AN OVERVIEW OF LOAD TEST TOOLS

Authors:
BURET Julien
DROZE Nicolas

Date:
Initial release February 27th 2003
Latest corrections May 21st 2003

SUMMARY

Introduction

This document has been designed in order to have an overview of existing software and applications that deal with load testing in all aspects. It has been written in the perspective of creating a new application in this domain, either from scratch or from existing parts. The role of this document is to highlight what it is possible to do, how are designed the applications, which are the parts we may benefit from, and which are the parts we may drop.

After a global search of existing software (but not exhaustive because of the necessary trade-off between the huge number of platforms and the available time for this preliminary study), we restricted this list to some products that appear interesting to us. We selected both open source and commercial software, since we can take good ideas from commercial software, and reuse parts of code from open source applications.

This document analyses these interesting applications. It identifies the specificity of each application, what are the possibilities and finally points advantages and drawbacks.

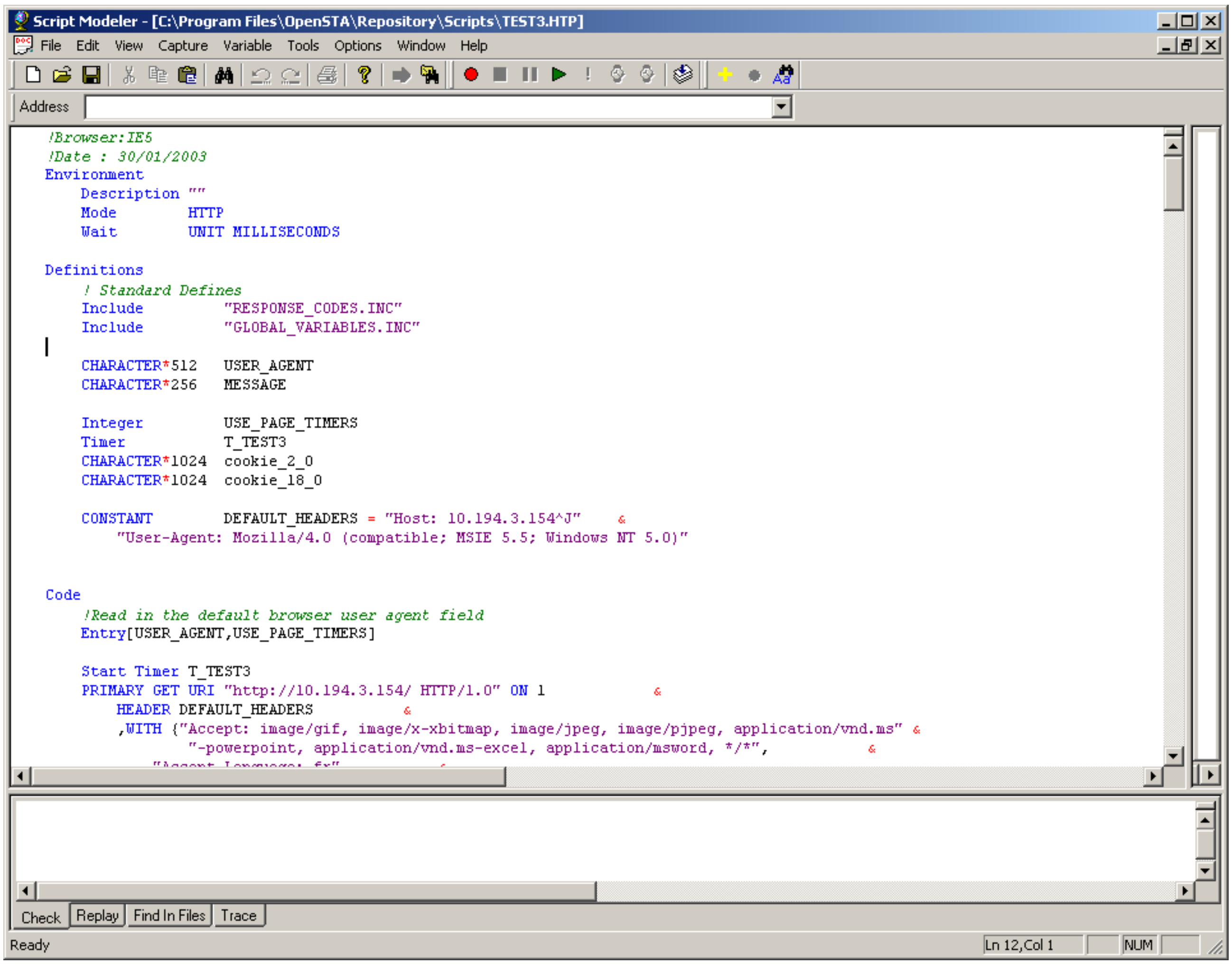
OpenSTA

Abstract

Open STA is an open source software developed in C++, and released under the GPL licence. It is an HTTP load test application with scenario creation and monitoring, which can be distributed on different computers. The version evaluated below is the v1.4.1 from July 2002. More information can be found on the web site home page: http://opensta.org/

Scenario

OpenSTA provides a script language which permits to simulate the activity of a user (sequence of accessed files, form filling, delay between each user interaction (think time), …). This language can describe HTTP/S scenario, with OpenSTA you can only build load tests for web applications.
You can write scripts with an editor provided with the application or you can use a module which records the request and response of a browser and generate a script automatically. To simulate a great number of users, this script is executed by each virtual user at the same time.


Script Modeler - [C:\Program Files\OpenSTA\Repository\Scripts\TEST3.HTP]
File Edit View Capture Variable Tools Options Window Help
Address
Check Replay Find In Files Trace
Ready
Ln 12,Col 1
NUM

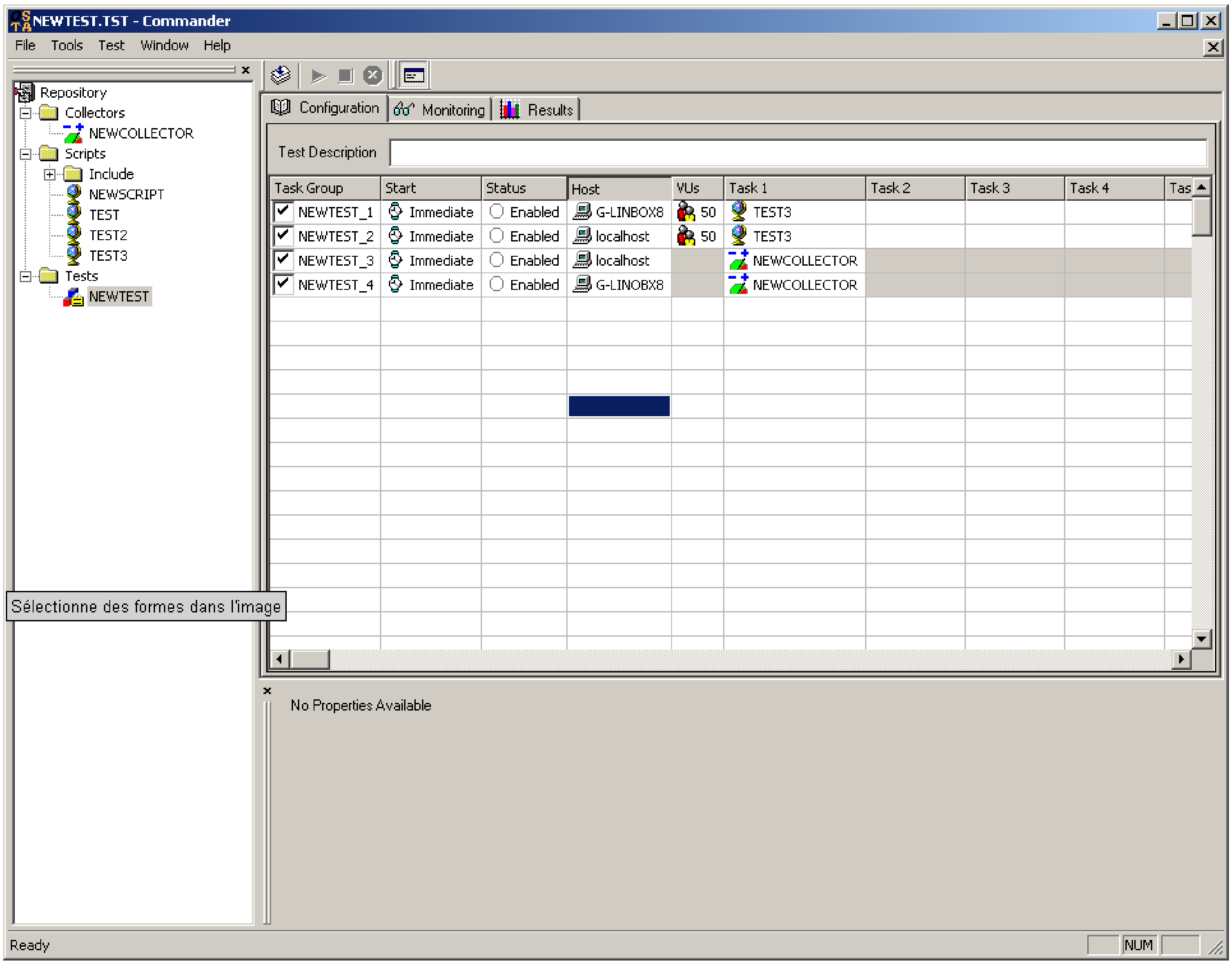
```
!Browser:IE5
!Date : 30/01/2003
Environment
    Description ""
    Mode        HTTP
    Wait        UNIT MILLISECONDS

Definitions
    ! Standard Defines
    Include         "RESPONSE_CODES.INC"
    Include         "GLOBAL_VARIABLES.INC"

    CHARACTER*512   USER_AGENT
    CHARACTER*256   MESSAGE

    Integer         USE_PAGE_TIMERS
    Timer           T_TEST3
    CHARACTER*1024  cookie_2_0
    CHARACTER*1024  cookie_18_0

    CONSTANT        DEFAULT_HEADERS = "Host: 10.194.3.154^J"    &
        "User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)"


Code
    !Read in the default browser user agent field
    Entry[USER_AGENT,USE_PAGE_TIMERS]

    Start Timer T_TEST3
    PRIMARY GET URI "http://10.194.3.154/ HTTP/1.0" ON 1            &
        HEADER DEFAULT_HEADERS          &
        ,WITH {"Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms" &
                "-powerpoint, application/vnd.ms-excel, application/msword, */*",          &
```

Script editor

You cannot randomise the think time[1], the form filling or the user action like choosing a link in an HTML response.

Execution

All the test configuration is managed in the graphical interface. The composition of the test is very simple (drag & drop of elements…). You choose scripts for a test, a remote computer that will execute each test, the number of users, different monitoring possibilities and you can launch the test.
During the test execution, you can monitor a great number of parameters like CPU idle time, network traffic, number of processes, disk usage, error in HTTP response… For each user configured in the test, OpenSTA creates one thread which executes the script.



Task Group	Start	Status	Host	VUs	Task 1	Task 2	Task 3	Task 4	Tas
✓ NEWTEST_1	Immediate	Enabled	G-LINBOX8	50	TEST3				
✓ NEWTEST_2	Immediate	Enabled	localhost	50	TEST3				
✓ NEWTEST_3	Immediate	Enabled	localhost		NEWCOLLECTOR				
✓ NEWTEST_4	Immediate	Enabled	G-LINOBX8		NEWCOLLECTOR				

Composition of test

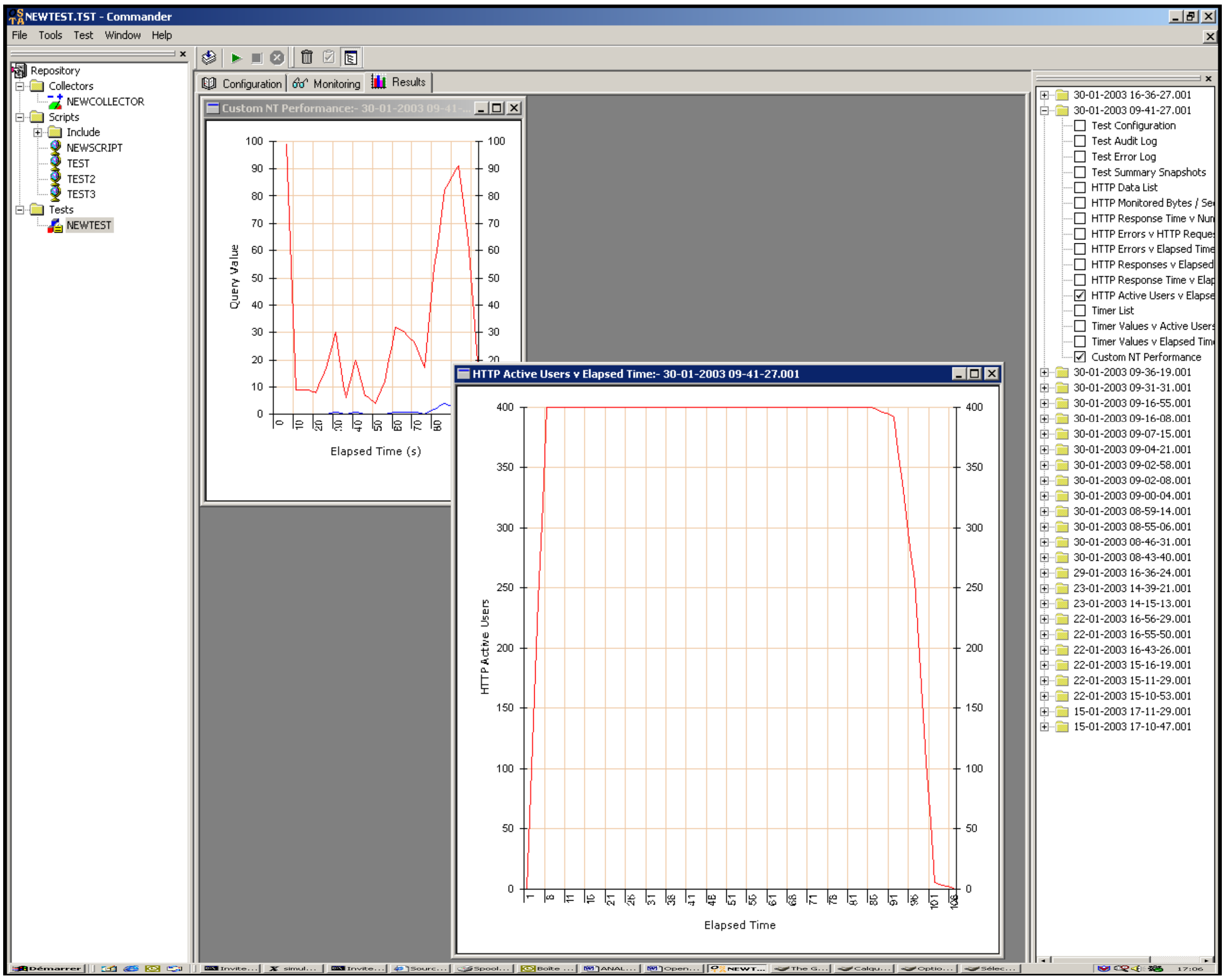
The test can be distributed on several computers. OpenSTA provides a name server for registering the different computers. During the building of the test we choose on which computer each scenario will be launched.

[1]The "think time" is a delay representing the time spent by the user thinking between two consecutive clicks.

All the distribution is based on CORBA. It uses the omniORB Object Request Broker (ORB) and Naming Service from AT&T Laboratories Cambridge.

Results

A great number of data can be caught during the test. First, all information of system monitoring (WinNT performance monitor or with SNMP). We can also catch information on the test, like number of requests, error, throughput of client…). All information can be reviewed with customizable graphics. All information can be exported in a file in order to be used with other software like Excel.



Result and analysis

Monitoring

You can make a lot of queries to monitor the computer. Two systems can be used for this: Windows performance monitor or SNMP server. You can only use the first one if the computer uses Windows NT and if OpenSTA is launched (with registration in the name server), it is not really a problem for all the clients because OpenSTA is designed only for

Windows. The second solution needs a SNMP server in the network and SNMP agent on each computer monitored.

Conclusion

Positive part of this software are the following:

-A user-friendly graphical interface.
-The script capture from the browser.
-The monitoring functionality
-You can make very complete test with the script language

Negative part of this software are the following:

-Only designed for Windows
-Only for HTTP

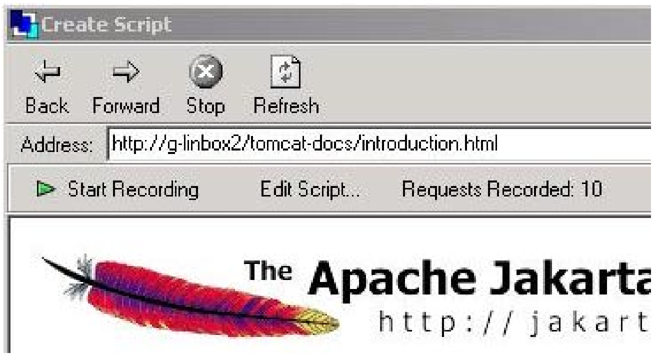
DieselTest

Abstract

DieselTest is a software designed in Delphi 5, for systems under NT environment. It is distributed under the GNU LGPL license. It is a load injector to test Internet web sites (HTTP and HTTPS requests), with monitoring and graphical representations.
http://www.dieseltest.com

Scenario

To carry out a scenario of tests, there is no specific script language or technical terms to know. The scenarios are recorded automatically in a very simple way, using a capture tool (graphical interface). This interface is a simplified representation of an Internet browser, in which you can enter a starting URL, visualize the page, launch the recording of the scenario and stop it. The scenario is recorded according to the user interactions in the integrated browser, by recording the links visited, the content and the moment it is visited.

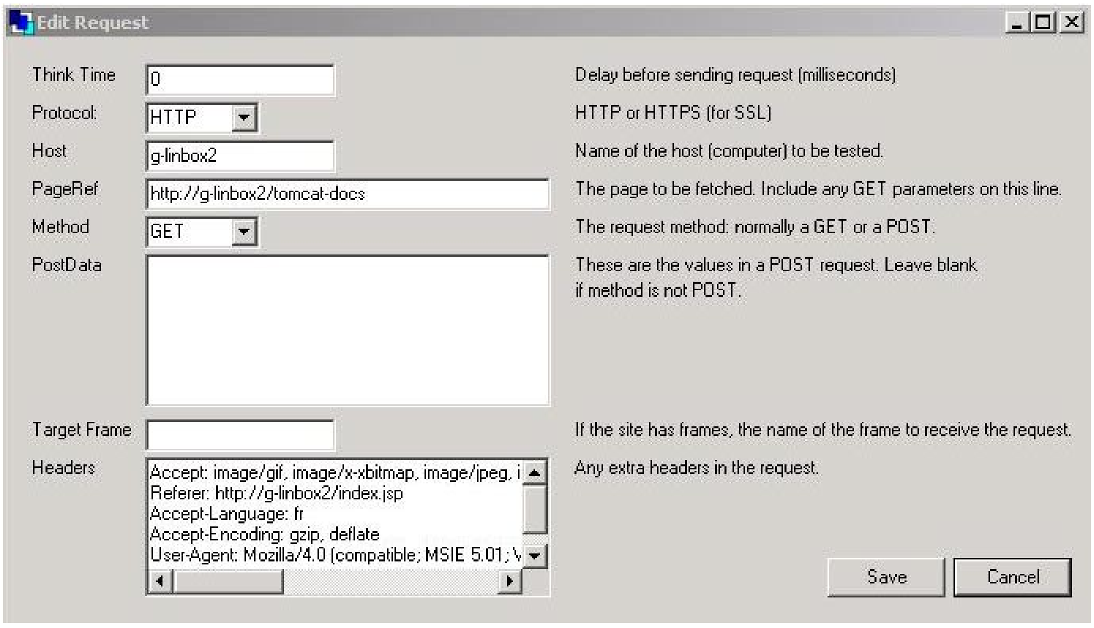


Address: http://g-linbox2/tomcat-docs/introduction.html

Requests Recorded: 10

Capture tool

There is the possibility to edit the recorded scenarios, by customizing the requests, for instance modifying the headers, the posted data, the link to visit, the parameters... This possibility can become tedious in the case of multiple modifications, because it is only possible to edit the scenario request by request.

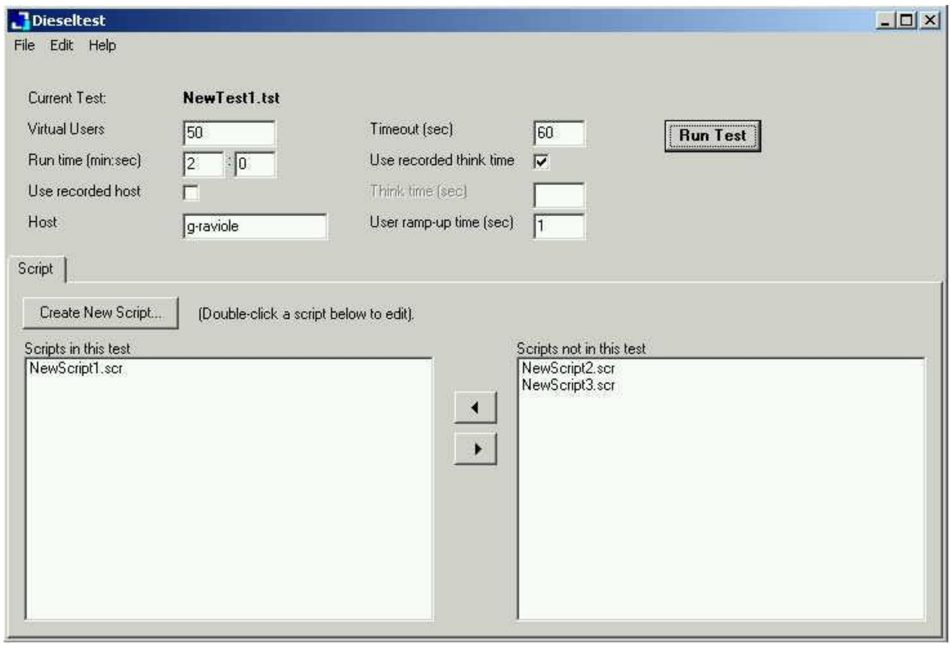

Edit Request

Think Time	0	Delay before sending request (milliseconds)
Protocol:	HTTP	HTTP or HTTPS (for SSL)
Host	g-linbox2	Name of the host (computer) to be tested.
PageRef	http://g-linbox2/tomcat-docs	The page to be fetched. Include any GET parameters on this line.
Method	GET	The request method: normally a GET or a POST.
PostData		These are the values in a POST request. Leave blank if method is not POST.
Target Frame		If the site has frames, the name of the frame to receive the request.
Headers	Accept: image/gif, image/x-xbitmap, image/jpeg, i Referer: http://g-linbox2/index.jsp Accept-Language: fr Accept-Encoding: gzip, deflate User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; \	Any extra headers in the request.

Request edition interface

The number of users to be simulated is chosen before starting the test, with a time delay before each new user creation.
The created scenario will be executed by each simulated user, until the end of assigned run time. The process can be terminated before the end of the creation of all the users or the requests if the assigned run time is too short, otherwise the scenario is played again through the end of the process.

A scenario can be saved in order to be integrated into others, to form a test plan and to define a sequencing, as in the following screen.


Dieseltest

Current Test:	NewTest1.tst		
Virtual Users	50	Timeout (sec)	60
Run time (min:sec)	2 : 0	Use recorded think time	✓
Use recorded host	☐	Think time (sec)	
Host	g-raviole	User ramp-up time (sec)	1

(Double-click a script below to edit).

Scripts in this test	Scripts not in this test
NewScript1.scr	NewScript2.scr
	NewScript3.scr

Sequencing of a test plan

Execution

The execution of the test plan is characterized by the creation of one thread per user, which will execute the recorded scenario request by request until the end of the assigned run time.

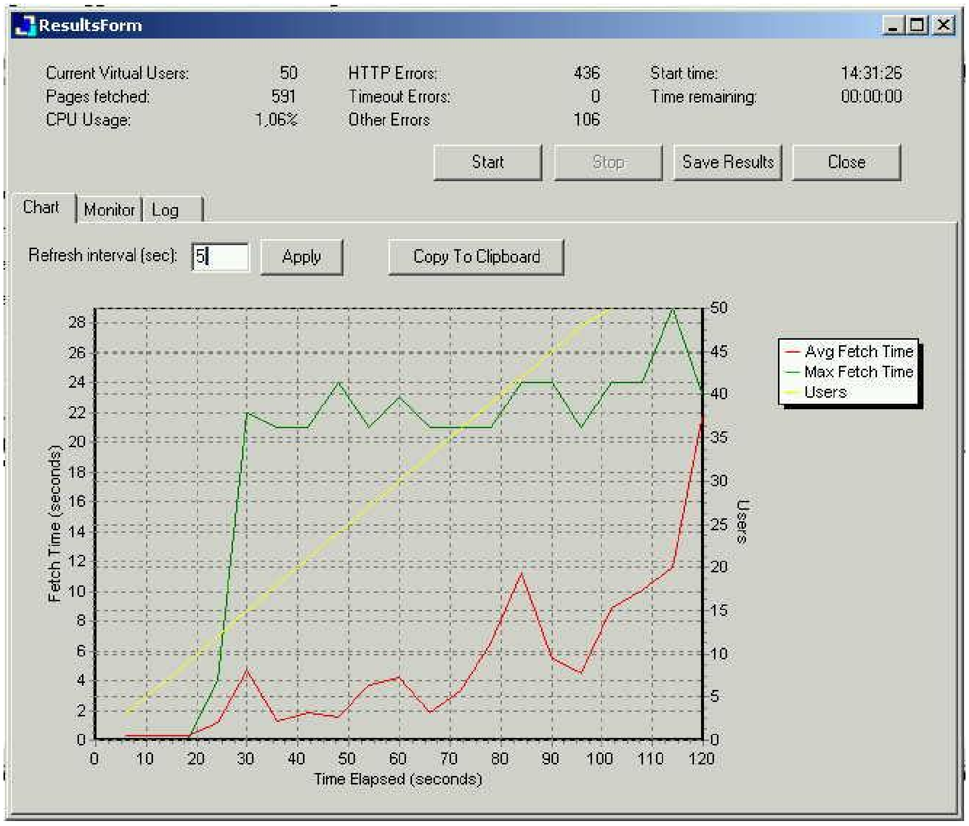
Results

During the running tests, there is the possibility to visualize two kinds of outputs: a chart display and a logging functionality.
During the execution of the tests, the graph is refreshed with regular time intervals (time in seconds, chosen by the user). There is also the possibility to have a written trace of the results and the events, with the logging functionality, updated in real-time.

The execution of the test plan allows to visualize the main following characteristics:

- Number of created users: This number can be less than the initial number desired, if the program did not have the time to create all the users in the assigned run time.
- Number of errors: Represents the number of responses with a response code different from 200.
- Number of pages fetched (errors included)
- Timeout: Number of timeout during the test.

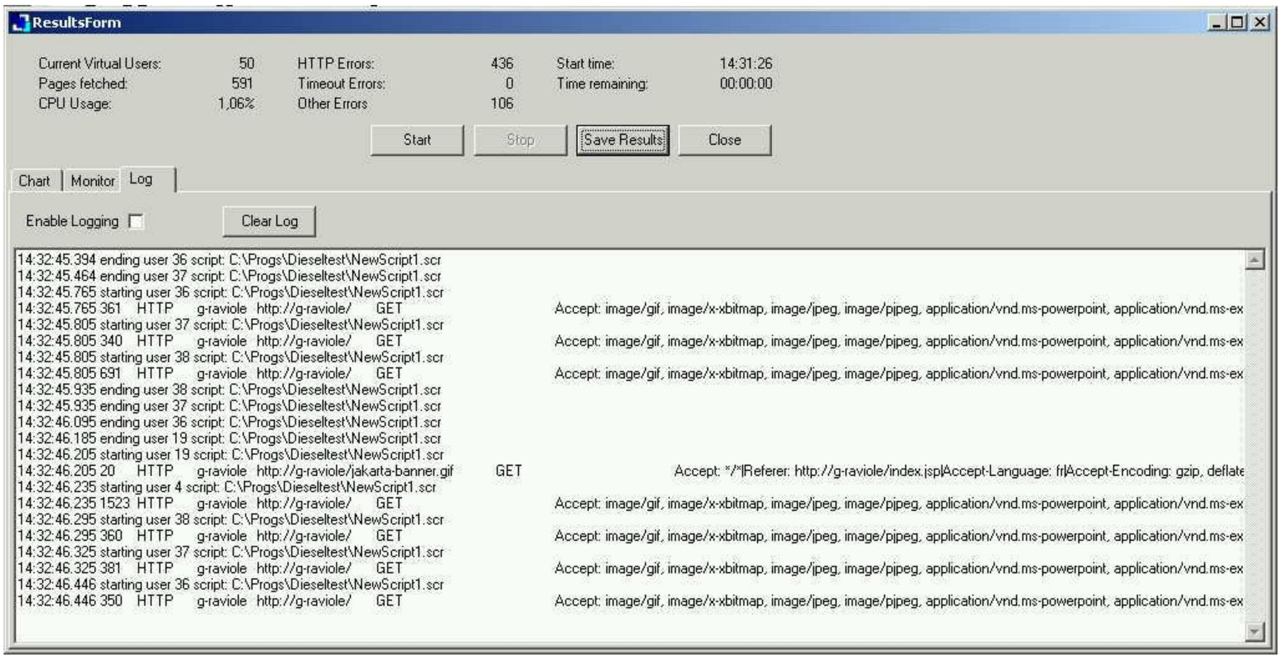
On the chart we can visualize the number of created users, the average fetch time, the maximum fetch time.



Current Virtual Users: 50
Pages fetched: 591
CPU Usage: 1.06%
HTTP Errors: 436
Timeout Errors: 0
Other Errors: 106
Start time: 14:31:26
Time remaining: 00:00:00
Refresh interval (sec): 5

Monitoring

The logging functionality details each response received: response code, content, time, link,…



```
Current Virtual Users:   50      HTTP Errors:      436    Start time:        14:31:26
Pages fetched:           591     Timeout Errors:   0      Time remaining:    00:00:00
CPU Usage:               1,06%   Other Errors      106

Enable Logging ☐
```

```
14:32:45.394 ending user 36 script: C:\Progs\Dieseltest\NewScript1.scr
14:32:45.464 ending user 37 script: C:\Progs\Dieseltest\NewScript1.scr
14:32:45.765 starting user 36 script: C:\Progs\Dieseltest\NewScript1.scr
14:32:45.765 361   HTTP    g-raviole   http://g-raviole/     GET              Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-ex
14:32:45.805 starting user 37 script: C:\Progs\Dieseltest\NewScript1.scr
14:32:45.805 340   HTTP    g-raviole   http://g-raviole/     GET              Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-ex
14:32:45.805 starting user 38 script: C:\Progs\Dieseltest\NewScript1.scr
14:32:45.805 691   HTTP    g-raviole   http://g-raviole/     GET              Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-ex
14:32:45.935 ending user 38 script: C:\Progs\Dieseltest\NewScript1.scr
14:32:45.935 ending user 37 script: C:\Progs\Dieseltest\NewScript1.scr
14:32:46.095 ending user 36 script: C:\Progs\Dieseltest\NewScript1.scr
14:32:46.185 ending user 19 script: C:\Progs\Dieseltest\NewScript1.scr
14:32:46.205 starting user 19 script: C:\Progs\Dieseltest\NewScript1.scr
14:32:46.205 20    HTTP    g-raviole   http://g-raviole/jakarta-banner.gif     GET        Accept: */*|Referer: http://g-raviole/index.jsp|Accept-Language: fr|Accept-Encoding: gzip, deflate
14:32:46.235 starting user 4 script: C:\Progs\Dieseltest\NewScript1.scr
14:32:46.235 1523  HTTP    g-raviole   http://g-raviole/     GET              Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-ex
14:32:46.295 starting user 38 script: C:\Progs\Dieseltest\NewScript1.scr
14:32:46.295 360   HTTP    g-raviole   http://g-raviole/     GET              Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-ex
14:32:46.325 starting user 37 script: C:\Progs\Dieseltest\NewScript1.scr
14:32:46.325 381   HTTP    g-raviole   http://g-raviole/     GET              Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-ex
14:32:46.446 starting user 36 script: C:\Progs\Dieseltest\NewScript1.scr
14:32:46.446 350   HTTP    g-raviole   http://g-raviole/     GET              Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-ex
```

Logging

It is possible to export results into a text file containing the number of the thread, the number of its executed request, starting time of the thread, ending time of the thread and HTTP return codes.

usernum	requestnum	fetchstart	fetchend	result
0	1	52288655	52288996	302
0	2	52289697	52290027	302
0	3	52290388	52290708	302
1	1	52290658	52290989	302
0	4	52291029	52291359	302
1	2	52291690	52292020	302
0	5	52291700	52292030	302
1	3	52292381	52292711	302
0	6	52292421	52292751	302
2	1	52292661	52293002	302
…				

Exporting the results

Monitoring

The most interesting results of this software are the average fetch time, the maximum fetch time, the CPU usage percentage of the computer launching the test.

Conclusion

The features of this software, although traditional, are presented in an easy way for the user. The positive parts are the following:

- The quality of the chart
- The representation of the users created on the chart for a given time
- The capture tool
- Simple and fast to use
- Exporting the results
- The logging functionality

The negative parts are the following:

- The manual edition of the tests is badly designed
- Some results seem incoherent, particularly fetch times (see chart)
- The ambiguity of certain results
- Distributed tests are impossible
- No feedback from the target system
- Specific technology environment (Delphi, NT)

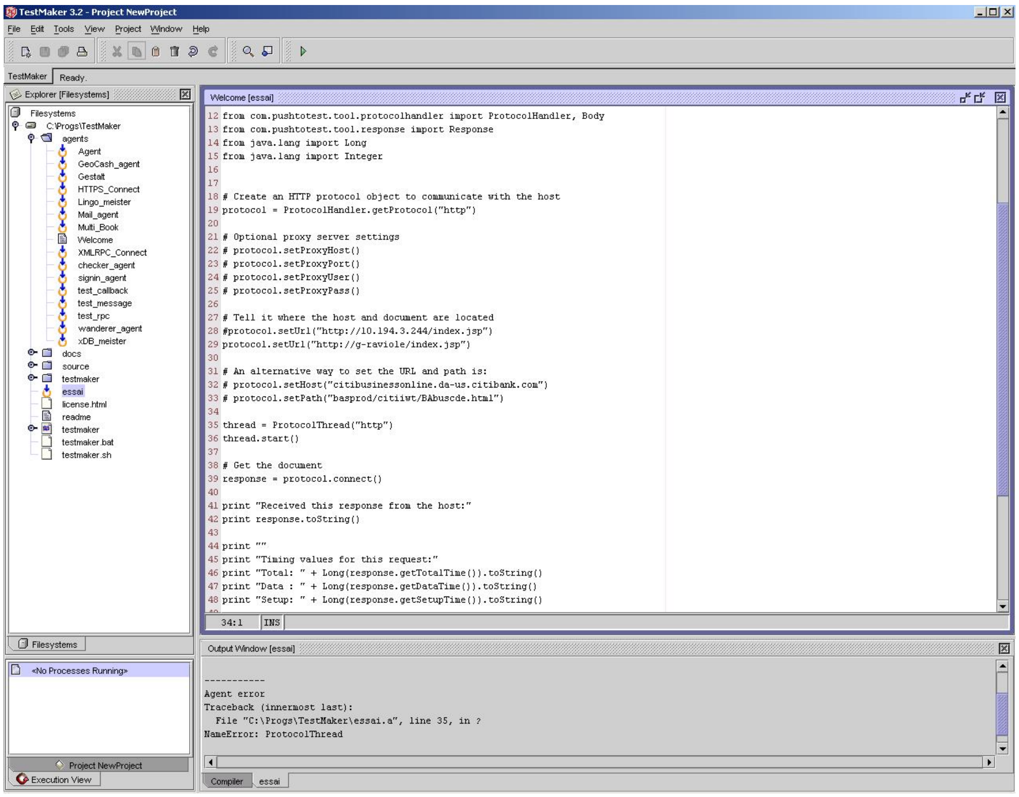
TestMaker

Abstract

TestMaker is a framework and utility that builds intelligent test agents which implement users interactions in several environments. This program is developed in Java (so it is multi platform) and it is available as open source.
The main systems tested are the following: HTTP, HTTPS, SOAP, XML-RPC, Mails (SMTP, POP3 and IMAP)
This version does not support remote clients for distributed testing, but the software TestNetwork sold by the same company is able to perform remote clients. But of course, it is not open source. All we know is that it includes parts of TestMaker source code.
This quick overview does not analyze the whole possibilities.
It seems that the developers are actively working on this software since the version we studied here was released on January 1st 2003 (version 3.2).
We also downloaded version 3.3 released at the end of January.
http://www.pushtotest.com/ptt

Scenario

The software is different from the others in the way they can build scenarios.
To build a test plan it is necessary to program everything in a specific language: Jython, an hybrid language between Java and Python.
In this software, the term agent is used to describe a file which contains a Jython script.
The language Jython permits to use Java usual methods, and simplify some other aspects such as statements, class definition, and other manipulations.
Moreover some specific classes are provided by this software. Utility classes can handle connection protocols, get connection response such as time, content, ...
The Jython web site can be useful for creating some agents: http://www.jython.org
This link is a JavaDoc style documentation of Testmaker's specific libraries: http://docs.pushtotest.com/tooldocs/



```
12 from com.pushtotest.tool.protocolhandler import ProtocolHandler, Body
13 from com.pushtotest.tool.response import Response
14 from java.lang import Long
15 from java.lang import Integer
16
17
18 # Create an HTTP protocol object to communicate with the host
19 protocol = ProtocolHandler.getProtocol("http")
20
21 # Optional proxy server settings
22 # protocol.setProxyHost()
23 # protocol.setProxyPort()
24 # protocol.setProxyUser()
25 # protocol.setProxyPass()
26
27 # Tell it where the host and document are located
28 #protocol.setUrl("http://10.194.3.244/index.jsp")
29 protocol.setUrl("http://g-raviole/index.jsp")
30
31 # An alternative way to set the URL and path is:
32 # protocol.setHost("citibusinessonline.da-us.citibank.com")
33 # protocol.setPath("basprod/citiiwt/BAbuscde.html")
34
35 thread = ProtocolThread("http")
36 thread.start()
37
38 # Get the document
39 response = protocol.connect()
40
41 print "Received this response from the host:"
42 print response.toString()
43
44 print ""
45 print "Timing values for this request:"
46 print "Total: " + Long(response.getTotalTime()).toString()
47 print "Data : " + Long(response.getDataTime()).toString()
48 print "Setup: " + Long(response.getSetupTime()).toString()
```

```
-----------
Agent error
Traceback (innermost last):
  File "C:\Progs\TestMaker\essai.a", line 35, in ?
NameError: ProtocolThread
```

Editing a scenario

Execution

It all depends on how the test plan is designed by the user. It is possible to program an agent with threads for a parallel execution, or sequentially, as well as continuously loop, create a thread by user, ...
Basically the software generates Java objects (classes) such as threads, protocols, responses, logging utility, ...

Results

Here again, everything depends on the programmer's choices. In the software it is possible to trace some operations in a specific window using a Jython command. Otherwise the programmer can handle his own results representation in a Java frame with graphics, using the appropriates Java libraries.

Monitoring

Not many characteristics are extracted from the tested system. The only available results are related to the evaluation of the response. It is possible to get the following parameters:

- The full contents of the response that was sent back from the server.
- The time it took from making the request to the host to when all the response data is saved to the response object.

- The response code value received from the host.
- The amount of time it took to set-up the request without including the time to make the request or receive the response.
- The total time it took to set-up the request, communicate with the host, and save the response data to a response object.

Conclusion

The positive parts of this software are the following:

- The possibility to build any kind of test agent.
- The power of Java programming with some Python simplifications.
- Multi-environment
- Open source

The negative parts of this software are the following:

- To build a test plan it is necessary to get familiar with the Jython scripting language, Java language and to write it from scratch.
- The monitoring tools are very basic, since it is limited to the response analysis.
- The complexity to quickly build some analysis results.
- Must pay for distributed testing

Grinder

Abstract

Grinder is a generic framework for load testing any kind of target systems, with scenario in Jython. It is developed in Java and released under BSD licence. The version evaluated below is v3 from December 2002
More information can be found on the home page: http://grinder.sourceforge.net/

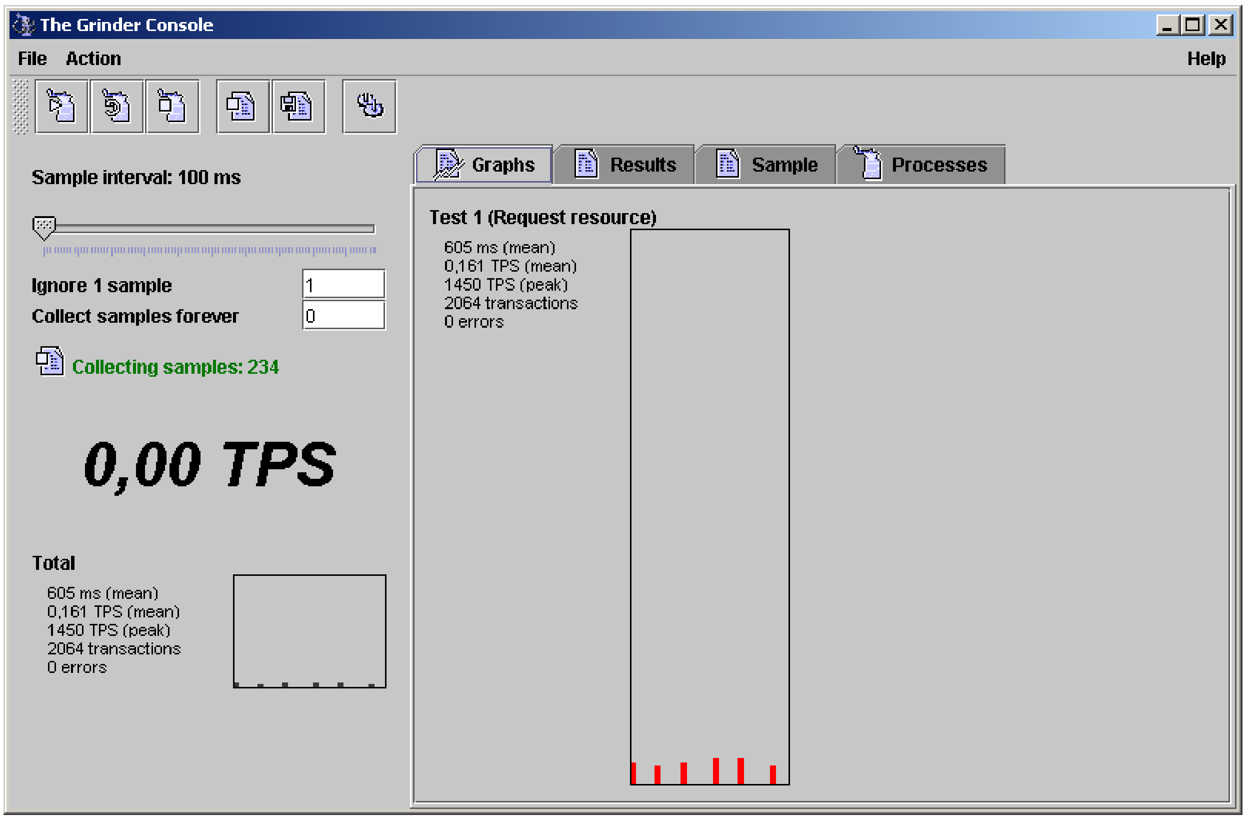
Scenario

The scenario test are written in Jython (Jython is a Python interpreter allowing manipulation of Java objects). We can create very complete scripts, because all Java classes can be used in scripts. For example you can create a scenario which accesses to an EJB server. (Example code with WebLogic).

```
from java.lang import String
from java.util import Properties,Random,HashMap
from javax.naming import Context,InitialContext
from net.grinder.script import Test
from weblogic.jndi import WLInitialContextFactory

log = grinder.logger.output

tests = {
    "home" : Test(1, "TraderHome"),
    "trade" : Test(2, "Trader buy/sell"),
    "query" : Test(3, "Trader getBalance"),
    }

# Initial context lookup for EJB home.
p = Properties()
p[Context.INITIAL_CONTEXT_FACTORY] = WLInitialContextFactory.name

home = InitialContext(p).lookup("ejb20-statefulSession-TraderHome")
homeTest = tests["home"].wrap(home)

random = Random()

class TestRunner:
    def __call__(self):
        trader = homeTest.create()

        tradeTest = tests["trade"].wrap(trader)

        stocksToSell = { "BEAS" : 100, "MSFT" : 999 }
        for stock, amount in stocksToSell.items():
            tradeResult = tradeTest.sell("John", stock, amount)
            log("Result of tradeTest.sell(): %s" % tradeResult)

        grinder.sleep(100)              # Idle a while

        stocksToBuy = { "BEAS" : abs(random.nextInt()) % 1000 }
        for stock, amount in stocksToBuy.items():
            tradeResult = tradeTest.buy("Phil", stock, amount)
            log("Result of tradeTest.buy(): %s" % tradeResult)

        queryTest = tests["query"].wrap(trader)
        balance = queryTest.getBalance();
        log("Balance is $%.2f" % balance)

        trader.remove()                 # We don't record the remove() as a test

        # Can obtain information about the thread context...
        if grinder.threadID == 0 and grinder.runNumber == 0:
            # ...and navigate from the proxy back to the test
            d = queryTest.__test__
            log("Query test is test %d, (%s)" % (d.number, d.description))
```

Execution

On each agent, you must copy a property file that contains all information for the test like console address, number of thread, etc and start the Grinder agent. On one host you must start the Grinder console. With this console you can control other agents. When you start the load test, the console makes a broadcast to the agents on the network and all the agents start the test.



Sample interval: 100 ms

Ignore 1 sample: 1
Collect samples forever: 0

Collecting samples: 234

0,00 TPS

Total
```
605 ms (mean)
0,161 TPS (mean)
1450 TPS (peak)
2064 transactions
0 errors
```

Test 1 (Request resource)
```
605 ms (mean)
0,161 TPS (mean)
1450 TPS (peak)
2064 transactions
0 errors
```

Results

Each agent creates a log file with the result of each transaction (HTTP code…), it can also save the result page in HTML. Another file contains time response of each transaction with the thread number, the number of run, the response time and if the transaction is successful.
The result file can be used with a spreadsheet like Excel in order to construct graphs or calculate other statistics.

Monitoring

Grinder can monitor only the transaction executed every second. Each agent sends information to the console during the load test.

Conclusion

Positive part of this software is the following:
- You can test everything with scripts in Jython

Negative parts of this software are the following:
- Deployment for distributed test.
- Poor results and graphical interface.

LoadSim

Abstract

LoadSim is an open source software developed in Java, which is designed for HTTP distributed load testing.
More information can be found on the home page: http://www.openware.org/loadsim/

Scenario

LoadSim uses Muffin (http://muffin.doit.org/) for creating scripts. It captures the requests made in a browser (You must change proxy configuration of your browser with a virtual gateway managed by Muffin). A filter provided with LoadSim allows Muffin to generate XML file with all the user requests on the web application.

```
<sequence>
<link id="admin"
host="http://10.194.3.154"
pathroot="/admin">
</link>
<link id=""
host="http://10.194.3.154"
redirect="static"
pathroot="/admin/">
</link>
<link id="index.jsp"
host="http://10.194.3.154"
redirect="static"
pathroot="/admin/index.jsp">
</link>
<link id="login.jsp;jsessionid=9EA8CF9806E69D71FAAD2ADE416C75FD"
host="http://10.194.3.154"
redirect="static"
pathroot="/admin/login.jsp;jsessionid=9EA8CF9806E69D71FAAD2ADE416C75FD">
</link>
<link id="j_security_check"
host="http://10.194.3.154"
pathroot="/admin/j_security_check">
<formdata>
<data>
<name>j_username</name>
<value><constant value="admin"/></value>
</data>
<data>
<name>j_password</name>
<value><constant value="admin"/></value>
</data>
</formdata>
</link>
<link id="index.jsp"
host="http://10.194.3.154"
redirect="static"
pathroot="/admin/index.jsp">
</link>
</sequence>
```

This scripting language can fill the forms. Data used for the form can be defined in the script, in another file (with a lot of choices for each field) or can be parsed from a result page. You can also define the number of virtual users and the time distribution of this user.

Execution

There is not any GUI for this application, all command are executed from a textual console. This console is only started on one client. All hosts (client) are defined in a configuration file. For each other host the configuration can be different.
Scenarios are replicated automatically on remote hosts.

```
<use-sequence filename="sequence.xml" hostname="//localhost:8000" num-iterations="1"
   num-virtual-users="50" />
  <use-sequence filename="sequence.xml" hostname="//10.194.3.244:8000" num-
   iterations="1" num-virtual-users="50" />
```

All hosts start LoadSim in server mode. The test is launched from the console.

Results

LoadSim generates results in a file (like CSV format), but provides no tool to analyze or draw the results. The format of result file can be customized from a configuration file.

Monitoring

No monitoring.

Conclusion

Generic load-test, no GUI, no results representation.
Positive parts of this software are the following:
    -Generation of script
    -Each client have a different configuration (user, script…)
Negative parts of this software are the following:
    -No graphical interface
    -Poor results
    -No graphical representation of result.
    -No monitoring

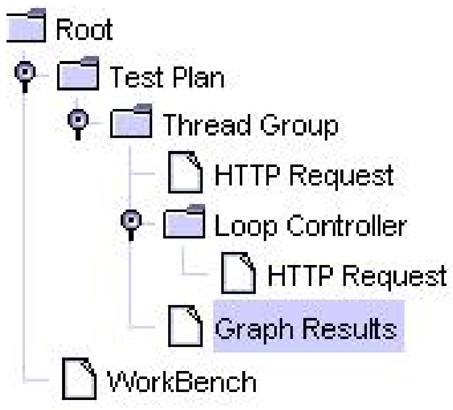
Jmeter

Abstract

Apache Jmeter is a 100% Java desktop application designed to load test functional behavior and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions (FTP, Java, SOAP/XML-RPC, JDBC).
More information are available on the web site: http://jakarta.apache.org/jmeter/index.html

Scenario

In order to build a scenario it is necessary to get familiar with Jmeter terminology. Then it is possible to create a scenario adding some built-in operations hierarchically.
A *thread group* represents a sequence of operations which will be executed for each virtual user. It is possible to customize parameters like the number of threads to execute, the ramp up period and the duration of the test.
Some samplers designate a kind of request (HTTP, FTP, ...) which has to be customized (filling a form with server address, port number, ...)
Some other elements can be added to the thread group such as charts representation, timers, logical controllers.
One of Jmeter's peculiarities is the possibility for the user to develop his/her own controllers, listeners and samplers in order to add them to a Jmeter test plan. The new sampler is added simply in Jmeter by adding the JAR file in a specific directory.



A simple test plan

Execution

A test run may be distributed on several computers which have an RMI registry running. However it is only possible to start remote computers one by one, and stop them one by one too. (at the time we are publishing this document, new developments seem to fix this problem)
One virtual user is represented by a thread which is executed according to the “thread group” parameters.

Results

Listener elements can be added to the “thread group” to visualize request responses. For instance it is possible to record results in a graph which will record the average time of the response, the throughput, ...
The graph representation is a little confusing since it sometime traces new values on old ones, if data do not fit in the original graph size.
Another representation is a table, which contains the response code, response time and if the operation succeeded or not.
It is also possible to export the results in a text file, containing basics information such as response code, thread name, page fetched, timestamp, ...
It is important to note that Jmeter has been designed to be extendible, even as far as data viewing and monitoring is concerned. So, custom collectors and visualizers may be developed in Java and added to Jmeter.

Monitoring

The only possible monitored data are the response time and throughput. The current number of virtual users simulated is not represented. A developer may add other monitoring tools (see section above).

Conclusion

Positive parts of this software are the following:

- -The distributed testing
- -Various target systems
- -Extensibility: Pluggable samplers allow unlimited testing capabilities

Negative parts of this software are the following:

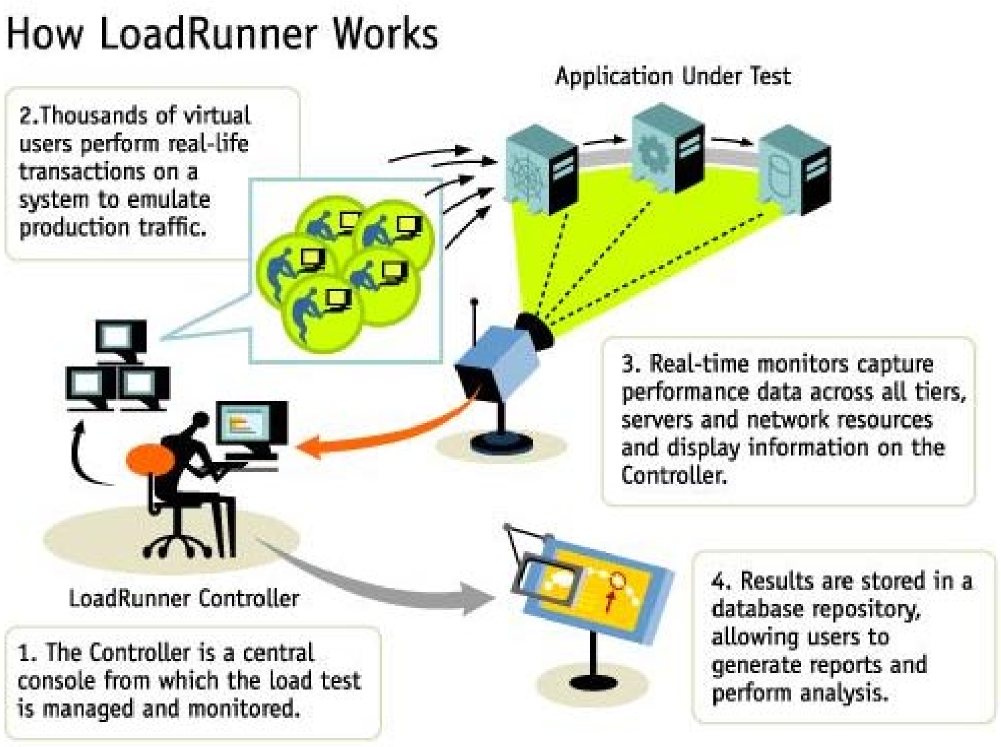
- -Chart representation quite confuse
- -Terminology not very clear
- -Necessary to start remote machine one by one
- -Remote machines must be declared in a property file before starting application

LoadRunner (LoadTest)

Abstract

Mercury Interactive's LoadRunner is a load testing tool that analyses system behavior and performance. It exercises the entire enterprise infrastructure by emulating thousands of users and employs performance monitors to identify and isolate problems.
LoadRunner is distributed by Mercury Interactive company under a commercial license. The product web site is available at **http://www-svca.mercuryinteractive.com/products/loadrunner**

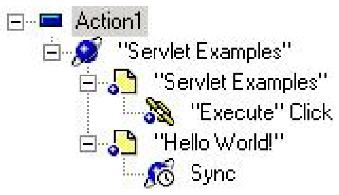
The target systems are various, and most of existing technologies are supported, like Web servers, Web application servers, streaming media servers, databases servers, Java, ERP, ...


How LoadRunner Works
Application Under Test
2.Thousands of virtual users perform real-life transactions on a system to emulate production traffic.
3. Real-time monitors capture performance data across all tiers, servers and network resources and display information on the Controller.
LoadRunner Controller
1. The Controller is a central console from which the load test is managed and monitored.
4. Results are stored in a database repository, allowing users to generate reports and perform analysis.

Another Mercury Interactive product is available for evaluation. AstraLoadTest is a part of LoadRunner, but it is designed only to check web servers scalability.
The AstraLoadTest software proposes three different modules specialized in creating a scenario, running a scenario and viewing the results.
The following of this chapter is about LoadTest.

Scenario

Building a scenario is quite simple with an automatic recording tool which records user interactions in the Internet browser. The result is an automatic creation of the scenario structured as a tree, with each action to execute. It is also possible to manually add interactions elements to build a hierarchical test plan.



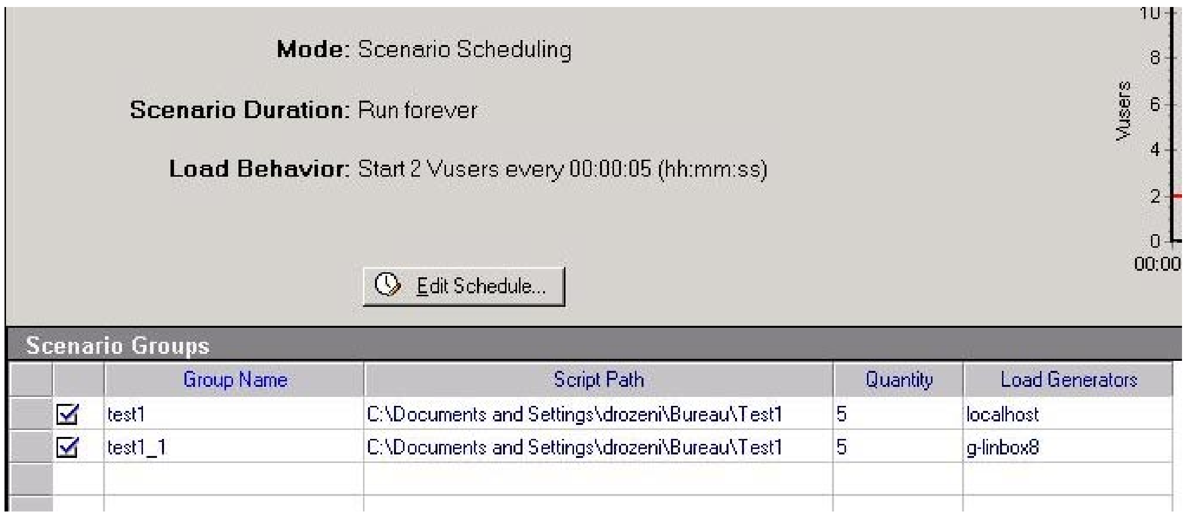
Recorded scenario

Basically, the results displayed for such a test represent response time for each action, but there is the possibility of creating and ending some transactions. This specificity has the effect of grouping the response results of the actions comprised between the transaction tags. It allows for instance to get the response time of a group of actions.
Another functionality is a “rendezvous point”. This functionality waits for all the virtual users simulated to arrive at this point (or to reach the time out period).

Execution

The scheduling of the test plan can take various aspects.
Here the term “group” represents the name of a computer from which the scenario will be executed.
There are two main types of scheduling, but both allow to customize a ramp up period, a duration period and a ramp down period.
The schedule by scenario or schedule by group are two different visions for running the test.
The schedule by scenario allows for instance to start a given number of virtual users every fixed time.
The schedule by group allows to delay a test on a remote host, by setting the start time after another host started.

The following snapshot represents a distributed test from two computers, which will emulate 5 users each. The scheduling is in scenario mode and two virtual users will be emulated every 5 seconds.


Mode: Scenario Scheduling

Scenario Duration: Run forever

Load Behavior: Start 2 Vusers every 00:00:05 (hh:mm:ss)

Scenario Groups

		Group Name	Script Path	Quantity	Load Generators
	☑	test1	C:\Documents and Settings\drozeni\Bureau\Test1	5	localhost
	☑	test1_1	C:\Documents and Settings\drozeni\Bureau\Test1	5	g-linbox8

Results

Once the test is finished, results are collected from all the distributed computers, and merged into one document.
This report summarizes some key results such as number of users emulated, total throughput, average throughput, total hits, average hits, response codes and transactions information.
These data are also available on charts.
A lot of graphics are available depending on the targeted system.

Monitoring

The monitoring provides a lot of system values such as the percentage of processor time, number of threads, memory usage but it is only available for NT or Linux systems.

Conclusion

LoadTest seems to be one of the most complete tool for monitoring results.
Positive parts of this software are the following:

-Monitoring capabilities
-Charts representations
-Scenario capture tool
-Variety of target systems

Negative part of this software is the following:

-Commercial license

Rubis

Abstract

Rubis is an auction site prototype (Ebay-like), that can be used to evaluate different application implementations and different application servers scalability. Rubis is provided with some load-test tool (designed for Rubis, but some parts of code could be re-used) and a monitoring system. It is an open source application developed in Java.
http://www.objectweb.org/rubis

Scenario

The benchmark tool simulates users, that browse the web site. For each customer session, the client emulator opens a persistent HTTP connection to the Web server and closes it at the end of the session. Each emulated client waits for a certain think time before initiating the next interaction. The next interaction is determined by a state transition matrix that specifies the probability to go from one interaction to another one. A think time can be randomly generated between two consecutive interactions, or retrieved from the transition matrix.

Execution

The execution profile is set up in a property file which defines all the parameters for the test, such as the names of the computers to use, the delay between each new client emulation, slow down factor, ramp up period,...
Then all threads are created before the test execution, but requests are executed according to a slow down factor defined in the settings.

Results

The test is started under Linux environment with an ssh connection, which will activate scripts on the other computers, start the test plan according to the property file and record all the results in some HTML pages with charts representations. The results files are stored on a common NFS mount.

Monitoring

A monitoring tool is also developed for the observation of clients and server during the test. A tool monitors a lot of data like CPU idle, number of processes, memory usage, network, … in order to guarantee the validity of the test. It also reminds the test execution configuration parameters. All the monitored data are obtained by the SAR Unix utility.

Conclusion

Positive parts of this software are the following:

- -Monitoring capabilities
- -Charts representations and automatic generation of HTML report

Negative part of this software is the following:

- -Specific to Unix environment and Rubis application

Conclusion

Many projects and small applications exist, but very few are well designed and complete. Moreover they mainly only test HTTP servers. Commercial software are more powerful with the possibility to test many other systems.

Here is a synthesis of each part.
We can distinguish three aspects in a scenario creation:

- Those that allow the user to write entirely his/her scenario, using a scripting language such as Jython, or using XML (like TestMaker, Grinder, LoadSim, Jmeter).
- Those that allow the user to build his/her scenario with some graphical objects representing some built-in interactions (like Jmeter, LoadTest). The scenario is represented as a JTree (see Java Swing GUI library).
- Applications that automatically record a scenario with a capture tool often integrated in an Internet browser (like DieselTest, OpenSTA, LoadSim, LoadTest).

Some applications also combines the last two parts (LoadTest).

The execution aspect is similar for every application. The model used is the creation of one thread by virtual user. Then it is possible to schedule the execution and creation of the virtual users with a delay, a ramp up period. Scenarios written in Jython can be arranged manually to deal with this aspect.

The monitoring part, if it exists, represents main characteristics: number of virtual users, response time, average response time, … . But only LoadTest and Rubis propose very detailed information on the target system: number of processes, processor characteristics, memory usage, …
Information about target system characteristics are available on Windows NT systems only with some external libraries. This problem does not exist under Linux or Unix environment.

In conclusion, we can say that each software has his advantages and drawbacks.
The most complete tool tested in this document is probably **LoadTest** which offers powerful capabilities to the user. The building of the scenario can be automatic or customized with scripts or graphical elements. The load test is very simple to launch, and monitored data are very detailed. This solution is adapted for testing web servers and visualize accurate system data.

DieselTest also proposes to test web servers, but with a more simple interface, which also means with less features. A capture tool simplifies the task of recording the scenario. Collected data are very basic, that is why this software is designed to test small web sites.

TestMaker is a powerful tool since the scenario is created with scripts in Jython which allow to use all the Java features. This framework permits to build a customized scenario and very precise test plan, with specific classes. However the monitoring is very poor, and distributed testing is not possible. But it seems interesting to have a look at the classes structure and hierarchy.

Grinder is similar to TestMaker, except that it allows distributed tests, so it can simulate heavy loads.

Jmeter offers some good features and can perform tests on several systems, but it is a little more difficult to use than the other software. It is possible to realize a wide variety of load tests, even to distribute the scenario. However, the application is weaken by the management of Java objects and memory concerns which affect global performance.
The major advantage is the possibility to add new features very simply in the application, it is also a good example of modularity and customization.

Rubis is characterised by the capability to test HTTP servers using a particular model of execution (transition matrix). Running only under Unix environment, it provides detailed information and charts on the target system. This application is adapted for a specific simulation.

OpenSTA provides some good performances for HTTP testing, with simple charts and distributed tests. It is a good application for simple and reliable HTTP tests.

LoadSim does not propose any graphical interface. With this application, it is only possible to create scenarios in XML or with a capture tool, execute the test eventually on remote hosts, and finally get the results in a text file.

Finally, in this overview we found a couple of applications that provide maximum efficiency for building a test plan. The keys features include:

- Easy, complete and customizable building of the scenario.
- Distributed tests and customizable sequencing of the scenario.
- Accurate charts and data from monitored systems.
- Customizable protocols.

APPENDICES

Extended list of applications (not exhaustive)

Nom	Lien
123 Load Test	http://www.wpidalamar.com/projects/123loadtest/
ANTS	http://www.red-gate.com/advanced_dotnet_testing_system.htm
Apache Bench	http://httpd.apache.org/docs/programs/ab.html
Apache JMeter	http://jakarta.apache.org/jmeter/
Astra Loadtest	http://www-svca.mercuryinteractive.com/products/
Bean-Test	http://www.empirix.com/Empirix/
Benchmark Factory	http://www.benchmarkfactory.com/
CapCal	http://www.capcal.com/
Deluge	http://deluge.sourceforge.net/
Dieseltest	http://dieseltest.com/
DigitalBees Load	http://www.digitalbees.com/uses.htm
EasyWebLoad	http://www.easywebload.com/
ECPerf	http://java.sun.com/j2ee/ecperf/
EJBSPEC	http://sourceforge.net/projects/ejbspec/
e-Load Expert	http://www.empirix.com/Empirix/Web+Test+Monitoring/Testing+Solutions/
EtherApe	http://etherape.sourceforge.net/
eValid	http://www.soft.com/
FirstACT	http://www.scl.com/products/empirix/datasheets/firstact.html
Flight	http://flight.sourceforge.net/
FORECAST	http://www.facilita.co.uk/
Fungus	http://valvassori.free.fr/unix/fungus/
Grinder	http://grinder.sourceforge.net/
Hammerhead	http://hammerhead.sourceforge.net/
http-Load	http://www.acme.com/software/http_load/
Jabez	http://sourceforge.net/projects/jabez/
JavaLoad/JavaStart	http://www.products.datamation.com/development/java/916161638.html
Jblitz	http://www.clanproductions.com/jblitz/index.html
Jmeter	http://jakarta.apache.org/jmeter/index.html
JunitPerf	http://www.clarkware.com/software/JUnitPerf.html
LMBench	http://www.bitmover.com/lmbench/
Load	http://www.pushtotest.com/ptt
LoadPro	http://www.keynote.com/solutions/solutions_pt_loadpro_tpl.html
LoadSim	http://www.openware.org/loadsim/
Mercury LoadRunner	http://www-svca.mercuryinteractive.com/products/
Microsoft Application Center Test	http://msdn.microsoft.com/library/default.asp?url=/library/en-us/act/htm/actml_main.asp
Microsoft WCAT load test tool	http://support.microsoft.com/support/kb/articles/Q231/2/82.ASP
MTLoadGenerator	http://sourceforge.net/softwaremap/trove_list.php?form_cat=138&page=4
NetPressure	http://www.syntheticnets.com/
Netwok Traffic Generator	http://galileo.spaceports.com/~rsandila/traffic.html
Open DTE	http://opendte.sourceforge.net/
Open Performance Suite	http://sourceforge.net/projects/performancenet/
Open STA	http://opensta.org/
OpenLC	http://openlc.sourceforge.net/
OpenLoad	http://openload.sourceforge.net/
PerfectLoad	http://www.freakysoft.com/

Portent Web Load test tool	http://www.loadtesting.com/
Postal	http://www.coker.com.au/postal/
Project OpenLoad	http://www.opendemand.com/openload/
PureLoad	http://www.minq.se/
QALoad	http://www.compuware.com/products/qacenter/
QuotiumPro	http://www.quotium.com
Radview's WebLoad	http://www.radview.com/
Siege	http://www.joedog.org/siege/index.shtml
SilkPerformer	http://www.segue.com/
Site Tools	http://www.softlight.com/sitea/index.asp
SiteTester1	http://www.pilotltd.com/eng/index.html
SiteTools Loader	http://www.softlight.com/sitea/index.asp
SSLider	http://sourceforge.net/projects/sslider/
Test Perspective Load Test	http://www.keynote.com/solutions/html/testing.html
TestMaker	http://www.pushtotest.com/ptt
Torture	http://stein.cshl.org/~lstein/torture/torture.html
TPTest	http://tptest.sourceforge.net/index.php
UrbanCode EJBBenchmark	http://www.urbancode.com/projects/ejbbenchmark/default.jsp
VeloMeter	http://www.velometer.com/
WAGON: A Web Server Benchmarking Tool	http://www-sop.inria.fr/mistral/personnel/Zhen.Liu/wagon.html
Web Avalanche	http://www.cawnetworks.com/
Web Polygraph	http://www.web-polygraph.org/
Web Roller	http://www.webapplicationstesting.com/index.html
WebART	http://www.oclc.org/webart/
Webhammer	http://www.genusa.com/iis/webhamr2.html
Webmaster Solution: Site Stress	http://www.webmastersolutions.com/loadtesting.shtml
WebPerformance Trainer	http://www.webperformanceinc.com/
WebServer Stress Tool	http://www.paessler.com/
WebSizr/WebCorder	http://www.technovations.com/
WinRunner	http://www-svca.mercuryinteractive.com/products/winrunner/features/
Zeus: Web Server Benchmarking	http://www.zeus.com/products/zws/capacity/benchmarking.html

Synthesis

Name	Home page	License	OS	Language	Protocols	Scenario designing	Distributed tests
OpenSTA	http://opensta.org/	open source	Windows	C++	HTTP	GUI, capture, scripting	yes
DieselTest	http://dieseltest.com/	open source	Windows	Delphi	HTTP	GUI, capture	no
TestMaker	http://www.pushtotest.com/ptt	open source	all	Java	HTTP, POP, SMTP, IMAP	GUI, scripting	yes
Grinder	http://grinder.sourceforge.net/	open source	all	Java	all	scripting	yes
LoadSim	http://www.openware.org/loadsim/	open source	all	Java	HTTP	XML, capture	yes
JMeter	http://jakarta.apache.org/jmeter/index.html	open source	all	Java	HTTP, FTP, JDBC, Java	GUI, XML	yes
LoadTest	http://www-svca.mercuryinteractive.com/products/loadrunner	commercial	Windows	?	HTTP	GUI, capture, scripting	yes
Rubis	http://www.objectweb.org/rubis	open source	Unix	Java & sh scripts	HTTP	CSV (text)	yes