



From Technologies to Solutions

Apache JMeter

A practical beginner's guide to automated testing and performance measurement for your websites

Emily Halili

PACKT
PUBLISHING

Chapter 5. Load/Performance Testing of Websites.....	1
Preparing for Load Testing.....	2
Using JMeter Components.....	3
Running the Test Plan.....	18
Interpreting the Results.....	18
Remote Testing with JMeter.....	21
Monitoring the Server's Performance.....	22
Summary.....	24

5

Load/Performance Testing of Websites

So you are ready now to take on Web Load/Performance Testing on your target web application.

Before we embark on this significant journey, I am obliged to highlight the fact that load testing and performance testing do somewhat differ. IEEE 90 defines Performance Testing as, *Testing conducted to evaluate the compliance of a system or component with specified performance requirements*. Some of its goals include to identify bottlenecks in a system, finding which can later support performance tuning efforts, to aid in auditing the system's performance, and/or to collect other relevant data to help stakeholders make informed decisions related to the quality of the application under test.

Load Testing, being a part of performance testing, is simply the process of subjecting a component or a whole system to a work level approaching its limits. Another point of concern is that some quarters use load testing interchangeably with stress testing. This may not be an entirely accurate practice, as there is a thin line between these two terms. Load testing is performed within the capacity of the resource(s) being subjected, while stress testing is normally performed to evaluate the performance of resources and behavior at or beyond normal capacity. In summary, both load testing and stress testing are part of what makes performance testing.

For sake of argument (and so as not to overwhelm you – the reader), this chapter will focus on using JMeter to perform Web Application Load Testing as being a part of Performance Testing. First, it will give you some general guides to help you prepare and plan for a load test. The remaining part of the chapter will give you a step-by-step walkthrough in bringing together JMeter components to build a load-test plan. The final section of this chapter will capture the test results after running the test.

Preparing for Load Testing

In preparing for load testing it is utterly important to address a number of concerns with regards to the target server under test. As load testing helps to benchmark performance behavior of a server, it is important to be able to identify the general expectations and other matters that would normally be taken into account in order to carry out a successful load testing.

What You Need to Know

As noted earlier, load testing in this matter subjects the application server to work approaching its limits. Obviously, the limits will need to be clearly defined, understood, and agreed upon by the stakeholders, namely your superior(s). In addition, performance metrics need to be clear in order to keep the performance goals in check.

Important expectations as for any load testing include:

- A suitable time to load-test the application, for instance when no development work is taking place on the server (load testing may cause the server to crash) and/or no other users are accessing the server (else the testing results would not yield the correct measures)
- The performance metrics, accepted levels, or SLAs and goals
- Objectives of the test
- The Internet protocol(s) the application is(are) using (HTTPS, HTTP, FTP, etc.)
- If your application has a state, the method used to manage it (URL rewriting, cookies, etc.)
- The workload at normal time and at peak time

It is often advisable that any form of performance testing, inclusive of load testing, is performed on a functionally stable application, regardless of the environment where the application is located. Load testing is best done when the functionality of the **Application Under Test (AUT)** is stable enough to yield consistent and correct results.

Some Helpful Tips to Get Better Results

- Use meaningful test scenarios (use cases are helpful) to construct test plans with 'real-life' test cases.
- Run JMeter on a machine other than that running the application you are testing.

- Make sure that the machine running JMeter has sufficient network bandwidth, so the network connection has little to no impact on the results. Also, the machine running JMeter should have enough computing power (memory, CPU) to generate load.
- Let JMeter Test Plan run for long time periods, hours or days, or for a large number of iterations. This may yield a smaller standard deviation, giving better average results. In addition, this practice may test system availability rate and may highlight any decay in server performance.
- Ensure that the application is stable and optimized for one user before testing it for concurrent users.
- Incorporate 'thinking time' or delays using Timers in your JMeter Test Plan.
- Conduct tests under a monitored and controlled environment, to prevent other users from affecting JMeter results.
- Keep a close watch on the four main things: processor, memory, disk, and network.
- Only run JMeter against servers that you are assigned to test, else you may be accused of causing DoS attacks.

Using JMeter Components

For practical and realistic reasons, we will use an existing remote server to test its performance. First of all, we will create some useful scenarios as the baseline of our test.

First, we will need to determine the test cases. Generally, we will test five key scenarios:

1. **Homepage**
2. **Keyword Search**—New Visitor making a keyword search
3. **Create Account**—New Visitor creating an Account
4. **Select A Title**—Registered Visitor selecting a featured title
5. **Add To Cart**—Registered Visitor adding selection to cart

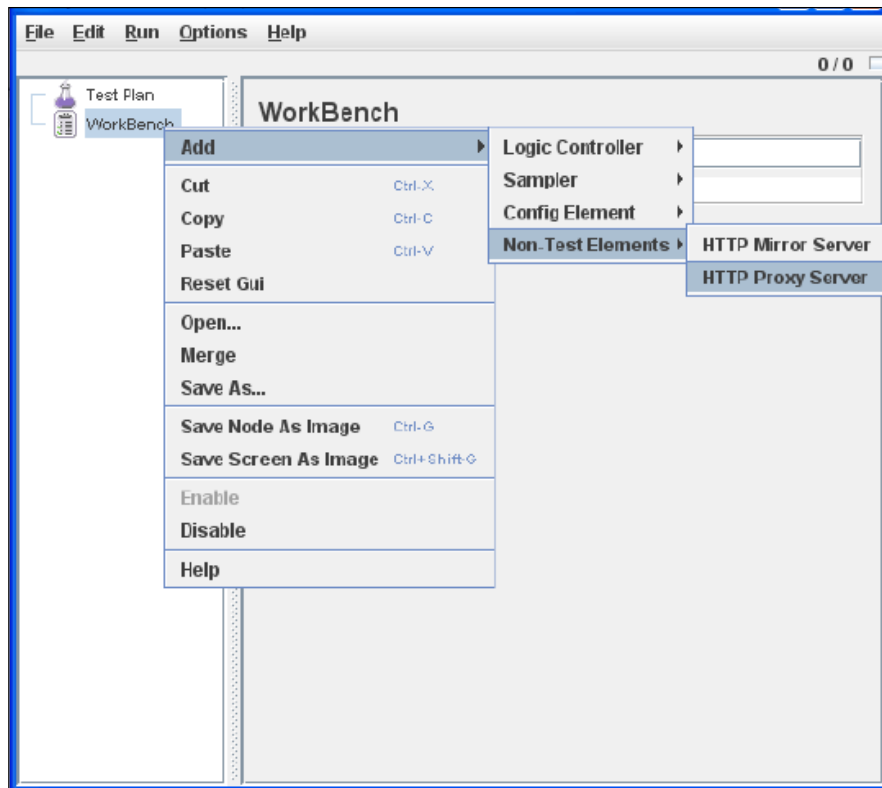
These scenarios will be included in a single JMeter Test Plan, for simplicity reasons.

Recording HTTP Requests

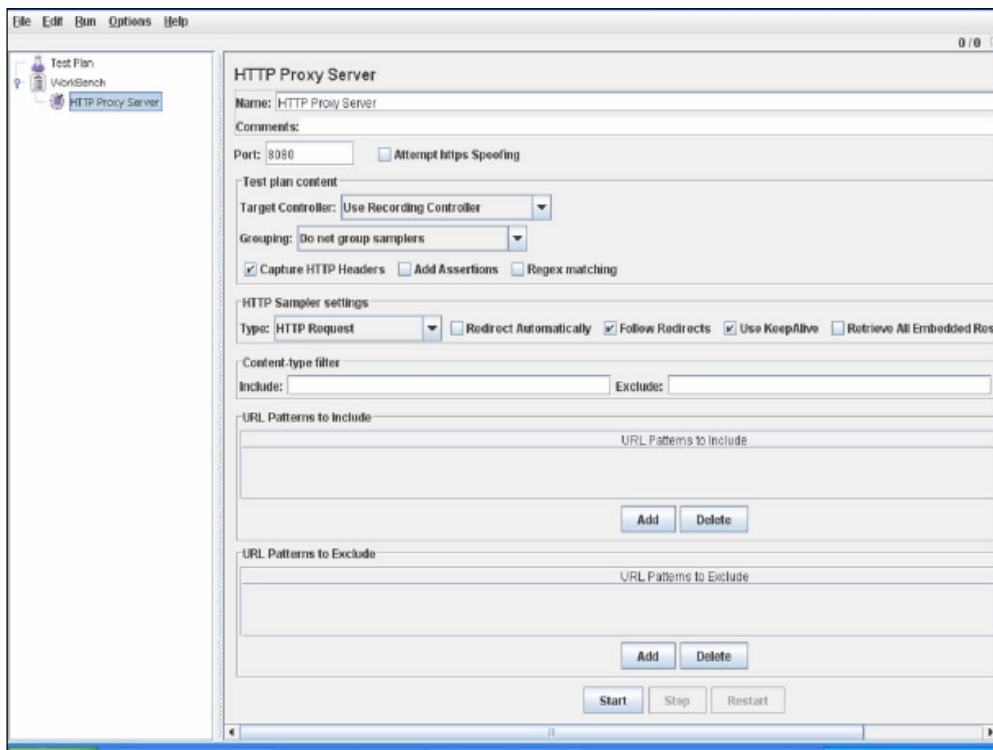
A fast way to capture the HTTP pages of this application is to record every request made to the server. For this we will need to use a special-purpose Configuration Element: HTTP Proxy Server. Note that the Proxy Server element is only available in the Workbench element.

Since Proxy Server element records any page requests, besides user requests, it will also record background requests made by the browser as modern browsers do. As we would like the Proxy element to record only requests to the target server, consider setting web filter to allow the current browser to make requests only to that server. Otherwise, unnecessary requests will only clutter your recording. You may find these web filters as an option in any Internet security tool or software currently running on your machine.

Run JMeter (double-click `JMeter.bat` from `jmeter/bin` folder). You will see the default Elements, which are **Test Plan** and **Workbench**. Right-click on **Workbench** and select **Add | Non Test Elements | HTTP Proxy Server**.

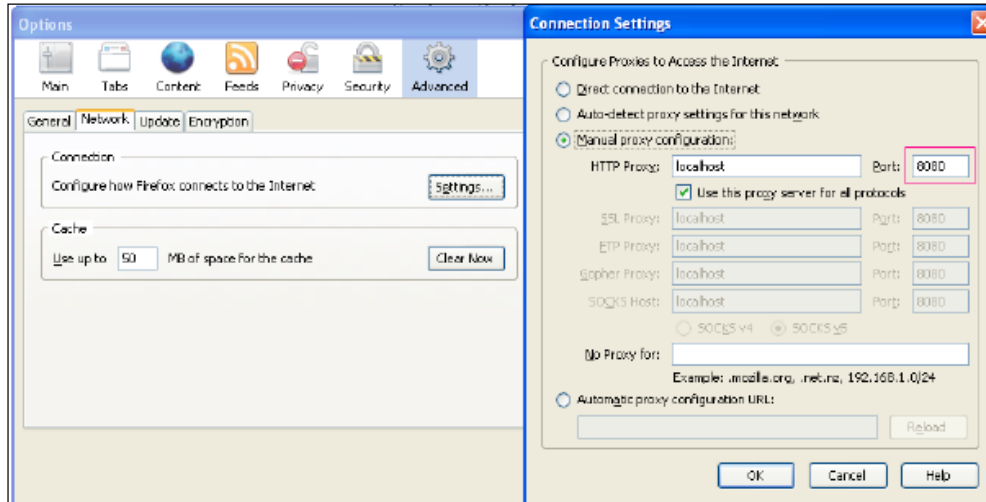


An HTTP Proxy Server configuration Element will appear as a child of WorkBench. Bear in mind that child elements of WorkBench are not saved as part of Test Plan, but you can save them separately. In the HTTP Proxy server configuration Target Controller lets you determine where the recorded requests will be placed and the Grouping option allows the recorded pages to be grouped or left as individual requests. *Notice that the port we are using is the default port, following the browser setting in the next instructions.*

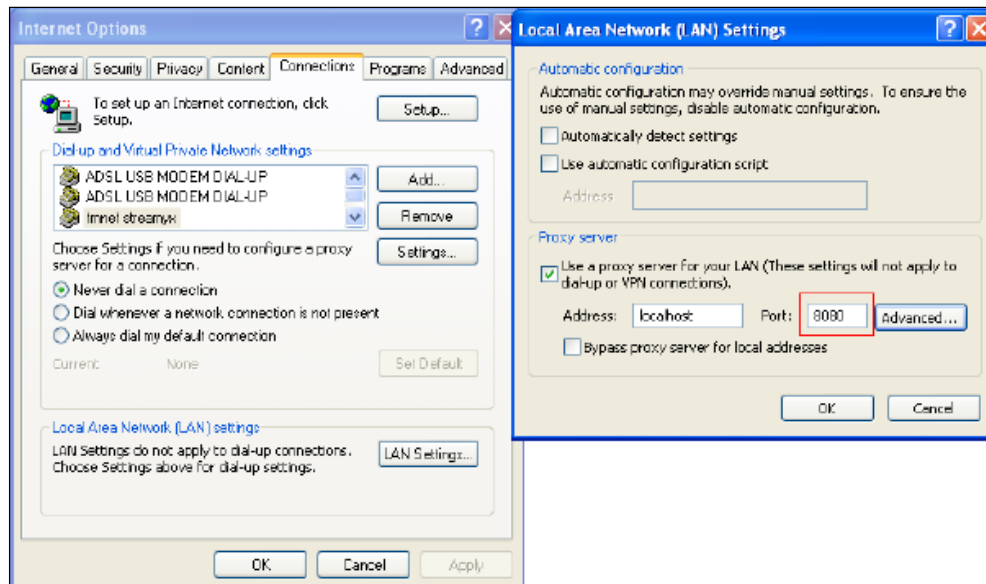


Next, we will change the setting of your favorite browser to access a proxy (in our case JMeter HTTP Proxy), listening to the same port. You may also use port 90 for both the browser and JMeter Proxy setting.

The following figure shows the setting for Mozilla Firefox:

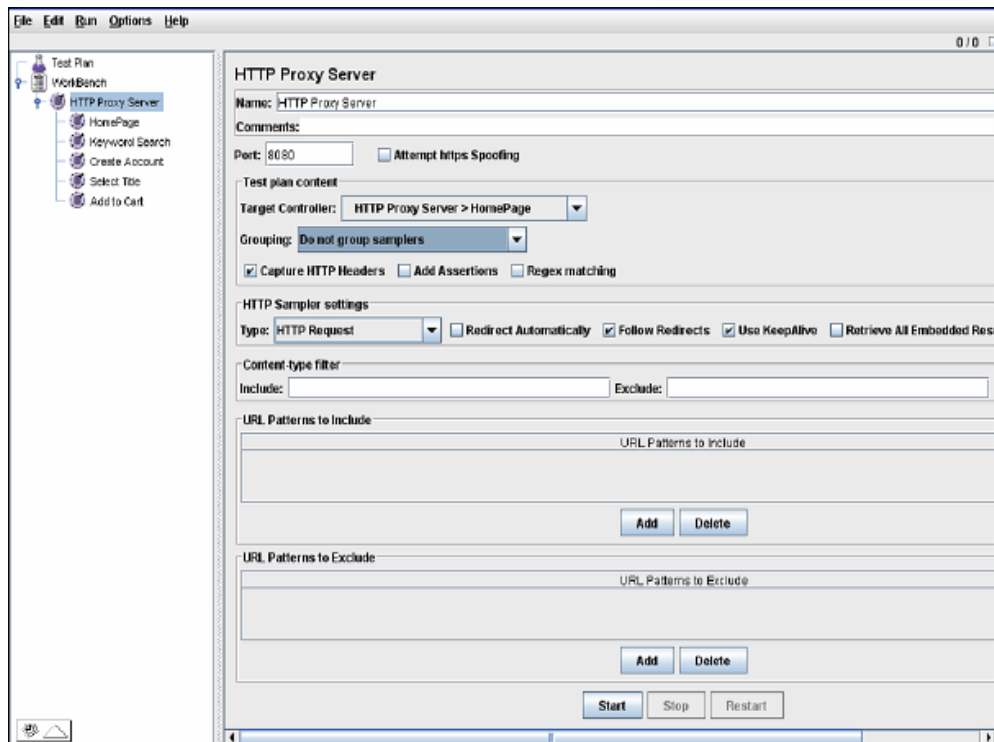


The following screenshot shows the setting for IE:



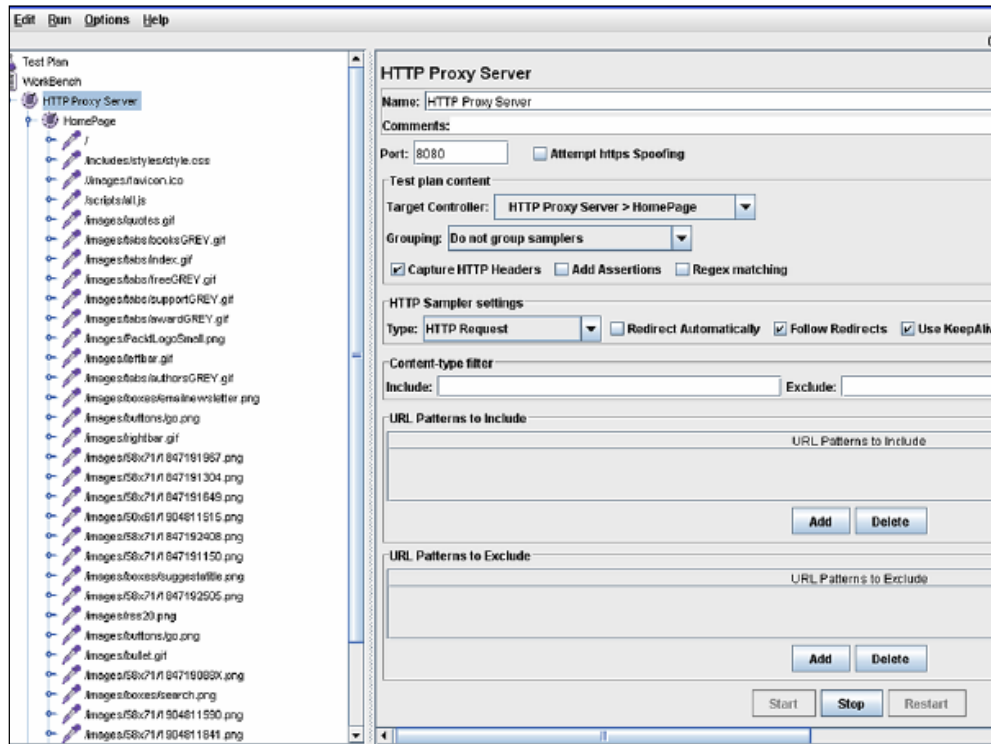
We are expecting that a single page request will make several embedded requests for images, JavaScript files, CSS files, etc. Therefore for a more managed recording, it is a good practice to create controllers that can contain the sub-requests for each request.

Right-click on **HTTP Proxy Server** and select **Add | Logic Controller | Simple Controller**. Repeat so we have five Controllers, or you can simply copy and paste. Name each Controller: **Homepage**, **Keyword Search**, **Create Account**, **Select Title**, and **Add to Cart**. Then configure the Target Controller to **HTTP Proxy Server | Homepage**, while the other defaults remain.

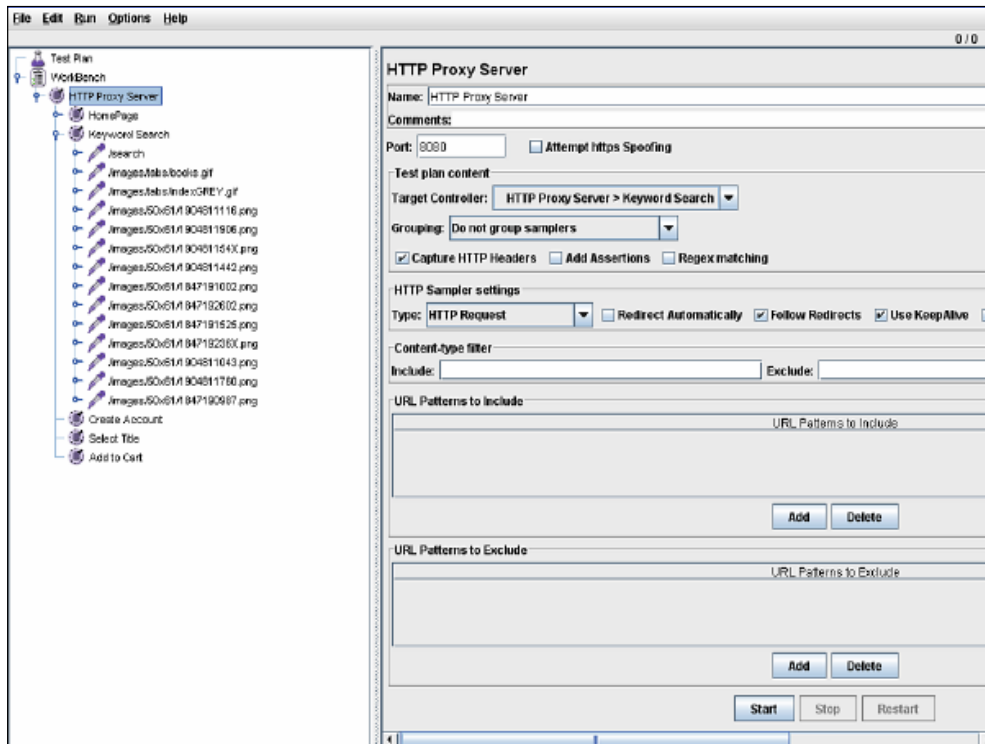


Now we are ready to record our first page request. As we return to JMeter, simply press the **Start** button at the Element Controller, and use your browser. Remember that JMeter records all HTTP requests from the browser you are using; therefore, with all the rich browsers and with all the add-ons we have today, you might want to filter that only requests to the targeted server are allowed. You may configure your browser or firewall to do so.

Click the **Start** button, and type the **URL** of the server you want to test in the Address bar. You will see that JMeter has begun to record the request to the homepage and its sub-requests into Homepage Controller.

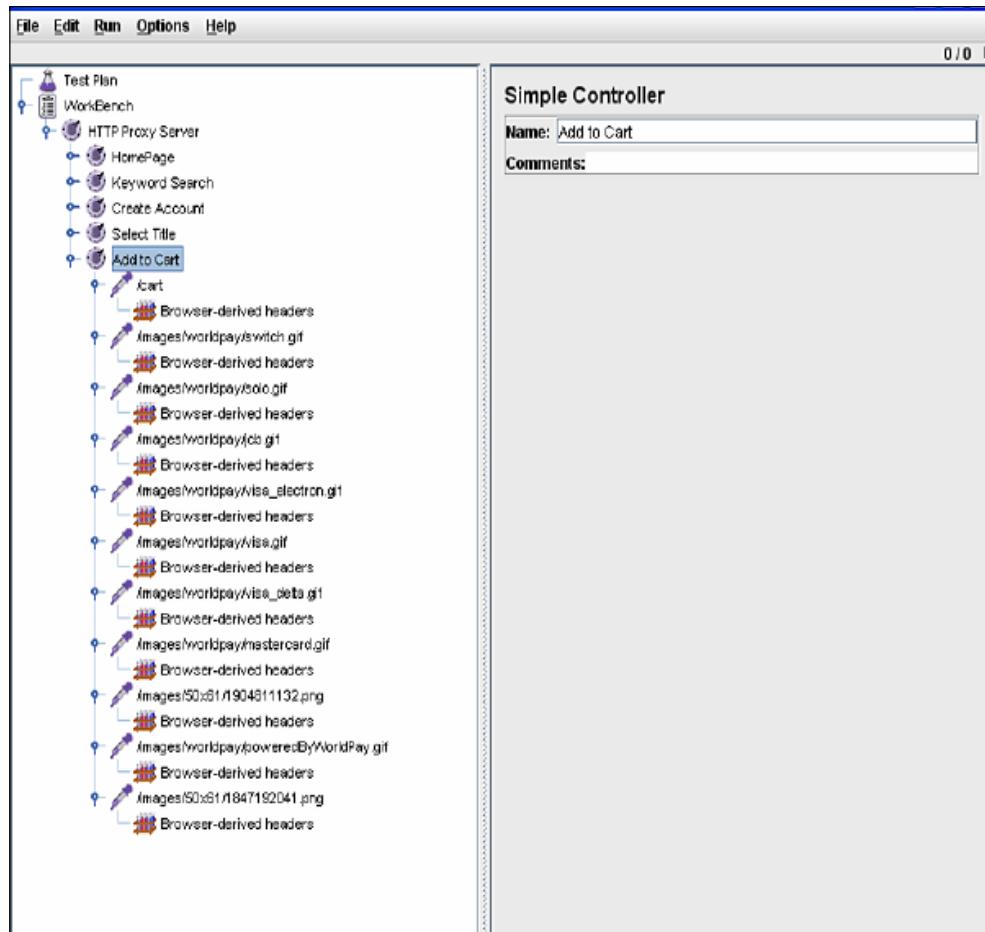


Next, on the Target Controller, select **HTTP Proxy Server | Keyword Search**, return to the browser and type in a keyword and click search. JMeter will record the following search page and its sub-requests in the **Keyword Search** Controller and nowhere else.



Repeat for the **Create Account**, **Select Title**, and **Add to Cart** Controllers, switching to the respective Target Controllers.

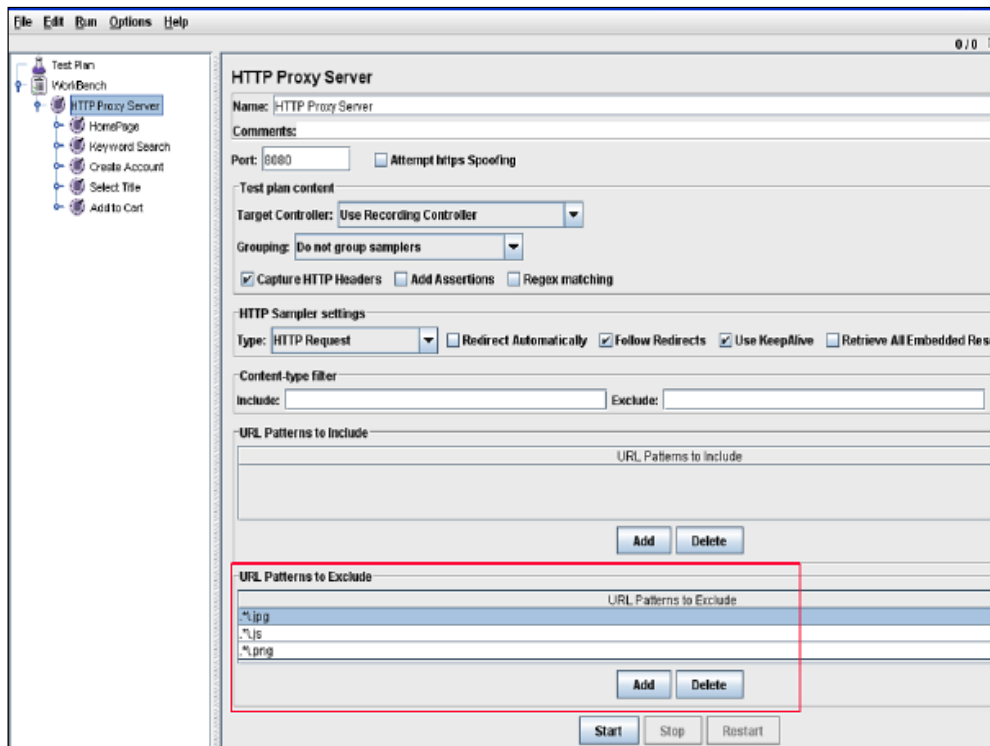
The final recording screenshot is as shown in the following figure:



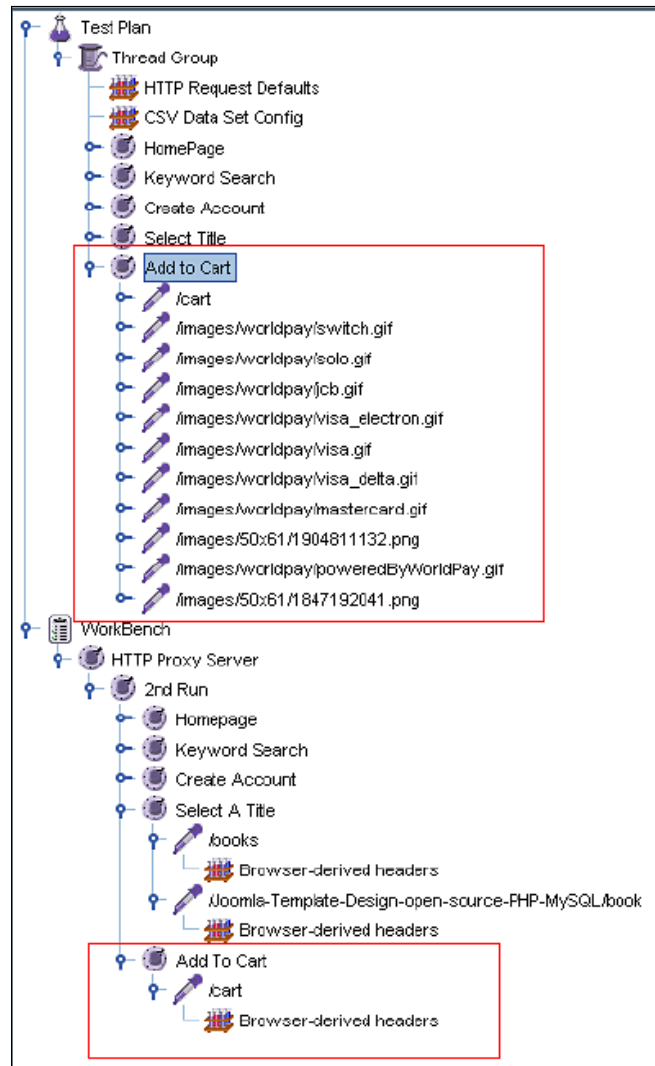
Once the recording is over, we will save these by right-clicking on the **HTTP Proxy Server** Controller and saving it in a folder of your choice. As you will see, each request may or may not generate sub-requests for files. The caching feature of today's web browsers allows these files (*.png, *.jpg, *.css, *.js, etc.) to be stored in the browser's cache the first time they are downloaded. Unless there were changes in the main request, the browser will not make new requests for these cached files as it will simply load them from the local cache. How is this feature helpful in benchmarking

the performance of an application? To iterate, this has become a test goal decision to make whether the application should be tested to evaluate performance for first-time visitors or existing visitors. If we are testing for first-time visitors, we can use the first recording to simulate first-time users. Subsequent recordings can be used to simulate existing users.

Alternatively, we can configure the Proxy Server Configuration Element to exclude recording of particular file type(s), as the following figure indicates.



Subsequent recording of similar actions using the new configuration of HTTP Proxy Server will exclude the caching files. Let us perform another round of recording and see how that turns out. The following screenshot highlights the first and second round of recording.



For our purpose, we will simulate 10 existing visitors, while running the test plan for a minimum of 10 iterations, with a think time between one and three seconds. We will use the second recording too, as it is the closest to emulate existing visitors,

as we would expect that the caching files would have already been stored locally. To remove elements, highlight the Controllers in the Test Plan and press delete. To move the new Controllers to the Test Plan, simply highlight and Copy Paste in the Thread Group.

Creating the Test Plan

We will begin by creating a single Thread Group (Users Group) that we will configure later as we expand the Test Plan.

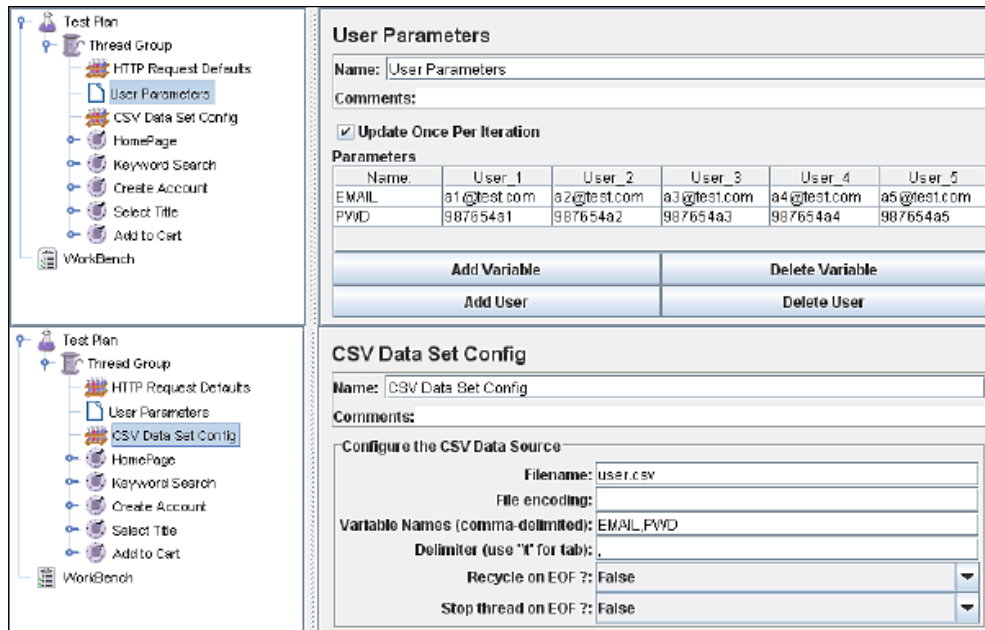
Right-click on the **Test Plan** element and select **Add | Thread Group**. A Thread Group Element will appear. Configure the **Number of Threads** to **10**, **Ramp-Up Period** to **1 second**, and **Loop Count** to **50**. If you wish, you can set the **Scheduler** so your test plan can run automatically on the pre-determined time and date.

We want each request to target only one server, therefore, we set a request default to serve this purpose. Add to the **Test Plan Config Element | HTTP Request Defaults**:



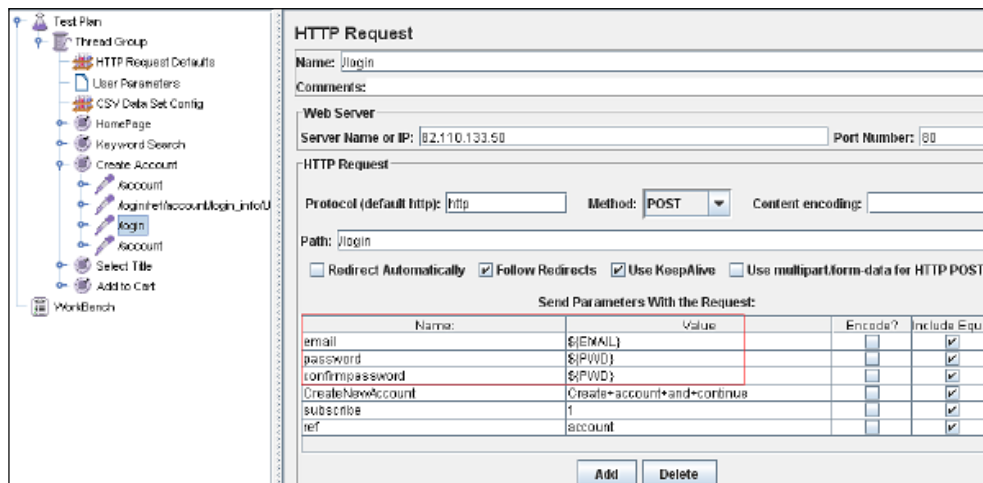
As this test should emulate real-life scenarios, this test requires that 500 unique accounts be created at the **Create Account** Request. Since we will need to generate multiple new users as JMeter creates 'new' accounts, one option is to parse unique account pairs into the appropriate Sampler, in this case the `\login` request in the **Create An Account** Controller. JMeter provides this capability by **Pre-Processor | User Parameter Element** or **Config Element | CSV Data Set Config**. The **User Parameter Element** allows the user to specify unique values for User Variables specific to individual threads. The **CSV Data Set Config** element serves the same purpose; however, these values are read from lines from a file, and split into variables that later can be used throughout the life of the thread.

For providing a large number of users, CSV Data Set Config would be a better choice. The following figure will compare both the elements.



Since we need to create 500 unique accounts, the natural choice would be using **CSV Data Set Config Element**. It is advisable that the CSV file created for this test be located in the same directory as the Test Plan. This element gives you the option to define the delimiter of your data file. Since we want only these 500 unique accounts created and nothing more, we do not choose **recycle on EOF**, which would reread these pairs from the beginning of the file once the whole file is parsed.

These data will need to be parsed into the appropriate parameter(s) using the function syntax: `${VARIABLE-Name}`. We will use these account pairs in **Create Account | /login Sampler** where you see the corresponding variable names or parameters (email, password, confirm password) are captured in the Sampler. In the following figure you may notice the **Value** for these parameters corresponds with Variable Names defined earlier in the CSV Data Set Config Element.



Adding Listeners

We are now ready to add Listeners to our Test Plan. As we are evaluating performance based on scenarios, each scenario Controller will have its own Listener. One Listener is sufficient to capture the performance data, as the saved data can be represented in various ways according to the Listener selected to view these data. The following steps will give you a better walk through.

Right-click on the **Homepage** Controller and select **Add | Listener | View Results in Table**. In the Filename text-field simply type the name of the XML or JTL file, along with its extension, that will store the results for the requests in this Controller. By default this will be located in the \bin folder of your JMeter installation path; however, you may choose to specify a different location. Follow similar steps for all other Controllers.

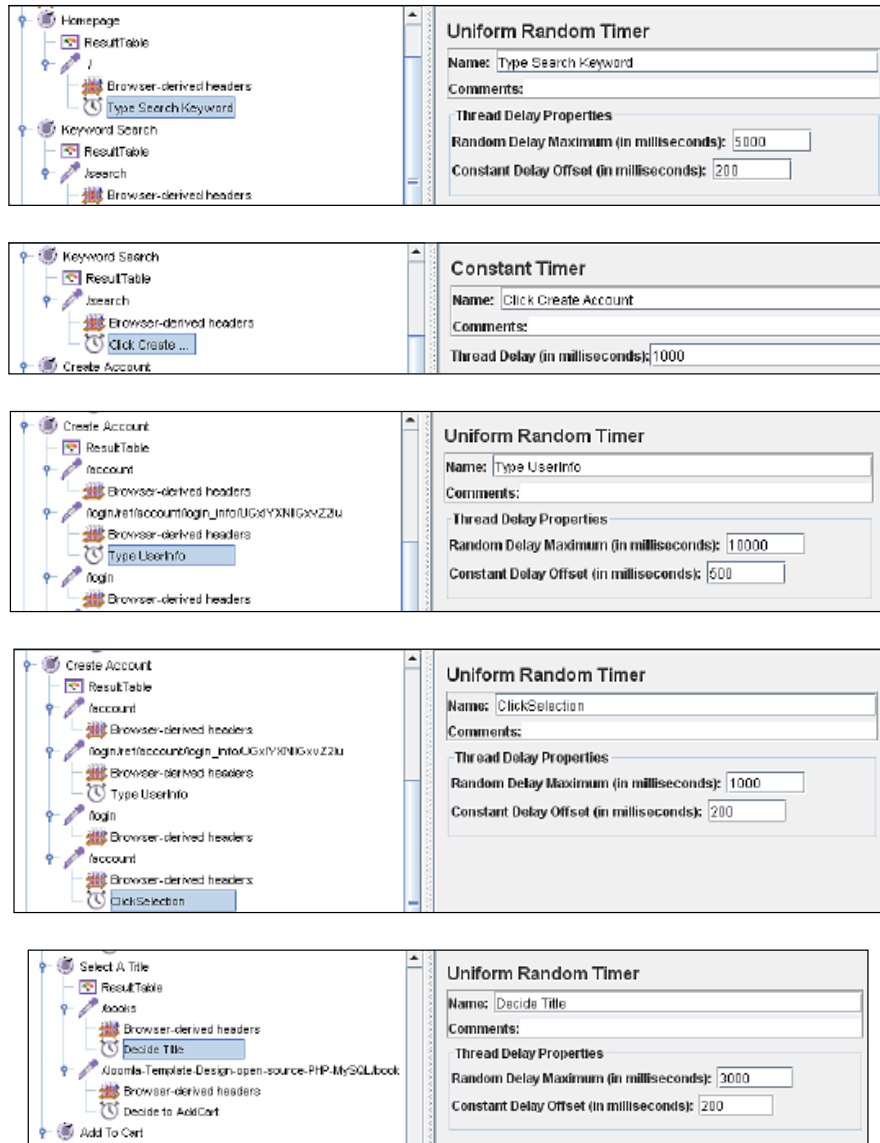
Adding Timers

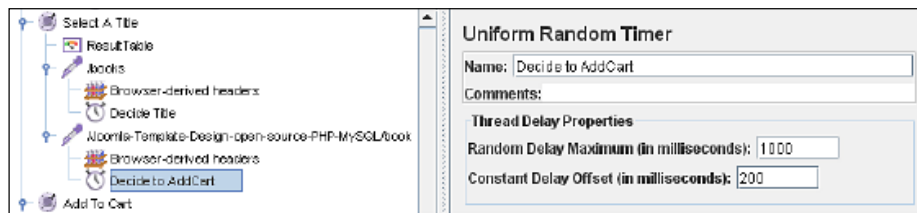
As we are emulating real-life visitors/users, we will need to consider delays between requests. Such delay is also known as **think time**, which a real user will take as he/she will need to decide the next action, be it a click to some other link, or pressing some button, etc. that causes a new request to the target server.

To add a Timer, right-click on the element for which we want to simulate the think-time, select **Add | Timer**, and choose the type of timer. For this exercise, we will configure the two timers we are using, **Uniform Random timer** and **Constant timer**, to emulate real-user actions as closely as possible. 'Uniform Random' timer pauses

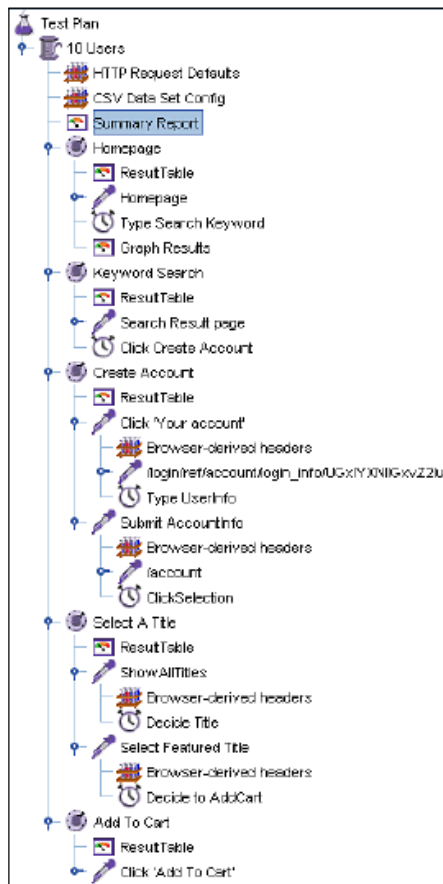
each sampler request for a random amount of time, with each time interval having the same probability of occurring. The total delay is the sum of the random value and the offset value. Meanwhile, 'Constant timer' allows the thread to pause for the same amount of time between requests.

The following screenshots explain the above paragraph.





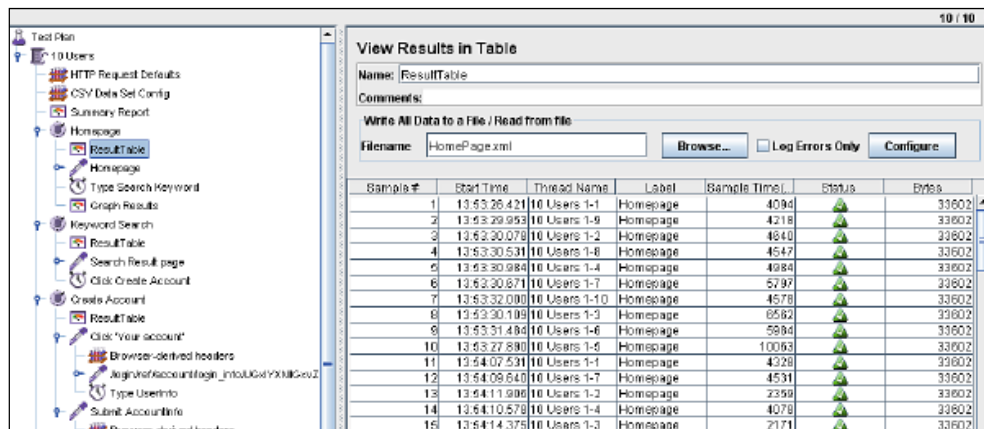
We may need to organize the Test Plan tree, by placing sub-requests of a request Sampler as children of the Sampler, so that we can measure the performance better according to particular actions of a user. To view the Results for all Samplers, we may add a Summary Listener of the scope of the Test Plan and configure the Filename so the test summary data is saved separately. The final Test Plan may look as in the following snapshot:



Running the Test Plan

We are now ready to run the Test plan we have built. We may rename the Thread Group to "10 Users" for better documentation. Look at the tiny gray indicator box at the top right of the Control Panel. The numbers beside the box indicate number of active threads vs. total number of threads in the Thread Group.

JMeter requires saving the Test Plan before running; unless indicated otherwise, it will save the Test Plan in the `\bin` folder of your JMeter installation path. To run the Test Plan, go to the **Run** menu and select **Start**. As soon as it runs, the gray box will turn green as JMeter ramps up the total number of active threads to 10. You will see that JMeter takes approximately 5 seconds to activate the total number of users with a delay of 100ms (1000ms per 10 users) between subsequent threads. This demonstrates the 1 second ramp-up time we have set in Thread Group earlier.



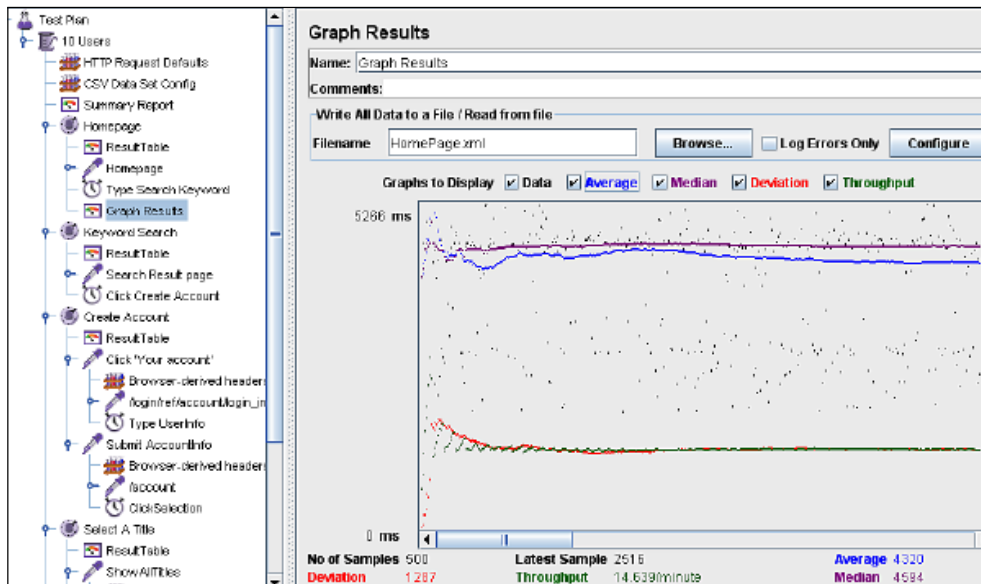
Sample #	Start Time	Thread Name	Label	Sample Time	Status	Bytes
1	13:53:26.421	10 Users 1-1	Homepage	4094	✓	33602
2	13:53:29.953	10 Users 1-9	Homepage	4218	✓	33602
3	13:53:30.079	10 Users 1-2	Homepage	4640	✓	33602
4	13:53:30.531	10 Users 1-8	Homepage	4547	✓	33602
5	13:53:30.984	10 Users 1-4	Homepage	4984	✓	33602
6	13:53:30.871	10 Users 1-7	Homepage	5797	✓	33602
7	13:53:32.000	10 Users 1-10	Homepage	4578	✓	33602
8	13:53:30.109	10 Users 1-3	Homepage	6582	✓	33602
9	13:53:31.484	10 Users 1-6	Homepage	5984	✓	33602
10	13:53:27.890	10 Users 1-5	Homepage	10063	✓	33602
11	13:54:07.531	10 Users 1-1	Homepage	4326	✓	33602
12	13:54:09.640	10 Users 1-7	Homepage	4531	✓	33602
13	13:54:11.906	10 Users 1-2	Homepage	2359	✓	33602
14	13:54:10.579	10 Users 1-4	Homepage	4078	✓	33602
15	13:54:14.375	10 Users 1-3	Homepage	2171	✓	33602

When the test is complete, the indicator box will return to gray.

Interpreting the Results

Once the test is completed, we can now retrieve the results we have saved for each Controller. With the exception of the Assertion Result Listener, the saved data can be viewed in numerous forms. Let us use `HomePage.xml` as our specimen dataset. Add more Listeners to this Controller: Summary Report, Aggregate Result, and Graph Results. To retrieve the results for this Sampler, type in the name of the file to which you saved data for this Sampler, and press *Enter*. The following snapshots show the result views.

For **Graph Results**, the **Data** legend shows us the widely dispersed data, representing the large value of the Standard **Deviation** across all samples for this Homepage Sampler. In the case where the results are highly skewed or not symmetrical using 'mean' would result in inaccurate representation of response time. The **Median** value, which is found in Aggregate Report and Graph Results would closely approximate the response time.



The saved results can be viewed in various forms. The following snapshot is of the saved test result viewed using the **Summary Report** Listener.

Summary Report

Name: Summary Report

Comments:

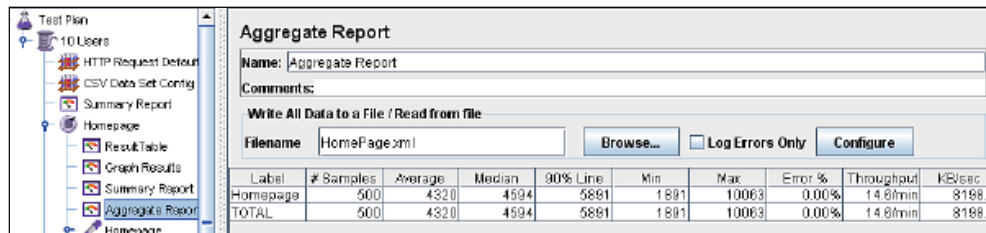
Write All Data to a File / Read from file

Filename: HomePage.xml

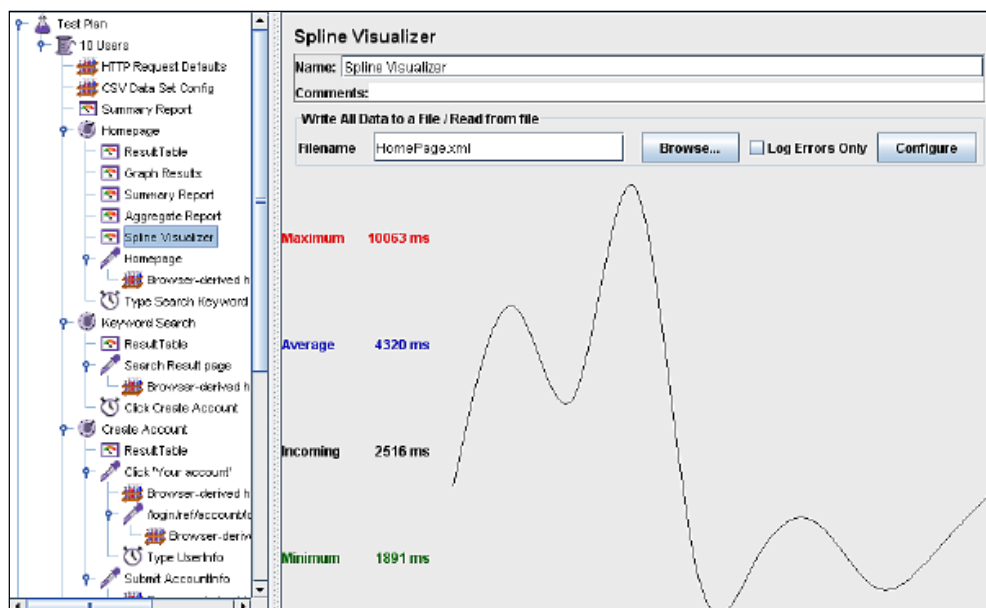
Browse... Log Errors Only Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
Homepage	500	4320	1891	10063	1287.89	0.00%	14.7/min	9.02	33602.0
TOTAL	500	4320	1891	10063	1287.89	0.00%	14.7/min	9.02	33602.0

The following snapshot is of the saved test result viewed using the **Aggregate Report** Listener. Note that Summary Report and Aggregate Report display the same set of test results differently, following different computation over the same data.



In the case where data distribution is even, or follows the 'bell' distribution, median and average will have the same or only slightly different values. We can see the data distribution pattern by adding a **Spline Visualizer** Listener using the filename of the file used to store the sampler results. The following shows us the distribution graph for requests to this page.



In this test, we can approximate that for the 10 simultaneous users, the approximate response time would be the median figure, since the data is widely spread.

How do we measure the point at which the server's performance degrades, or bottlenecks may appear? This important key question may be resolved if we run several thread groups that represent increasing number of 'Users', for example 10, 50, 100 users or even 500 or more for a large-scale, mission-critical server. We can then identify the points where performance degradation begins taking place. These User groups can be placed in the same Test Plan, or in a separate Test Plan, while maintaining the default configurations. This sort of practice allows us to benchmark the server's performance to find the bottlenecks in the server, hence creating a baseline for future tests.

Our test, however, does not seem to show any performance degradation, simply because it is running only a handful of users almost simultaneously. We run the test for 50 times so that we can get a better shot at the figures, in case failed requests occur, which normally would happen in real-life. You may want to expand the Test Plan further to include 50, 100, or even 500 users to find out if any performance degradation could occur.

With an average of 4.3 and median of 4.5 seconds, its speed of response is generally acceptable as it meets the threshold of acceptability for retail web page response times. See http://www.akamai.com/html/about/press/releases/2006/press_110606.html for more insight on this finding.

You may find that this analysis is limited as it yields results for only a minimum number of scenarios and concurrent users. But fear not, there are plenty more analyses you can make out of your future tests, if you create larger Thread groups, and maybe opt for stress testing and/or monitoring the server for performance.

Remote Testing with JMeter

Although this chapter does not give further details about stress testing your target server, this topic may capture your interest. This remote testing capability is very useful when your machine alone may have performance issues as it tries to simulate a very large number of concurrent users. This may well affect the speed and frequency of requests made to the target server, therefore affecting testing goals and subsequently, the results.

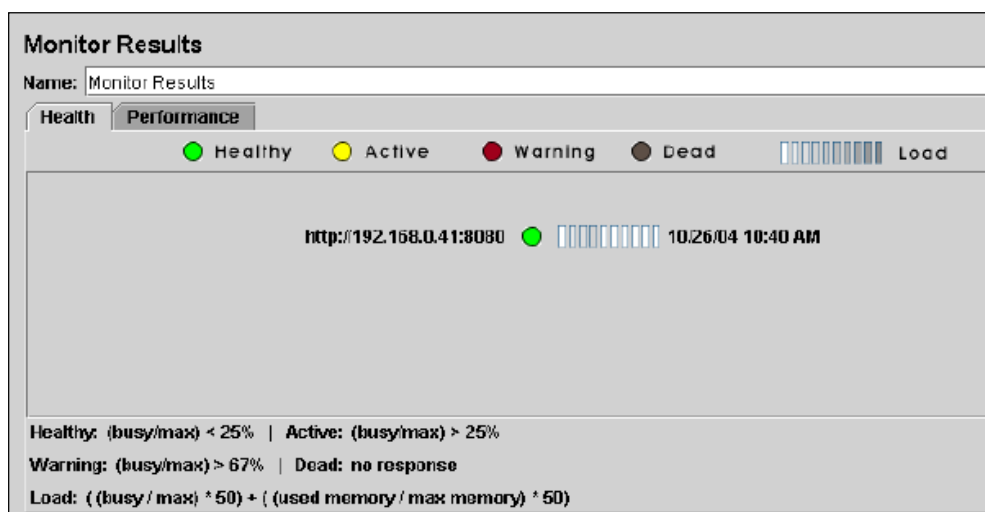
For more details on this attractive feature, you may refer to the online manual on remote testing on <http://jakarta.apache.org/jmeter/usermanual/remote-test.html>. JMeter Wiki also provides a simple and easy to follow guide (PDF) – http://jakarta.apache.org/jmeter/usermanual/jmeter_distributed_testing_step_by_step.pdf.

These guiding documents highlight following settings and configurations. You may remotely assign 'slave' machines to make requests to the target server while your machine becomes the 'master'. In other words, one machine controls the execution of the specified JMeter tests, as the results are collected at the 'master' machine. This approach lets us benchmark the target server's performance even more closely and effectively than otherwise.

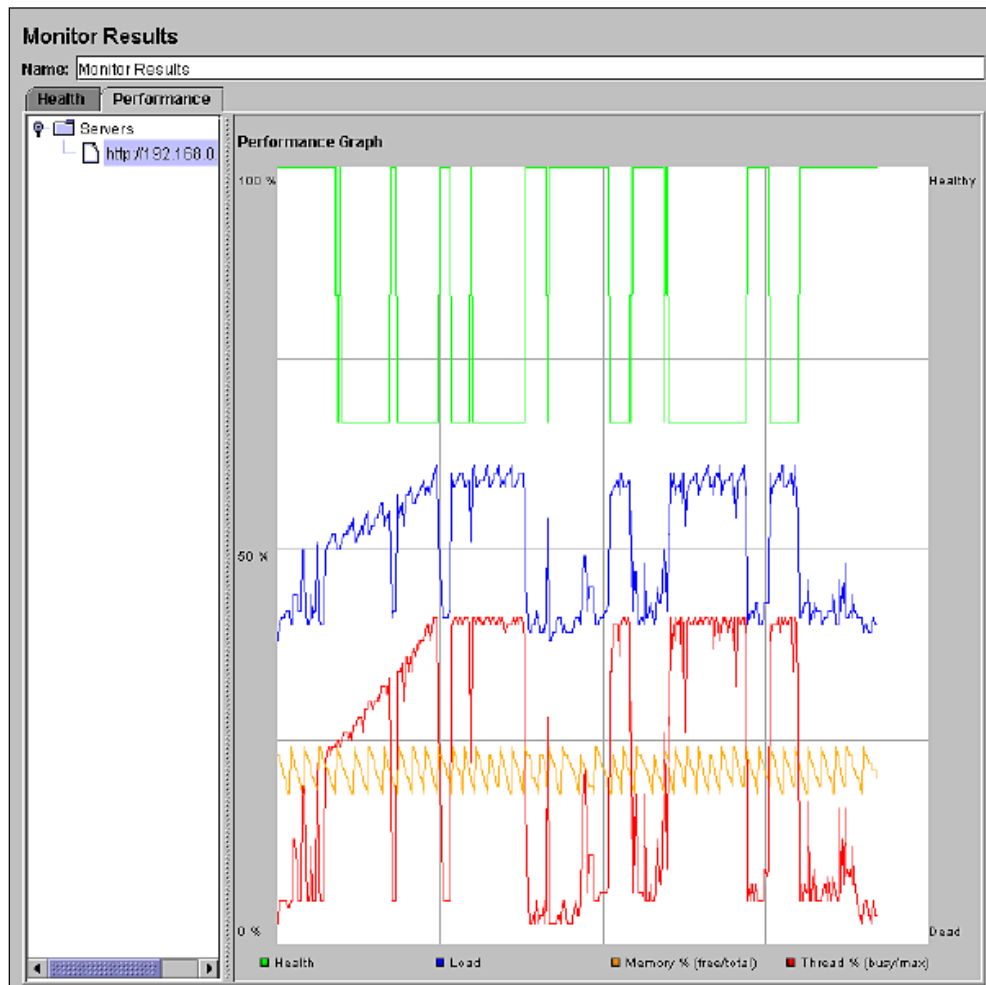
Monitoring the Server's Performance

There is a special Listener that allows you to monitor the target server's performance as Samplers make requests. However, this Monitor Result Listener is designed to work with Apache Tomcat, and only Apache Tomcat application server version 5 and above. You may refer to <http://jakarta.apache.org/jmeter/usermanual/build-monitor-test-plan.html> to find out more. For any other application server, you may use any other available open-source tools, or commercial tools.

The following are snapshots of the **Monitor Result** Listener, taken from JMeter's official user manual. The **Health** tab shows the general status of one or more servers.



This snapshot from the **Performance** tab of this Listener displays the specific performance indicators for the selected server affected by a load test for the last 1000 samples. These graphs indicate the general health of the server (green), the load capacity that the server was able to process (blue), ratio of server memory being utilized – free vs. total memory (yellow), and thread ratio capacity (red).



Summary

This test helps us to find out if the performance goals and/or SLA are reached given the total threads and scenarios. As we highlight the key pages of the website/application, JMeter running our Test Plan allows us to use the mean or median response time, depending on the type of data distribution, to approximate how fast the target server responds to concurrent requests. If the target server is Tomcat 5.0 or above, then you can easily monitor the server's general health in terms of its computing resources, such as memory use, workload, etc. As you explore JMeter's capability for remote testing, you can conveniently extend your Test Plan to support stress testing purposes as well. The following chapter will also make use of the Test Plan that we have just built to support functional testing — a real time-saver.