IBM Tivoli Service Management Products

Performance Test Best Practices With HP LoadRunner

White Paper

Document version 1.1

NOTICES

The information contained in this document is distributed AS IS, without warranty of any kind.

This paper presents best practices for performance testing IBM Tivoli service management products by using HP LoadRunner.

This document applies to the following IBM Tivoli service management products and releases:

- IBM Maximo Asset Management 7.1 and 7.5
- IBM Tivoli Change and Configuration Management Database 7.1 and 7.2
- IBM Tivoli Provisioning Manager 7.1 and 7.2
- IBM Tivoli Service Automation Manager 7.1 and 7.2
- IBM Tivoli Service Request Manager 7.1 and 7.2

CONTENTS

1 Importance of Performance Testing IBM Tivoli Service Management Products

A common misconception is that performance testing of IBM Tivoli service management products in a customer environment is not necessary, because the product has been thoroughly tested by the IBM Tivoli Performance and Scalability teams, and the product is initially configured to provide optimal performance when it is installed.

The problem with that misconception is that internal tests performed by a development organization might not reflect conditions experienced in a production environment.

Internal tests focus on discovering defects, determining whether the product is scalable to accommodate the largest organizations, and determining overall performance of each of the applications. Customers who deploy IBM Tivoli service management products in a production environment should not exclusively rely on the results of internal testing. Instead, they should use the internal results as a guideline when determining the optimal configuration needed to support the mix of applications, number of JVMs, number of users, and number of transactions that are typical of the actual production environment.

This document provides guidelines for conducting performance tests of IBM Tivoli service management products in a customer environment.

Four common reasons to conduct your own performance tests are:

- To validate sizing estimates for a new IBM Tivoli service management products implementation.
- To ensure that IBM Tivoli service management products meet the current requirements of the business processes and to document the performance of the existing system.
- To ensure that IBM Tivoli service management products will meet the business demands for future growth and to assist the organization with predicting performance bottlenecks.
- To analyze, tune and debug current performance issues in production environments.

2 Defining Performance Benchmark Test Objectives

A well-defined set of performance test objectives is critical to the success of any performance test project. Careful planning and consideration of all factors should be the first step of all projects. The most common reason for failed projects is that too little time is spent on planning.

Define the test objectives with a set of business questions that are expected to be answered by the results of the tests. All further details of test design should map back to answering a specific business question.

The following are examples of some business questions that you should address for a new deployment:

- Will the architecture being deployed meet our business requirements?
- Will IBM Tivoli service management products provide a satisfactory response time, given your own requirements for a certain number of concurrent users, performing a given number of transactions in a period of time, on the hardware that you currently have in place?
- Given the combination of IBM Tivoli service management products that you intend to run, and given the transaction volume and concurrent users that you expect, which component is likely to be a bottleneck to overall performance?
- Is your existing hardware sufficient to provide acceptable performance, given the expected number of transactions and concurrent users, or is new hardware or reconfiguration of existing hardware necessary?

If you are planning a new installation, or if you expect changes to your current IBM Tivoli service management products environment, such as an increase in the number of users, you should load test the entire application. Stress and stability tests are performed before you configure the product for a production environment. However, this type of testing is not always possible because of time, cost, or other constraints. Consequently, it is critical that you identify the business questions with the highest risks and rewards and test accordingly.

Document the test objectives, including the business questions to address, in a project test plan. When testing is complete, you should be able to provide concise answers to each of the business questions in a test report.

3 Defining Test Methodology

3.1 Defining Test Types

The type of test you perform should be determined by the business questions you ask. The following test types are commonly performed during IBM Tivoli service management products performance test projects, but your unique business questions might require additional testing.

3.1.1 Baseline Measurements

Baseline measurements should be performed before any other tests are run. Baseline measurements consist of a single user test that you run for several iterations. The purpose of this type of test is to verify that the test is working as expected, and to allow you to identify aspects of an application that are not performing well, even when only one user is using it. If an application does not meet your requirements in this situation, obviously the problems must be resolved before you attempt additional tests with more users. You might have to reconfigure both the hardware and the software to eliminate the failures.

Once a set of single user baseline measurements has been completed with satisfactory results, a “benchmark-under-load” test is performed. In this type of test, your test script should simulate a subset, such as 25%, of the total activity that you expect to see in the production environment. Like the initial test, any problems with the performance of the application must be resolved before you conduct additional tests with higher load levels.

There are certain situations where it is advantageous to schedule a scenario with a single functional flow. One situation is when the performance testing coincides with a delivery of a functionally stable customization and you want to see how this particular function performs before it is included in a workload mix. Another situation is when performance issues are identified during the mixed mode scenario. In order to isolate where the performance issues are in a specific functional flow, you might need to run single functional flow tests.

3.1.2 Performance Load Tests

After the baseline and benchmark-under-load tests are completed, the full suite of load tests can begin. For these load tests, the Performance and Scalability team suggests that you run at least 5 load points, where the tests simulate 50%, 75%, 100%, 125%, and 150% of the expected system load. By doing this, the knee of the performance curve can be clearly identified. You can test each load point as an individual test, or use the performance test tool scheduling options to drive all 5 load points from a single test that presents multiple load levels. There are advantages and disadvantages to each method. Each load point should be allowed to plateau and remain at a constant user load/transaction rate for a period long enough to make clear observations before scaling to the next load point.

The objective of the load test is to measure the response times of transactions and verify that business requirements and service level agreements are met under load conditions. Another result of load tests is that they can expose defects in functionality that might not otherwise be revealed. By monitoring the success and failure rate of transactions, you can determine if the application is performing as expected and identify bottlenecks that may cause functional issues.

The design of your load tests is crucial to understanding test results and tuning the environment for optimal performance. Poorly defined workloads and test approaches can have dramatic effects on test results and lead you to false conclusions.

3.1.3 Stress Tests

Load tests focus on testing the performance of software using simulations that approximate conditions of real users using the application. Stress tests focus on identifying the server break points and how the application behaves when a failure occurs. Customized deployments of IBM Tivoli service management products are good candidates for stress tests.

For this type of testing, you gradually increase the number of concurrent users until the server breaks. Unlike load testing, you do not need to consider “think time” or the browser cache during stress testing. The duration of the stress test is not predetermined, because you increase concurrent users gradually to a point where all server resources are exhausted. You identify the conditions that caused the failure by monitoring the server resources during stress testing and analyzing system logs after a failure. Stress test results indicate whether the system remains stable under extreme loads or whether its performance degrades or the system crashes.

Detailed result analysis with project architects and implementers should be routine with stress tests due to the potential significant impacts to customer environments when a system becomes stressed.

3.1.4 Endurance Tests

Endurance testing differs from load and stress testing in that endurance tests examine how the application performs when running a significant number of concurrent users for extended periods of time. The purpose of endurance tests is to identify memory leaks, performance degradation over time, and overall system stability. It is common for these tests to be executed over a period of time, ranging from 12 hours to 1 week.

Key indicators to monitor during an endurance test are consistent transaction response times, stable memory usage, stable CPU utilization, constant throughput levels, and server crash avoidance.

3.1.5 Sizing/Capacity Tests

Sizing and capacity tests are important when the results of the tests will be needed by deployment teams to properly size new deployments and to determine the capability of existing environments to absorb anticipated increases to system load.

For IBM Tivoli service management products, the results of sizing tests are commonly reported in concurrent-users-per-JVM, and as transaction rates per server.

Sizing and capacity tests are designed differently from the standard performance tests. It is important to design series of tests that focus on each tier as well as the individual components of those tiers. For example, to determine the maximum concurrent users per JVM for a given workload, it would be necessary to ensure that the JVM itself will be the limiting factor. One way to achieve this is to restrict the number of JVMs on a system so that the JVM will reach capacity well before the CPU reaches its limitations. Once the JVM capabilities have been determined, the JVM count should be increased so that it is no longer a bottleneck, but the CPU could be. A continual increase in concurrent users should be staged to the point that the CPU reaches 100% utilization. At that point, the maximum number of concurrent users per CPU can be reported. This approach should be continued until all components of each tier have been fully characterized.

In addition to the components and tiers, the various combinations of IBM Tivoli service management products can be sized by following the same methodology. Once characteristics of each component within each tier of each application have been established, integrated testing of various applications should be executed to add another dimension to the results. This will provide additional data that can be used to estimate sizing "rules of thumb" as combinations of applications are expected to be deployed.

3.1.6 Performance Tuning/Debugging

Through creative manipulation of the hardware resources on the IBM Tivoli service management products system, bottlenecks can be controlled. For example, if JVM memory is determined to be a bottleneck, it might be easier and faster to debug and tune if you only start a single JVM in a server cluster. That way, a smaller load can recreate the problem.

If one particular use case is not performing as expected, it is often useful to test that use case by itself under load to gain insight into the resources consumed. By weeding out the other traffic, it can be much easier to review traces and logs.

For details about logging, tracing, and debugging, refer to the *System Administrator Guide*.

For the latest IBM Tivoli service management products tuning information, refer to the *Best Practices for System Performance* white paper.

3.1.7 Batch Testing

Batch testing refers to testing the components of IBM Tivoli service management products that are non-interactive. Batch testing plays a vital role in successful IBM Tivoli service management products performance analysis and it should not be excluded. Data loads during customer deployments, integrations with outside vendors, and use of the integration framework can significantly affect both the middleware and database servers.

These tests focus heavily on system resource consumption of the server tiers and throughput rates. However, response time Service Level Agreements (SLAs) should also be considered if required. The type of records to simulate, size of messages, and transaction feed rates should be clearly defined in the test approach because mistakes in design can vary results to a significant degree.

Batch tests typically do not use the standard performance simulation tools and typically require some level of programming to develop. It is important to work with an implementer, architect, or someone that is familiar with the components to be simulated.

4 Test Cases, Workloads, and Scenarios

4.1 Test Case Development

When considering test cases for the performance tests, the goal is to keep the use cases to as small a number as possible, while ensuring that they include the most frequently used functions or the functions that present the most risk. Remember that the goal of the performance test is different from functional testing. It is not possible to conduct performance tests with the level of coverage found in functional testing, because producing performance test scripts from the use cases can be time consuming and labor intensive. A guideline is to concentrate on testing 20% of the scenarios that are most frequently accessed. Expending effort on the remaining 80% of scenarios does not typically bring considerable performance improvement.

If this is not the first performance test for the application, you should use the previous performance benchmarks and existing production data as input for comparing results from release to release.

Logons and logoffs should be structured in the scripts to match how they occur in a production environment. If the end user typically logs on once in the morning and then maintains that connection throughout the day, then the scripts should be recorded to reflect that behavior. If the end user typically logs on, performs some work, and then logs off, the script should reflect that behavior. If a specific logon/logoff use case is wanted, add a script specifically for that so those transactions can be controlled and measured by the workload mix.

4.2 Workloads

Developing a valid workload can be a challenge but can usually be determined from a combination of historical data from production systems, database queries of live systems, and interviews with application support personnel and end users. The workload is comprised of two elements: the workload distribution and the workload rate. The workload distribution is the percentage mix of all use cases in the test scenario. The workload rate is determined by transaction rates: for example, as ‘x’ number of transactions, or as a given type created per month, and then estimated down to the peak hour.

It is not enough to know that 20% of your users are creating work orders. You also need to know the rate at which work orders are created. For example, are 20 created per hour, or 100?

Knowing the correct figures of simultaneous and concurrent users is vital to perform performance testing of an application.

Scenario	Scenario description	Weight factor
S01	Simple Search	20%
S02	Advanced Search	10%
S03	Simple Create	30%
S04	Advanced Create	20%
S05	Simple Close	10%
S06	Advanced Close	10%
		100%

Sample workload distribution

Average Hourly Transaction Rates	Peak Hour
S01 – Simple Search	2000
S02 – Advanced Search	650
S03 – Simple Create	900
S04 – Advanced Create	750
S05 – Simple Close	1700
S06 – Advanced Close	2000
Totals	**8000**

Sample workload transaction rates

It is possible to have multiple workloads during a test project, depending on the test type and objectives of the test effort. The most common workload type simulates the concurrent users and transactions for a peak hour during the target production environment.

When building the test scenarios for execution, the number of concurrent users to be simulated should follow the workload mix that has been defined. Transaction rates per hour during the simulation should correspond with the analysis done to derive the workload mix. These rates can be reported as transactions per second, minute, or hour per user.

Concurrent and Simultaneous Users

There are two distinctions to make when reporting users. Concurrent users are the number of users that have an active, but possibly idle, session with the application. Simultaneous users are those actually doing work at a given time. This classification of users is important when analyzing results.

The terms "concurrent users" and "simultaneous users" are not tightly defined terms. If you are told that a web site had 150 concurrent users, you should not assume that means 150 requests were being processed all at once. Instead, it means that 150 users were logged on and actively interacting with pages on that web site. In reality, many of them could be reading the page, thinking, or typing responses, and thus would not be loading the server simultaneously.

Real-time and Virtual User Mapping

It is important to map simulated virtual users to the equivalent real-time users to provide realistic load levels. For applications that have the real-time usage history (for example, a version of the application is already in production), the server load (requests handled per second) during the peak traffic hour needs to be compared to the server load created on the server during the performance test. This helps to confirm that the load simulated during the performance test is realistic. The think times provided in the test scripts need to be adjusted to create the appropriate load on the server. This is explained more in the Think Time paragraphs below.

Analyzing the server load in terms of requests per second is recommended rather than the user load measured in number of users because it is a more granular and appropriate unit to measure the load handled by the server. The number of users handled by the server is a user-perceived metric that needs to be reported to the business stakeholders, whereas performance testers should focus on the number of requests handled by the server.

Think Time

"Think time" is the time taken by the users to think or to navigate to different pages in the application. Think time must be considered when deciding the test strategy.

Users will have different think times, depending on what part of the application they are using. When you run the load test, apply a randomization factor on the configured think time. Performance test tools provide a mechanism for randomizing think times during a run. Various tools use different terms, such as "think time", "delay time", and so on, to describe this pause between interactions.

Another method of introducing randomization consists of adding code to the top of each script's `Action()` section to wait a random amount of time before executing that particular iteration of the test. The Performance and Scalability teams use this technique and calculate the random think time so that the transactions per second/user can be controlled to provide the projected transaction rates over the test time.

Think time has an impact on the server load. Decreasing the think time will increase the server load. Increasing think time reduces server load. By reducing the think time, you can simulate the load of a large number of users, even though only a few users are used to run the test. In order to calculate the appropriate think times, you must understand the desired transactions per second, per user.

Also, make sure that think times are placed in the script outside of the start and end of the transactions being measured. If this is not done, the think times will be included in the response time results.

4.3 Test Scenarios

Test scenarios should be designed to start off using few system resources, then increase resource use, but at a stable level, and then taper down the use of resources. During the stable period, the target user load needs to perform various operations on the system with realistic think times. All the metrics measured should correspond only to the stable period and not during the ramp up and ramp down periods. Ensure that enough data samples have been gathered before making conclusions about the response time metrics, because there could be some reason for faster or slower response times at any single point of time. Using the 90th percentile response time to report the response time metrics is another guideline to follow to eliminate response-time spikes.

5 Benchmark Test Environment Considerations

Many components make up the performance test environment. Consider each component during the planning phase of the project. Having a mirror image of the production system would be ideal, but it is often not possible for many reasons. Therefore, special attention should be paid to each component to ensure that extrapolations from test environments can be applied to production environments. A comprehensive understanding of the intended production environment is a prerequisite to planning the test environment.

At a minimum, the test environment should have:

- The same overall architecture as the production environment, including:
    - The same OS and middleware platforms. For example, if the production system uses AIX and DB2, so should the test configuration.
    - Similar tiers, hardware proportions, and topology. For example, the same number of JVMs on the application server, the same distributed database and application servers, and so on.
    - The same configuration for the integration framework and cron settings. For example, using a dedicated server as opposed to dedicated JVMs on the same physical server that has JVMs being used for the user interface.
- Matching software stack versions (the same versions of the OS, the middleware, and IBM Tivoli service management products).
- Comparable data volumes and distributions in the database.
- Identical server configurations. For example, both environments use SSL, and both have the same maximo.properties file and security layers, such as policies, firewall rules, and so on.

If you do not closely match the characteristics of the production environment, you cannot accurately correlate results from the test environment to the production environment, and CPU and memory consumption could be different between the two environments.

Always identify the system configuration details of the server machines in the production and test environments, including the number of CPUs, CPU capacity (clock speed), RAM capacity, disk capacity, free space available on the disk, NIC card capacity, and network bandwidth. These details need to be identified before scheduling the performance test, and should be documented in the test plan document.

Having a dedicated testing environment is imperative to successful performance testing. Random application traffic or rogue processes can skew performance results and create confusion and variances in the tests. The dedicated resources should include both the server and network components. Network isolation is critical when the primary focus of the test includes bandwidth testing, because general traffic on the LAN (traffic that is not caused by the testing) can affect the test results.

When you configure the test environment for the first time, apply the best practices to the IBM Tivoli service management products and to the other hardware and software. Document any modifications you make during the testing process.

5.1 Application Servers

Unless the testing is focused on tuning or debugging a specific application, the standard load tests performed should include all of the applications and processes that will be used on the production system. This comprehensive type of test will be the only way to validate the overall performance expectations.

For the latest IBM Tivoli service management products tuning information, refer to the *Best Practices for System Performance* white paper.

5.2 Database Servers

When preparing for a performance test, you must have sufficient data in the test database. The execution of a SQL query could be significantly less when a database contains 1000 records as compared with 100000 records. Database performance in the production environment may be significantly worse than in the test environment if the production database has a higher volume of data than what was tested.

Ensure that a representative amount of data exists in the test database to allow for data usage during the tests. The way that table records are arranged and indexed will affect performance. Run database maintenance utilities before creating a database backup. Use the back up to restore to the same state before each test run to allow test repeatability.

For the latest IBM Tivoli service management products tuning information, refer to the *Best Practices for System Performance* white paper.

5.3 Build Administration Servers

During performance testing, you might want to adjust values in the properties files and build multiple EAR files to deploy for testing. Keep copies of any files you might change in case you need to fall back to an earlier configuration. Document the changes that you make.

5.4 Monitoring Considerations

Some monitoring tools can affect system performance. Be selective about what you monitor to minimize the effects of the monitoring tool on performance. If you set the sampling interval too small, that too can negatively affect performance; however, setting the interval too high will miss potential bottleneck situations.

6 LoadRunner Script Development Essentials

6.1.1 Load Simulation

Load testing tools such as LoadRunner are typically thought of as having two parts:

1. The load test controller, which is used to create and drive the required load.

2. The load test generators, which send the actual load to the server.

Depending on the number of virtual users you plan to simulate, the load test controller and the load test generators can reside on the same hardware. The LoadRunner documentation might provide guidance for when hardware should be dedicated or shared.

Load Test Controller

The load test controller is sometimes referred to as the test scheduler. When the load test controller is configured to simulate some number of virtual users, LoadRunner, depending on the configuration, will generate the same number of threads or processes to send the client requests to the server based on the think time defined in the scripts. Either the load test controller needs to have enough hardware resources (CPU and memory) to handle the running threads/processes, or a separate load test generator needs to be used.

Load Test Generators

A load test generator, sometimes called an agent or driver, can be installed on several systems to spread the load. To perform a 1000 user test, load generators can be installed on four machines, and each load generator could be configured to create 250 virtual users. The user load, created on four different IP addresses is then sent to the server under test. From the load test controller, the load generated by each load test generator can be collated when viewed.

Load Test Data

Depending on the complexity of the test data, data population could be done by the performance test engineer or by a database administrator. A database dump from a production system should be used, when possible, to populate the test database.

6.1.2 Recording a Script

6.1.2.1 Before Recording

Before recording your script, document the steps you will be taking. For example, after entering text into a filter, will you tab to the next field, press the Return key, or click the binocular icon? Define the names of your transactions so that when you execute the test in the controller and the results are processed by the analysis tool, the transaction names are displayed in the correct order. A useful format would be `<ScriptName>_<StepNumber>_<StepDescription>`. For example, `AM01_01_Login` or `AM01_05_EnterStatusWAPPR`.

6.1.2.2 VuGen Settings

After opening the LoadRunner Virtual User Generator, select File > New. When the New Virtual User dialog appears, select New Single Protocol Script from the left pane, select Popular Protocols from the Category pull-down, select Web (HTTP/HTML), and then click OK. When the Start Recording dialog appears, click the Options button, then select Recording from the left pane. Select HTML-based script for the HTTP / HTML Level, then click the HTML Advanced button. Then select **Recording** from the left pane. Click the button for **HTML Advanced** and select the option to produce a script that contains explicit URLs.

6.1.2.3 Recording

If you are performing an action against a specific value (such as an item or asset number), you should precede the value with an equal sign in the filter. Once the script is recorded, save the script as-is and do your modifications from a copy of the script.

6.1.3 Script Clean-up

Comment out all `web_add_cookie` and `lr_think_time` entries that were recorded. You will likely want to replace the recorded `lr_think_time` entries with those containing the appropriate think times. Make sure that your script includes `web_set_max_html_param_len("1024");` just inside of the brackets in the `vuser_init()` section.

6.1.4 Correlating Values

The most important areas to make LoadRunner scripting of IBM Tivoli service management products successful are text checks, correlating the uissessionid, and correlating the dynamic UI values (the mx values).

A uisessionid is generated for each user at login time to associate that user’s login session with each request submitted to the server. If the uisessionid is not correlated, the server will not be able to process any request sent, resulting in an error.

Without correlating the mx values, the scripts are unlikely to work from one build (release) to another. Without the text checks, you will get a false sense of accuracy in your tests.

Depending on the version of IBM Tivoli service management products you are running, you may not need to do the correlations on all MXid values. If you are running IBM Tivoli service management products 7.5 or greater, and are using static MXids (`update maxpropvalue set propvalue='true' where propname='mxe.webclient.staticid'`) you will not need to correlate most MXids.

After recording a script, you will notice that most of the `web_submit_data` stanzas contain entries similar to the following:

```
"Name=targetid", "Value=mx282", ENDITEM,
"Name=changedcomponentid", "Value=mx450", ENDITEM,
"Name=currentfocus", "Value=mx450", ENDITEM,
"Name=uisessionid", "Value=3", ENDITEM,
```

There will even be some variation on the above:

```
"Name=targetid", "Value=mx520[R:24]", ENDITEM,
"Name=currentfocus", "Value=mx206_anchor", ENDITEM,
```

You will also see entries similar to the following in the `web_url` stanzas:

```
"URL=http://<servername>/maximo/ui/?event=loadapp&value=asset&uises
sionid=3",
```

Each of the uisessionid and mx values needs to be correlated. Generally you find the value to correlate in the server response from the previous `web_submit_data` stanza. If it is not there, it will be further back in the script, in the server response from the previous `web_url` stanza. Start in the `vuser_init()` section of your script and look for the correlation for each value in order. Position your cursor in the stanza you want to search, and click on the tree view icon, then on the Server Response tab. Once in the Server Response tab, search for the mx value. The first mx value you want to search for will likely correspond to the GoTo menu. For example, in a search for mx43, several occurrences are found:

```
id="mx43_compHolder" class="bc"
ctype="label"   id="mx43"    tabindex="0"      ev="goto"
ev="goto" targetid="mx43" mxevent="click" accesskey='G'
<img id='mx43_image' src='..
new Array(false, true, "mx43", "click")
```

The second and third occurrences look to be unique enough that if the value were to change in a future build, LoadRunner would be able to correlate to the correct value. Consider the second line for this example. Switch back to script view and, from the menu bar, select `Insert > New Step > web_reg_save_param`.

Name the parameter MxValue1 for the first mx value you are correlating. The left boundary should be ctype=`\"label\"    id=\"`. The right boundary should be `\"    tabindex=\"0\"        ev=\"goto\"`. Click **OK**. The following stanza should have been created.

```
      web_reg_save_param("MxValue1",
            "LB=ctype=\"label\"   id=\"",
            "RB=\"    tabindex=\"0\"      ev=\"goto\"",
            LAST);
```

Place the following comment before the stanza:

```
//    Original value mx43
```

Search through the entire script and change all `mx43` values to `{MxValue1}`. Be careful not to change a value like `mx4312` to `{MxValue1}12`.

Once you have changed this value, you can go on to the next mx value. Usually, the mx values on a `"Name=currentfocus"`, line do not need to be correlated. The exception would be a value that is later used in a `"Name=targetid"`, line. To be sure you are correlating those values to the most accurate location, it is best to correlate the values on the "`Name=currentfocus"`, lines also.

You will encounter lines in the format of

```
"Name=targetid", "Value=mx520[R:18]", ENDITEM,
```

The entry corresponds to a selection choice you made while recording. It will likely be either a menu selection from a list of statuses, or a selection from a screen of items. Correlate the `mx520` in the same manner you correlate the other values. If it is from a selection of items on a screen, you should correlate the value from the initial list (find the code with the value `mx520[R:0]`), even if it is back several `web_submit_data` stanzas.

You will likely want to put in code to make a random selection from 0-19, or always take the first item `[R:0]` from the list. If the value were above 19, you would have paged forward one or more screens of items. You might consider putting in code to randomly go forward some number of screens and multiply the random number between 0-19 by 1 + the number of screens you paged forward. In the case of a selection from various statuses, you will not want to change the recorded value.

Entries in the format

```
"Name=currentfocus", "Value=mx206_anchor", ENDITEM,
```

are from changing tabs within the application. Those do not need correlating but are useful to note when creating text checks.

6.1.5 Text Checks

For every `web_url` or `web_submit_data stanza`, you should insert a text check by using a `web_reg_find` stanza. Use the Server Response tab in tree view to help decide what text to use. You need to locate text that is unique to that screen. A possible choice of text for the initial screen is Welcome to Maximo®. The stanza would look like:

```
      web_reg_find("Text=Welcome to Maximo&reg;",
            LAST);
```

When trying to find text to use, search for `BEGINNING of returned STREAM` or `Begin WARNING DIALOG`. You should be able to find some unique text after that point. Here are some styles of text you can look for:

```
window.name="startcntr";
event=loadapp&value=xxxx
xxx_xxx_dialog
menu0_xxxxx_xxxxxx
menuxxx
title=”xxxxx
dialogtoRemove
Record has been saved
```

6.1.6 Parameters

Parameterize the URL in a data file (for example, `URL.dat`). If you will have multiple scripts that you intend to execute in the same scenario, the `URL.dat` file should be shared. If your script performs an action against a specific item that was not the result of a search, that item number should be parameterized. If multiple users will be executing the script, parameterize the User/Password.

6.1.7 Think Time

In order to make your scripts run as if an actual user was performing the actions in a browser, you add think time to your scripts. Before every action where you enter text on the screen, or where you would normally pause to read the screen, insert a specific amount of think time, such as 10 seconds; `lr_think_time(10);`. Be sure the think time is outside of the transaction timer, between the `lr_end_transaction` and `lr_end_transaction` entries. If you plan to run the script multiple times, you might also consider putting some amount of random think time after the login, but before the first action from the Start Center. This will allow you to control the iteration rate to achieve your desired transactions/sec/user as discussed in Section 4.2.

Place:
```
int rand_time_int;
```

before `Action(){`.

Place this text:

```
rand_time_int=atoi( lr_eval_string("{RandThink}") );
lr_think_time(rand_time_int);
```

after `Action() {`.

Create a parameter `RandThink` that is a random number with a value from 0 – XXX that is updated each iteration.

7 Troubleshooting Performance Problems

You should use the troubleshooting procedures in a development or test environment for performance analysis and debugging. The procedures generally should not be used in a production environment. Use these procedures in a production environment only if you cannot isolate the problem in a test environment.

7.1 Performance Problem Determination

If possible, do load testing during the implementation phase to expose performance problems before you put IBM Tivoli service management products into production.

If you have the equipment to perform load testing, you can use load testing after IBM Tivoli service management products are in production to determine if there is any performance impact from patches or from data growth over time.

7.1.1 Problem Determination Techniques

Application Server

WebSphere logs should be reviewed for any errors. For load-balanced implementations, attention should be paid to the distribution of users across the JVMs. Monitor the JVM memory utilization on the application server by enabling verbose garbage collection.

WebSphere Application Server Logs

SystemOut.log and *SystemErr.log* – any application errors, long running SQL queries, and so on, can be found in these logs.

Native_system_err.log – verbose garbage collection information will be in this log if verbosegc is enabled.

http_plugin.log – can be reviewed for any load balancing issues in a clustered environment.

WebSphere Application Server Configuration

Ensure that the JVM heap size is set adequately and that heap exhaustion does not occur.

Ensure that thread pool sizes for the default and WebContainer thread pools are set to appropriate levels.

Enable the Connection Pool Watchdog to monitor the database connection pool to ensure connection leaks are not occurring by setting `log4j.logger.maximo.dbconnection` to **`INFO`**.

Web Server

Web server logs should also be reviewed for errors. View the maximum connections and total connection attempts to see if your web server can use as much of the connection as it needs. Compare these numbers to memory and processor use figures to determine whether a problem is caused by a connection, and not some other component.

On IBM HTTP Server, the logs of interest are *access.log, admin_access.log, admin_error.log, error.log*, and *http_plugin.log*.

It may be necessary to raise the *ThreadsPerChild* setting in the *httpd.conf* file.

Database Server

The `maxsession` table and the number of database connections should be analyzed to verify session activity is as expected.

Database server memory and instance memory should also be monitored. Database traces and snapshots can be gathered to assist with database tuning issues. Refer to DB2 Query Tuning and Index Creation and Tuning and Indexing Oracle Databases for more information.

Network

Monitoring the bytes per second processed by the network interface card can also give insight into potential network overload.

Should more detailed network information be required to understand bandwidth requirements, bandwidth-monitoring tools provide the ability to analyze HTTP requests, the number of round trips between tiers, and TCP/IP packet information.

CPU

Monitor the CPU to ensure that all processors are being utilized as expected and that overall CPU utilization remains healthy.

Memory

Be careful about monitoring total memory usage. For IBM Tivoli service management products, JVM heap size is the most important memory metric to monitor on application servers.

7.1.2 Problem Determination Tools

The following tools can be used to assist with performance problem determination of your IBM Tivoli service management products.

IBM Tivoli Service Management Products Tools

The following tools are provided by IBM Tivoli service management products to assist with problem determination:

- DB2 Snapshot Format Tool – is a Perl script that reads a DB2 snapshot file and produces a semicolon-delimited text file. You can load this file into a spreadsheet to assist with identifying slow-performing SQL queries.

- Activity Dashboard (also called PerfMon) – For instructions on enabling and using the activity dashboard, refer to Chapter 6 in DB2 Query Tuning and Index Creation.
- IBM Support Assistant Tool – provides a workbench to help you with problem determination.

Java Core, Heap Dump, and Garbage Collection Utilities

The following tools can be used to debug Java code:

- IBM Thread and Monitor Dump Analyzer for Java – analyzes javacores and diagnoses monitor locks and thread activities in order to identify the root cause of hangs, deadlocks, and resource contention, or to monitor bottlenecks.

  For information on collecting thread dump files for Websphere Application Server, refer to "*To force a thread dump*" in Web module or application server stops processing requests.

- HeapAnalyzer – allows the finding of a possible Java™ heap leak area through its heuristic search engine and analysis of the Java heap dump in Java applications.

  For information on collecting heap dump files for Websphere Application Server, refer to Generating heap dumps manually.

- IBM Pattern Modeling and Analysis Tool for Java Garbage Collector– parses verbose GC trace, analyzes Java heap usage, and recommends key configurations based on pattern modeling of Java heap usage.

Remote Connection Utilities

- PuTTY – is a Telnet and SSH client included in the PuTTY utilities that can be used to connect to remote UNIX platforms that are secured with SSH.
- Remote Desktop Connection – comes with Windows and allows remote connection to other Windows systems.

File Transfer Utilities

- WinSCP – is a free SFTP and FTP client for Windows that can be used for file transfers to/from UNIX platforms.
- psftp – is a SFTP client included in the PuTTY utilities that can be used for file transfers to/from UNIX platforms that are secured with SSH.

Application Profiling Utilities

The following tools can be used to profile and debug Java code:

- Performance Inspector – contains suites of performance analysis tools to assist in understanding the performance of applications and the resources they consume.
- YourKit – is a CPU and memory Java Profiler that supports J2EE/J2ME.

- Eclipse Memory Analyzer – helps you find memory leaks and reduce memory consumption.

- OProfile – is a system-wide profiler for Linux systems and can profile any running code with low overhead.

Other Utilities

- ActivePerl – provides a Perl programming environment that can be useful for writing scripts to assist with integration framework problem determination.

- Database SQL Query Tools – Each of the database platforms contain tools that can be used to analyze SQL queries to assist with any long-running SQL statements.

7.2 Monitoring the System

Ongoing monitoring of your system can prevent performance issues from arising in the first place. Have a monitoring strategy in place before you put IBM Tivoli service management products into production.

Monitoring the Maximo Platform provides an overview of how the Tivoli monitoring portfolio can be used to monitor your environment.

7.2.1 Monitoring Tools

The following tools can also be used to monitor your IBM Tivoli service management products.

Application Monitoring Tools

- Tivoli Monitoring Agent for Maximo is an optional feature pack that can be downloaded to use the capabilities of IBM Tivoli Monitoring. The feature pack includes Tivoli Monitoring software that you can use to monitor server memory statistics, cron tasks, user connections, database connections, installed products and licenses, and other system information.

  For more information on using IBM Tivoli Monitoring with IBM Tivoli service management products, refer to the IBM Maximo Monitoring Jam presentation.

- IBM Tivoli Composite Application Manager (ITCAM) – can be used to monitor applications at a deeper level for potential issues. For more information on using the ITCAM family with IBM Tivoli service management products, refer to System Performance Monitoring and Diagnosis using ITCAM Products.

Middleware Monitoring Tools

The following monitoring tools can be used to monitor the middleware components associated with IBM Tivoli service management products:

- Tivoli Performance Viewer – enables monitoring of the overall health of IBM WebSphere Application Server from within the administrative console.

- Database Monitoring – each database platform contains tools that can be used to monitor the database and provide useful information.

System Resource Monitoring Tools

The following tools can be used to monitor system resources while performing tests:

- perfmon – can be used to gather performance metrics on Windows-based systems. It is part of Windows and provides access to all of the Windows performance counters.

- nmon – can be used to gather performance statistics on AIX- or Linux-based systems. Nmon for AIX is included with AIX from 5.3 TL09, AIX 6.1 TL02, and Virtual I/O Server (VIOS) 2.1, and is installed by default. Versions of Nmon for previous versions of AIX and more information on using the tool can be found on IBM developerWorks. Nmon for Linux is released to open source and is available from the nmon for Linux wiki.

- rstatd – gathers performance metrics from a UNIX system kernel. Rpc.rstatd is shipped with AIX and can be downloaded from the rstatd 4 Linux site for Linux platforms.

- sysstat – is a package of utilities containing the sar, sadf, iostat, mpstat, and pidstat commands for AIX/Linux. Sar provides system information related to I/O, memory, paging, processes, network, and CPU utilization. Sadf formats data collected by sar. Iostat gives CPU and I/O data disks and TTY devices. Pidstat reports process data, and mpstat reports global and per-processor data.

- vmstat – reports statistics about kernel threads, virtual memory, disks, traps, and CPU activity on UNIX systems.

Bandwidth Monitoring Tools

The following monitoring tools can be used to monitor network and HTTP bandwidth while performing tests:

- Wireshark – is a network protocol analyzer that can be used to capture network traffic for bandwidth analysis.

- WinPcap – is a packet capture and filtering engine used by many network protocol analyzers, including Wireshark.

- HttpWatch – logs and allows inspection of HTTP and HTTPS traffic and can be used to understand the requests/responses between a browser client and a web server for bandwidth analysis.

8 Analyzing Results

When analyzing the results of a performance test, remember what the goal is. The typical goal of a performance test is to determine an optimal active concurrent user load for an average implementation, with a given hardware and software configuration, where end-user response times are within the criteria specified by the business requirements.

An optimal system environment should have sufficient CPU and memory capacities to support occasional peaks in the number of active users without taxing the system to its limits.

Since your workload mix is based on transaction rates, calculate the actually-executed transaction rates to determine if you have a valid test. Also, consider the transaction pass-and-fail rate to determine if your test should be rerun.

Graph your results for each load point. A good performance test will include results for response times, transactions per second, CPU, memory usage/paging, and throughput (mbps).

The following is an example graph that shows CPU use at different load levels:

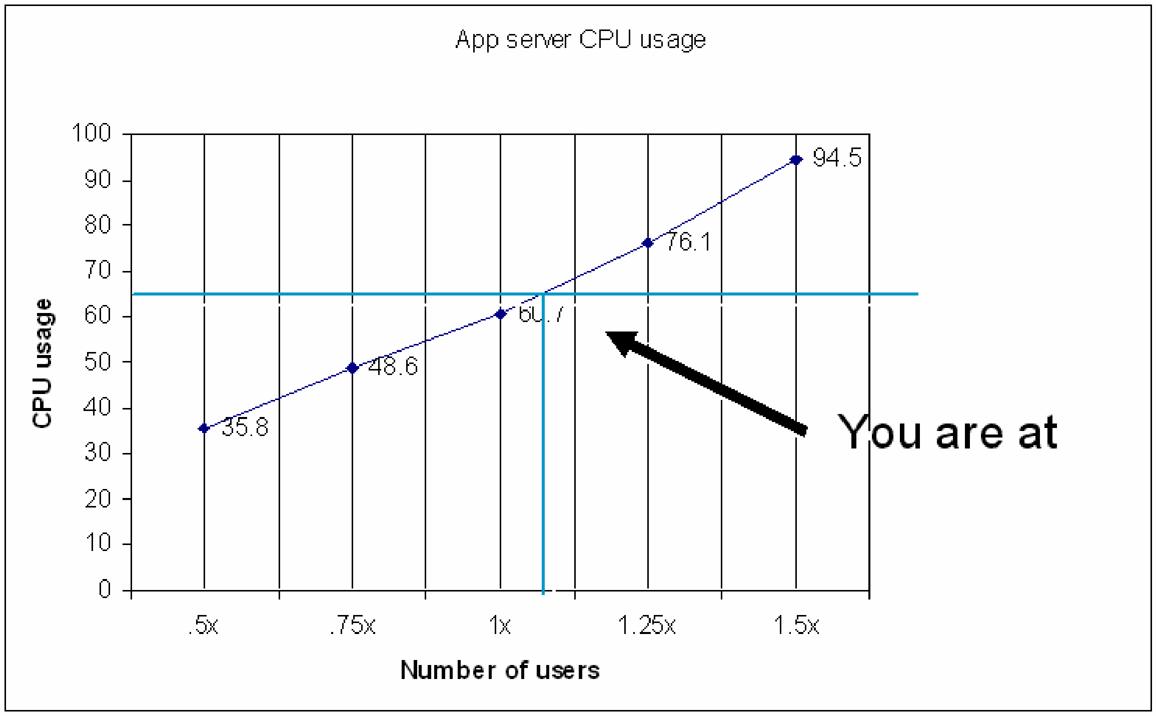

Figure 1: Example Results Graph

These graphical results should be included in a final report that ties back to the business requirements. The final report should document whether the performance requirements were met, and should outline the potential risks and future actions that should be taken.

About Tivoli software from IBM

Tivoli software provides a comprehensive set of offerings and capabilities in support of IBM Service Management, a scalable, modular approach used to deliver more efficient and effective services to your business. Meeting the needs of any size business, Tivoli software enables you to deliver service excellence in support of your business objectives through integration and automation of processes, workflows, and tasks. The security-rich, open standards Tivoli Service Management platform is complemented by proactive operational management solutions that provide end-to-end visibility and control. It is also backed by world-class IBM Services, IBM Support, and an active ecosystem of IBM Business Partners. Tivoli customers and partners can also use each other's best practices by participating in independently run IBM Tivoli User Groups around the world—visit www.tivoli-ug.org