Analyze the snapshots of the failed users in Vuser log folders.

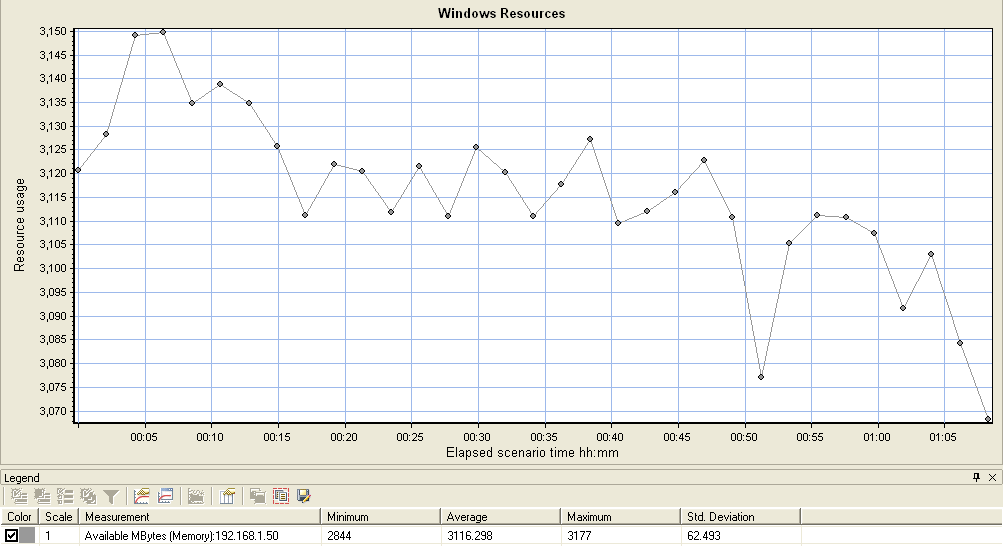
Note: To capture the error snapshots, we need to enable ‘*Generate Snapshot on Error’* in Run-Time settings during the test execution.

1. Server Side Metrics Analysis:

Note: Server User ID/Password – administrator/testqadmin

* 1. *Windows Level/System Level/Hardware Level:*

***Memory-Available Mbytes:***



***Description:*** Available MBytes is the amount of physical memory, in Megabytes, immediately available for allocation to a process or for system use. It is equal to the sum of memory assigned to the standby (cached), free and zero page lists. For a full explanation of the memory manager, refer to MSDN and/or the System Performance and Troubleshooting Guide chapter in the Windows Server 2003 Resource Kit.

Brief: it shows the amount of physical memory available during load test execution.

***Average (Avg):*** 3116.298

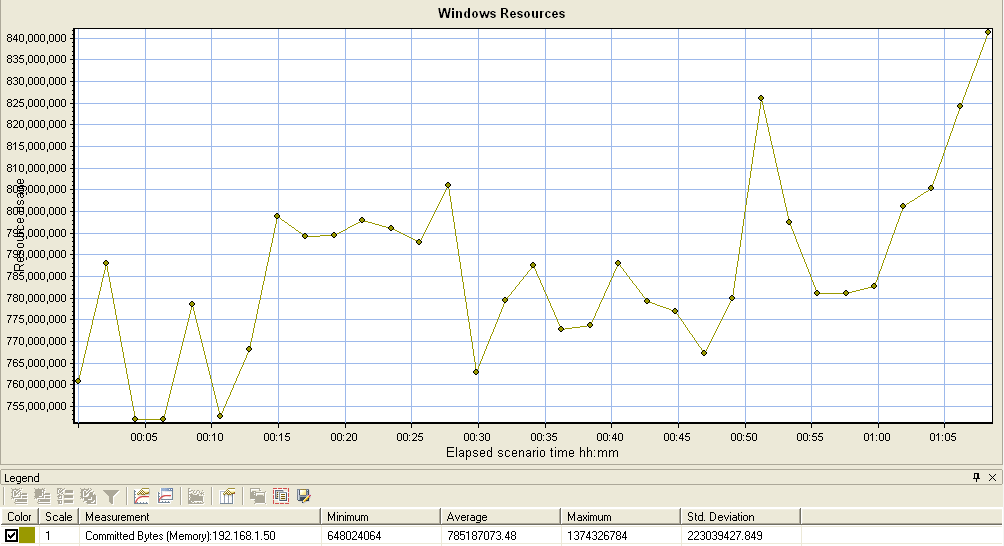
***Service Level Agreement (SLA):*** At least 20% of physical memory should be available during load test execution.

Total Configured System Memory is – 4GB (4096 MB)

Average Available Memory in execution is – 3116MB (76.08% of Total Configured Memory)

***Observation:*** I observed the average available Mbytes is 3116.29 MB (76%) which is normal in execution.

***Memory-Committed Bytes:***



***Description:*** *Committed Bytes is the amount of committed virtual memory, in bytes. Committed memory is the physical memory which has space reserved on the disk paging file(s). There can be one or more paging files on each physical drive. This counter displays the last observed value only; it is not an average.*

Brief: It shows the amount of virtual memory that is allocated during load test execution.

***Average (Avg):*** 785187073.48

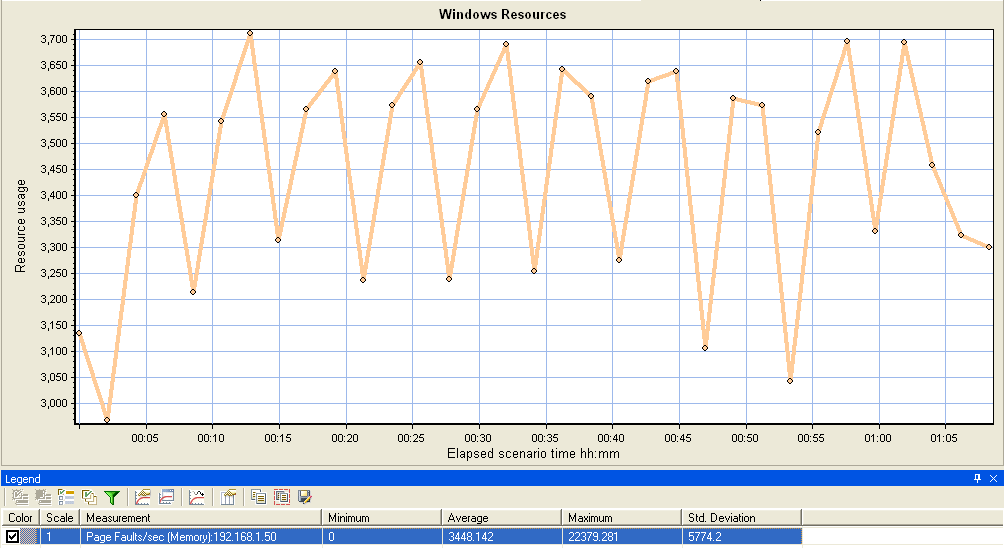
***Service Level Agreement (SLA):*** It can be allocated maximum of 80% of total virtual memory.

Total configured Virtual memory – 6GB (1.5 times of Available RAM = 1.5\*4GB = 6GB)

Average committed bytes in execution – 748.81 Mbytes (12.18% of Total Virtual memory)

***Observation:*** I observed the average allocated/Committed Mbytes is 748.81 Mbytes (12.18%) which is normal in execution.

***Memory-Page Faults/Sec:***



***Description:*** *Page Faults/sec is the average number of pages faulted per second. It is measured in number of pages faulted per second because only one page is faulted in each fault operation, hence this is also equal to the number of page fault operations. This counter includes both hard faults (those that require disk access) and soft faults (where the faulted page is found elsewhere in physical memory.) Most processors can handle large numbers of soft faults without significant consequence. However, hard faults, which require disk access, can cause significant delays.*

Brief: Whenever the required pages are not found in paged memory is called it as page fault.

Soft Faults – The faulted pages are found within the physical memory.

Hard Faults – The faulted pages are not found in physical memory so that it has to fetch from Virtual memory (Hard disk memory).

Hard Faults/Sec = Page Reads/sec + Pages Input/sec

= 0.065 + 0.494

= 0.559

Soft Faults/Sec = Page Faults/sec – Hard Faults/sec

= 3448.142 – 0.559

= 3447.583

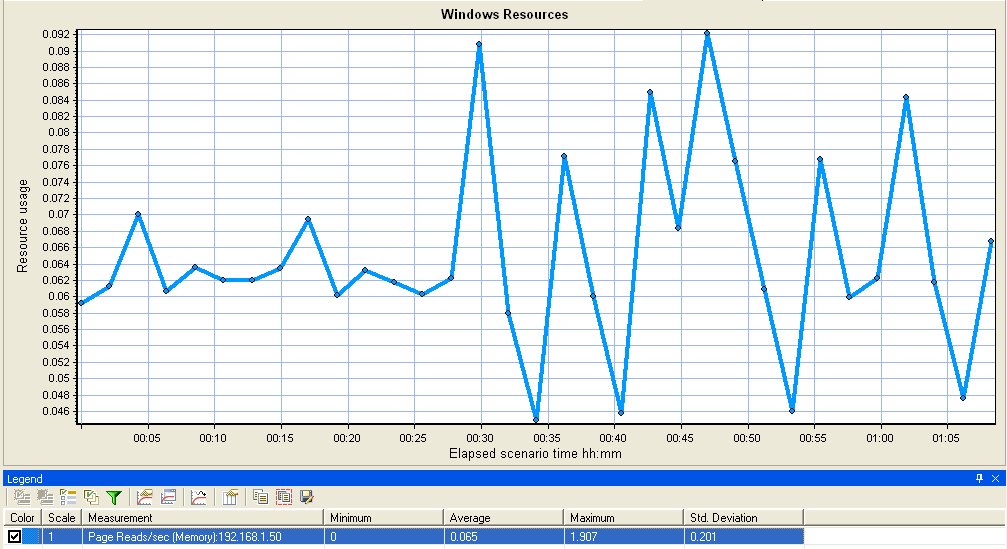
***Average (Avg):*** 3448.142

***Service Level Agreement (SLA):*** Always Hard Page Faults/sec rate should be less than the soft page faults/sec rate.

Note: Whenever more number of hard faults occurs in the execution, it will cause system delays.

***Observation:*** Hard Page Faults/sec is less than the Soft Page Faults/sec.

***Memory-Page Reads/Sec:***



***Description:*** *Page Reads/sec is the rate at which the disk was read to resolve hard page faults. It shows the number of reads operations, without regard to the number of pages retrieved in each operation. Hard page faults occur when a process references a page in virtual memory that is not in working set or elsewhere in physical memory, and must be retrieved from disk. This counter is a primary indicator of the kinds of faults that cause system-wide delays. It includes read operations to satisfy faults in the file system cache (usually requested by applications) and in non-cached mapped memory files. Compare the value of Memory\\Pages Reads/sec to the value of Memory\\Pages Input/sec to determine the average number of pages read during each operation.*

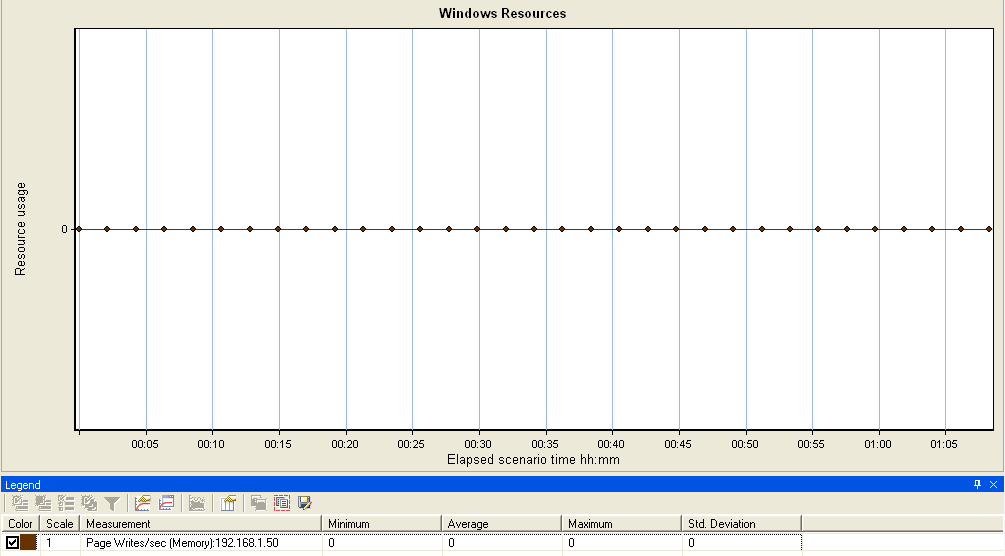
Brief: It shows the number of read operations per second to copy the pages from Virtual memory to Physical memory to resolve hard faults in the execution.

***Average (Avg):*** 0.065

***Service Level Agreement (SLA):***

***Observation:*** *Here, 0.065 Page reads/sec are observed. i.e. On an average, 0.065 page reads are done from Virtual memory to Physical memory per second.*

***Memory-Page Writes/Sec:***



***Description:*** *Page Writes/sec is the rate at which pages are written to disk to free up space in physical memory. Pages are written to disk only if they are changed while in physical memory, so they are likely to hold data, not code. This counter shows write operations, without regard to the number of pages written in each operation. This counter displays the difference between the values observed in the last two samples, divided by the duration of the sample interval.*

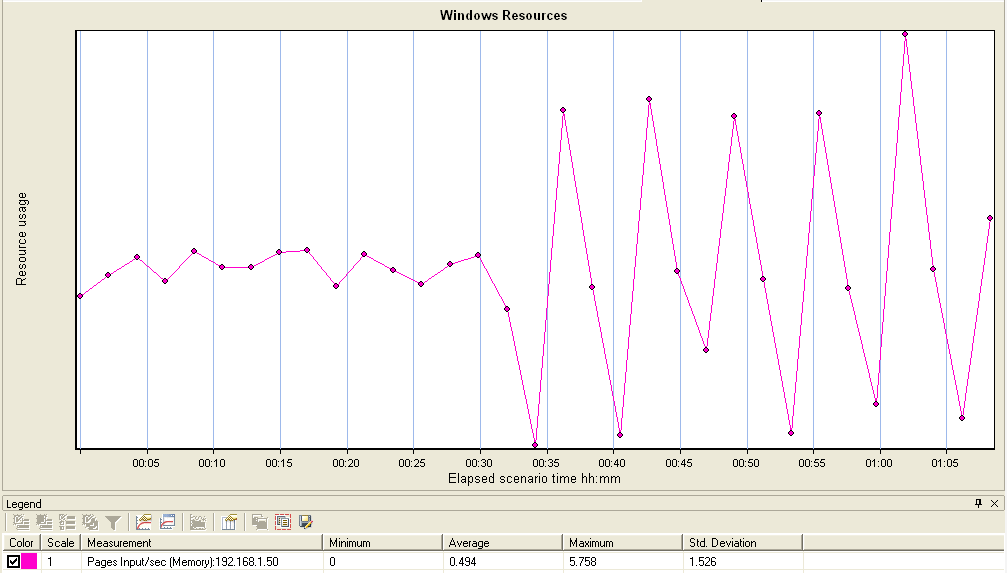
Brief: It shows the number of pages that are to be written back to disk (virtual memory) after resolving the hard page faults to free up the memory in physical memory (RAM).

***Average (Avg):*** 0

***Service Level Agreement (SLA):***

***Observation:*** *Here, Pages are not written to the disk (Virtual Memory). Hence, the average value is 0.*

***Memory-Pages Input/Sec:***



***Description:*** *Pages Input/sec is the rate at which pages are read from disk to resolve hard page faults. Hard page faults occur when a process refers to a page in virtual memory that is not in its working set or elsewhere in physical memory, and must be retrieved from disk. When a page is faulted, the system tries to read multiple contiguous pages into memory to maximize the benefit of the read operation. Compare the value of Memory\\Pages Input/sec to the value of Memory\\Page Reads/sec to determine the average number of pages read into memory during each read operation.*

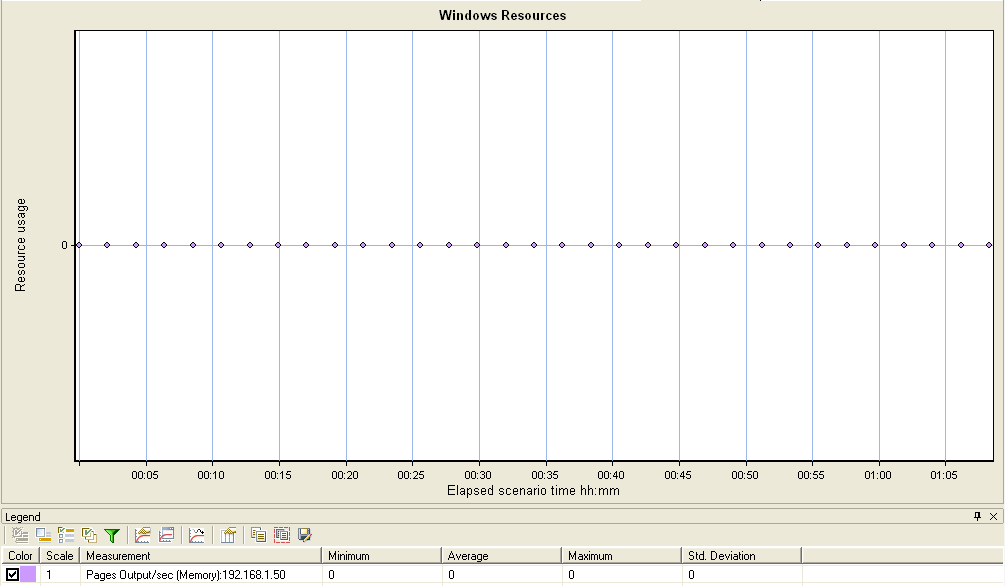
Brief: It shows the number of read operations per second to copy the pages from Virtual memory to Physical memory to resolve hard faults in the execution with regards to the multiple pages in each read operation.

***Average (Avg):*** 0.494

***Service Level Agreement (SLA):***

***Observation:***

***Memory-Pages Output/Sec:***



***Description:*** *Pages Output/sec is the rate at which pages are written to disk to free up space in physical memory. Pages are written back to disk only if they are changed in physical memory, so they are likely to hold data, not code. A high rate of pages output might indicate a memory shortage. Windows writes more pages back to disk to free up space when physical memory is in short supply. This counter shows the number of pages, and can be compared to other counts of pages, without conversion.*

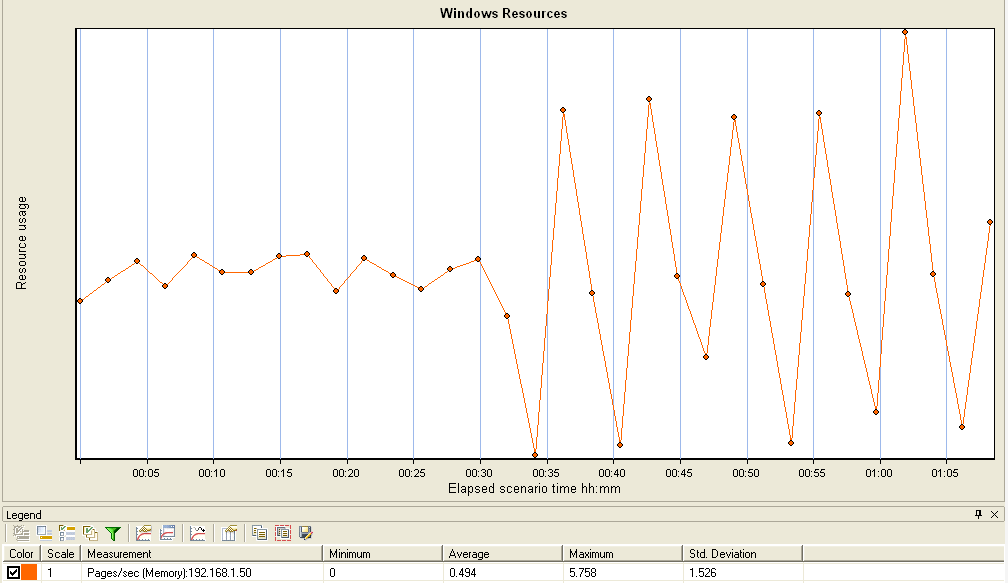
Brief: It shows the number of pages that are to be written back to disk (virtual memory) after resolving the hard page faults to free up the memory in physical memory (RAM) with regards to the multiple pages in each write operation.

***Average (Avg):*** 0

***Service Level Agreement (SLA):***

***Observation:***

***Memory-Pages/Sec:***



***Description:*** *Pages/sec is the rate at which pages are read from or written to disk to resolve hard page faults. This counter is a primary indicator of the kinds of faults that cause system-wide delays. It is the sum of Memory\\Pages Input/sec and Memory\\Pages Output/sec. It is counted in numbers of pages, so it can be compared to other counts of pages, such as Memory\\Page Faults/sec, without conversion. It includes pages retrieved to satisfy faults in the file system cache (usually requested by applications) non-cached mapped memory files.*

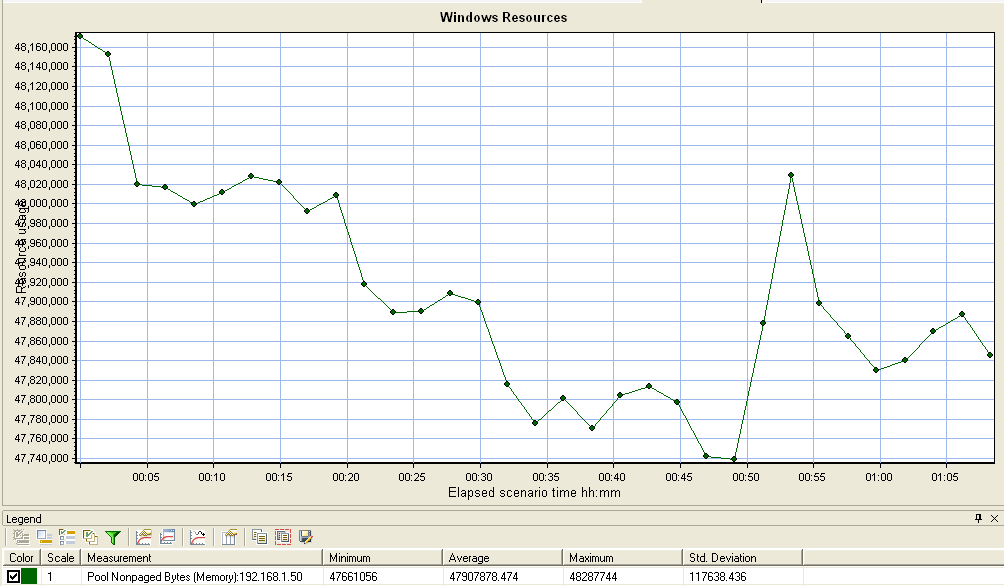
Brief:

***Average (Avg):*** 0.494

***Service Level Agreement (SLA):***

***Observation:***

***Memory-Pool/Non paged Bytes:***



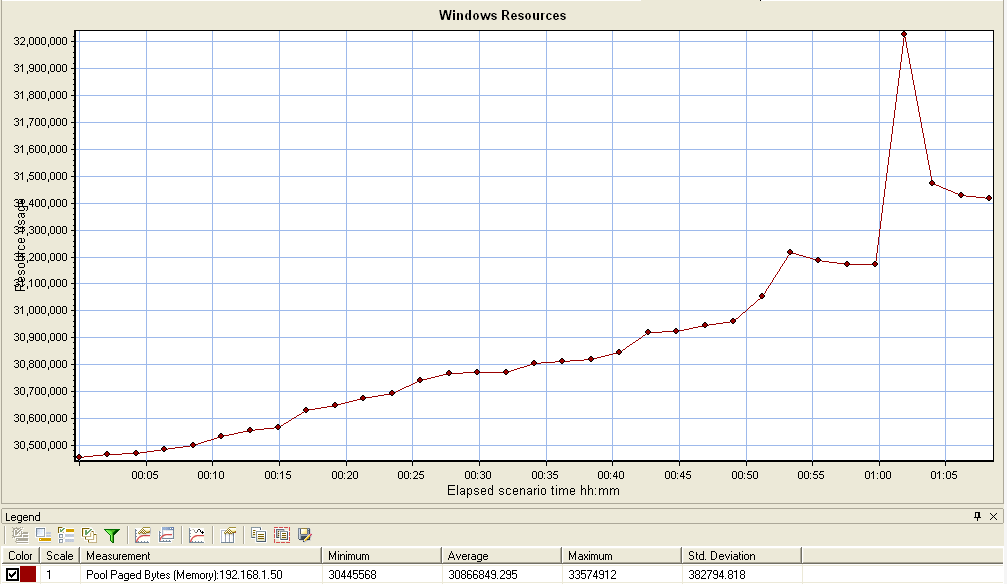
***Description:*** *Pool Nonpaged Bytes is the size, in bytes, of the nonpaged pool, an area of system memory (physical memory used by the operating system) for objects that cannot be written to disk, but must remain in physical memory as long as they are allocated. Memory\\Pool Nonpaged Bytes is calculated differently than Process\\Pool Nonpaged Bytes, so it might not equal Process\\Pool Nonpaged Bytes\\\_Total. This counter displays the last observed value only; it is not an average.*

***Average (Avg):*** 47907878.474

***Service Level Agreement (SLA):***

***Observation:***

***Memory-Pool/Paged Bytes:***



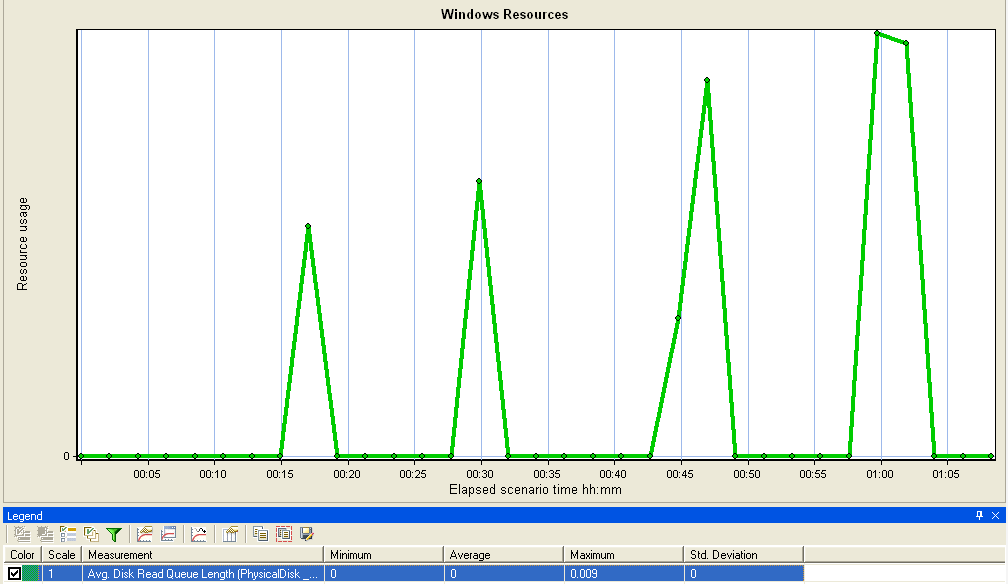
***Description:*** *Pool Paged Bytes is the size, in bytes, of the paged pool, an area of system memory (physical memory used by the operating system) for objects that can be written to disk when they are not being used. Memory\\Pool Paged Bytes is calculated differently than Process\\Pool Paged Bytes, so it might not equal Process\\Pool Paged Bytes\\\_Total. This counter displays the last observed value only; it is not an average.*

***Average (Avg):*** 30866849.295

***Service Level Agreement (SLA):***

***Observation:***

***PHYSICAL DISK-Avg. Disk Read Queue Length:***



***Description:*** *Avg. Disk Read Queue Length is the average number of read requests that were queued for the selected disk during the sample interval.*

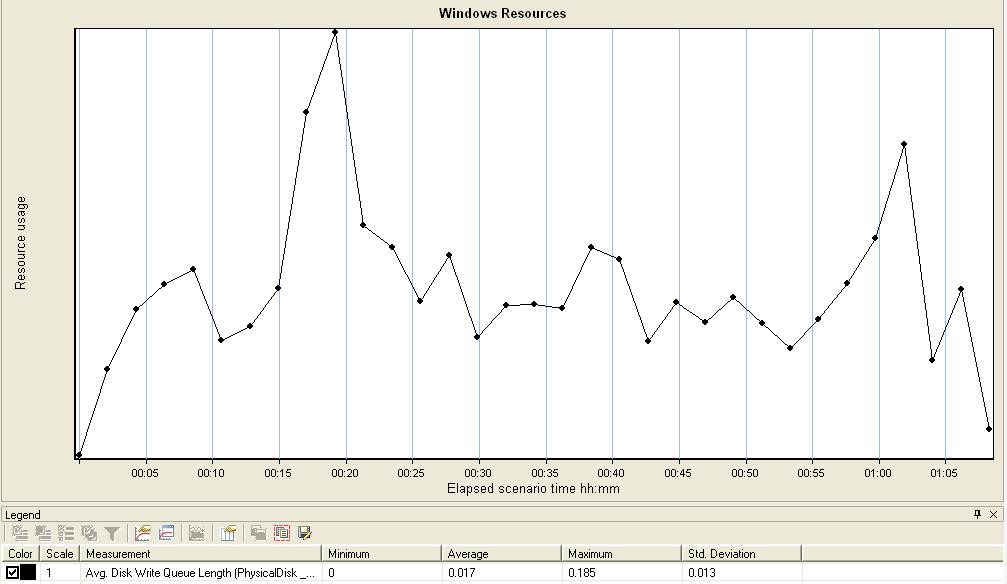
Brief: It shows the number of requests that are in the queue at hard disk while doing a read operation.

***Average (Avg):*** 0

***Service Level Agreement (SLA):*** Always Disk Read Queue Length should be less than 2.

***Observation:*** Average Disk Read Queue Length is 0 (<2) in this case which means that there are no requests in the queue.

***PHYSICAL DISK-Avg. Disk Write Queue Length:***



***Description:*** *Avg. Disk Write Queue Length is the average number of write requests that were queued for the selected disk during the sample interval.*

Brief: It shows the number of requests that are in the queue at hard disk while doing a write operation.

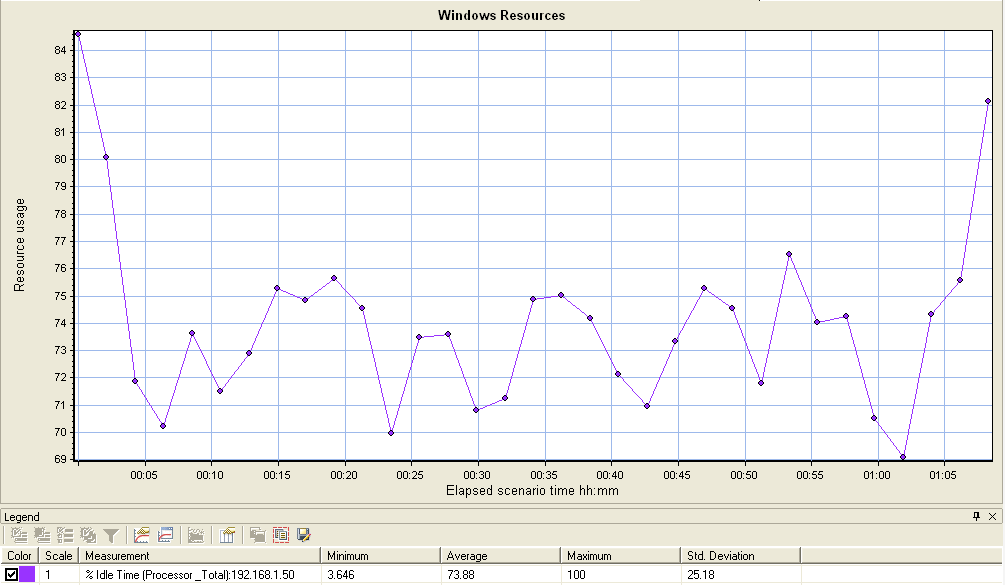
***Average (Avg):*** 0

***Service Level Agreement (SLA):*** Always Disk Write Queue Length should be less than 2.

***Observation:*** Average Disk Read Write Length is 0 (<2) in this case which means that there are no requests in the queue.

***PROCESSOR-%Idle Time:***

1. ***Total***



***Description:*** *% Idle Time is the percentage of time the processor is idle during the sample interval.*

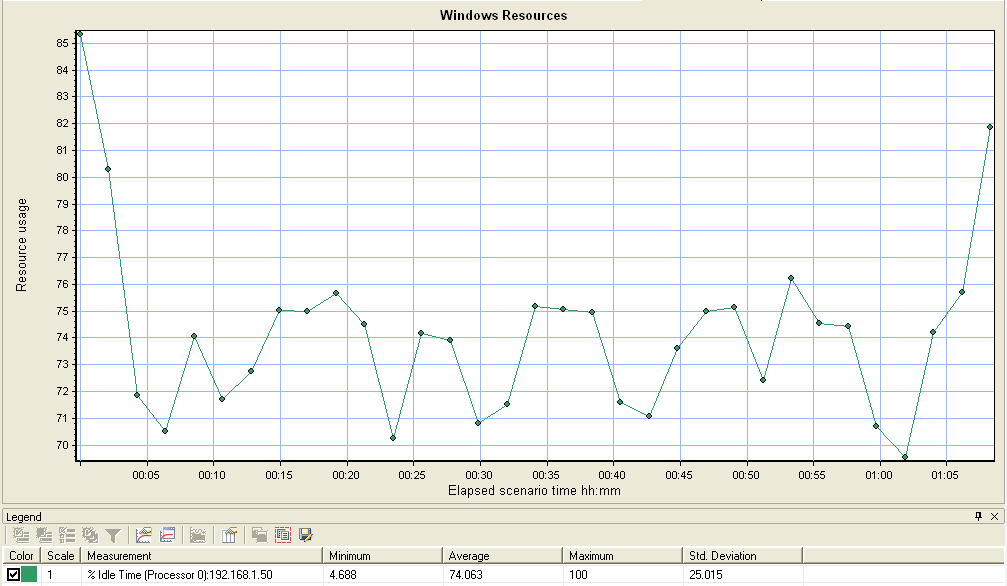
Brief: It shows the percentage of CPU which is free or idle during the test execution.

***Average (Avg):*** 73.88

***Service Level Agreement (SLA):*** At least, 20% of the CPU should be free or idle during the execution. (Cumulative of all the processors)

***Observation:*** Here, 73.88 is the Average % Idle Time. It is way above the threshold (20%).

1. ***0***



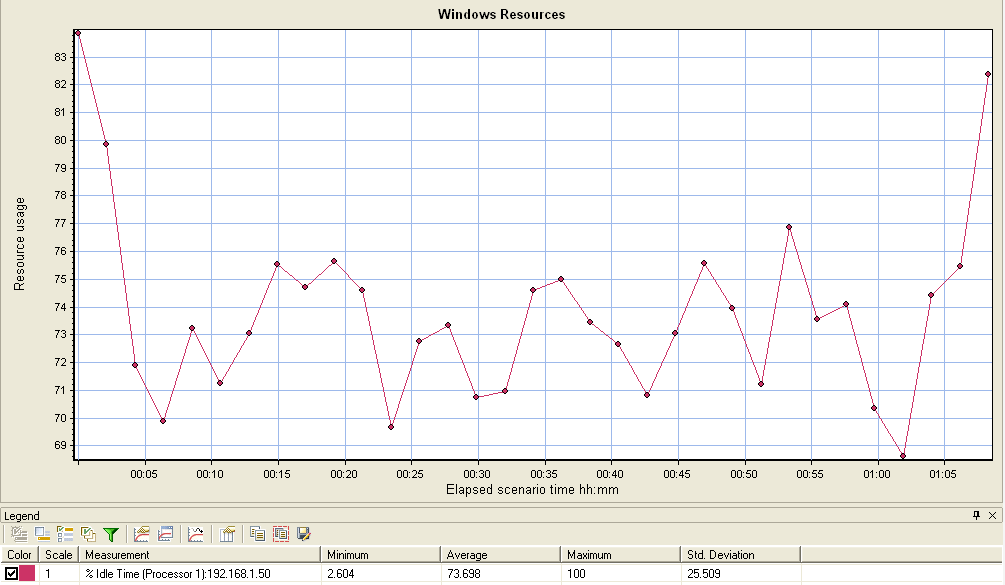
***Description:*** *% Idle Time is the percentage of time the processor is idle during the sample interval*

***Average (Avg):*** 74.063

***Service Level Agreement (SLA):***

***Observation:***

1. ***1***



***Description:*** *% Idle Time is the percentage of time the processor is idle during the sample interval*

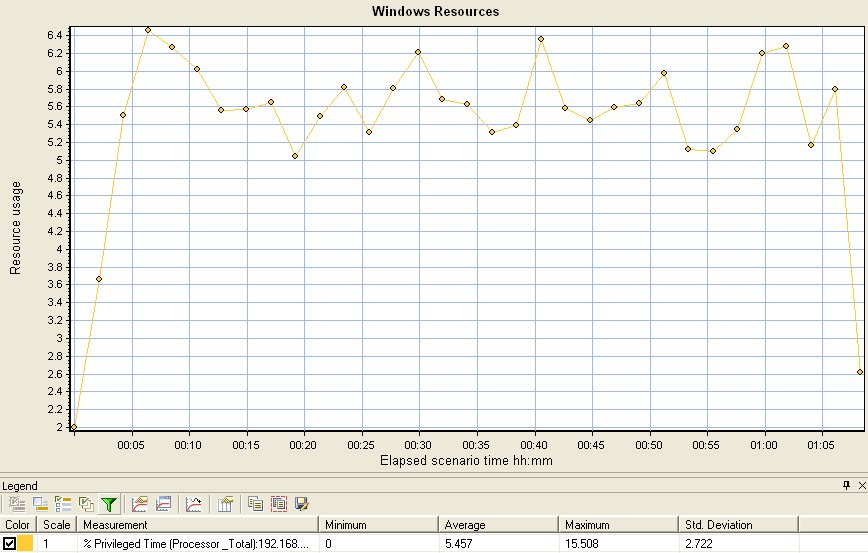
***Average (Avg):*** 73.698

***Service Level Agreement (SLA):***

***Observation:***

***Processor – %Privileged Time:***

1. ***Total***

******

***Description:*** *% Privileged Time is the percentage of elapsed time that the process threads spent executing code in privileged mode. When a Windows system service in called, the service will often run in privileged mode to gain access to system-private data. Such data is protected from access by threads executing in user mode. Calls to the system can be explicit or implicit, such as page faults or interrupts. Unlike some early operating systems, Windows uses process boundaries for subsystem protection in addition to the traditional protection of user and privileged modes. Some work done by Windows on behalf of the application might appear in other subsystem processes in addition to the privileged time in the process.*

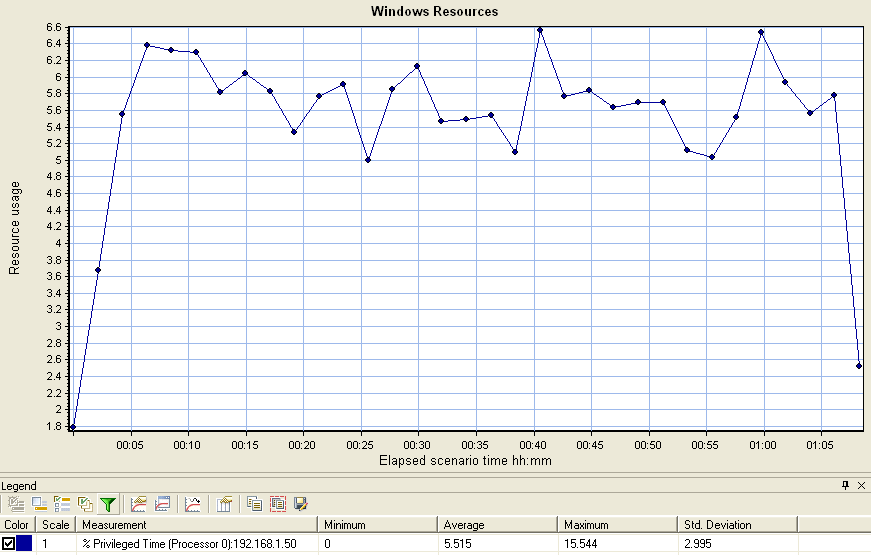
Brief: It shows the percentage of CPU utilized to execute all the processes running by the system mode or privileged mode.

***Average (Avg):***5.457

***Service Level Agreement (SLA):*** Always, % Privileged Time should be <= 40% (Cumulative of all the processors)

***Observation:*** Here % Privileged Time is 5.457 (<40%). This is an expected value.

1. ***0***

******

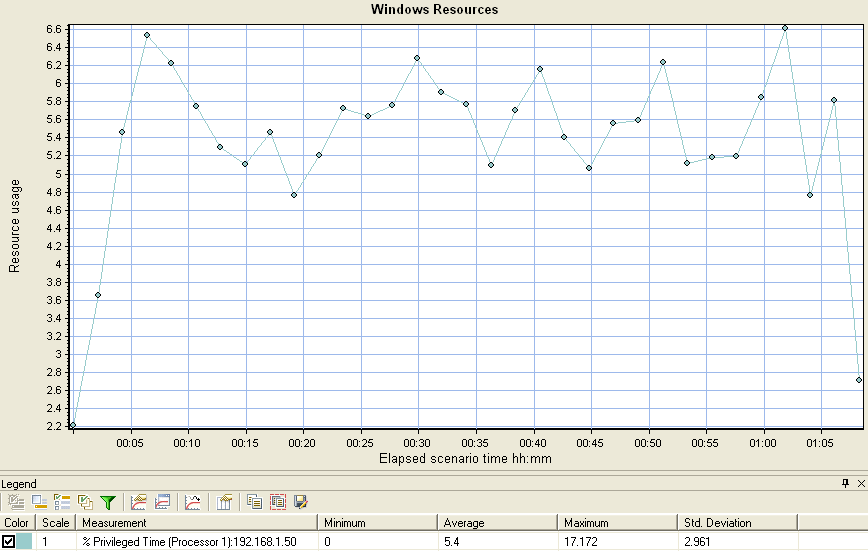
***Description:*** *% Privileged Time is the percentage of elapsed time that the process threads spent executing code in privileged mode. When a Windows system service in called, the service will often run in privileged mode to gain access to system-private data. Such data is protected from access by threads executing in user mode. Calls to the system can be explicit or implicit, such as page faults or interrupts. Unlike some early operating systems, Windows uses process boundaries for subsystem protection in addition to the traditional protection of user and privileged modes. Some work done by Windows on behalf of the application might appear in other subsystem processes in addition to the privileged time in the process.*

***Average (Avg):***5.515

***Service Level Agreement (SLA):***

***Observation:***

1. ***1:***

******

***Description:*** *% Privileged Time is the percentage of elapsed time that the process threads spent executing code in privileged mode. When a Windows system service in called, the service will often run in privileged mode to gain access to system-private data. Such data is protected from access by threads executing in user mode. Calls to the system can be explicit or implicit, such as page faults or interrupts. Unlike some early operating systems, Windows uses process boundaries for subsystem protection in addition to the traditional protection of user and privileged modes. Some work done by Windows on behalf of the application might appear in other subsystem processes in addition to the privileged time in the process.*

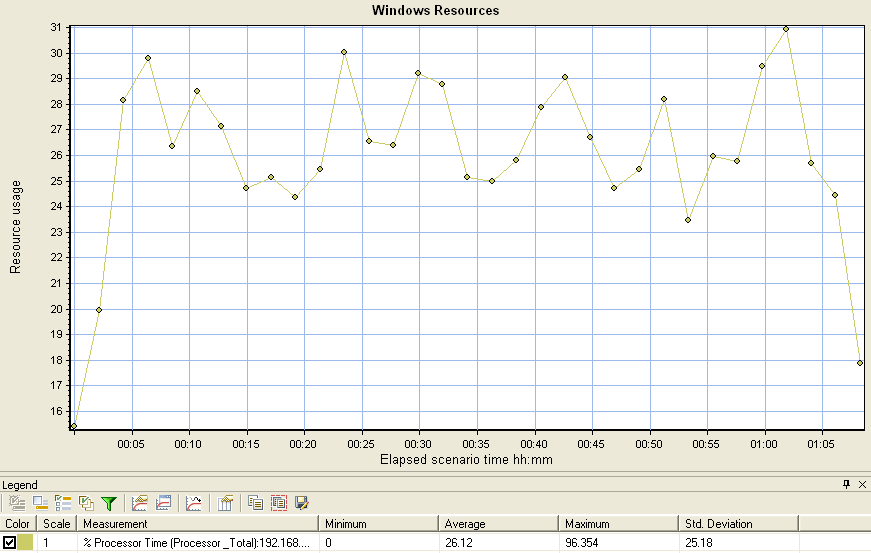
***Average (Avg):* 5.4**

***Service Level Agreement (SLA):***

***Observation:***

***Processor – %Processor Time:***

1. ***Total***

******

***Description:*** *% Processor Time is the percentage of elapsed time that the processor spends to execute a non-Idle thread. It is calculated by measuring the duration of the idle thread is active in the sample interval, and subtracting that time from interval duration. (Each processor has an idle thread that consumes cycles when no other threads are ready to run). This counter is the primary indicator of processor activity, and displays the average percentage of busy time observed during the sample interval. It is calculated by monitoring the time that the service is inactive, and subtracting that value from 100%.*

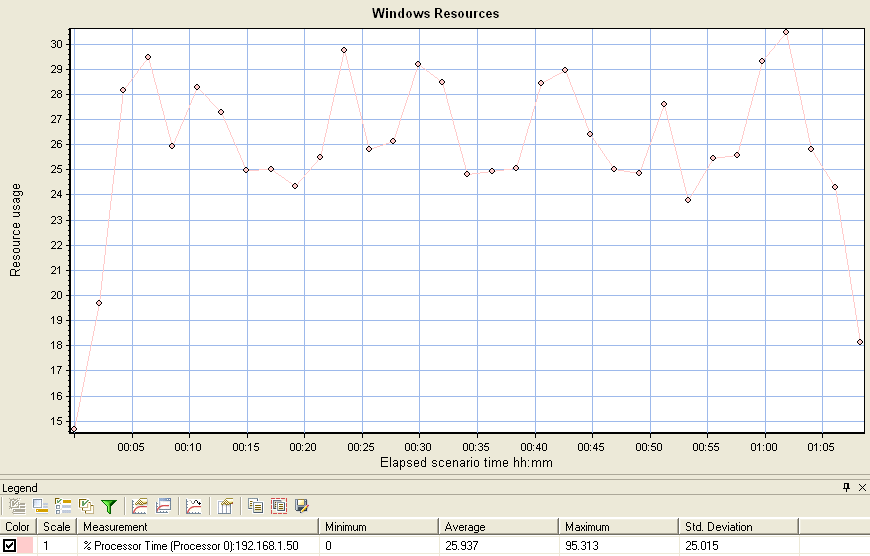
Brief: It shows the % of CPU used to run all the process running on the system (System level, user level, network level, local users. etc)

***Average (Avg):***26.12

***Service Level Agreement (SLA):*** % processor time should be less than are equal to 80% of CPU utilization

***Observation:*** I observed that average value is 26.12 so it is normal in execution.

1. **0**

******

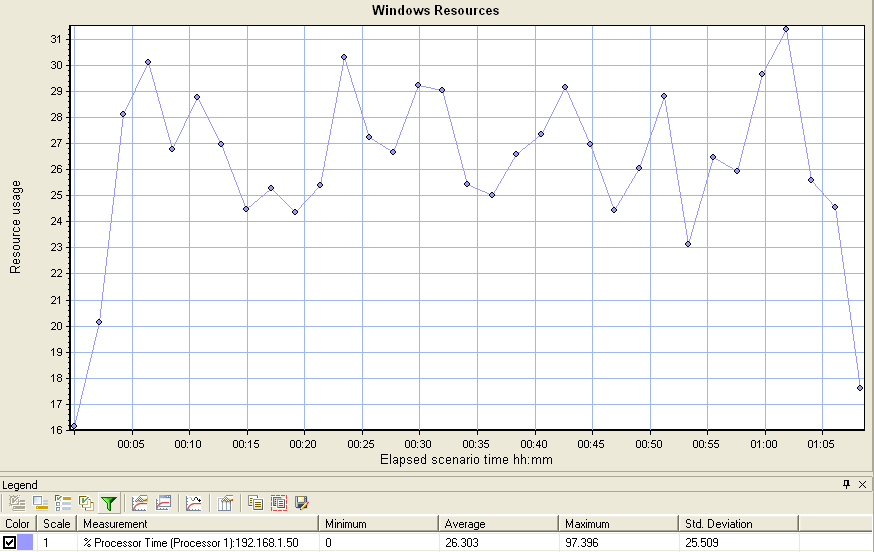
***Description:*** *% Processor Time is the percentage of elapsed time that the processor spends to execute a non-Idle thread. It is calculated by measuring the duration of the idle thread is active in the sample interval, and subtracting that time from interval duration. (Each processor has an idle thread that consumes cycles when no other threads are ready to run). This counter is the primary indicator of processor activity, and displays the average percentage of busy time observed during the sample interval. It is calculated by monitoring the time that the service is inactive, and subtracting that value from 100%.*

***Average (Avg):***25.937

***Service Level Agreement (SLA):***

***Observation:***

1. ***1 :***

******

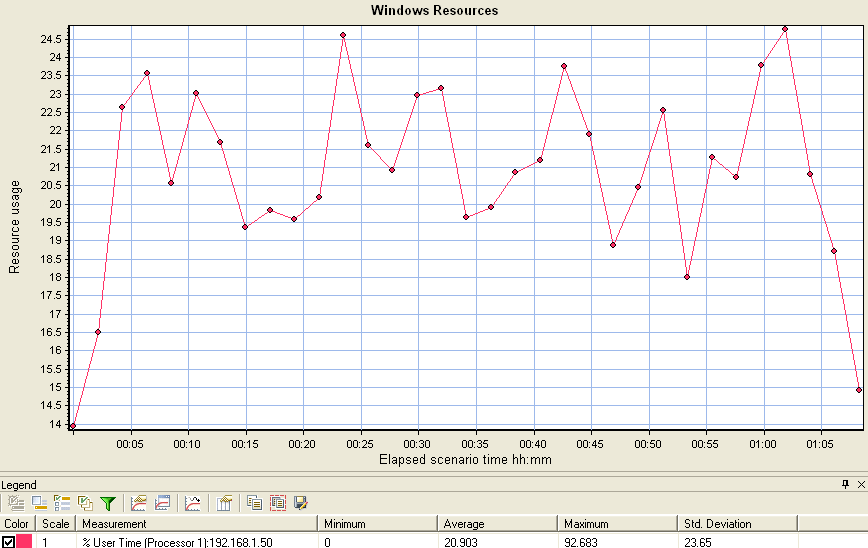
***Description:*** *% Processor Time is the percentage of elapsed time that the processor spends to execute a non-Idle thread. It is calculated by measuring the duration of the idle thread is active in the sample interval, and subtracting that time from interval duration. (Each processor has an idle thread that consumes cycles when no other threads are ready to run). This counter is the primary indicator of processor activity, and displays the average percentage of busy time observed during the sample interval. It is calculated by monitoring the time that the service is inactive, and subtracting that value from 100%.*

***Average (Avg):***26.303

***Service Level Agreement (SLA):***

***Observation:***

***Processor – %User Time:***

******

***Description:*** *% User Time is the percentage of elapsed time the processor spends in the user mode. User mode is a restricted processing mode designed for applications, environment subsystems, and integral subsystems. The alternative, privileged mode, is designed for operating system components and allows direct access to hardware and all memory. The operating system switches application threads to privileged mode to access operating system services. This counter displays the average busy time as a percentage of the sample time.*

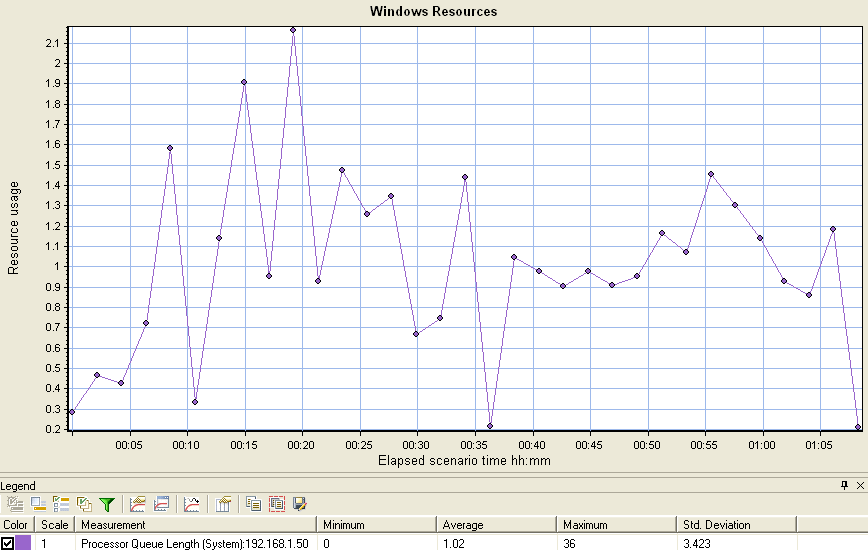
Brief: It shows the % of CPU utilization to run all the user level processes

***Average (Avg):***20.903

***Service Level Agreement (SLA):*** % user time should be less than are equal to 40%

***Observation:*** I observe 20.663% free so it is normal in execution

***SYSTEM – Processor Queue Length:***

******

***Description:*** *Processor Queue Length is the number of threads in the processor queue. Unlike the disk counters, this counter counters, this counter shows ready threads only, not threads that are running. There is a single queue for processor time even on computers with multiple processors. Therefore, if a computer has multiple processors, you need to divide this value by the number of processors servicing the workload. A sustained processor queue of less than 10 threads per processor is normally acceptable, dependent of the workload.*

Brief: it shows the queue length of all processor or CPUs to keep all the request are in queue before processing .

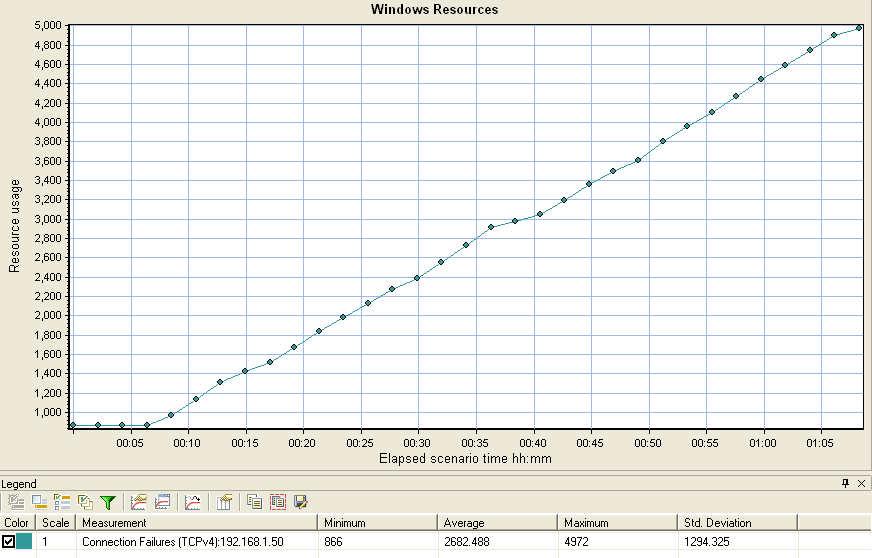
***Average (Avg):***1.02

***Service Level Agreement (SLA):*** the processor queue length should be less than are equal 10 for each processor.

Note: In the current system it contains two processors, so processer queue length can be 10\*2=20

***Observation:*** I observed that average queue length is 1.02 so it is normal in execution

***TCP v4 – Connection Failures:***

******

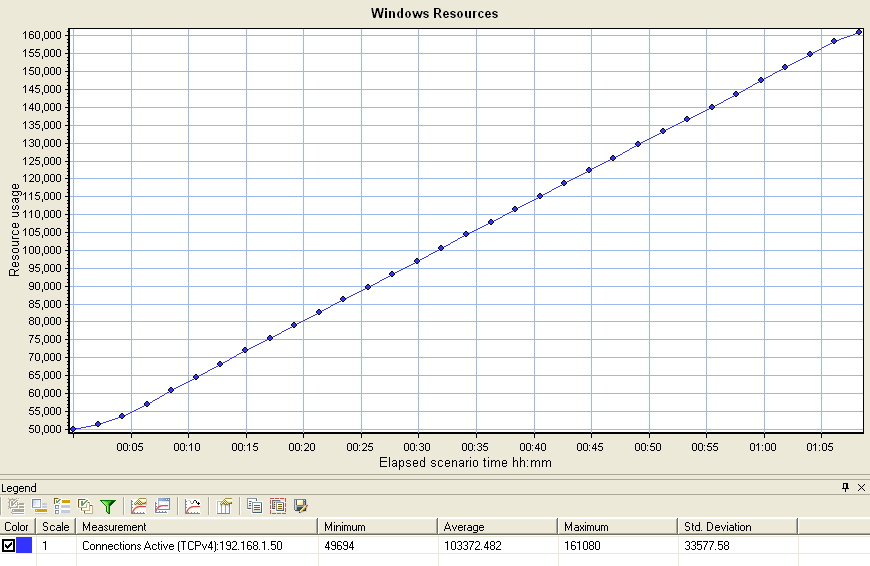
***Description:*** *Connection Failures is the number of times TCP connections have made a direct transition to the CLOSED state from the SYN-SENT state or the SYN-RCVD state, plus the number of times TCP connections have made a direct transition to the LISTEN state from the SYN-RCVD state.*

***Average (Avg):***2682.488

***Service Level Agreement (SLA):***

***Observation:*** I observed average value of TCP connection failure is 2682.488

***TCP v4 – Connection Active:***

******

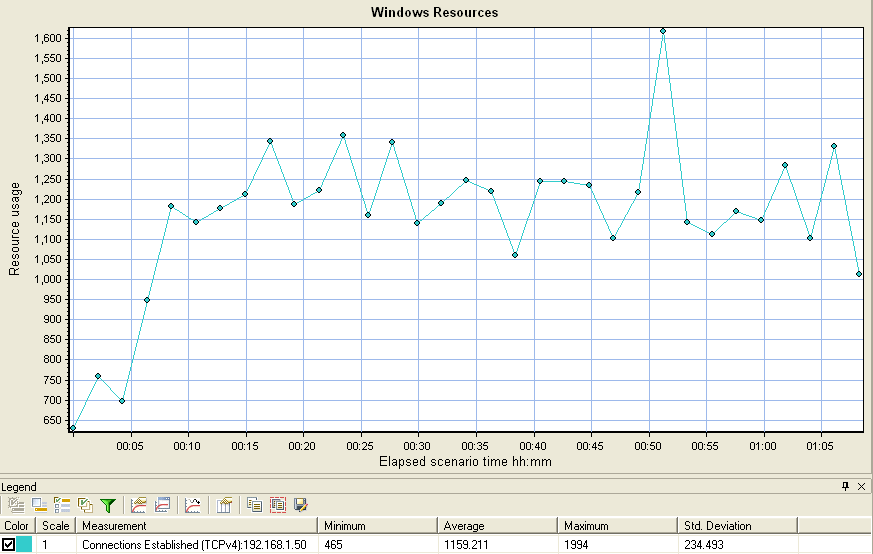
***Description:*** *Connections Active is the number of times TCP connections have made a direct transition to the SYN-SENT state from the CLOSED state.*

***Average (Avg):***103372.482

***Service Level Agreement (SLA):***

***Observation:***  I observed average value of TCP connection active is 103372.482

***TCP v4 – Connection Established:***

******

***Description:*** *Connections Established is the number of TCP connections for which the current state is either ESTABLISHED or CLOSE-WAIT.*

***Average (Avg):***1159.211

***Service Level Agreement (SLA):***

***Observation:*** I observed average value of TCP connection established is **1159.211**

**UNIX System level Metrics:**

Path: Controller -> Run -> Available Graphs -> System Resource Graphs -> Unix Resources

Note: To configure UNIX level metrics through controller we should ensure that “rstatd” (Remote Statistics Daemon )process should be running on the monitored machine (i.e on remote Unix System)

Configure the UNIX system level metrics using below .sh files

iostat.sh

vmstat.sh

netstat.sh

topstat.sh

1. Connect to remote UNIX box using PuTTY
2. Provide the system credentials as below (testQ/venkat12345)
3. Understand below type of basic UNIX commands
   1. ls: list of files and directories.   
      it is listing all the files and directories
   2. ls –l : it shows the long listing of each file and directory.

-rw-rw-r-- 1 testq testq 17678 Jun 17 19:03 Book1.xlsx

<Mode of permissions for each file/directory> <file/directory><created by><modified by> <size> <Modified time stamp> <Name of the file>

* 1. cd: change directory
  2. cd .. : Come back to immediate prev directory i.e parent directory
  3. mkdir: creates/makes a new directory
  4. cat: creates a new file
  5. rm: removes a file
  6. rm –r : removes a directory including all the files in it.
  7. Mv: moves a file/directory
  8. Chmod: use this to change the permissions of the file and directory

Read - 4

Write – 2

Execute – 1

Owner:Group:User

* 1. Vi: it is a editor to open any file.

**Example of vmstat.sh**

cd $2

if [ -f vmstat.log ]

then

mv vmstat.log vmstat\_log.old

fi

echo "interval=" $1 >> vmstat.log

while true

do

date >> vmstat.log

vmstat $1 7 >> vmstat.log

echo >> vmstat.log

done

Which file is used for what?????

vmstat – Virtual Memory + Processor

iostat – Disk input/output statistics

topstat – Process (Top consuming resource utilization)

netstat – Network statistics

**Test Execution Types:**

1. *Smoke Test:*

Test the behavior of the end to end environment at early stages of testing life cycle.

*Load:* 10 or 20 users (usually more than 1 and less than a real load to make sure scripts are running fine)

*Ramp Up:* 1 user with 30 seconds as the load is minimum. When users are more, ramp up time is more and vice-versa.

*Duration:* 30 mins

*Ramp Down:* Simultaneously

1. *Load Test:*

Test the behavior of an application with the expected users load.

*Load:* 1000 users

*Example:* Let us assume there are total 5 scenarios in scope so that the load distribution is as below –

Scenario1 – 300 users

Scenario2 – 250 users

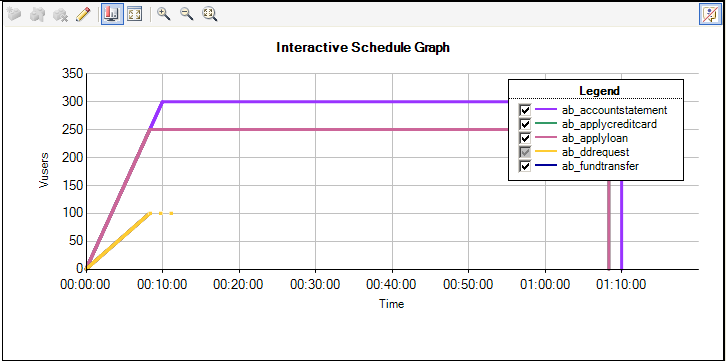
Scenario3 – 250 users

Scenario4 – 100 users

Scenario5 – 100 users

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Scenario Name** | **Load (Users)** | **Ramp Up** | **Steady State** | **Ramp Down** |
| Scenario1 | 300 | 1 User for every 2 seconds (600 sec) | 60 mins | Simultaneously |
| Scenario2 | 250 | 1 User for every 2 seconds (500 sec) | 60 mins | Simultaneously |
| Scenario3 | 250 | 1 User for every 2 seconds (500 sec) | 60 mins | Simultaneously |
| Scenario4 | 100 | 1 User for every 5 seconds (500 sec) | 20 Iterations in 60 mins | Simultaneously |
| Scenario5 | 100 | 1 User for every 5 seconds (500 sec) | 20 Iterations in 60 mins | Simultaneously |

*Sample Graph for the above scenario*



1. *Stress Test:* 
   1. *Application Level Stress Test:* Test the behavior of an application by gradually increasing the load until the application or servers crash.
   2. *Functional Level Stress Test:* Do the stress on a particular functionality to release all the Vusers on to the same functionality at the same time is called Functional level stress.

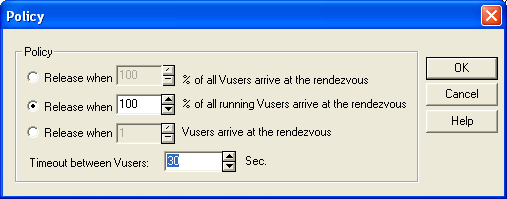
***Note:*** Use Rendezvous policy to release all the Vusers at the same time.

Use this Rendezvous function just before the lr\_start\_transaction statement of the desired transaction where the users need to wait for the other users to hit the server.

*lr\_rendezvous(“<Rendezvous Point Name>”)*

We can define ‘n’ number of Rendezvous policies.

**Policies –**

****

To set the rendezvous policy attributes:

1. Select **Scenario**> **Rendezvous**. The Rendezvous Information dialog box opens.
2. Select a rendezvous from the **Rendezvous** box, and click the **Policy** button. The Policy dialog box opens.
3. In the Policy section, select one of the following three options:
   * **Release when X% of all Vusers arrive at the rendezvous.** Releases the Vusers only when the specified percentage of all Vusers arrives at the rendezvous point.

Note: This option interferes with the scheduling of your scenario. If you select this option, your scenario will not run as scheduled.

* + **Release when X% of all running Vusers arrive at the rendezvous.** Releases the Vusers only when the specified percentage of all Vusers running in the scenario arrives at the rendezvous point.
  + **Release when X Vusers arrive at the rendezvous.** Releases the Vusers only when the specified number arrives at the rendezvous point.

1. Enter a timeout value in the Timeout between Vusers box. After each Vuser arrives at the rendezvous point, LoadRunner waits up to the maximum **timeout** period you set for the next Vuser to arrive. If the next Vuser does not arrive within the **timeout** period, the Controller releases all the Vusers from the rendezvous.

Each time a new Vuser arrives, the timer is reset to zero. The default **timeout** is thirty seconds.

1. Click **OK** to save your settings and close the Policy dialog box.
2. *Endurance Test (OR) Soak Test (OR) Pro-Long Duration Test (OR) Longevity Test :*

Test the behavior of an application for a long duration.

*Load:* 1000 users

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Scenario Name** | **Load (Users)** | **Ramp Up** | **Steady State** | **Ramp Down** |
| Scenario1 | 300 | 1 User for every 5 seconds (1500 sec) | 12 hours | Simultaneously |
| Scenario2 | 250 | 1 User for every 6 seconds (1500 sec) | 12 hours | Simultaneously |
| Scenario3 | 250 | 1 User for every 6 seconds (1500 sec) | 12 hours | Simultaneously |
| Scenario4 | 100 | 1 User for every 15 seconds (1500 sec) | 12 hours | Simultaneously |
| Scenario5 | 100 | 1 User for every 15 seconds (1500 sec) | 12 hours | Simultaneously |

1. *Spike Test:*

Test the behavior of an application with dynamic work load changes at run time.

**Service Level Agreement(s):**

* Configure the SLAs of each transaction – the response time, to validate the behavior at run time.

**IP Spoofing:**

* It allocates the different IP addresses for different users running in the execution.

**WebLogic Server Analysis:**

Analyze the performance metrics at weblogic server level to identify the application code level performance issues.

<http://192.168.1.50:7001/console> --> This is the URL to access WebLogic Server.

UserID/Password to access WebLogic Server – *weblogic/weblogic*

1. *Java Virtual Machine (JVM) Memory Utilization Statistics:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Heap Size Current: | |  |  | | --- | --- | | 345333760 |  | | The current size (in bytes) of the JVM heap.http://192.168.1.50:7001/console/images/spacer.gif[More Info...](javascript:launchAttributeHelp('core.server.servermonitoringperformance.heapsizecurrent','en-us')) |
| Heap Free Current: | |  |  | | --- | --- | | 91863568 |  | | The current amount of memory (in bytes) that is available in the JVM heap.http://192.168.1.50:7001/console/images/spacer.gif[More Info...](javascript:launchAttributeHelp('core.server.servermonitoringperformance.heapfreecurrent','en-us')) |
| Heap Free Percent: | |  |  | | --- | --- | | 26 |  | | Percentage of the maximum memory that is free.http://192.168.1.50:7001/console/images/spacer.gif[More Info...](javascript:launchAttributeHelp('core.server.servermonitoringperformance.heapfreepercent','en-us')) |
| Heap Size Max: | |  |  | | --- | --- | | 532742144 |  | | The maximum free memory configured for this JVM.http://192.168.1.50:7001/console/images/spacer.gif[More Info...](javascript:launchAttributeHelp('core.server.servermonitoringperformance.heapsizemax','en-us'))  Bottom of Form |

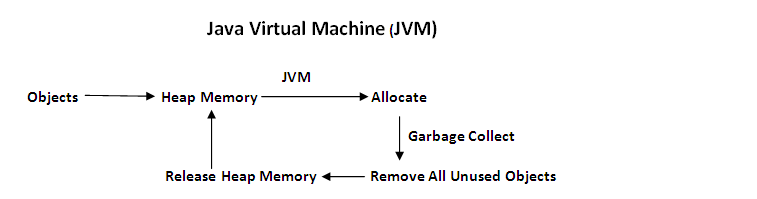
**Java Virtual Machine (JVM)**

**JVM**

**Objects Heap Memory Allocate**

**Garbage Collect**

**Release Heap Memory Remove All Unused Objects**

****

1. *JVM Heap Size Max:* It shows the maximum Heap memory configured in JVM
   * Current size is 532742144 Bytes
   * Converting the Bytes into Mega bits 🡪 (532742144/1024)/(1024) = 508.06 = approx 512 MB (Mega Bits)
2. *Heap Size Current:* It shows the current amount of available Heap memory which can be used in the execution.

* Heap Size Current = 345333760 Bytes = (345333760/1024)/(1024) = 329.33 MB (Mega Bits)

1. *Heap Free Current:* It shows the amount of free memory from the current size of the heap.

* Heap Free Current = 91863568 Bytes = (91863568/1024)/(1024) = 87.6 MB (Mega Bits)

1. *Memory Leaks:*

The unreleased memory which is no longer needed.

*Step 1:* For every new object creation, the JVM will allocate the Heap memory at run time.

*Step 2:* Once the object execution is completed (these objects are then called as unused objects), all unused objects will be removed by the GC (Garbage Collector).

*Step 3:* When the unused objects are not removed, the corresponding allocated memory also will not be removed. Then, it will be causing a memory leak.

1. *JDBC (Java Database Connectivity):*

***JDBC***

**WebLogic**

**SQL Server**

*Active Connections Average Count:*

* It shows the number of connections that are active at any point of time in the execution.

*Active Connections Current Count:*

* It shows the total number of active connections so far in the execution.

*Connections Total Count:*

* It shows the total number of JDBC connections that are configured in a JDBC connection pool.

Observation: I observed in one of my application that the total connections are configured as 100 for 1000 users load test.

**AJAX Click and Script Protocol:**

* It is a GUI Level Script. It is the scripting at Client side only.

Use this protocol to capture the script at GUI level. We will use the same protocol when the application was developed based on the AJAX controls.

Functions generally used in AJAX scripting –

1. web\_browser()
2. web\_edit\_field()
3. web\_image\_submit()
4. web\_text\_link()
5. web\_list()
6. web\_button()

Please find below – the sample AJAX script which shows only the client requests:

Action()

{

web\_browser("onlineshopping",

DESCRIPTION,

ACTION,

"Navigate=http://192.168.1.43/onlineshopping/",

LAST);

web\_edit\_field("txtName",

"Snapshot=t1.inf",

DESCRIPTION,

"Type=text",

"Name=txtName",

ACTION,

"SetValue=t@t.com",

LAST);

web\_edit\_field("txtPassword",

"Snapshot=t2.inf",

DESCRIPTION,

"Type=password",

"Name=txtPassword",

ACTION,

"SetEncryptedValue=4e75e03bf85c4b31d473cb",

LAST);

web\_image\_submit("imgLogin",

"Snapshot=t3.inf",

DESCRIPTION,

"Alt=",

"Name=imgLogin",

ACTION,

"ClickCoordinates=46,17",

LAST);

web\_text\_link("KITCHEN",

"Snapshot=t4.inf",

DESCRIPTION,

"Text=KITCHEN",

ACTION,

"UserAction=Click",

LAST);

web\_image\_submit("ctl00$ContentPlaceHolder1$grdProduct",

"Snapshot=t5.inf",

DESCRIPTION,

"Alt=",

"Name=ctl00$ContentPlaceHolder1$grdProducts$ctl00$imageitem",

ACTION,

"ClickCoordinates=98,114",

LAST);

web\_edit\_field("ctl00$ContentPlaceHolder1$gridSubpro",

"Snapshot=t6.inf",

DESCRIPTION,

"Type=text",

"Name=ctl00$ContentPlaceHolder1$gridSubproducts$ctl02$txtQty",

ACTION,

"SetValue=1",

LAST);

web\_image\_submit("ctl00$ContentPlaceHolder1$ImgRegiste",

"Snapshot=t7.inf",

DESCRIPTION,

"Alt=",

"Name=ctl00$ContentPlaceHolder1$ImgRegister",

ACTION,

"ClickCoordinates=70,2",

LAST);

web\_image\_submit("ctl00$ContentPlaceHolder1$ImageButto",

"Snapshot=t9.inf",

DESCRIPTION,

"Alt=",

"Name=ctl00$ContentPlaceHolder1$ImageButton1",

ACTION,

"ClickCoordinates=58,10",

LAST);

web\_image\_submit("ImageButton4",

"Snapshot=t10.inf",

DESCRIPTION,

"Alt=",

"Name=ImageButton4",

ACTION,

"ClickCoordinates=29,42",

LAST);

web\_edit\_field("txtCardNumber",

"Snapshot=t11.inf",

DESCRIPTION,

"Type=text",

"Name=txtCardNumber",

ACTION,

"SetValue=2345345654647458",

LAST);

web\_edit\_field("txtFirstName",

"Snapshot=t12.inf",

DESCRIPTION,

"Type=text",

"Name=txtFirstName",

ACTION,

"SetValue=hdrhdfhd",

LAST);

web\_edit\_field("txtSurname",

"Snapshot=t13.inf",

DESCRIPTION,

"Type=text",

"Name=txtSurname",

ACTION,

"SetValue=drtjhtrthrt",

LAST);

web\_list("ddlMonth",

"Snapshot=t14.inf",

DESCRIPTION,

"Name=ddlMonth",

ACTION,

"Select=9",

LAST);

web\_list("ddlYear",

"Snapshot=t15.inf",

DESCRIPTION,

"Name=ddlYear",

ACTION,

"Select=2009",

LAST);

web\_list("DropDownList1",

"Snapshot=t16.inf",

DESCRIPTION,

"Name=DropDownList1",

ACTION,

"Select=5",

LAST);

web\_list("DropDownList2",

"Snapshot=t17.inf",

DESCRIPTION,

"Name=DropDownList2",

ACTION,

"Select=2013",

LAST);

web\_edit\_field("txtSecurityCode",

"Snapshot=t18.inf",

DESCRIPTION,

"Type=text",

"Name=txtSecurityCode",

ACTION,

"FireEvent=onfocus",

LAST);

web\_edit\_field("txtSecurityCode\_2",

"Snapshot=t19.inf",

DESCRIPTION,

"Type=text",

"Name=txtSecurityCode",

ACTION,

"SetValue=3434",

LAST);

web\_button("INPUT",

"Snapshot=t20.inf",

DESCRIPTION,

"Type=submit",

"Tag=INPUT",

"ID=Button1",

ACTION,

"UserAction=Click",

LAST);

web\_browser("Navigate",

"Snapshot=t21.inf",

DESCRIPTION,

ACTION,

"Navigate=http://192.168.1.43/onlineshopping/PayPoll/CardDetails.aspx",

LAST);

web\_button("INPUT\_2",

"Snapshot=t22.inf",

DESCRIPTION,

"Type=submit",

"Tag=INPUT",

"ID=Button2",

ACTION,

"UserAction=Click",

LAST);

web\_edit\_field("txtPassword\_2",

"Snapshot=t23.inf",

DESCRIPTION,

"Type=password",

"Name=txtPassword",

ACTION,

"SetEncryptedValue=4e75e073e1542e8c291388d5",

LAST);

web\_button("INPUT\_3",

"Snapshot=t24.inf",

DESCRIPTION,

"Type=submit",

"Tag=INPUT",

"ID=btnSubmit",

ACTION,

"UserAction=Click",

LAST);

web\_text\_link("Log Out",

"Snapshot=t25.inf",

DESCRIPTION,

"Text=Log Out",

ACTION,

"UserAction=Click",

LAST);

web\_browser("Sync\_2",

"Snapshot=t26.inf",

DESCRIPTION,

ACTION,

"Sync",

LAST);

return 0;

}

**SAP Web Protocol:**

web\_save\_timestamp\_param("test");

lr\_output\_message(lr\_eval\_string("{test}"));

//1216286618009 --> These are the milli seconds from 1970 to till date

// Initially, before LR 9.5 there are no methods to get these milli seconds

// From LR 9.5, web\_save\_timestamp\_param("<Parameter Name>"); is used to

//get the milliseconds from midnight of Jan 1st 1970 to Till Date

web\_save\_timestamp\_param("<Parameter Name>");

This function saves the current timestamp.

int **web\_save\_timestamp\_param**( const char \**tmstampParam*, LAST );

|  |  |
| --- | --- |
| **Part** | **Description** |
| *tmstampParam* | The name of the parameter in which to store the timestamp. |

**General Information**

**web\_save\_timestamp\_param** saves the current timestamp. In some applications, VuGen replaces all non–empty timestamps in the script with a parameter. To save the value of this parameter, VuGen automatically generates a call to **web\_save\_timestamp\_param**. The value saved is the number of milliseconds since midnight January 1st, 1970.

Use this protocol to develop the script for the SAP based applications.

**Note:** In SAP Web protocol, WindowID will be generated dynamically by the client. So, use the below function to generate the WindowID dynamically at client side.

**Manual Scripting:**

*Below are the Network Sniffers OR Network Analyzers when* ***Browsers are available****–*

In Mozilla Firefox – *Http Fox*

In Internet Explorer – *Http Watch*

*Below are the Network Sniffers OR Network Analyzers when* ***Browsers are NOT available****–*

*Wire Shark*

Throughout the script, we see only 2 functions – web\_url() and web\_submit\_data().

Now, we have to know when to use web\_url() and when to use web\_submit\_data(). This can be known from the POST Data.

* If there is a POST Data for the request, then use web\_submit\_data()
* If there is no POST Data for the request, then use web\_url()

**Image Checkpoint:**

Enable *Image and Text Check* option in Preferences section of Run Time Settings

*src* is the source where the image is located. To get the *src* value, right click on the page where the image is displayed and then view source and search for the *src* in the source of the page.

web\_image\_check(“ImageCheck”,

“src=images/hp\_logo.png”,

LAST);

**Perfmon Analysis:**

1. Available Mbytes: