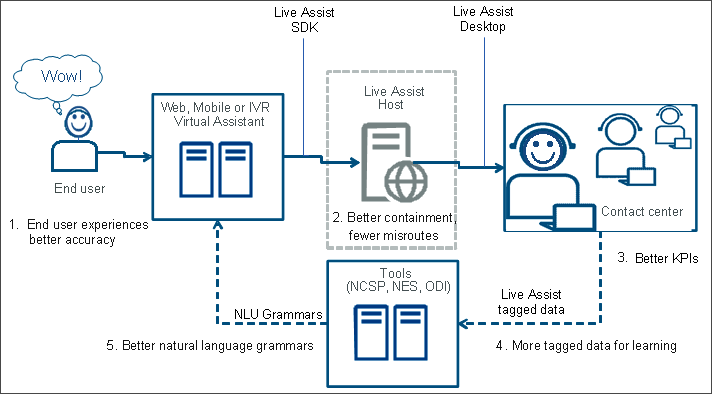
**Live Assist API + GUI concurrent QA Load Test**

1. ***Test setup and env basic:***
   1. ***Test machine/OS resource for the 8 api + 8 agent (with 2 client QA test machines setup) concurrent load test on 1 Live Assist server:***

* Live Assist server (CentOS or Windows)
* api + gui test machine 1 (Win2k3): test script, nodejs, protractor, grunt, selenium hub and nodes – 1
* api + gui test machine 2 (win2k3): selenium nodes – 2
  1. ***Concurrent load test workflow:***

Live Assist product basic work flow:



To simulate Web/IVR (api call) -> Live Assist -> Agent (GUI test), there were two QA api + gui concurrent load test approaches implemented:

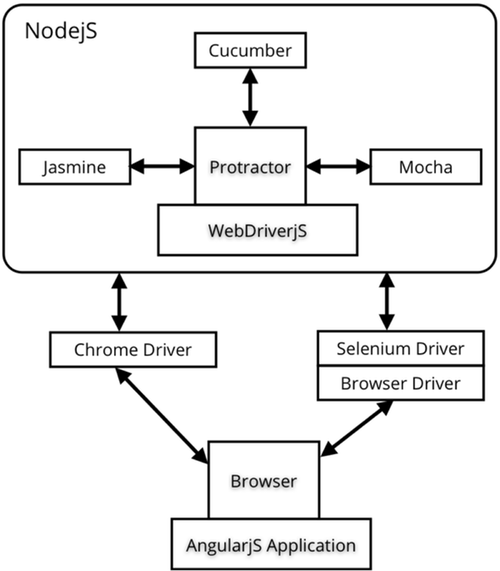
1. *Each api + gui agent channel has a complete workflow*

Agent channel 1: Test iteration setup/initialization -> agent gui login -> api call -> agent response selection -> check agent outcome via session/api call -> agent logout -> test iteration done

…

Agent channel x: Test iteration setup/initialization -> agent gui login -> api call -> agent response selection -> check agent outcome via session/api call -> agent logout -> test iteration done

The basic agent angularjs GUI test using protractor + selenium approach as follow:



**Fig. The agent GUI protractor + selenium test architecture**

1. *Separate (decoupled) api and gui handling workflow into independent api call channel and independent agent gui outcome response channel*

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

Ch8: api test driver

Ch2: api test driver

Ch1: api test script

HTTPHTTPHTTP

Live Assist Server

HTTPHTTPHTTP

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

Ch2: agent browser

Ch8: agent browser

Ch1: agent browser

Selenium/Protractor

Selenium/Protractor

Selenium/Protractor

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

Ch2: gui test driver

Ch8: gui test driver

Ch1: gui test script

**Fig. The basic QA api + gui concurrent load test architecture/setup**

Basically, for QA api + gui concurrent load test, QA implemented both approaches for testing, however, to support higher agent/api channel tests and call iterations, and also based on Live Assist call escalation and agents handling design which after api call triggered, it will be escalated to any available agent gui, so the 2) test approach is appropriated and finally used in the Live Assist QA api + gui concurrent load test.

The 2) test approach also used selenium grid to make sure the agent gui running with firefox browser can be run and tested parallel with at least 8 browser instances support in two QA test machines (client side). A reference for the selenium grid setup:

We use selenium standalone server with latest version firefox browser (note: they should be upgrade together, now it is selenium 2.41.0 with firefox 28 used in the QA api+gui load test). Selenium server can be setup to start as hub/node, but actual test we recommend to use nodes or separate selenium process each with different ports (e.g. in separate command windows processes/driver instances) and corresponding firefox driver profiles. There are 4 selenium nodes running on one QA test machine to support 4 agent GUI test, as 1st test group, and another 4 selenium nodes running on another QA test machine to support another 4 agent GUI test, so totally 8 agent GUI test with 8 different firefox profiles can be run and tested concurrently. Also the QA test scripts (8 gui scripts) all point to the 8 selenium nodes with 8 different ports.

* 1. ***Test software/scripts/framework:***
* Live Assist server: Live Assist build
* QA test machine (master):
  + batch script files, to setup selenium hub, nodes, browser driver and start selenium servers:
  + setup under c:\liveassist\_test\test, including batch files:
    - start-api-gui-load2.bat (recommended): start selenium standalone servers separately and each with different ports and profiles for firefox driver
    - start-api-gui-load3.bat (optional for selenium hub/nodes setup test): start selenium server with hub and nodes setup. Each node started with different ports and profiles for firefox driver

Note: if setup test go with selenium hub/nodes, you need to change TC000<x>\_gui.js scripts to point to the selenium hub URL, default setup to work with start-api-gui-load2.bat

* + start selenium hub and first 4 selenium nodes group (for agent gui testing): ivr-gui-load.bat
  + nodejs test scripts, including api scripts, and gui test scripts:
    - setup under c:\liveassist\_test\test\QA\_LOAD3
    - api nodejs scripts (x range from 1 - 9):
      * TC000<x>\_ivrapi.js
      * TC000<x>\_webapi.js
    - gui nodejs scripts (x range from 1 - 9):
      * TC000<x>\_gui.js
  + grunt files, to setup and start concurrent load test:
    - setup Gruntfile.js under c:\liveassist\_test\test,
    - config concurrent ivr/agent running number, names and tasks

Example for current api-gui 8 agent load test setup:

shell:

run\_TA0001: { // Target

options: { // Options

stdout: true

},

command: 'api-gui-load1.bat'

},

run\_TA0002: { // Target

options: { // Options

stdout: true

},

command: 'api-gui-load2.bat'

},

…

run\_TA0008: { // Target

options: { // Options

stdout: true

},

command: 'api-gui-load8.bat'

},

run\_TA0009: { // Target

options: { // Options

stdout: true

},

command: 'api-gui-load9.bat'

}

},

concurrent: {

load\_test: ['loadtest-TA0001', 'loadtest-TA0002', 'loadtest-TA0003', 'loadtest-TA0004', 'loadtest-TA0006', 'loadtest-TA0007', 'loadtest-TA0008', 'loadtest-TA0009'], //run 8 agents/channel test

options:{

logConcurrentOutput: false,

limit: 11

// limit: 6

}

}

});

// Load the plugin for tasks

grunt.loadNpmTasks('grunt-concurrent');

grunt.loadNpmTasks('grunt-execute');

grunt.loadNpmTasks('grunt-shell');

// register task(s)

grunt.registerTask('loadtest-TA0001', ['shell:run\_TA0001']);

grunt.registerTask('loadtest-TA0002', ['shell:run\_TA0002']);

grunt.registerTask('loadtest-TA0003', ['shell:run\_TA0003']);

grunt.registerTask('loadtest-TA0004', ['shell:run\_TA0004']);

grunt.registerTask('loadtest-TA0005', ['shell:run\_TA0005']);

grunt.registerTask('loadtest-TA0006', ['shell:run\_TA0006']);

grunt.registerTask('loadtest-TA0007', ['shell:run\_TA0007']);

grunt.registerTask('loadtest-TA0008', ['shell:run\_TA0008']);

grunt.registerTask('loadtest-TA0009', ['shell:run\_TA0009']);

grunt.registerTask('loadtest-TA0000', ['shell:run\_TA0000']);

grunt.registerTask('loadtest-node', ['concurrent:load\_test']);

* + Test config:
    - setup c:\liveassist\_test\test\config, TestConfig.js
    - config file to setup: Live Assist server addr, api call iteration, protractor timeout, api call wait timeout, etc. Example setup for 8 agent total 1000 call iteration test:

exports.options = {

server : "10.3.41.54", //mt-ray-vm01, load test

port : 8080, //default Live Assist server port

test\_location : "c:\\LiveAssist\_test\\test", //test base folder

selenium\_server : "http://localhost:4444/wd/hub", //selenium server URL

ptor\_timeout\_regression : 40000, // protractor driver & script time out

// For load test setup -------------------------------

second\_selenium\_host : "mtl-bl1-12-vm08",

ptor\_timeout\_load : 35000, // protractor driver & script time out

ptor\_timeout\_load2 : 150000, //for each iteration, good one: 140000, 150000

api\_call\_wait\_iter: 160, //100 x 1 sec or above, like 120, 125, 130

api\_call\_number : 1500

gui\_run\_number : 1700

};

* + Framework/Test runner/standalone server: nodejs (>0.10.26), protractor(>0.20), grunt (>0.4.4) with concurrent & shell plugin, selenium server (>2.41) running as hub & nodes, and other modules for api + gui test(see detail in the script)
  + Firefox (>27): setup 10 profiles (load test need 8) with name format: TA000<x>, x range from 0 to 9
  + setup test server URL, port, api/gui iteration #, and wait timers in TestConfig.js under c:\liveassist\_test\test\config
* QA test machine (slave):
  + batch script files, to start second selenium group nodes, browser driver:
    - All under c:\liveassist\_test\test, including batch files: ivr-gui-load<x>.bat (x range from 6 - 9, and 0)
    - start secnd 4 selenium nodes group: ivr-gui-load-slave.bat
  + Selenium standalone server: selenium server (>2.41.0) running as nodes
  + Firefox (>27): setup 10 profiles (load test need 8) with name format: TA000<x>, x range from 0 to 9

***Note***: *if test machine changed (ip, port), test batch files need to be updated following the new changes, such as: if master machine changed, the parameter which connects to selenium grid hub server address in ivr-load-test-slave.bat should be changed: "SET SeleniumServerHubAddr=http://10.3.41.73:4450/grid/register"*

* 1. ***Test Setup (example from 07/28/2014)***

*Test environment:*

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

            api+gui test master machine: mtl-bl1-12-vm04, win2k3, 4 core, 4GB; 10 channel api load test driver, 9 channel gui load test driver, protractor + selenium + firefox driver (3 instances)

            api+gui test slave machine 1: mtl-bl1-12-vm08, win2k3, 2 core, 4GB; protractor + selenium + firefox driver (3 instances)

            api+gui test slave machine 2: mtl-bl1-12-vm12, win2k3, 2 core, 4GB; protractor + selenium + firefox driver (3 instances)

*API load test driver*:

api call type: ivr api

api channel #: 10 (with 3 type of emma: high conf, low conf, no-match)

API calls channel assignments with 3 emma type used to trigger IVR api calls (high conf, low conf, and no-match):

            Ch0 – conf. = 0.9, ‘OPERATOR’

            Ch1 – conf. = 0.3, ‘WAIVE\_FEE\_REQUEST’

            Ch2 – conf. = 0.4, ‘UPDATE\_CONTACT\_INFORMATION’

            Ch3 – conf. = 0.2, ‘OPERATOR’

            Ch4 – conf. = 0.2, ‘EXPLAIN\_BALANCE’

            Ch5 – conf. = 0.3, ‘EXPLAIN\_FEE\_REQUEST’

            Ch6 – conf. =0.4, ‘CHECK\_BALANCE’

            Ch7 – conf. = 0.1, ‘BOOK\_FLIGHT’

            Ch8 – conf. = 0.3, ‘no-match’

            Ch9 – conf. = 0.4, ‘no-match’

API calls emma example:

emma from ivr app:

*- low conf, escalated*

*<?xml version='1.0'?><emma:emma version="1.0" xmlns:emma="http://www.w3.org/TR/2007/CR-emma-20071211" xmlns:nuance="http://nr10.nuance.com/emma"><emma:grammar id="grammar\_1" ref="session:http://10.3.41.59:8080/liveassist/data/vxmldemo/ivrapi/callsteering/mainmenu.grxml -1 -1 10000"/><emma:interpretation id="interp\_1" emma:uninterpreted="true" emma:confidence="0.18" emma:grammar-ref="grammar\_1" emma:tokens="book" emma:duration="1340" nuance:eos-silence="1260"/></emma:emma>*

*- high conf, no escalation*

*<?xml version='1.0'?><emma:emma version="1.0" xmlns:emma="http://www.w3.org/TR/2007/CR-emma-20071211" xmlns:nuance="http://nr10.nuance.com/emma"><emma:grammar id="grammar\_1" ref="session:http://10.3.41.59:8080/liveassist/data/vxmldemo/ivrapi/callsteering/mainmenu.grxml -1 -1 10000"/><emma:interpretation id="interp\_1" emma:confidence="0.82" emma:grammar-ref="grammar\_1" emma:tokens="book a flight" emma:duration="1220" emma:mode="voice" nuance:eos-silence="1340"><INTENT*

*conf="0.82">BOOK\_FLIGHT</INTENT><SWI\_literal>book a flight</SWI\_literal><SWI\_meaning>{INTENT:BOOK\_FLIGHT}</SWI\_meaning></emma:interpretation></emma:emma>*

*&utterance=http://mtl-bl1-12-vm02:90/Nuance/callLogs/IvrApiCallSteering/2014/06June/18/15/NUAN-09-13-mtl-bl1-12-vm02-0a032969\_00002680\_53a1e3d9\_0007\_0001-utt001-SAVEWAVEFORM.wav*

*GUI load test driver:*

gui channel #: 9 (with 7 different intents on 7 channels)

GUI calls channel to simulate agent selection/return assignment with 7 intents outcome:

            Ch1 – Intent = ‘BOOK\_FLIGHT’

            Ch2 – Intent = ‘CHECK\_BALANCE’

            Ch3 – Intent = ‘EXPLAIN\_BALANCE ‘

            Ch4 – Intent = ‘EXPLAIN\_FEE\_REQUEST ‘

            Ch5 – Intent = ‘OPERATOR‘

            Ch6 – Intent = ‘UPDATE\_CONTACT\_INFORMATION ‘

            Ch7 – Intent = ‘WAIVE\_FEE\_REQUEST ‘

            Ch8 – Intent = ‘WAIVE\_FEE\_REQUEST ‘

            Ch9 – Intent = ‘OPERATOR ‘

            Ch0 – Intent = backup (not-in-use)

Each channel set call iterations on api: 2200; gui: 2800

            [Notes: api valid # should < gui target #; each escalated ivr call need one gui response, so api call response # depends on gui response #; some api calls can go to the same one gui channel; gui result iter# can be varied among different channels and should be set larger #]

Performance monitor setup on the QA api+gui client test machines to monitor:

- system available memory (memory usage)

- total processor time (cpu usage)

- Private byte on all the selenium server (java process)

1. ***Steps Before Test:***

* Test checkup/config:
  + check the test batch files, test scripts and configuration files ready and put in the correct folders
  + check the test framework/Test runner/standalone server is ready
  + config test server URL, port, api/gui iteration #, and wait timers in TestConfig.js under c:\liveassist\_test\test\config
* Stop live assist server (from NMS or command line)
* clean up
  + quit last running script (if has)
  + run “clean\_client.bat” to kill java process (protractor/selenium), firefox driver instances and quit all the opened command line windows (QA nodejs load test scripts).
* backup last run logs(if necessary):
  + call logs
  + diagnostic logs
  + performance log (if have)
* start live assist server (and audio server if used)
  + starting Live Assist server:
    - su -s /bin/bash nuance --session-command liveassist
  + starting Live Assist Audio server:
    - su -s /bin/bash nuance --session-command liveassistaudio
* start performance monitor on Live Assist server
  + The perfmon script& result location: /opt/performance\_monitor
  + 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)
  + Run the perfmon script for Live Assist: ‘sh liveassist\_perfmon.sh’
  + 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)

A screenshot on Live Assist server (mt-ray-vm01) as follow before the api+gui load test started:

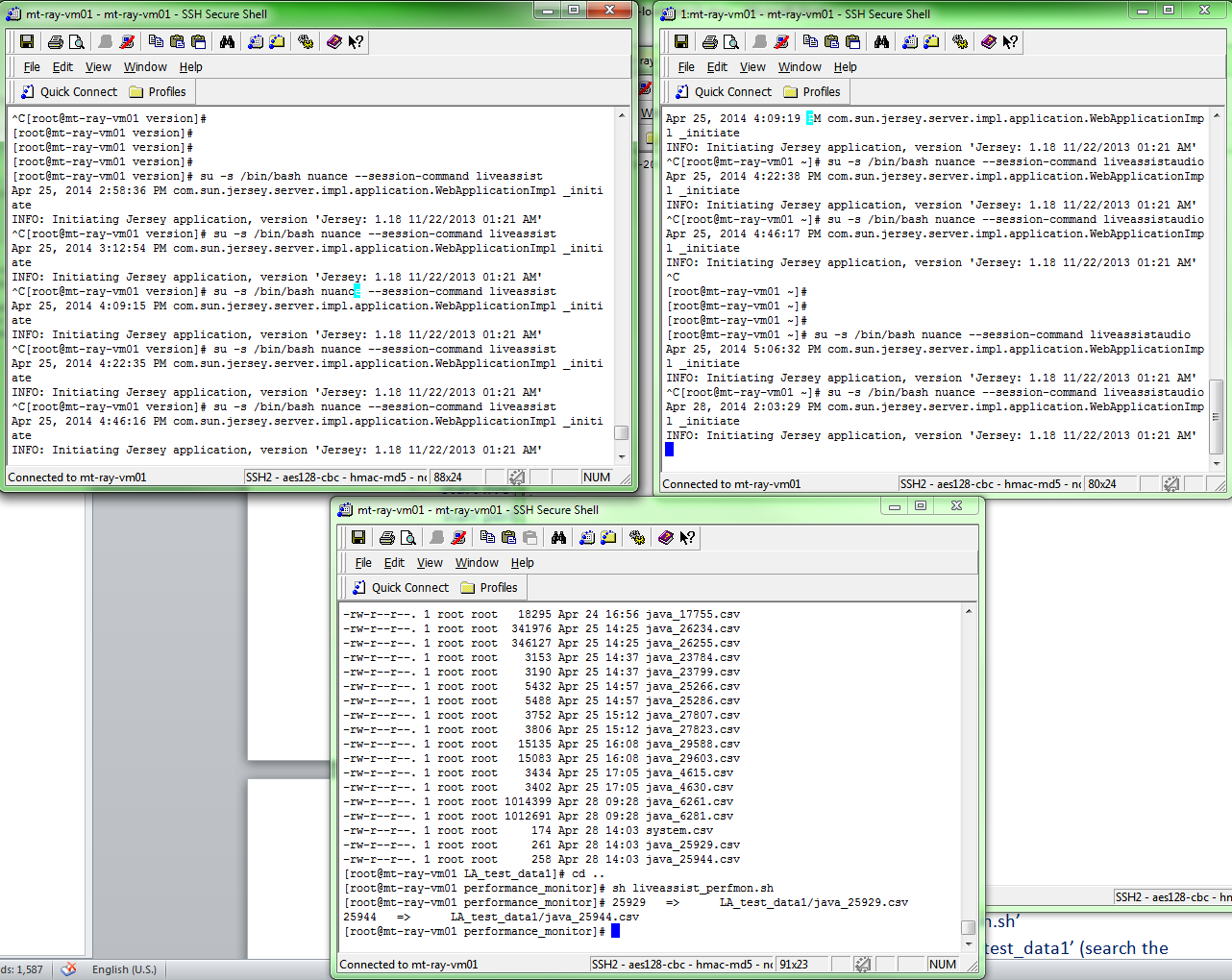


Fig. screenshot from Live Assist server: started audio server, live assist server and perfmon script

* Suggested settings for all the selenium/protractor web-driver (firefox) profiles (TA0000 – TA0009)

1. From “View\Toolbars” tab, uncheck “Bookmarks Toolbar”
2. Right click from toolbar and click “Customize”
3. Remove “Google search” by dragging it to the “Customize Toolbar” window
4. From the “Customize Toolbar” window, click “Use Small Icons” check box then hit “Done”
5. Click “Tools\Options” then set the following:  
   a. “Main” tab  
   - set Home Page to “about:blank”  
   - uncheck “Show the Downloads..” option  
   b. “Tabs” tab  
   - Select “a new window” for new pages  
   - Uncheck all warning options  
   c. “Content” tab  
   - uncheck “Block pop-up” windows option  
   d. “Privacy” tab  
   - uncheck all “History” options  
   e. “Security” tab  
   - uncheck all “Security” options  
   - click “Settings” and uncheck all warning options  
   f. “Advanced” tab  
   - Uncheck “autoscrolling” option from “General” tab  
   - uncheck “warn me …” and “Ssearch Engines”option from “Update” tab
6. From “Tools\Add-ons” install the following:  
   - Firebug: allows you to edit, debug, and monitor CSS, HTML, and JavaScript on your application under test  
   - Selenium IDE: allows you to record, edit, and debug Selenium tests  
   - ScreenGrab: saves entire webpages as images.
7. From the address bar type “about:config” and add the following by right-click anywhere on the page and selecting “new”  
   - extensions.update.notifyUser (type=boolean; value=false)  
   - extensions.newAddons (type=boolean; value=false)
8. ***Concurrent api + gui load test run:***

* All test scripts are nodejs/JS files:
  + The grunt file and batch file under base folder, c:\liveassist\_test\test
  + api and gui nodejs load test script under: c:\liveassist\_test\test\QA\_LOAD3
  + Test config script TestConfig.js, c:\liveassist\_test\test\config
* Configure the api + gui load test:
  + Configure the load test using TestConfig.js with following major parameters:
    - server: Live Assist server IP or hostname
    - port: Live Assist server port
    - test\_location: load test base folder
    - second\_selenium\_host: selenium slave server IP or hostname
    - api\_call\_number: api calls triggered number for each channel
    - gui\_run\_number: gui automation response simulation iteration numbers for each channel
    - ptor\_timeout\_load2, timeout for protractor on each iteration, good one: 100000 or above like 140000, 150000
    - api\_call\_wait\_iter, api call wait for gui response iterations, - 100 x 1 sec or above, like 120, 125, 130

Note: Due to api calls may escalate to any agent GUI, so user need to assign separate api call number and gui run number based on the target expected call iteration number, for example: web api - target each chl 1400 calls, total 11200 call iter, for overnight load:

api\_call\_number : 1500,

//for api each 1400 calls x 8 agents = total 11200 iter (note: api\_call\_number = expected call iter number + 100, to make sure all the api test iteration can be completed)

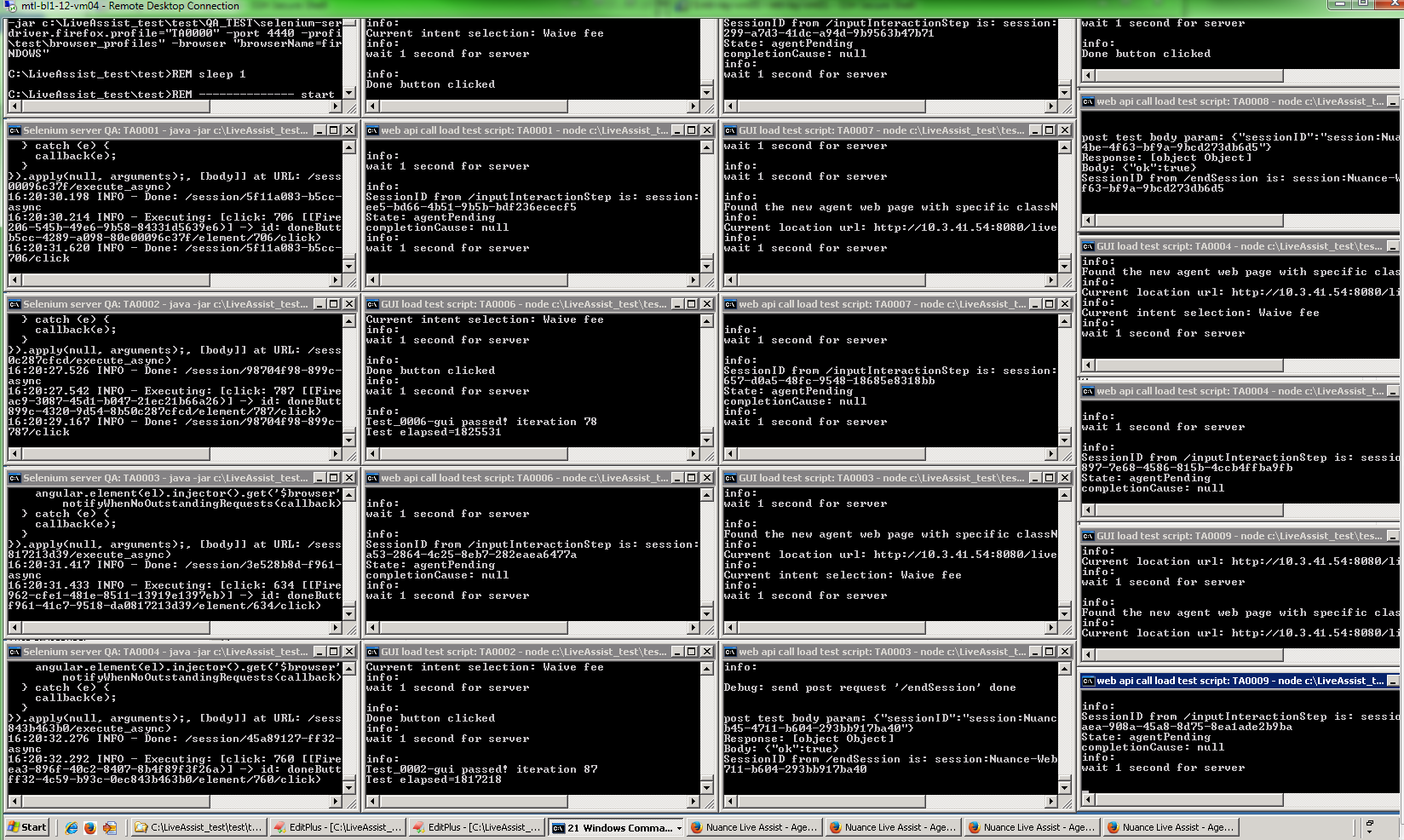
gui\_run\_number : 1700 //for gui response, each 1400 x 8 agents = total 11200 call iterations (note: gui\_run\_number = api\_call\_number + 200, to make sure all the GUI test iteration can be completed)

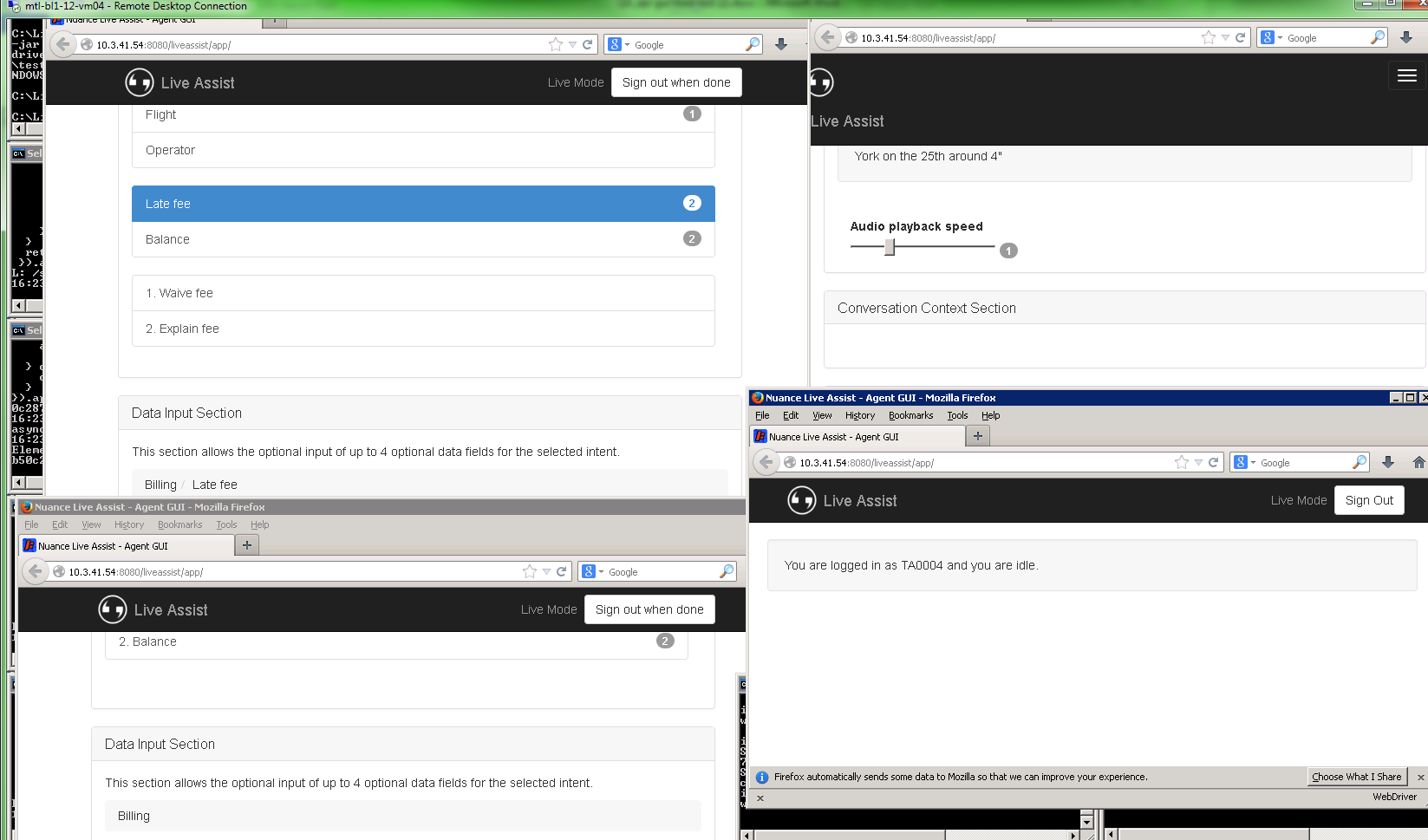
The above setup load test in current QA env may run about 9 hours.

* Start selenium master and slave server in the two QA test machines
  + master test machine: start.bat, to start selenium grid central hub server, and 5 selenium node servers
  + slave test machine: ivr-gui-load-slave.bat, to start another 5 selelnium node server
* Start concurrent api+gui load test:

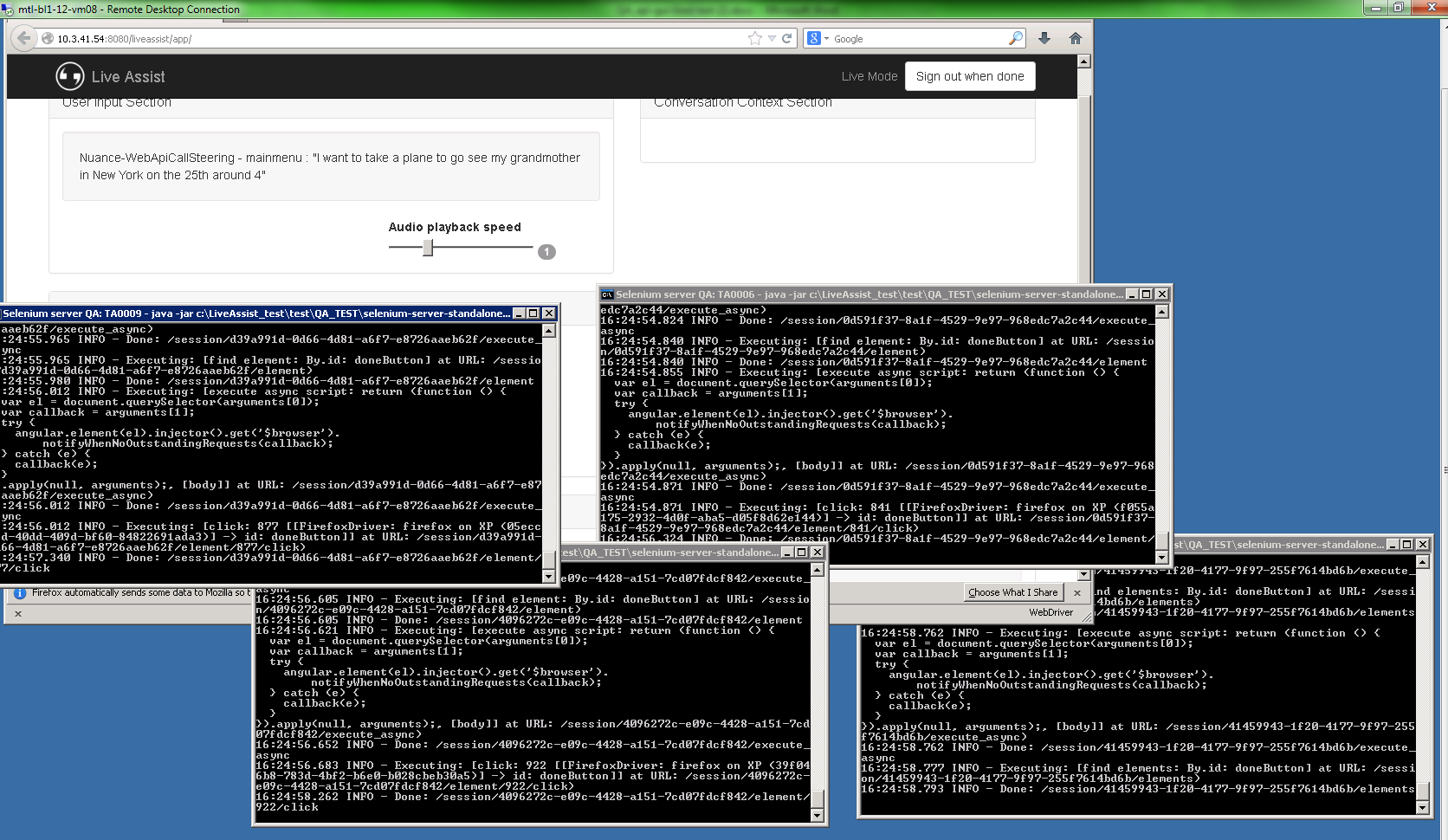
Run “grunt loadtest-node”

Example api + gui load test run (8 agents) screen shot as follow:





***Fig. screenshot from master test machine after api+gui load test started with 1st 4 agent group and all channel api calls.***



***Fig. screen shot from slave test machine from api+gui load test (2nd 4 agents group)***

***Note****: each of above steps need to wait specific time (at least >5 sec) to make sure servers started properly then go to the next steps.*

* *Start performance monitor on client side for firefox browsers (optional, for client browser cpu/memory monitor)*

1. ***Steps After Test Complete:***

* Stop live assist server
* Stop performance monitor script running on Live Assist server
* Stop grunt load test running on the master api+gui test machine (ctrl + c then confirmed ‘y’), then run ‘clean\_client.bat’ to close all the opened command windows for testing (such as for 8 agent load test, 21 cmd windows) and selenium server(java processes).
* Stop test and closed opened command windows on the slave test machine, then run’clean\_client.bat’ to close all the opened windows and selenium server(java processes).
* Backup logs:
  + api call test log (must)
  + gui test log (must)
  + performance log (must)
  + call logs (optional, if required)
  + diagnostic logs (optional, if required)
  + copied all the logs and QA result summary file to public server(optional, especially if test run on RC or feature complete build)
* Analyze data:
  + api call test logs (must) -> test pass/fail (<2%)
  + gui test logs (must) -> test pass/fail (<2%)
  + performance log (optional, if perfmon used during the test)
  + error/exception/issue in diagnostic log (optional, for failed test)
  + Run specific perl QA tool to check and analyze result logs:
    - Copy all the api/gui log results under specific folder under c:\liveassist\_test\test
    - Run perl script to parse GUI logs:
    - ‘perl parseAPITestlog.pl <target\_gui\_load\_test\_result\_files\_dir> <target\_each\_chl\_gui\_iteration>’
    - Run perl script to parse API logs:

‘perl parseAPITestlog.pl <target\_api\_load\_test\_result\_files\_dir> <target\_each\_chl\_call\_iteration> ’

* + Run specific call log parse DEV tool to check and analyze load test call logs:

The script to parse call log can be found and run from LA server, CentOS6, setup 90 port to support remote check:

calllog-stats.py -r -p -i http://<LA\_server>/Nuance/callLogs

ex.: calllog-stats.py -r -p -i http://10.3.41.54:90/Nuance/callLogs

Notes: a) you can add re-direct command at the end of perl command line to output the summary into a specific file like ‘> api\_test\_summary.txt’; b) perl script report each channel pass/fail and actual call iterations and also give summary on all the channels. C) run the gui test first to have the actual valid gui response iterations then base on that to set realistic api call target iterations. The current Live Assist escalate calls not spread event to different agents, some agents (gui) may have more calls than others.

An example api+ gui load test report for 9200 calls, 8 agent, ~7 hour:

api call: web api

api channel: 8

gui channel: 8

each api/gui channel valid call iterations: 1150

each api/gui channel set call iterations (higher # than target): 1500-1700

Total valid call iteration: 1150 x 8 = 9200

each api channel failed calls (check iter < 1150 as valid range, no matter which agent gui reponse):

ch1 ch2 ch3 ch4 ch5 ch6 ch7 ch8

0 1 1 0 0 0 1 0

API LOG PARSING SUMMRY:

Log dir: ivr\_8chl\_1150each\_load\_04252014

Target iteration: 1150

Total calls: 9200

total calls passed: 9197

total calls failed: 3

calls pass rate is 99.97 %

calls fail rate is 0.03 %

GUI LOG PARSING SUMMRY:

Log dir: ivr\_8chl\_1150each\_load\_04252

Target each channel iteration: 1200

Total calls: 9207

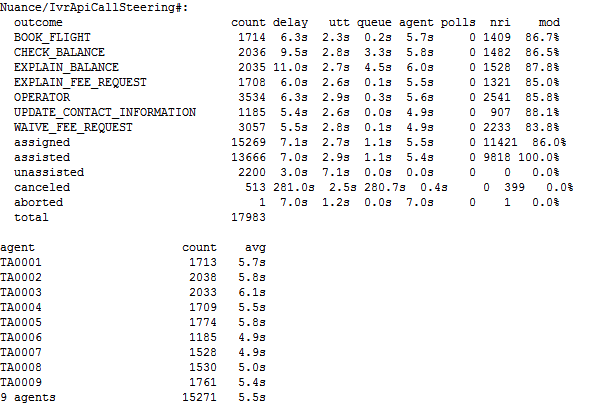
total calls passed: 9207

total calls failed: 0

calls pass rate is 100.00 %

calls fail rate is 0.00 %

**An analyze result example from call log parse tool:**



1. ***Test resource (updated for iteration 5.1 demo build)***

* The api + gui concurrent load test batch files: [\\mt-nasrops01\qa\LiveAssist\Test\_Set\Iteration5.1\LiveAssist\_Test\test](file:///\\mt-nasrops01\qa\LiveAssist\Test_Set\Iteration5.1\LiveAssist_Test\test)
* The api + gui concurrent load test nodejs scripts:
  + Iteration 5.1:[\\mt-nasrops01\qa\LiveAssist\Test\_Set\Iteration5.1\LiveAssist\_Test\test\QA\_LOAD3](file:///\\mt-nasrops01\qa\LiveAssist\Test_Set\Iteration5.1\LiveAssist_Test\test\QA_LOAD3)
  + Iteration 5.2: [***\\mt-nasrops01\qa\LiveAssist\Test\_Set\Iteration5.2\Load\_Test\_Bak\LiveAssist\_test\test***](file:///\\mt-nasrops01\qa\LiveAssist\Test_Set\Iteration5.2\Load_Test_Bak\LiveAssist_test\test)
* All the latest load test scripts/batch files can be downloaded or sync up from the Perforce as well, QA source stream entrd\_liveassqa, and download all the load test batch files under \LiveAssist\_test\test, and all the load test nodejs script under \LiveAssist\_test\test\QA\_LOAD3
* Ref: The current test results on iteration 5.1 RC build as reference (>9000 call iterations, 8 agent, ~7 hours, passed): [\\mt-nasrops01\qa\LiveAssist\QA\_tech\Test\_logs\api-gui-load\_result\_iter51-rc\ivr\_8chl\_1150each\_load\_04252014](file:///\\mt-nasrops01\qa\LiveAssist\QA_tech\Test_logs\api-gui-load_result_iter51-rc\ivr_8chl_1150each_load_04252014)

***Notes:***

* *All the load test batch files and scripts can be found from Perforce QA stream (entrd\_liveassistqa) as well.*
* *The above load test setup and scripts satisfied the basic QA concurrent test functions and it passed 1000 iteration calls with 8 agent on the last iteration demo build, however, those scripts will be continued to be improved and add more supports/functions in the future*
* *Web api also provided for testing, instead run wit prefix ‘ivr’ test scripts/batch files, user can run ‘web’ prefix or just ‘api’ prefix test scripts (nodejs) or batch files. The IVR and Web api calls can be configured to run parallel as well.*
* *Ref. current dedicated QA api+gui load test env setup (used by end of iteration 5.1)*
  + *Live Assist server: mt-ray-vm01 (CentOS)*
  + *api+gui load test machine 1 (with master load test scripts/batches files): mtl-bl1-12-vm04*
  + *api+gui load test machine 2 (with slave server load test batches files): mtl-bl1-12-vm08*
* *Some reference links:*
  + *Nodejs:* [*http://nodejs.org/api/*](http://nodejs.org/api/)
  + *Webdriver:* [*https://code.google.com/p/selenium/wiki/WebDriverJs*](https://code.google.com/p/selenium/wiki/WebDriverJs)
  + *Protractor:* [*https://github.com/angular/protractor*](https://github.com/angular/protractor)
  + *Selenium grid (hub/node):* [*http://docs.seleniumhq.org/docs/07\_selenium\_grid.jsp*](http://docs.seleniumhq.org/docs/07_selenium_grid.jsp)
  + *Grunt:* [*http://gruntjs.com/*](http://gruntjs.com/)