Published on **dev2dev** (http://dev2dev.bea.com/)
http://dev2dev.bea.com/pub/a/2006/08/jmeter-performance-testing.html
See this if you're having trouble printing code examples

Using JMeter to Performance Test Web Services

by Dmitri Nevedrov
08/02/2006

Abstract

Performance testing is an important part of any distributed or Web application testing plan. Inclusion of performance estimates into planning and development cycles ensures that the application delivered to a customer satisfies high load, availability and scalability requirements. Early identification of software load limitations helps to configure the system appropriately to avoid unexpected crashes. Several questions should be addressed at system performance analysis: will the system or server be able to process simultaneous requests coming from hundreds, or thousands of clients, and, what is the frequency of requests the system can handle. This type of test not only provides an absolute measure of system response time, but also targets the regressions on server and application code, examines if the response from the server matches the expected result, and helps to evaluate and compare middleware solutions from different vendors.

Apache JMeter—a performance testing framework from Apache, has been widely accepted as a performance testing tool for Web applications. It can be used to analyze overall server performance under simulated heavy load. The software features FTP and HTTP requests and extensible custom scripting features. In this article we show how JMeter can be used to load test Web services. In particular we demonstrate it by deploying a simple Web service on BEA WebLogic Server 9.0. The example test plan illustrates the creation of a test plan, thread group, loop, and a Web service request. We also discuss how to measure the data and interpret results displayed on graphical tools provided with JMeter chart.

JMeter

Apache JMeter is a tool that can be used to test applications utilizing HTTP or FTP servers. It is Java based and is highly extensible through a provided API. A typical JMeter test involves creating a loop and a thread group. The loop simulates sequential requests to the server with a preset delay. A thread group is designed to simulate a concurrent load. JMeter provides a user interface. It also exposes an API that allows you to run JMeter-based tests from a Java application. To create a load test in JMeter build a test plan, which is essentially a sequence of operations JMeter will execute. The simplest test plan normally includes the following elements:

- Thread group - These elements are used to specify number of running threads and a ramp-up period. Each thread simulates a user and the ramp-up period specifies the time to create all the threads. For example with 5 threads and 10 seconds of ramp-up time, it will take 2 seconds between each thread creation. The loop count defines the running time for a thread. The scheduler also allows you to set the start and end of the run time.
- Samplers - These elements are configurable requests to the server HTTP, FTP, or LDAP requests. This tutorial focuses on the Web service requests only.
- Listeners - These elements are used to post process request data. For example, you can save data to a file or illustrate the results with a chart. At the moment the JMeter chart does not provide many configuration options; however it is extensible and it is always possible to add an extra visualization or

data processing module.

A more detailed description of the available elements is given on the Apache JMeter Web site. In some cases, when the available elements are not suitable for a particular test, a developer can write his or her own script or Java class and embed it into a test plan by placing a jar file into the JMeter installation `\lib\ext\` directory.

In this article we use JMeter version 2.1. Download a binary executable from the Web site, unzip it, and the application is ready for use on Windows or Unix platforms. Go to the `bin` folder and start the application with `jmeter.bat` or `jmeterw.bat` if you work on the Windows operating system. The initial user interface is shown in Figure 1.

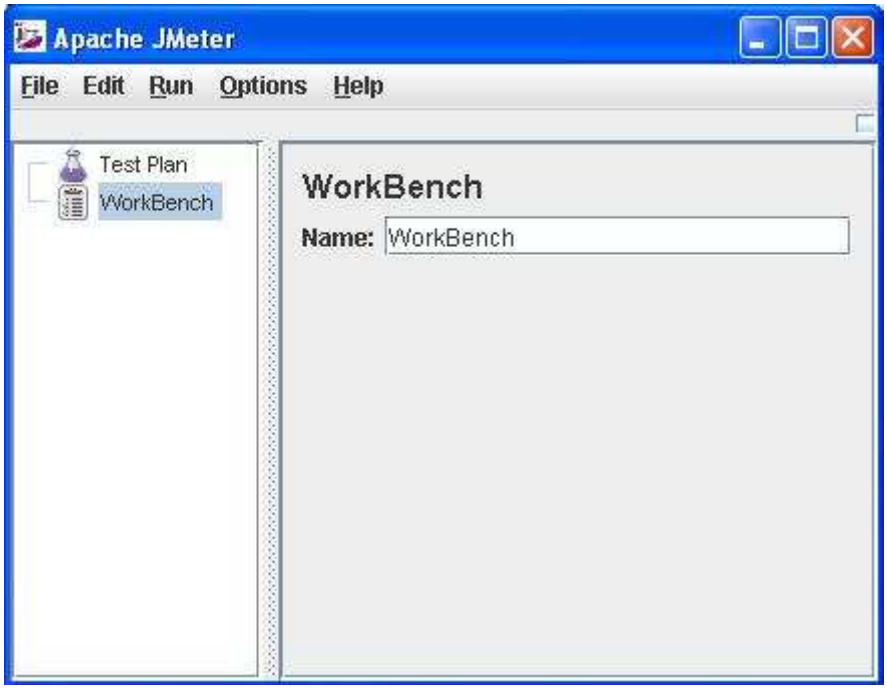

Figure 1. Starting Apache JMeter

Creating a load test

The JMeter load test feature allows you to generate a high load on a server and determine its capacity and limitations. Note: To use the Web services samples you also need the `mail.jar` and `activation.jar` available from Sun Microsystems (see the links below). Due to licensing restriction Apache does not distribute these libraries. After downloading these two jars, place them into the Java classpath or into the JMeter installation `lib` directory.

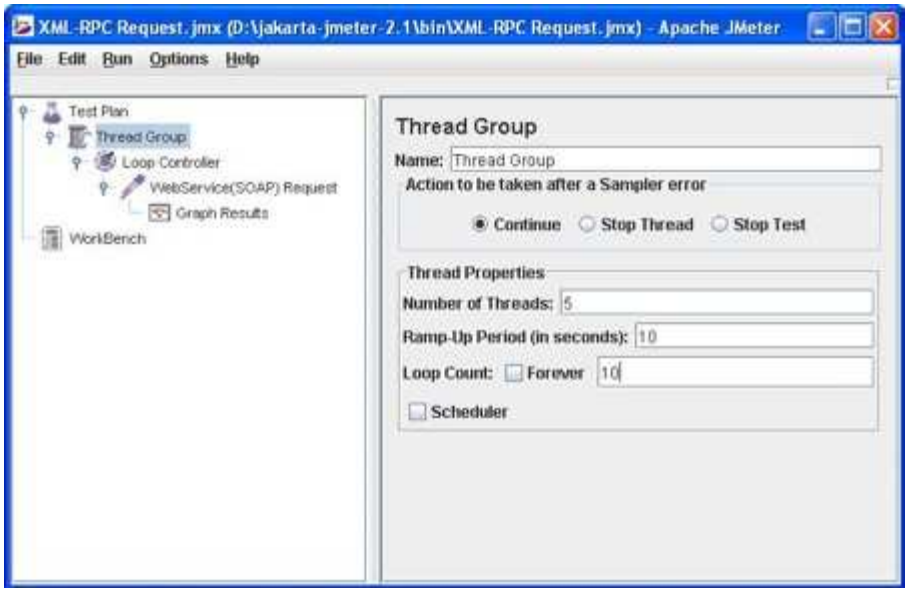

Figure 2. Creating thread groups and a basic Web service test plan.

Now, right-click Test Plan and add a Thread Group and a Loop Controller. We use these two elements to set the number of simulated concurrent users and duration of the test. In a tree structure, under Loop Controller, add a "WebService (SOAP) Request" and a Graph, as shown in Figure 2. If you cannot add a WebService request to the test plan, you probably don't have either `mail.jar` or `activation.jar` in your path.

Type in the number of threads, ramp-up periods, and loop count. For this tutorial we used 5, 10, and 100 respectively. Set the loop controller count to 1. If the WebLogic Server with the deployed Web service is not running yet, start it manually or from within WebLogic Workshop.

Configuring a load test

As shown in Figure 3, we need to set the parameters of the SOAP request sent to the server. If the link (URL) to the WSDL file is available, paste this link into the WSDL URL field and click Load WSDL. The available methods will be displayed in the Web Methods combo box. Next, you need to click Configure, so that Server Name or IP, Port Number, Path, and SOAPAction get populated.

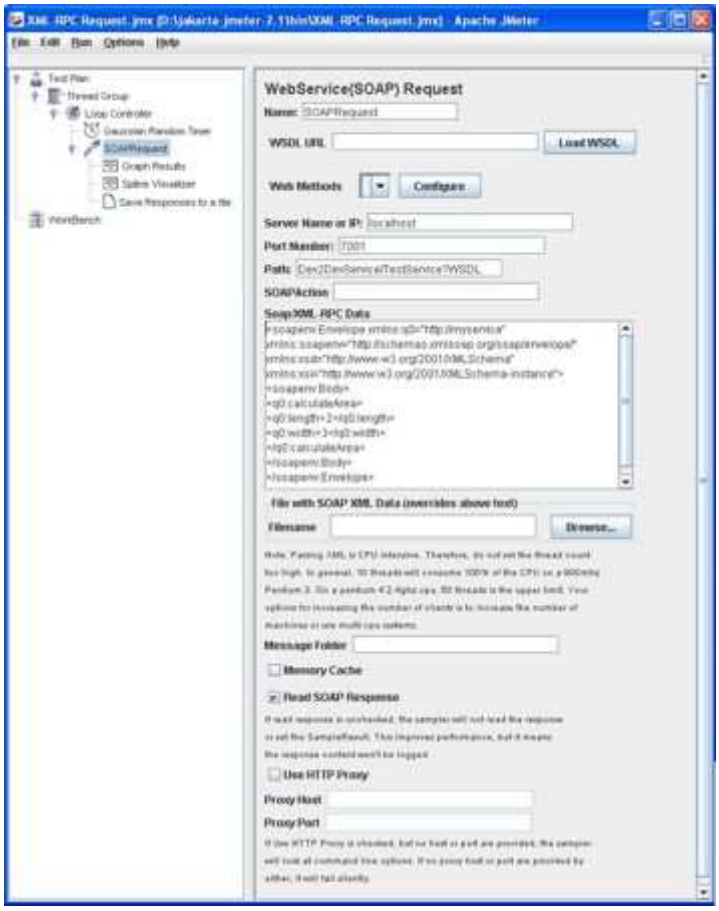

Figure 3. WebService(SOAP) request dialog (click the image for a full-size screen shot)

When no WSDL link is available, you can also manually type in values for server name, port number, path, and SOAP action. As a last step, fill in the SOAP request in the SOAP/XML-RPC Data area. You can also load it from a file using the File with SOAP XML Data option.

After all the fields in the Web service request dialog have been entered, save the JMeter project clicking Ctrl+S. To visualize the data, we added chart elements "Graph results" and "Spline Visualizer." In addition we can also send the responses to a file by adding a "Save responses to a file" element; this is helpful when examining errors in SOAP responses. To simulate a more realistic sequence of client requests we inserted a timer element in the tree—a "Gaussian Random timer." By doing this the client requests have a more chaotic distribution and will hit the server at random rather than at equal intervals. We set the Gaussian random times to deviation of 100ms and constant delay offset - 300ms.

Running the load test

You can run the load test by either clicking Ctrl+R or choosing Run from the menu and clicking Start. Click the graph element and you will see that the chart is being populated with the data representing requests to the server, as shown on Figures 4 to 6.

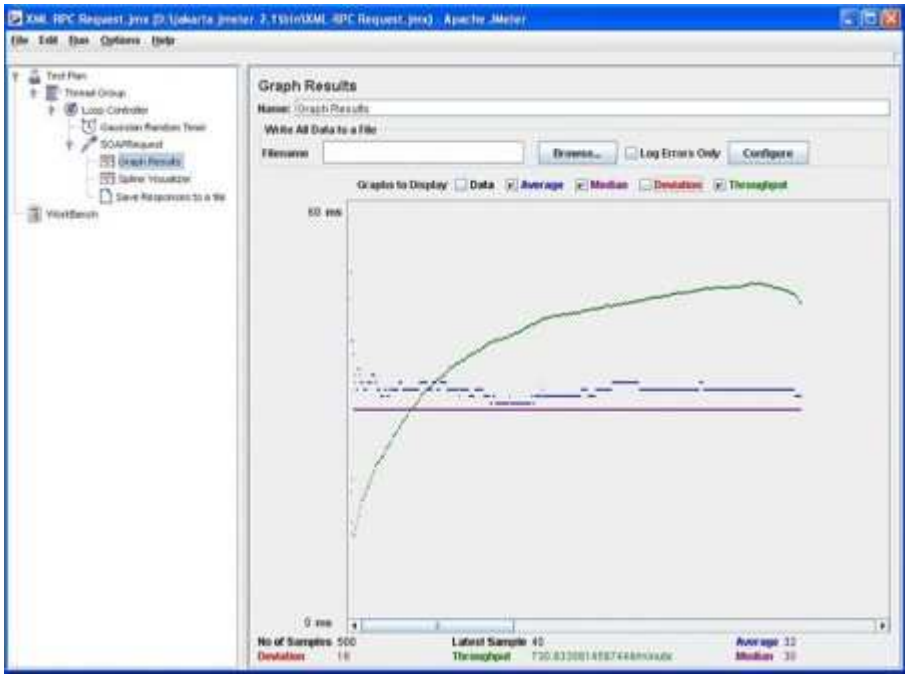

Figure 4. Chart illustrating test results with number of threads = 5, ramp-up time = 10, loop count = 100, and loop controller set to 1 (click the image for a full-size screen shot)

We chose to display the three parameters on the charts - **throughput** (green line), **median** (purple line) and **average** (blue line). Let us modify the parameters of the test. To simulate a higher load on a server, let us increase the number of threads to 10 and 50 and compare server response time. Click Start again and observe the results displayed on a chart; see Figures 5 and 6.

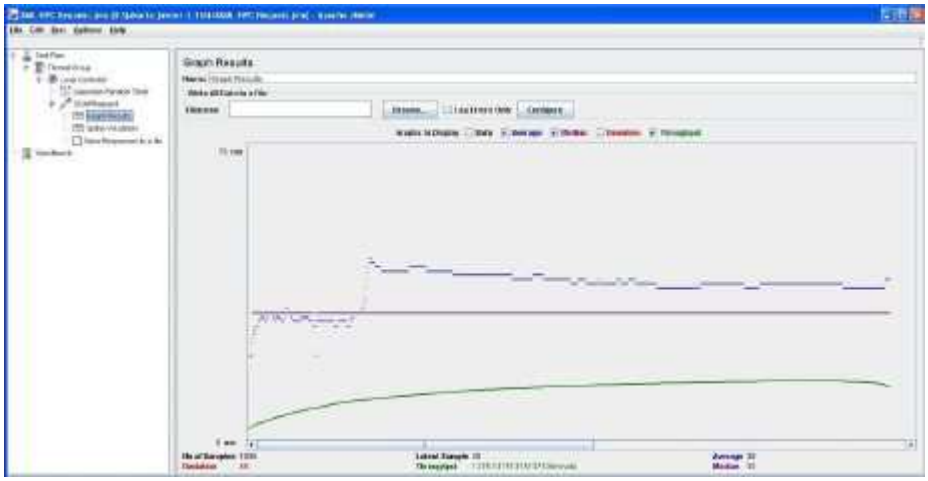

Figure 5. Chart illustrating test results with number of threads = 10, ramp-up time = 5, loop count = 100, and loop controller set to 1 (click the image for a full-size screen shot)

Figure 6. Chart illustrating test results with number of threads = 50, ramp-up time = 5, loop count = 20, and loop controller set to 1 (click the image for a full-size screen shot)

The parameters at the bottom of the chart have the following meaning:

- Throughput is the number of requests per minute the server has processed.
- Average is the total time running divided by number of requests sent to the server.
- Median is the number that represents the time, where half of server response time is lower than this number and half is higher.
- Deviation shows how much the server response time varies, a measure of degree of dispersion, or, in other words, how spread the data are.
- Latest sample is just the last request completed.

Just looking at these three runs and their corresponding charts we have the following valuable results:

Number of threads	**Throughput, *responses/minute***	**Average, *ms***	**Median, *ms***
5	731	30	32
10	1375	38	30
50	3479	115	100

The increase in response time with the addition of threads is obvious. To continue testing we can change the number of threads, ramp-up period, and loop count. Note that we have not changed or adjusted configuration settings of the server. WebLogic Server 9.0 has an automatically configurable thread pool with configurable constraints (see Workload Management in WebLogic Server 9.0 by Naresh Revanuru, Dev2Dev, July 2006), and the table above shows that increasing the number of clients has a nonlinear impact on server response times. In fact, increasing this number two or ten times does not make a significant impact! It would, however, be interesting to run the same experiments with the number of threads close to or exceeding the default server constraints. To verify that the response received from the server is an actual SOAP response and not an HTTP error let us look at the content of the output file. This is a SOAP response that corresponds to the request shown above:

```
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <m:calculateAreaResponse xmlns:m="http://myservice">
      <m:return>12</m:return>
    </m:calculateAreaResponse>
  </soapenv:Body>
```

```
</soapenv:Envelope>
```

To further visualize the test results, add the Spline Visualizer into the plan, right after Graph Results. The Spline Visualizer provides a view of all sample times. It is constructed as a continuous line, a piecewise interpolating function, and is drawn across 10 points where each point represents 10 percent of samples. Instead of connecting points with straight lines, spline charts provide a smoothed view of distribution based on polynomial approximations. The result is shown in Figure 7

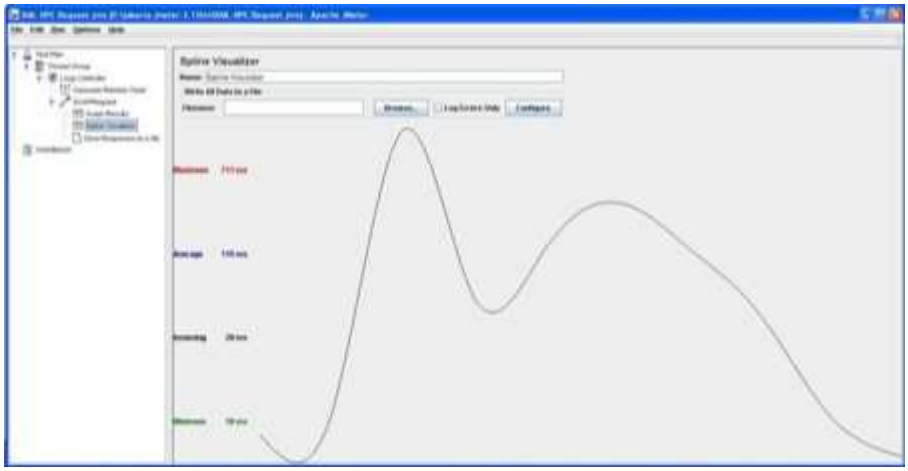

Figure 7. Chart illustrating the same test results as shown in Figure 6 using Spline Visualizer (click the image for a full-size screen shot)

Multiple JMeters

JMeter also has a useful Remote Start feature, which allows you to launch JMeter tests from multiple machines. The client host addresses can be entered in a "jmeter.properties" file located in the `bin` folder. Find the `remote_hosts` attribute and add there the names of remote hosts separated by commas. Restart JMeter and click either Remote Start or Remote Start All from the Run menu. Some Web and application servers treat multiple requests coming from the same IP address sequentially, and requests coming from different IP addresses in parallel, therefore if it is critical to have requests coming from different machines or distribute the test load among several clients you can use this option to do so.

Other Features

As mentioned, JMeter has HTTP, FTP, and LDAP samplers. Creating these requests is a straightforward task and is well described in the JMeter manual; it usually involves creating a Thread Group, adding the sampler and the timer and the listener. We would like to mention some of the features that go beyond these standard samplers and that may need some non-standard configuration steps.

Scripting

Custom tests and scripting, particularly support for Java requests and BeanShell scripts, is another feature of JMeter that gives more flexibility to the load test developer. We can create a scripted test and compile it as a Java class and pass it to JMeter to run. To use the BeanShell scripting feature it, you need to download the BeanShell jar and place it under the /lib directory so JMeter will be able to pick it up at runtime. You can use the beanshell API from Java sampler, or create a BeanShell sampler that either reads the script from a file or processes the commands typed in a text box. The code sample below illustrates the BeanShell assertion added to an HTTP sampler. You can analyze and control the test execution based on the response from an HTTP

request. A good scenario would be to create a test plan with a thread group and a loop starting the load test at a scheduled time. The request will hit the Web server and pick up its response:

```
print("HTTP return code is: " + ResponseCode);
print("HTTP return message is: " + ResponseMessage);
if (SampleResult.isSuccessful())
{
     print("Success");
     SampleResult.setStopTest(true);
     SampleResult.setStopThread(true);
}
```

The response can be logged with BeanShell assertions added to the test plan right after the HTTP sampler, and if the Web server response matches certain criteria, that is, if it is successful or it contains an expected string, the script will either stop the test or thread or continue execution.

JMS applications

To use JMeter to load test JMS applications, download the ActiveMQ jar and copy it into the /lib directory. As in the previous test first we create a thread group and add a JMS Point-to-Point sample. The JMS Point-to-Point dialog needs to be populated with the following parameters: QueueConnection Factory, JNDI Name Request queue, and JNDI Name Receive queue. For example we can use the parameters from an installed JMS module on the WebLogic examples server; in this case they are `weblogic.examples.jms.QueueConnectionFactory` and `weblogic.examples.jms.exampleQueue`. We can also create a custom JMS connection factory and queue. For WebLogic server we also need to add `weblogic.jndi.WLInitialContextFactory` as an Initial Context Factory value and the Provider URL, which normally looks like `t3://hostname:7001`, for example `t3://localhost:7001`.

Custom samplers

Many test developers sooner or later face the question: What if I have to create a test plan and JMeter does not provide necessary samplers or listeners. For example, you may need to test Enterprise JavaBeans applications. You can still use JMeter's Java Request, which essentially is a Java class to which we can add any logic we need. Let's create a simple Java sampler that will send the request to the application server and call the session bean. First, let us create a package `mytest` and a Java class called `EJBTest.java`:

```
package mytest;

import mybeans.LoginBeanRemoteHome;
import mybeans.LoginBeanRemote;

import java.io.Serializable;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
import java.util.Hashtable;
import javax.naming.*;
import javax.rmi.*;
import java.rmi.RemoteException;
import javax.ejb.CreateException;

import org.apache.jmeter.protocol.java.sampler.*;
import org.apache.jmeter.config.Arguments;
import org.apache.jmeter.protocol.java.sampler.AbstractJavaSamplerClient;
import org.apache.jmeter.protocol.java.sampler.JavaSamplerContext;
import org.apache.jmeter.samplers.SampleResult;
import org.apache.jmeter.testelement.TestElement;

public class EJBTest implements JavaSamplerClient ,
```

```
                    Serializable {

  public EJBTest() {
  }
  public void setupTest(JavaSamplerContext context) {
  }
  public Arguments getDefaultParameters() {
    Arguments params = new Arguments();
    return params;
  }
  public SampleResult runTest(JavaSamplerContext context) {
    SampleResult results = new SampleResult();
    try{
      Context env = getInitialContext("t3://localhost:7001");
      LoginBeanRemoteHome home = (LoginBeanRemoteHome)
      PortableRemoteObject.narrow(
      env.lookup("ejb.LoginBeanRemoteHome"),
        LoginBeanRemoteHome.class);
        LoginBeanRemote bean = (LoginBeanRemote) home.create();
        bean.login("TestUser","TestPassword");
        results.setSuccessful(true);
    }
    catch (NamingException ne){
      ne.printStackTrace();
    results.setSuccessful(false);
    }
    catch (RemoteException re){
      re.printStackTrace();
    results.setSuccessful(false);
    }
    catch (CreateException ce){
      ce.printStackTrace();
    results.setSuccessful(false);
    }
    return results;
  }
  static Context getInitialContext(String url) throws NamingException {
    Hashtable env = new Hashtable<String,String>();
    env.put(Context.INITIAL_CONTEXT_FACTORY,
      "weblogic.jndi.WLInitialContextFactory");
        env.put(Context.PROVIDER_URL, url);
    return new InitialContext(env);
  }
  public void teardownTest(JavaSamplerContext context) {
  }
}
```

To use this code, compile it to create a jar file `myEJBTest.jar`, copy the jar to the JMeter classpath (such as the lib directory), or add the classpath location to the `jmeter.properties` file - something like:

```
user.classpath=d:\jakarta-jmeter-2.1.1\lib\ext\myEJBTest.jar
```

Restart JMeter, add a Java Request element to a test plan, and choose `mytest.EJBTest` from the dropdown menu. By default JMeter provides a "SleepTest" and "JavaTest." Now we can add the thread group, loop, and listeners to our test plan and run it.

Testing databases

Evaluating database server performance is another feature supported by JMeter. You have a choice of using the JDBC Request element provided by the software or of creating your own test using either a script or Java class. For example, you could test calls to stored procedures using something similar to the examples shown before. Many database-tuning techniques are available that include using vendor-specific optimization such

as parallel processing of queries, using joins, or indexing. Moreover, knowledge of data organization is useful when creating queries with multiple boolean evaluations. Database and query tuning are particularly critical in applications dealing with large amounts of data, and, JMeter is a tool that is capable of providing some metrics in such evaluations. For example, you can execute performance or load tests before and after data or query optimization and compare results.

Let us demonstrate how the simple database performance measurement plan works by creating a test plan for a MySQL instance. First, download a JDBC driver from the MySQL Web site and copy it into `/lib` directory so JMeter is able to access a database. Now start JMeter, create a thread group, and set the loop count and the number of threads accordingly. Add JDBC Connection Configuration, JDBC Query Defaults, and JDBC Request elements. In these dialogs we need to configure the database connectivity and query. Enter the database URL value, which will look something like `jdbc:mysql://hostname/databaseName`, and enter `com.mysql.jdbc.Driver` for the JDBC Driver Class. For our experiment a database `customers` was created with a table `customer` that has three fields: `name`, `address`, and `account`. Type in a query in a Query box, for example `select name from customer`.

To visualize the results you may add either a chart with a response time or a response assertion if you want to verify that the response matches a certain pattern. For example, in Response Assertion add a text pattern "Smith." The Assertion Result window will show an error such as "Test failed, text expected to contain /Smith/," or "Response was null" if we have no connection due to errors in settings. Nothing will be returned if the test was successful. In addition to chart and assertion listeners we may use an Aggregate Report that summarizes the number of samples, average, median, throughput in a table.

Summary

JMeter is a flexible tool that not only allows you to test the HTTP servers but also to load-test Web services. A skilled developer can write their own scripts to simulate or customize the client requests or add a customized visualization of test results. Web service and SOAP samplers are new features of JMeter; hopefully they will evolve as Web services get a wider acceptance in industry and among developers.

Using the existing functionality of JMeter and its provided user interface we were able to simulate a load of 5 concurrent threads hitting the server with 10 and 5 ms delays. This tutorial demonstrated how the tool can be used for measuring the response time of Web services. Utilizing JMeter scripting, for example, we can address client authentication and authorization. Both the tested application server and JMeter client were physically installed on the same workstation, however the same setup can be used for testing remote servers.

The results and charts presented do not yet provide a valuable performance report; they only illustrate a plain distribution of WebLogic Server response time to Web service requests. To get an idea of whether an optimization was achieved, the same load test should be run before and after optimization, or against two different servers, or using different loads with a variable number of clients-threads. In other words, only the comparative results have value and help to understand server performance which then allows conclusions based on these metrics.

References

- View the Jakarta Apache JMeter - binary distribution download and tutorial.
- Visit the WebLogic Server Product Center on Dev2Dev.
- Read the Programming Web Services for WebLogic Server tutorial (product documentation) for more information on building Web services.
- Approaches to Performance Testing by Matt Maccaux (December 2005, Dev2Dev) is a great introduction to testing.

Dmitri Nevedrov works as a software engineer (at BEA systems) and is based in Denver, Colorado.

Return to dev2dev.