ORACLE
ENTERPRISE MANAGER

An Oracle White Paper
June 2014

Oracle Load Testing Deep Dive

ORACLE

Product Overview

Oracle Application Testing Suite is an integrated, full lifecycle solution which ensures application quality and performance with complete end-to-end testing and test management capabilities. Oracle Application Testing Suite helps deliver high quality applications with three separately licensed products.

- Oracle Functional Testing for automated functional and regression testing of Web applications, Web Services and Oracle packaged applications.

- Oracle Load Testing for automated load and performance testing of Web applications, Web Services, Oracle packaged applications and databases.

- Oracle Test Manager for process management throughout the testing lifecycle, including test planning, requirements management, test management, test execution and defect tracking.

Introduction

An IT organization faces many challenges from its day to day operation. One of the challenges is to ensure their applications and infrastructure can scale to meet the needs of the end users. To achieve this, it is necessary to review performance of the application and detect bottlenecks, if any. Oracle Load Testing, one of the components of Oracle Application Testing Suite, helps simulate realistic load on the Application under Test thereby helping IT organization identify bottlenecks and scalability issues before going to production. This paper addresses the inner workings of Oracle Load Testing (OLT) and also highlights the optimal configuration to create load test scripts and scenarios. It looks into the aspect of creating robust load test scripts and focuses on the architecture and life cycle of Oracle Load Testing. The runtime aspects of OLT and the behavior of databank record during load tests will also be discussed.

Generating Realistic Load using OpenScript Load Test Script

In order to make the right conclusions from the load test report, it is imperative to create load test scripts that generate realistic traffic on the Application under Test. This section looks at the OpenScript[1] recording and playback preferences which let users create scripts that generate realistic traffic on execution.

By default, OpenScript recording preferences prevent static web resources from getting recorded into the script. Not having static web resources in the script keeps the script clean and makes it easier to understand, enhance and debug. At the same time, to simulate a real transaction, OpenScript automatically sends requests for these static resources while executing the script. Default Record and Playback configuration for Load Test modules may not always generate realistic traffic while executing the script. The default configuration can be modified to make sure that the script playback generates all the requests using the simulated cache.

Default Record Configuration

To understand how default recording works, imagine two filters placed between OpenScript and the Application under Test. As each http request/response passes through these filters, OpenScript decides if the request should be recorded into the script.

URL filters, as seen in Figure 1, comprise the first filter. URL filters essentially match the request URL or the content-type of the response based on the regular expression or wild card. Once a request/response is filtered by one of these rules, the request will not get recorded into the script. For example, if a rule filters all images based on response content type, then the recorded script will not have any images.

TABLE 1. LIST OF TAGS PARSED BY DOWNLOAD MANAGER

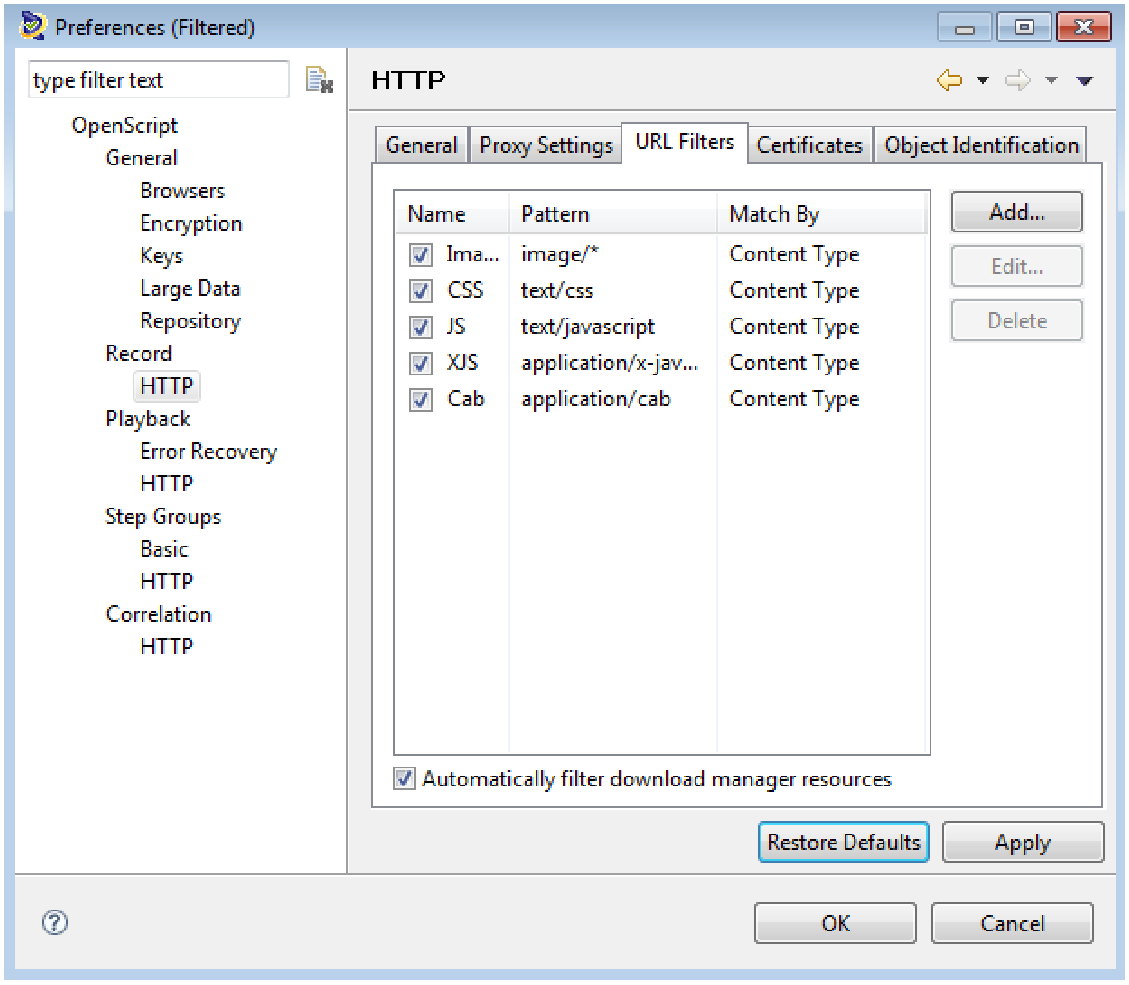
STATIC RESOURCE TYPE	PARSED TAGS
CSS	<link>
Images	<img>, background attribute of a tag, <style> with background url
Embedded object resource	<embed>, <object>
Script resource	<script>
Applet	<applet>

[1] OpenScript is a standalone application based on Eclipse RCP. It is a component of Oracle Functional Testing and is used to generate both load and functional test scripts.

Download Manager (DM), enabled in Figure 1, makes up the second filter. During recording, whenever non-static resources like HTML pages are encountered, Download Manager parses the response using the rules mentioned below to know the static resources embedded in that page.

If a request/response is not filtered by the rules in URL filters and if the response is a static web resource (css, js, swf etc), Download Manager checks if the static web resource is in the list of already parsed static web resources. If not, the request for this response will be recorded in the script.

During playback, Download Manager, if enabled, automatically retrieves all the static resources that it can, from each containing page using multiple connections.



Name	Pattern	Match By
✓ Ima...	image/*	Content Type
✓ CSS	text/css	Content Type
✓ JS	text/javascript	Content Type
✓ XJS	application/x-jav...	Content Type
✓ Cab	application/cab	Content Type

✓ Automatically filter download manager resources

Figure 1. Default record configuration

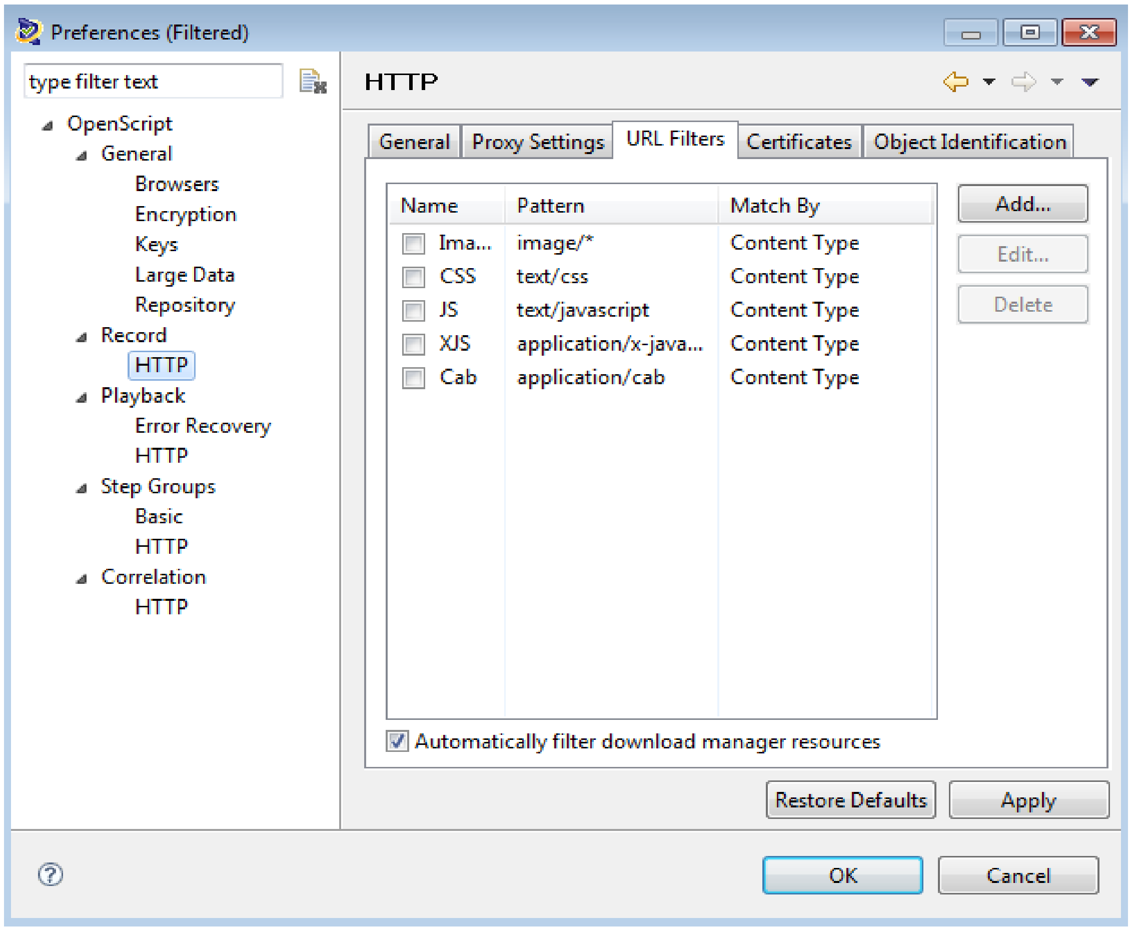
Limitations

Although default configuration works well in most cases, there are cases where it will miss retrieving some resources.

- Certain resource URLs that are not inside an HTML tag are not parsed by the Download Manager. For example, a resource URL that is dynamically composed by JavaScript cannot be parsed by Download Manager.
- Certain resource URLs that do not appear directly in the HTML page contents are not parsed by the Download Manager. For example, an HTML page that imports '.css' files will not be parsed. When the browser loads the HTML page, it automatically loads the '.css' file and downloads any '.gif' resources.

Modified Record Configuration

In view of the limitations, a change in the record and playback configuration along with some custom code will make sure that all resources are downloaded during script playback.



Name	Pattern	Match By
☐ Ima...	image/*	Content Type
☐ CSS	text/css	Content Type
☐ JS	text/javascript	Content Type
☐ XJS	application/x-java...	Content Type
☐ Cab	application/cab	Content Type

☑ Automatically filter download manager resources

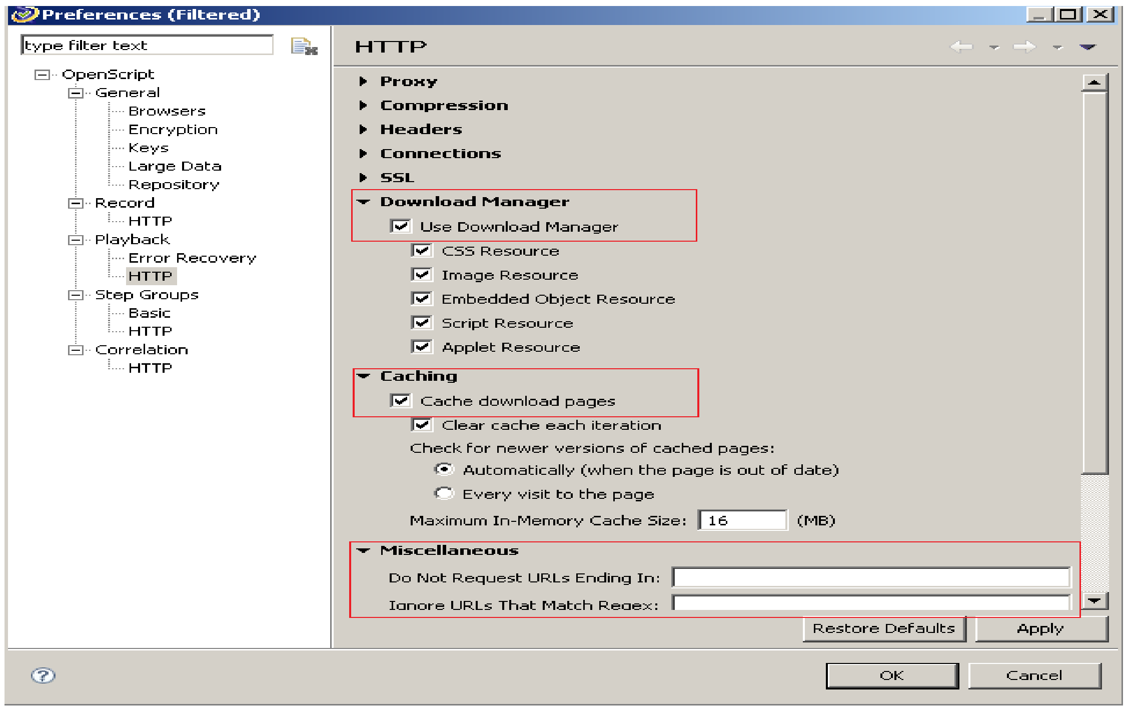
Figure 2. Modified record configuration

In the new configuration shown in Figure 2, all responses go through only the Download Manager filter, since all URL filters are disabled. Resources which can be parsed by Download Manager from the HTML page will now be recorded in the script. Because the browser downloads static web resources in parallel using multiple connections, static resources in each step group should be wrapped in concurrent blocks, as shown below.

```
beginStep("Landing Page");
{
    http.beginConcurrent("homepage_resources");
    // requests for static resources
    http
    .get(
        178,
        "http://xyz.com/i/themes/theme_2/images/bt-gray-l.png",
        null,
        true, "UTF8", "GB18030");

    http.endConcurrent("homepage_resources");
}
endStep("Landing Page");
```

Figure 3 shows the playback configuration that retrieves static web resources during script playback. Besides Download Manager, caching should be enabled to simulate real browser session. Make sure 'Do Not Request URLs Ending In:' is blank.


Preferences (Filtered)

type filter text

- OpenScript
  - General
    - Browsers
    - Encryption
    - Keys
    - Large Data
    - Repository
  - Record
    - HTTP
  - Playback
    - Error Recovery
    - HTTP
  - Step Groups
    - Basic
    - HTTP
  - Correlation
    - HTTP

HTTP

- Proxy
- Compression
- Headers
- Connections
- SSL
- Download Manager
  - [✓] Use Download Manager
  - [✓] CSS Resource
  - [✓] Image Resource
  - [✓] Embedded Object Resource
  - [✓] Script Resource
  - [✓] Applet Resource
- Caching
  - [✓] Cache download pages
  - [✓] Clear cache each iteration
  - Check for newer versions of cached pages:
    - ● Automatically (when the page is out of date)
    - ○ Every visit to the page
  - Maximum In-Memory Cache Size: 16 (MB)
- Miscellaneous
  - Do Not Request URLs Ending In:
  - Ignore URLs That Match Regex:

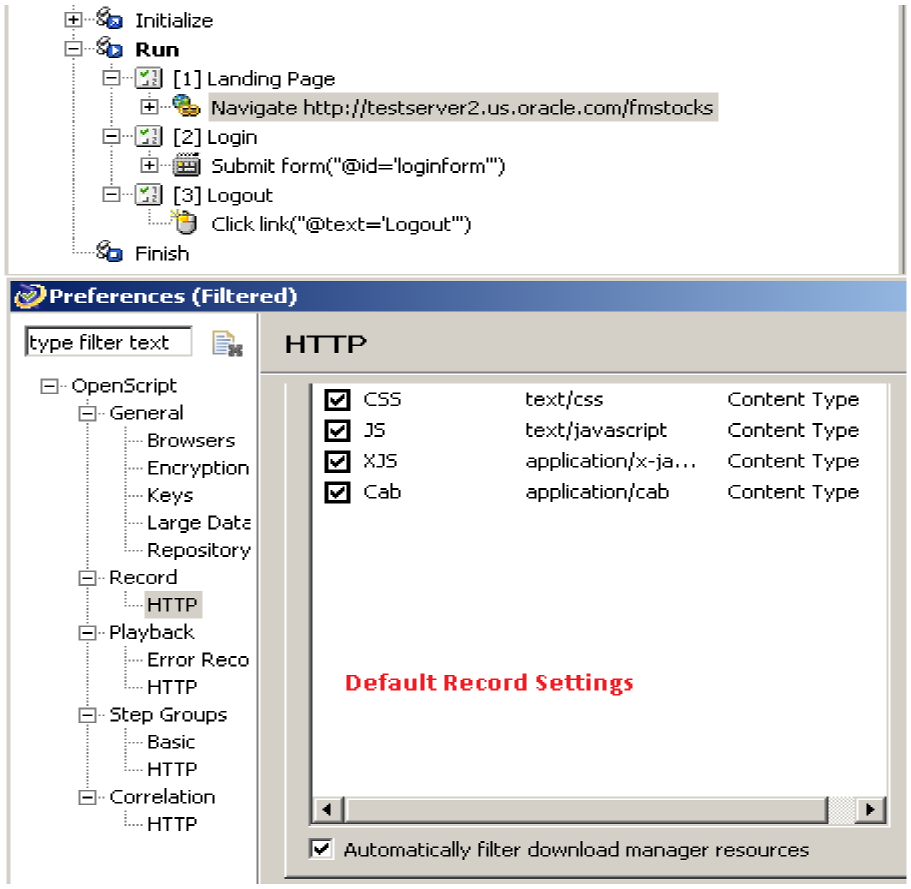
Figure 3. Playback configuration

Example

In this example, the following CSS snippet requests for an image when a user logs into the application

```
TD.logo
{
     BACKGROUND-COLOR: #006699;
     BACKGROUND-IMAGE: URL(images/FMBanner.gif);
     BACKGROUND-POSITION: left top;
     BACKGROUND-REPEAT: no-repeat;
     TEXT-ALIGN: right
}
```

Shown in Figure 4 is a simple workflow recording using default record configuration. Figure 5 shows additional request for a static resource in the script when the same workflow was recorded with modified record configuration (i.e. filters are removed and only Download Manager is enabled).



```
Initialize
Run
  [1] Landing Page
    Navigate http://testserver2.us.oracle.com/fmstocks
  [2] Login
    Submit form("@id='loginform'")
  [3] Logout
    Click link("@text='Logout'")
Finish
```

Preferences (Filtered)

type filter text

OpenScript
- General
  - Browsers
  - Encryption
  - Keys
  - Large Data
  - Repository
- Record
  - HTTP
- Playback
  - Error Reco
  - HTTP
- Step Groups
  - Basic
  - HTTP
- Correlation
  - HTTP

HTTP

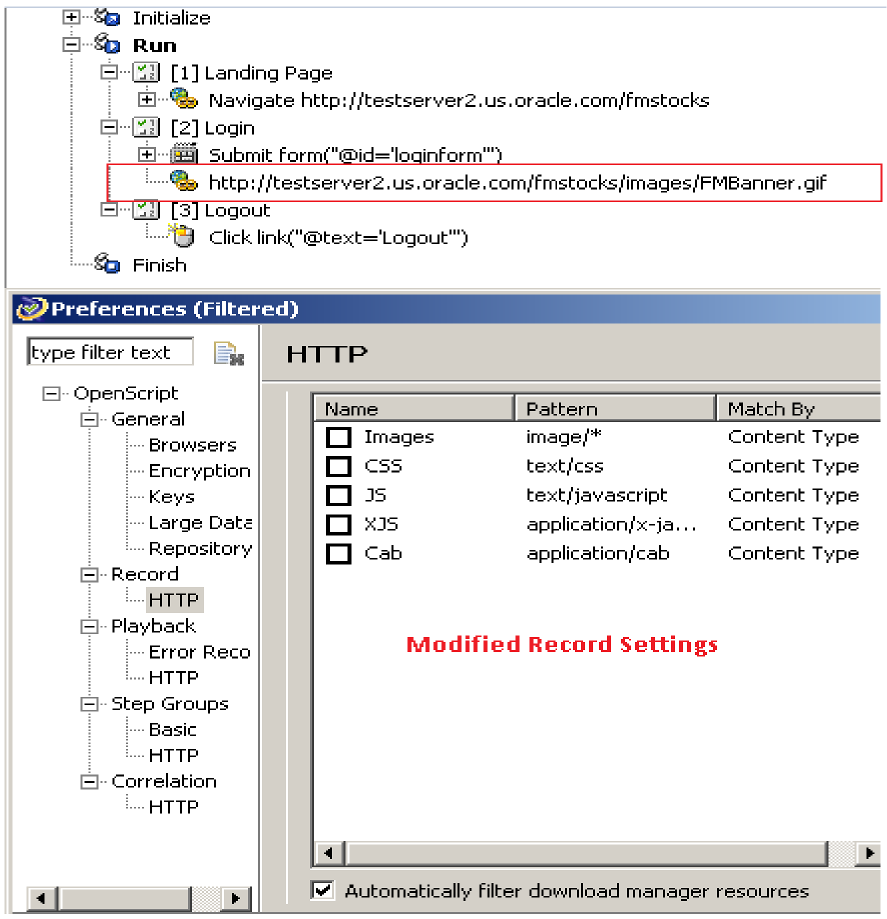
☑ CSS	text/css	Content Type
☑ JS	text/javascript	Content Type
☑ XJS	application/x-ja...	Content Type
☑ Cab	application/cab	Content Type

Default Record Settings

☑ Automatically filter download manager resources

Figure 4. Script with default record configuration

Since URL filters were removed, Download Manager applies its rules on all the resources of the application. Because Download Manager is not able parse the image URL from the CSS file, it lets the recorder write this request to the script. Now, playback of the script recorded with modified configuration sends all the resource requests to the server, unlike the script recorded with default configuration.



```
Initialize
Run
  [1] Landing Page
    Navigate http://testserver2.us.oracle.com/fmstocks
  [2] Login
    Submit form("@id='loginform'")
    http://testserver2.us.oracle.com/fmstocks/images/FMBanner.gif
  [3] Logout
    Click link("@text='Logout'")
Finish
```

Name	Pattern	Match By
☐ Images	image/*	Content Type
☐ CSS	text/css	Content Type
☐ JS	text/javascript	Content Type
☐ XJS	application/x-ja...	Content Type
☐ Cab	application/cab	Content Type

☑ Automatically filter download manager resources

Figure 5. Script with modified record configuration

Once the load test script is recorded and playback is verified in OpenScript, the next phase of the load test project is to setup the Oracle Load Testing infrastructure and configure it.

Oracle Load Testing Architecture

This section looks at the architecture of Oracle Load Testing. It highlights the components that will be installed for the chosen installation type and takes a closer look at the life cycle of OLT.

Figure 6 depicts a high level component overview of Oracle Load Testing. All of the components will be installed with a full installation of ATS. On remote machines which are exclusively used to generate load on the Application under Test, it is sufficient to have only ATS Agent installed. ATS Agent installs and configures Data Collector, Agent Manager, and OLT Agent components.

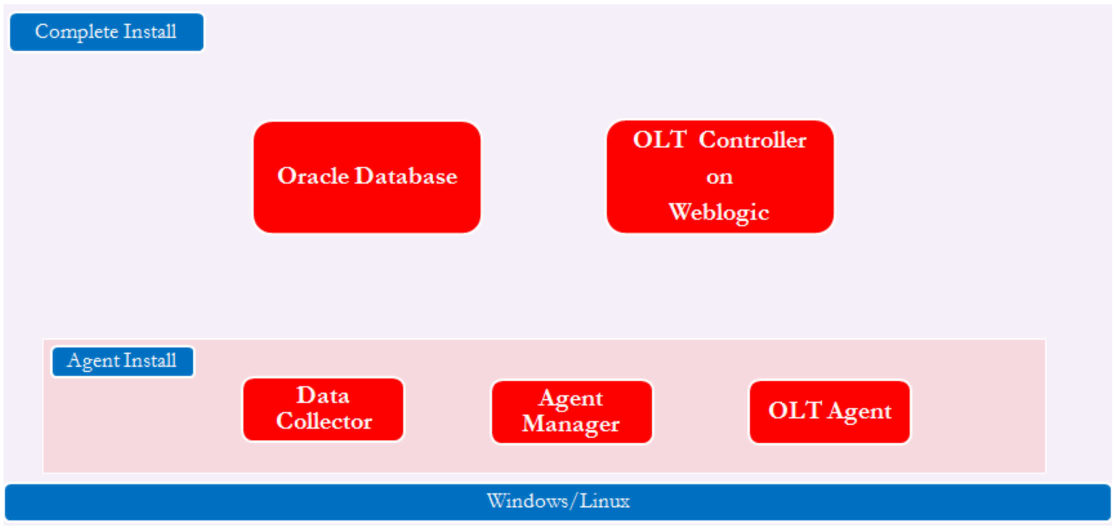


Figure 6. Oracle Load Testing Architecture

- Oracle Database – A complete installation of ATS will install Oracle Database 10g Express Edition (XE). Oracle XE database is useful for demo purpose however for production load testing it is recommended to use Oracle Database 11g Enterprise Edition and have it installed on a separate server.
- Oracle Load Testing Controller – A web application deployed on Weblogic running on Windows/Linux. Once successfully authenticated by Agent Manager, OLT Controller issues commands to the Agent Manager to control the load test.
- Agent Manager – Agent Manager manages the life cycle of OLT Agent and Data Collector.
- OLT Agent – Agent process generates load on the Application under Test. Each thread of the agent process simulates Virtual Users. The number of agent processes spawned by the Agent Manager depends on the configuration of the load test scenario.
- Data Collector – If ServerStats is configured in the load test scenario, OLT Controller requests Agent Manager to spawn the Data Collector process. Data Collector process connects to the monitored systems and retrieves configured metrics at specified intervals.

Load Testing Life Cycle in OLT

Understanding how different components of OLT work together to execute a load test scenario is helpful in setting up an environment for a load test and to troubleshoot common connectivity issues between OLT Controller and Agent Manager. Figure 7 depicts the life cycle of a load test in OLT.

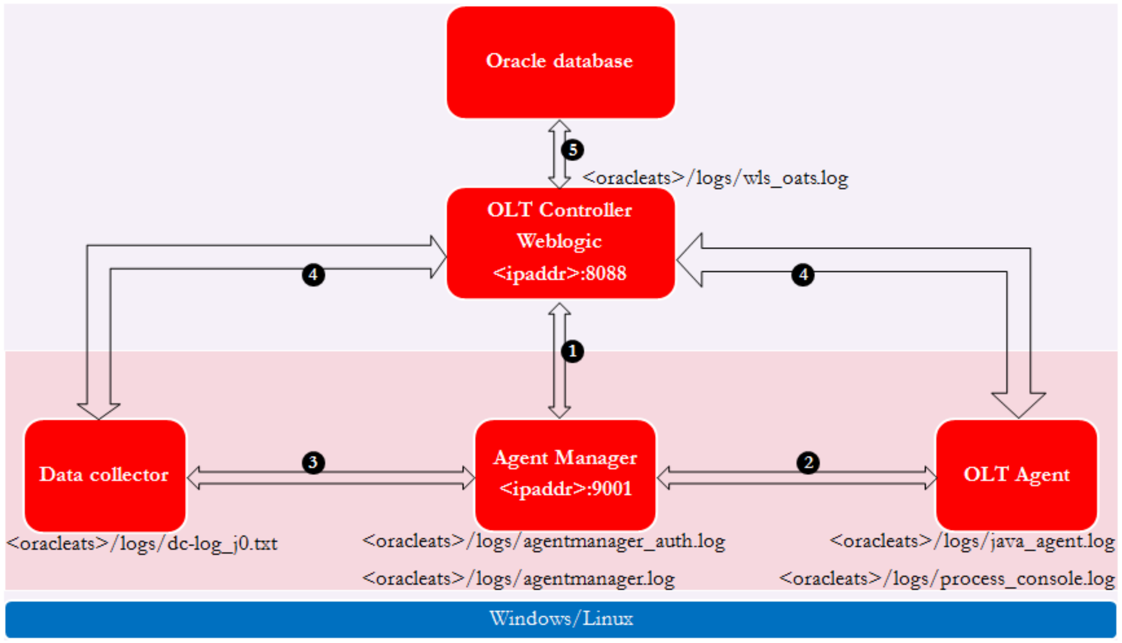

Oracle database
5
<oracleats>/logs/wls_oats.log
OLT Controller
Weblogic
<ipaddr>:8088
4
4
1
Data collector
3
Agent Manager
<ipaddr>:9001
2
OLT Agent
<oracleats>/logs/dc-log_j0.txt
<oracleats>/logs/agentmanager_auth.log
<oracleats>/logs/agentmanager.log
<oracleats>/logs/java_agent.log
<oracleats>/logs/process_console.log
Windows/Linux

Figure 7. Communication between OLT components

1. When a load test is started, OLT Controller initiates a connection to the Agent Manager listening on port 9001. Once the connection is established, Agent Manager checks if OLT Controller has provided the right credentials to make use of its service. If the load test fails to start, check <oracleats>/logs/*agentmanager_auth.log* file on the agent machine to see if OLT Controller has failed to get authenticated. Logging output related to OLT Controller can be seen in <oracleats>/logs/*wls_oats.log* file.

2. On successful authentication, OLT Controller requests Agent Manager to spawn an agent process which generates load on the Application under Test and simultaneously captures transaction metrics. Each agent process establishes a connection to the OLT Controller, listening on port 8088. All errors and warning messages generated during the scenario execution will be logged to <oracleats>/logs/*java_agent.log* file on the agent box. All other output will be directed to <oracleats>/logs/*process_console_<xxx>.log* files. Each agent process creates one process_console_xxx.log file for all of its logging needs.

3. If ServerStats is configured in the scenario, Agent Manager spawns a Data Collector process. Data Collector process connects to monitored systems to gather metrics at regular intervals. Data Collector sends logging output to <oracleats>/logs/*dc-log_j0.txt* file.

4. OLT Agent process and Data Collector establishes a connection with OLT Controller to transfer transaction and ServerStats metrics to the Controller.
5. OLT Controller saves the data it receives from OLT Agent and Data Collector to the database.

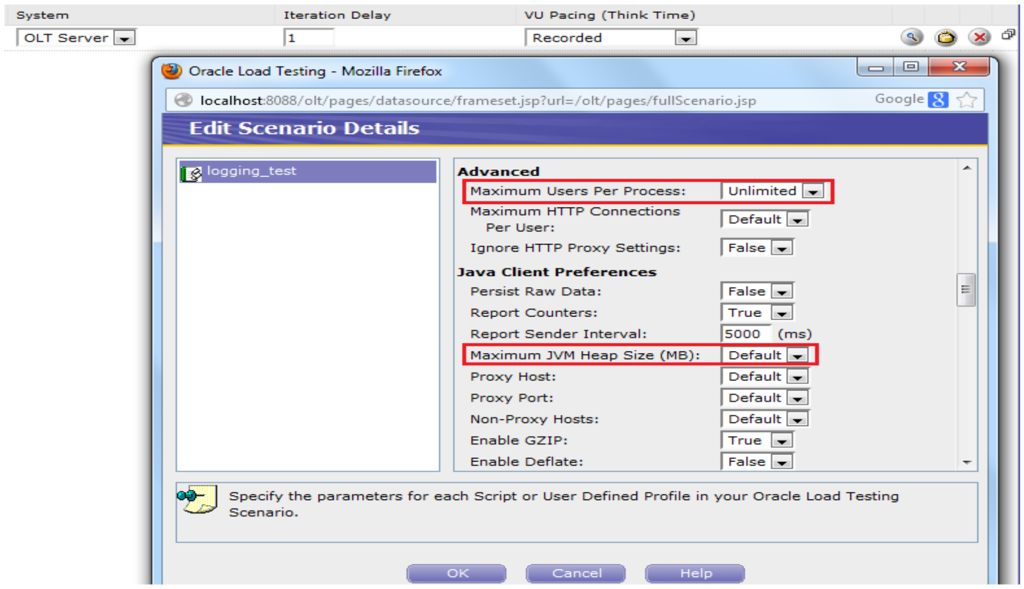
When the load test is completed or when the user stops/aborts a running test, OLT Controller sends a kill signal to OLT Agent and Data Collectors.

OLT Controller and Agent Runtime Behavior

This section looks at how OLT scenario configuration affects the runtime behavior of the OLT and agent processes. Controlling this runtime behavior is critical when there is a need to efficiently utilize the available system resources for the load test.

Agent Process Invocation

As explained in the Load Testing Life Cycle section, OLT Controller spawns an agent process during the load test. The Number of agent processes spawned by OLT Controller and the heap size of the spawned agent process depends on the '*Maximum Users per Process*' and '*Java Client Preferences*' settings (see Figure 8).


System
OLT Server
Iteration Delay
1
VU Pacing (Think Time)
Recorded
Oracle Load Testing - Mozilla Firefox
localhost:8088/olt/pages/datasource/frameset.jsp?url=/olt/pages/fullScenario.jsp
Edit Scenario Details
logging_test
OK
Cancel
Help

Advanced	
Maximum Users Per Process:	Unlimited
Maximum HTTP Connections Per User:	Default
Ignore HTTP Proxy Settings:	False
Java Client Preferences	
Persist Raw Data:	False
Report Counters:	True
Report Sender Interval:	5000 (ms)
Maximum JVM Heap Size (MB):	Default
Proxy Host:	Default
Proxy Port:	Default
Non-Proxy Hosts:	Default
Enable GZIP:	True
Enable Deflate:	False

Specify the parameters for each Script or User Defined Profile in your Oracle Load Testing Scenario.

Figure 8. Edit Scenario Details window

With default *Java Client Preferences*', OLT Controller spawns an agent process with a heap size of 1G, which runs Vusers of all the scripts in the scenario. If '*Maximum Users Per Process*' is set to 2 and if the process is executing more than half of the total Vuser load for the scenario, then OLT Controller spawns a second agent process with identical properties as that of the first one. The key thing to note is that since all of the scripts in the scenario have identical 'Java Client Preferences', one agent process executes Vusers of all the scripts in the scenario until another agent process shares the load.

If a scenario has 2 scripts (A and B) and the heap size for script A is default (1G) and for Script B is 512MB, then OLT Controller will spawn 2 agent processes because 'Java Client Preferences' configuration is different for each script. In this case, all Vusers of script A are run by agent process with heap size of 1G and all Vusers of script B are run by agent process with heap size of 512MB. '*Maximum Users per Process*' in this case affects all of the spawned processes.

Tuning JVM Heap

The default JVM configuration shipped with ATS may have to be modified to match the needs of a load test scenario. In this context, it is important to keep in mind that except 64-bit Linux builds, all other ATS builds are shipped with 32-bit JVM. Two primary areas where user can modify the JVM parameters are:

- Heap size of Weblogic server where OLT is deployed. This is done to make sure that OLT is not overloaded during large load tests.
- Heap size of OLT Agent processes which generate load on the Application under Test.

Modifying JVM Heap Size for OLT Controller

The following table presents the recommended heap size for Weblogic server based on Operating System and architecture.

TABLE 2. HEAP SIZE OF OLT CONTROLLER

OPERATING SYSTEM	32-BIT	64-BIT
Windows	Do not modify. Heap size defaults to 1G (-Xmx1g) With 4GT enabled, set heap size to 2G (-Xmx2g) [2]	Set maximum heap size to 3G (-Xmx3g) [3]
Linux (OEL, RHEL)	Modify heap size to 2G (-Xmx2g)	Heap size can be set to a large value with ATS build for Linux 64-bit.

Steps to change heap size in Windows

1. Stop Oracle ATS Server service using services.msc.
2. Navigate to HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\OracleATSServer \Parameters
3. Locate the key named CmdLine.

[2] When 4GT is enabled, a process can use up to 3GB of virtual address space. http://msdn.microsoft.com/en-us/library/windows/desktop/bb613473%28v=vs.85%29.aspx has instructions to enable 4GT on 32-bit Windows.

[3] ATS uses 32-bit JVM on Windows (32/64 bit). 64-bit Windows allows 32-bit process to address the entire 4GB of the virtual address space.

4. Modify the maximum value for the heap from -Xmx1024m to the desired value.

   E.g. To use 2GB Heap, modify the value to -Xmx2g

Steps to change heap size in Linux (OEL, RHEL)

1. Stop ATS Server daemon (/sbin/service OracleATSServer stop)
2. Edit /etc/init.d/OracleATSServer
3. Find the line: "export USER_MEM_ARGS="-Xms256m -Xmx1024m"
4. Modify the maximum value for the heap from -Xmx1024m to desired value.

   E.g. To use 2GB Heap, modify the value to -Xmx2g

Modifying Heap Size for OLT Agent Process

Heap size for OLT agent process is configured in '*Edit Scenario Details*' dialog. After adding the script in OLT, click '*Configure All Parameters*' next to the script to configure heap size. The default heap size for the agent process is 1G.

TABLE 3. HEAP SIZE OF OLT AGENT PROCESS

OPERATING SYSTEM	32-BIT	64-BIT
Windows	Use *Default* for '*Maximum JVM Heap size*'. Use '*Maximum Users Per Process*' to distribute load across multiple agent processes.	Same as for 32-bit since ATS does not use 64-bit JVM on Windows.
Linux	Use *Default* for '*Maximum JVM Heap size*'. Use '*Maximum Users Per Process*' to distribute load across multiple agent processes.	Heap size can be set to a large value. But it is not recommended to set the heap size to more than 4G. [4] Use '*Maximum Users per Process*' to distribute load across multiple agent processes, if necessary.

[4] JVM may have to be tuned to efficiently use large heap sizes. Running multiple load agent processes with small heap size helps users focus on the load test rather than spending time tuning the agent process.

With the introduction of 'Hardware Estimation' feature in OLT, estimating the system resources/machines needed to carry out a load test becomes a trivial task. This feature also helps in figuring out the optimal heap size for a set of Vusers instead of relying on trial and error or a monitoring exercise to determine the value.

Distribution of Databank Records during a Load Test

Scenario configuration is a required setup in order to achieve accurate load test results. Among the various parameters available in the scenario details, the distribution of databank records should be configured for each script before a load test starts. For performance reasons, the agent process reads records from the databank in blocks of 20. When Vusers simulated by the agent process need more records from the databank, agent process requests for a set of 20 more records. This behavior has a harmless consequence which is obvious only when multiple agent processes, running locally or remotely, access the same databank file. To understand this, consider a load test with the following attributes.

- Script reads a record from the databank once per execution.
- Script takes at least 15 minutes per iteration.
- Each Vuser runs only for 1 iteration.
- Total load of 30 Vusers with a ramp up rate of 5 Vusers every 30 seconds.
- '*Maximum Users per Process*' is set to 10.
- Databank has 100 records.

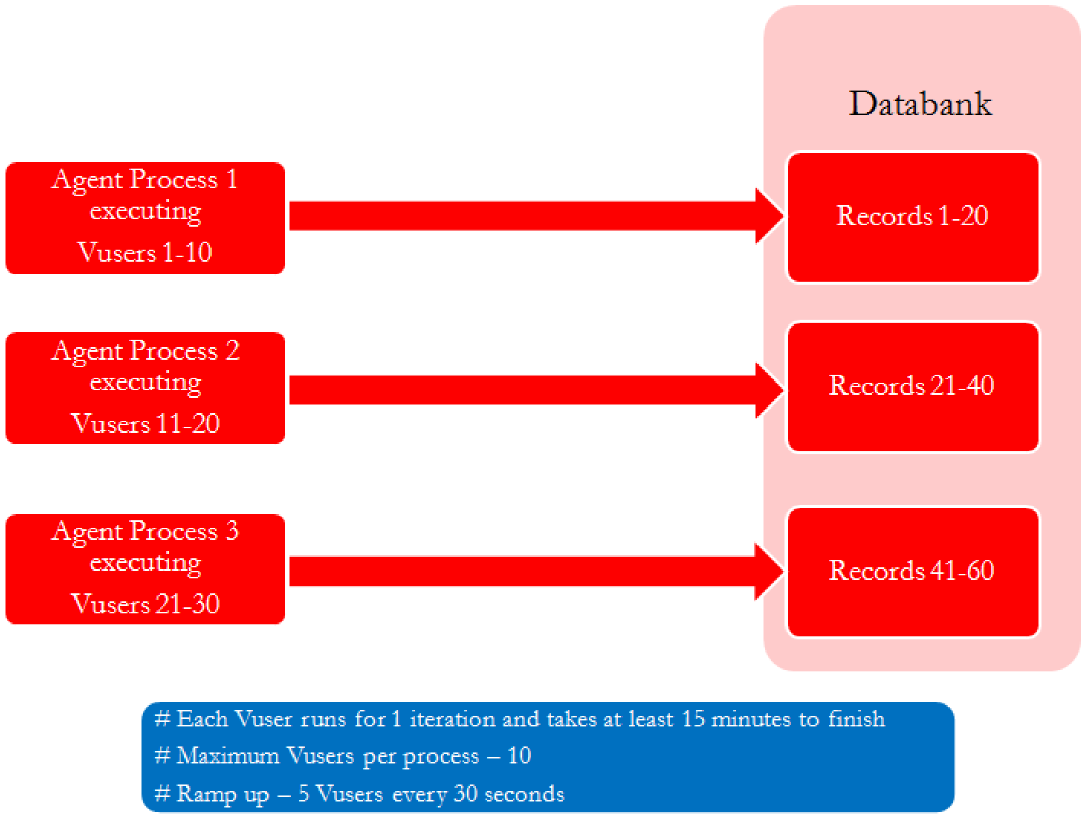

Databank
Agent Process 1 executing Vusers 1-10
Records 1-20
Agent Process 2 executing Vusers 11-20
Records 21-40
Agent Process 3 executing Vusers 21-30
Records 41-60
# Each Vuser runs for 1 iteration and takes at least 15 minutes to finish
# Maximum Vusers per process – 10
# Ramp up – 5 Vusers every 30 seconds

Figure 9. Databank cache and OLT

Essentially, there will be 3 load agent processes each executing 10 Vusers. It is important to keep in mind that a new agent process gets spawned only when the active Vuser count in all running agent processes reach the target value of '*Maximum Users Per Process'* (i.e. 10 in this use case). Since all 30 Vusers access the same databank, when agent-process-1 starts to execute the first set of 10 Vusers, it has a cache of the first set of 20 records from the databank. When the 11th Vuser is about to start, agent-process-2 gets spawned and caches the second set of 20 records from the databank. Similarly, agent-process-3 reads the third set of 20 records from the databank.

Effectively, each agent process uses only 10 records. Vusers 1-10 use record set 1-10, Vusers 11-20 use record set 21-30, and Vusers 21-30 use record set 41-50. Because of the caching mechanism, Vusers split across multiple processes and accessing the same databank may not use records sequentially.

The '*Maximum Users Per Process*' should be set to 20 if Vusers need to use records sequentially from the databank. Using this configuration there will be only 2 agent processes: agent-process-1 executing Vusers 1-20 using databank record set 1-20, and agent-process-2 executing Vusers 21-30 using record set 21-40 (with records from 21-30 range being utilized).

Conclusion

Setting up OLT load testing infrastructure is not a complex task. However, with so many subsystems of OLT at play and their interaction with equally complex network and IT infrastructure, end users may encounter challenges. Having a good understanding of OLT configurations and how the OLT subsystems interact with one another will allow the user to overcome challenges related to environment setup. Similarly, the ease of creating robust load test scripts is important for the success of a realistic load test. As discussed above, OpenScript way of creating robust load test scripts is both easy and intuitive.

Oracle Application Testing Suite is the recommended solution for organizations looking to improve the performance and identify the bottlenecks in their implementation of packaged Oracle applications, Web Services and web applications.

ORACLE

Oracle Load Testing Deep Dive
June 2014
Author: Raja Vengala
Contributing Authors: Karilyn Loui, Yutaka Takatsu

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com