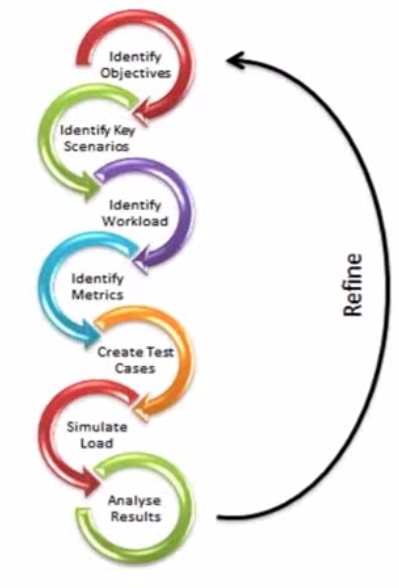
**Live Assist QA JMeter Load Test**

1. ***Test Goals and tools selection:***
   1. ***Goal: high agent gui channel desity (>=100)***

* We want to know if the product support many agents logged in and run concurrently on one Live Assist server
* For large number agents logged in and concurrently running, what about the performance (response time/delay/latencies) of the Live Assist server?
* Ensure a good user experience which satisfy specific criteria such as CPU/memory/https call/responses/error percentage etc under many users logged in load test;



*Fig.1 basic performance test and tuning procedure*

* 1. ***Test tools selection/considerations***
* Target:
  + Simulate agent gui communication with Live Assist: Live Assist server ⬄ Browser;
  + To simulate the traffic (HTTP) between browser communicate with LA server
    - How does a browser works:
      * Browsers sends a request (URL, button click, AJAX)
      * Server responses (HTML, images, etc)
      * Browser parses responses and executes JS
      * Repeat
* Protractor/Selenium + firefox driver solution:
  + Pros: simulated real agent response via real browser; The test covered by separated Live Assist QA api+gui load test with 10 API channel with 9 agent GUI channel running call iterations > 10,000
  + Cons: HW resource limited the test run with high agent # load test with real browser;
* JMeter solution:
  + JMeter can simulate all the browser doing with server:
    - JMeter send request (HTTP)
    - Server responds
    - JMeter parse responses (optional, for JS, simulate with JMeter scripts)
    - Repeat

Live Assist product basic work flow:

SIP call/QA-User Test DriverSIP call/QA-User Test Driver**SIP call/QA-User Test Driver**

ChN: JMeter simulate api test

Ch2: JMeter simulate api test

Ch1: JMeter simulate api test

HTTPHTTPHTTP

Live Assist Server

HTTPHTTPHTTP

SIP call/QA-User Test DriverSIP call/QA-User Test DriverSIP call/QA-User Test Driver

ChN: JMeter simulate agent response

Ch2: JMeter simulate agent response

Ch1: JMeter simulate agent response response browser

**Fig. 2 Basic QA api + gui concurrent load test architecture/setup**

1. ***Live Assist high agent channel density load test***

***2.1 Test env and setup***

*Test environment:*

Live Assist server: mt-ray-vm04, CentOS6, 2 core, 4GB RAM

JMeter API requests simulated load test machine: mtl-bl1-12-vm10, win2k3, 2 core, 6GB RAM

JMeter Agent response simulated load test machine: mtl-bl1-12-vm10, win2k3, 2 core, 6GB RAM

*Test setup:*

JMeter simulate API requests:

API type: IVR api, no-match emma

Channel #: 100 (note: match to GUI channels)

JMeter simulate Agent response:

Agent outcome: ‘Operator’

Channel #: 100

* 1. ***Test steps and scripts***

Test Configuration:

* For JMeter API and agent simulation test script:
  + Number of Threads: 100 threads (agent users)
  + Each thread: 100 iterations
  + Ramp-up period (sec): 300
* For JMeter test setup:
  + Change the JMeter JVM settings again:
    - set HEAP=-Xms512m -Xmx1400m
    - set NEW=-XX:NewSize=128m -XX:MaxNewSize=512m
  + Disabled all the listeners (note: status code from each request/response still be checked).
  + Save test data into a file, and checked with aggregate report with graph after test done, instead of collecting and showing date real time with summary graph report during the test run before.
  + run JMeter as non-GUI mode (from command line)

Test Scripts:

1. JMeter simulate API load test script:

ivr api trigger call -> extra session ID -> while loop to check agent outcome intent -> while loop to check agent state -> ivr api call again to assert check agent intent return -> end session and reset user variables (agent\_intent, agent\_state, and session ID)

1. JMeter simulate agent outcome load test script:

agent login -> websock connection -> (idle mode) -> poll for interaction -> while loop to check poll check status result (busy?) -> busy mode -> poll for interaction -> set agent outcome (post) -> agent logout

1. Test folders/batch files/scripts location:

API JMeter test folder: C:\JMeter\_Test

JMeter scripts: C:\JMeter\_Test\LiveAssist\_Perf\LA\_stress\_ivr-api\_test1.jmx

Test start batch file: C:\JMeter\_Test\start\_ivr-stress\_test1.bat

Results (saved to): C:\JMeter\_Test

Agent JMeter test folder: C:\JMeter\_Test2

JMeter scripts: C:\JMeter\_test2\LiveAssist\_Perf\LA\_stress\_agent\_test1.jmx

Test start batch file: c:\JMeter\_test2\start\_agent-gui-stress\_test1.bat

Results (saved to): C:\JMeter\_Test2

1. Basic controller/listener/config manager used for JMeter test scripts:

Configuration managers used:

1. CSV Data set config – load Live Assist user info for each thread (up to 100 users)
2. HTTP cache manager – simulate agent browser setup, select ‘Clear cache each iteration’, and ‘Use Cache-Contorl/Expires header when processing GET requests’ with max number of elements in cache set to ‘5000’
3. HTTP cookie manager – simulate agent browser setup, select ‘Clear cookies each iteration’
4. HTTP Requests defaults – set web server (live assist server) IP and protocol (https)
5. User Defined Variables – set for temporary variables used for api test and agent test, such as user name, password, server IP/port, and status variable; call sessionID, agent intent, agent state variables etc.

Controller used:

1. Simple controller – group JMeter basic test controllers such as IVR api and result verification, agent login, agent outcome, agent logout
2. Transaction controller – triggered http/https request
3. While controller – test flow logic controller, such as: check ivr api agent outcome intent, check agent state, check if call escalated to agent, etc

Listener used:

1. Summary Report – Create a table row for each differently named request
2. Response Time Graph – Draws a line chart showing evolution of response time during the test for each labelled request.
3. Aggregate Graph – Generate bar graphs and save graphs as PNG file
4. View Result Tree – Show a tree of all sample responses, and it only used for smoke JMeter test and debug purpose

All above scripts/listeners/controllers/configuration managers which used in the QA JMeter load test can be viewed and updated in JMeter GUI console.

Notes:

* Run non-gui mode for high agent thread/users load test, less affect load test perf
* The two JMeter load test scripts run parallel with 2 JMeter instances.
* The basic workflow difference between QA api+gui load test and this JMeter api+agent outcome request load test is that JMeter need to send poll to get call escalation state, but api+gui load test can use JS event driven mechanism to check gui element change to know if call escalated or not
* The actual call escalation check from agent side simulation was currently done by periodically poll and check return response for state change (to reduce burden on Live Assist server, the check period set >= 2 sec).
* A webSocket sampler used in this JMeter test just for login only, which just simulate the creation of the websocket connection, the test poll interaction (get request) to periodically check call escalation … it will be improved to use websocket (ping/notify) to control if need to poll for escalated call or not in the future.
* The above 2 instances run parallel on one test machine (or two) with the same Live Assist server.

Test steps:

1. Test checkup/config:
   * check the test batch files, test scripts and configuration files ready and put in the correct folders
   * check the JMeter machine and Live Assist server ready and communication is ok
   * check JMeter simulated API test configuration and agent test configuration setup properly
2. Stop live assist server
3. clean up
   * call logs
   * diagnostic logs
   * performance log (if have)
4. start live assist server
   1. starting Live Assist server:
      1. su -s /bin/bash nuance --session-command liveassist
   2. starting Live Assist Audio server:
      1. su -s /bin/bash nuance --session-command liveassistaudio
5. start performance monitor on Live Assist server
   1. The perfmon script& result location: /opt/performance\_monitor
   2. It can monitor all the java process (Live Assist server, audio server if used) and system on CPU and memory usage from the test server (CentOS6)
   3. Run the perfmon script for Live Assist: ‘sh liveassist\_perfmon.sh’
   4. Result can be found under ‘/opt/performance\_monitor/LA\_test\_data1’ (search the latest 3 csv files, each for Live Assist server, audio server and system perfmon log)
6. Run JMeter test batch files:
   1. JMeter simulated API load test: : C:\JMeter\_Test\start\_ivr-stress\_test1.bat
   2. JMeter simulated agent load test: c:\JMeter\_test2\start\_agent-gui-stress\_test1.bat
7. Test result collection and analysis:
   1. Collect 3 basic type results for both JMeter api and agent load test: request/response summary; Aggregate response latency graph; Response Time Graph; see the examples from the following section 2.3;
   2. Save graph (if available) to corresponding JMeter test folders
   3. Pass/fail criteria: No CPU or memory leaking, CPU usage < 80%, no channel hang; JMeter test percentage of error on request < 1% (refer to other project), response latency/delay under reasonable range (accepted by team)

Notes:

* Before the multiple iteration multiple agent JMeter test, optional for new build testing is that run smoke test with 1 channel 2 loops to make sure test scripts still works on new build.
* If the new Live Assist build requested to drop DB and re-create DB, user need to re-setup 100 agent users:
  + Live Assist server:
    - To add 100 new agents:

    wget <http://mtl-da55-vm6/sipcallclient/add_load_test_data_to_db.sql>

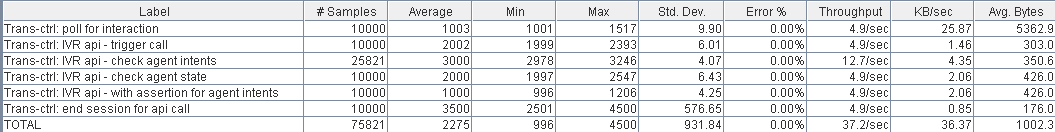
    mysql < add\_load\_test\_data\_to\_db.sql

* + - The agent usernames are LT0000 to LT0099 and the passwords are ‘changeit’.
  + JMeter machine (optional if done before already):
    - Run QA Perl script ‘generate\_LA\_user\_JMeter\_csv\_100.pl ‘ to generate the necessary 100 user CSV files under: C:\JMeter\_Test\LiveAssist\_Perf\data

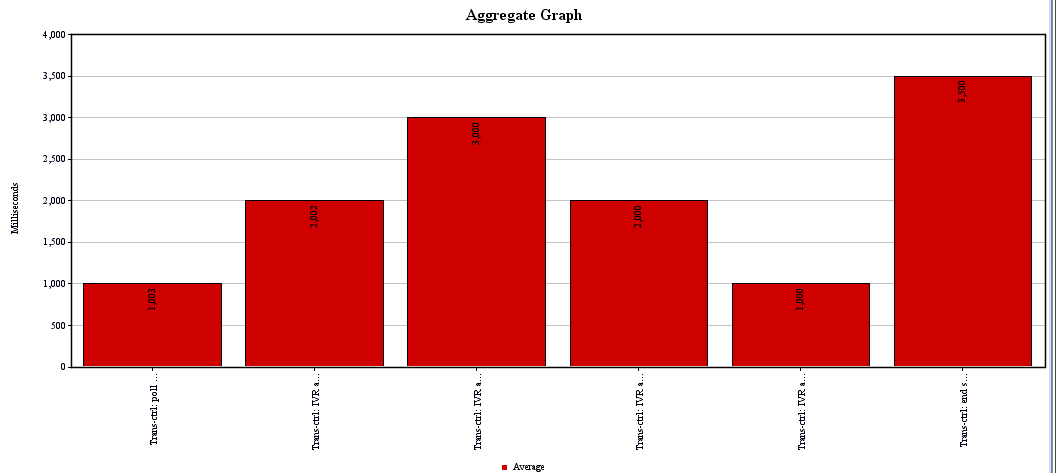
***2.3 Test result/report***

The following results/metrics can be collected as part of this JMeter stress/performance test report. Example referred here from iteration 6.4 JMeter load test result:

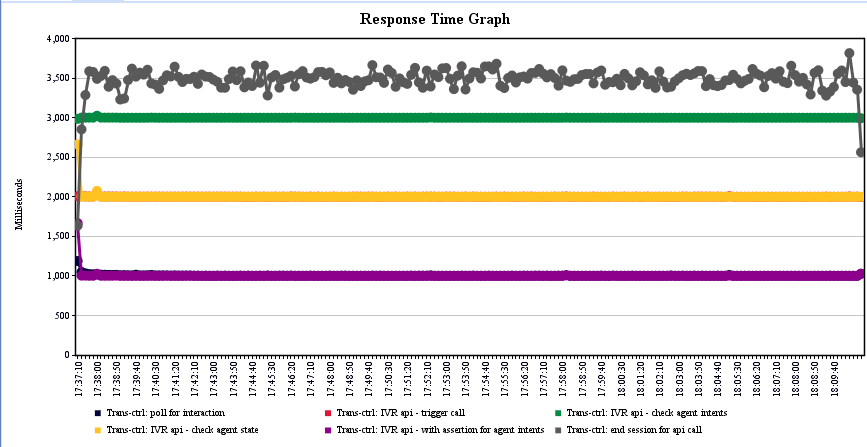
1. JMeter simulated API requests/response summary:



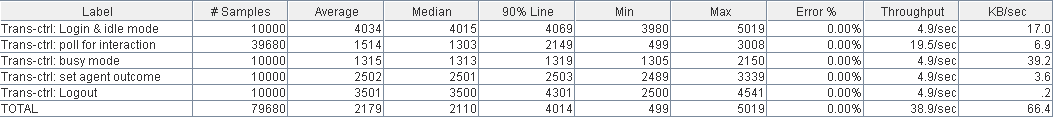
Aggregate API call response latency graph:



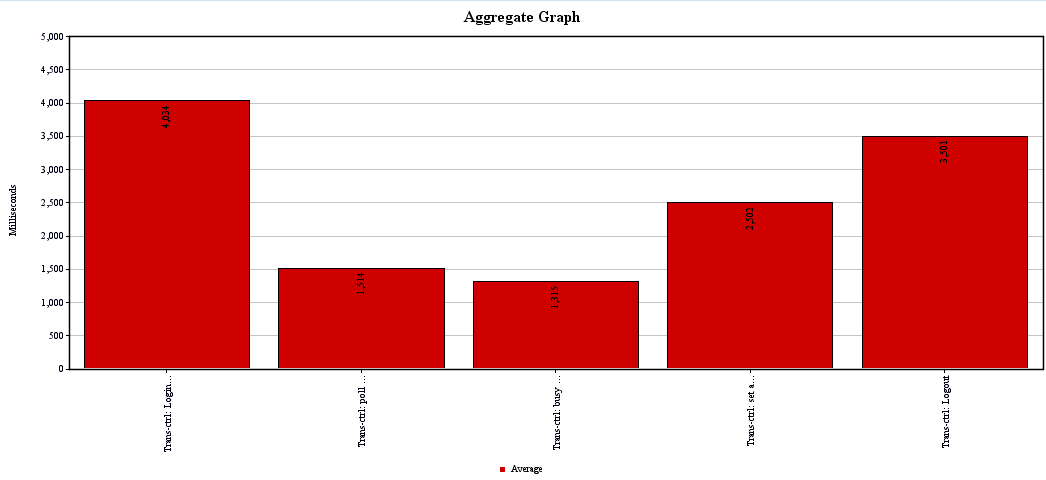
Response Time Graph:



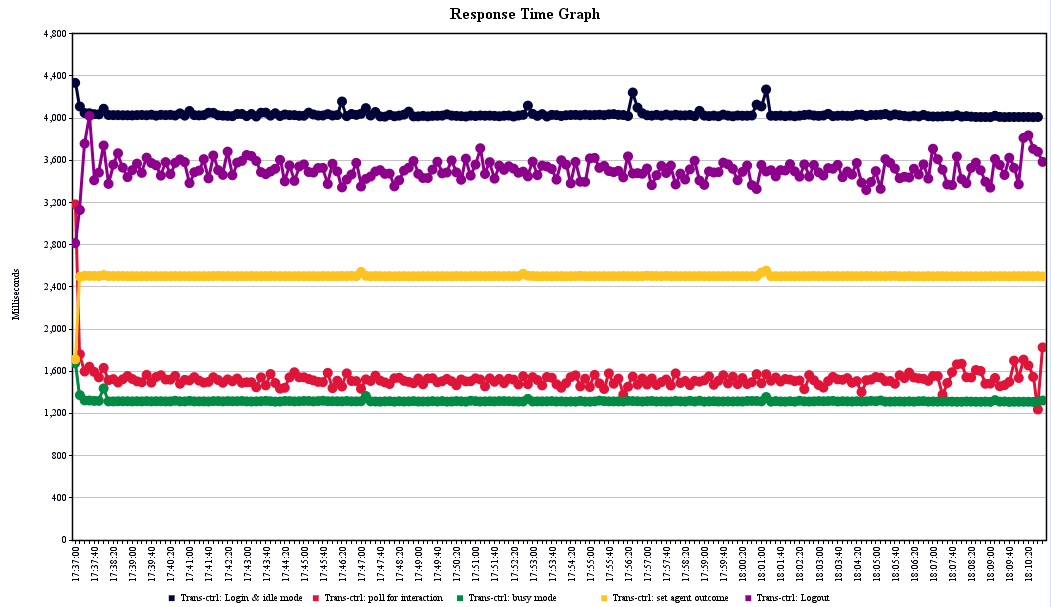
1. JMeter simulated agent outcome requests/responses summary:



Aggregate simulated agent call/response latency graph:



Response Time Graph



C) verify the CPU & memory usage (system & Java process) on Live Assist server. The basic test criteria: CPU avg usage < 80%, and no CPU or memory leaking found during the test.

1. ***Test resource and references:***
2. Test scripts/result ref:
   * JMeter Iteration 6.4 Load Test scripts reference: [\\mt-nasrops01\qa\LiveAssist\Test\_Set\Iteration6.4\Jmeter\_load](file:///\\mt-nasrops01\qa\LiveAssist\Test_Set\Iteration6.4\Jmeter_load)
   * JMeter Iteration 6.4 test result reference: [\\mt-nasrops01\qa\LiveAssist\QA\_tech\Test\_logs\JMeter-load\_result\_iter64\stress\_ivr-agent\_jmeter\_load\_09152014](file:///\\mt-nasrops01\qa\LiveAssist\QA_tech\Test_logs\JMeter-load_result_iter64\stress_ivr-agent_jmeter_load_09152014)
3. Install JMeter WebSocket Sampler:

* Download the latest version of the extension from the [Releases](https://github.com/maciejzaleski/JMeter/releases) page. Drop the .jar to /lib/ext folder of your JMeter installation
* Check Project [Wiki](https://github.com/maciejzaleski/JMeter/wiki) for any dependencies, to be completely sure all the jars from target/site/plugins directory should go either to /lib or to /lib/ext folder of your JMeter installation. Currently (for release 1.0.1) the required libraries live in [Jetty Bundles 9.1.1.v20140108](http://download.eclipse.org/jetty/updates/jetty-bundles-9.x/9.1.1.v20140108/).
* For websocket traffic capture, the Firefox firebug can’t capture all the request/header correctly on websocket, as https, should be wss:// or ws://. Instead using chrome DEV tool with websocket filter to capture.

1. Reducing resource usage during JMeter load test:

* Use non-GUI mode: jmeter -n -t test.jmx -l test.jtl
* Use as few Listeners as possible; if using the -l flag as above they can all be deleted or disabled.
* Don't use "View Results Tree" or "View Results in Table" listeners during the load test, use them only during scripting phase to debug your scripts.
* Rather than using lots of similar samplers, use the same sampler in a loop, and use variables (CSV Data Set) to vary the sample. [The Include Controller does not help here, as it adds all the test elements in the file to the test plan.]
* Don't use functional mode
* Use CSV output rather than XML
* Only save the data that you need
* Use as few Assertions as possible
* Use the most performing scripting language (see JSR223 section)