IBM Tivoli Service Management Products

Performance Test Best Practices With Rational Performance Tester

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 IBM® Tivoli® Service Management Products best practices for performance testing with Rational® Performance Tester.

This document applies to the following IBM Tivoli Service Management products:

- IBM Maximo® Asset Management 7.1 and 7.5
- IBM SmartCloud Control Desk 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.

However, internal tests that a development organization performs might not reflect the conditions 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 meet the business demands for future growth, and to help the organization predict performance bottlenecks.
- To analyze, tune, and debug current performance issues in production environments.

Best Practices Is a Cooperative Effort

This document is the result of a cooperative effort between IBM Tivoli Performance and Scalability teams and a core group of IBM Rational Performance Tester engineers. This document was developed in response to customer requests. The collaborative effort is ongoing.

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 objective with a set of business questions that are intended to be answered by the results of the test. All further details of test design should be intended to answer a specific business question.

The following are examples of some business questions that you should address:

- If this is a new deployment, will the architecture that is being deployed meet your business requirements?
- If this is a new deployment, 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?
- If this is a new deployment, which component is likely to be a bottleneck to overall performance, for the combination of IBM Tivoli service management products that you intend to run, and for the transaction volume and concurrent users that you expect?
- If this is an existing deployment, is your existing hardware sufficient to provide acceptable performance, for the expected number of transactions and concurrent users? Or is new hardware or reconfiguration of existing hardware necessary to provide acceptable performance?

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

3.1.1 Baseline Measurements

Perform baseline measurements before you run any other tests. 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, 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.

After you complete a set of single-user baseline measurements with satisfactory results, perform a “benchmark-under-load” test. 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. As with the initial test, any problems with the performance of the application must be resolved before you conduct additional tests with higher load levels.

In some situations, it is helpful to schedule a scenario with a single functional flow. One such situation is when the performance testing coincides with delivery of a functionally stable customization and you want to see how the customization performs before it is included in a workload mix. Another situation might be when you have identified performance issues during the mixed mode scenario. To isolate whether the performance issues are from a specific functional flow, you might need to run single functional flow tests.

3.1.2 Performance Load Tests

After you complete baseline and benchmark-under-load tests, you can begin the full suite of load tests. For these load tests, IBM Tivoli Performance and Scalability teams suggest that you run at least five load points: simulate 50%, 75%, 100%, 125%, and 150% of the expected system load. The five load points help you identify the “knee” of the performance curve. You can test each load point as an individual test, or use the performance test tool scheduling options to drive all five load points from a single test that presents multiple load levels. There are advantages and disadvantages to each method. Allow each load point to plateau and remain at a constant user-load/transaction rate for a period that is long enough to make clear observations before scaling to the next load point.

The objectives of the load test are to measure the response times of transactions and to verify that business requirements and service level agreements are met, under load conditions. Load tests also can find defects in functionality that otherwise might not be revealed. By monitoring the success and failure rate of transactions, you can determine if the application is performing as expected, and you can identify bottlenecks that might 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.

In stress 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 browser cache during stress testing. The duration of the stress test is not predetermined; you gradually increase concurrent users to the point where all server resources are exhausted. You identify the conditions that caused a failure by monitoring the server resources during stress testing and analyzing system logs after the failure. Stress test results indicate whether the system remains stable under extreme loads, or whether its performance degrades or the system crashes.

Detailed results analysis with project architects and implementers should follow stress tests because of 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. 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. Endurance tests are typically executied over time periods that range from 12 hours to a week.

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

3.1.5 Sizing and Capacity Tests

Sizing and capacity tests are important when the results of the tests are needed by deployment teams to properly size new deployments and to determine whether the existing environments can 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.

You design sizing and capacity tests differently than you design standard performance tests. It is important to design series of tests that focus on each tier as well as on the individual components of those tiers. For example, to determine the maximum concurrent users per JVM for a given workload, you must ensure that the JVM itself is the limiting factor. One way to achieve this is to restrict the number of JVMs on a system so that the JVM reaches capacity well before the CPU reaches its limitations. After the JVM capabilities are determined, increase the JVM count so that it is no longer a bottleneck, but the CPU could be. Continue to increase concurrent users until the CPU reaches 100% utilization. At that point, the maximum number of concurrent users per CPU can be reported. Continue this approach until all components of each tier have been fully characterized.

In addition to the components and tiers, you can size the various combinations of IBM Tivoli service management products with the same methodology. After you establish characteristics of each component within each tier of each application, run integrated testing of various applications to add another dimension to the results. This provides additional data that you can use to estimate sizing “rules of thumb” as combinations of applications are expected to be deployed.

3.1.6 Performance Tuning and Debugging

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

If a 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 and then pursue deeper analysis. Restricting other traffic can make it much easier to review traces and logs.

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

For the latest IBM Tivoli service management products tuning information, see the *Best Practices for System Performance* 7x white paper.

3.1.7 Batch Testing

Batch testing refers to testing components of IBM Tivoli service management products that are not interactive. Batch testing plays a vital role in the performance analysis of IBM Tivoli service management products. It should not be excluded. Data loads during customer deployments, integrations with outside vendors, and tools such as the Integration Framework can significantly affect both the middleware and database servers.

Batch tests focus on system resource consumption of the server tiers and throughput rates. If necessary, also consider response-time SLAs. In the test approach, clearly define the type of records to simulate, the size of messages, and transaction feed rates. Mistakes in design can cause results to vary significantly.

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, an architect, or someone that is familiar with the components that are to be simulated.

4 Test Cases, Workloads, and Scenarios

4.1 Test Case Development

When you consider test cases for performance tests, keep the number of use cases to be tested as small as possible. But you must ensure that the cases include the most frequently used functions and the functions that present the most risk. The goal of the performance test is different than the goal of functional testing. It often is not possible to conduct performance tests that have the level of coverage that is in functional testing, because producing performance test scripts from use cases can be time-consuming and labor-intensive. Focus on testing the 20% of scenarios that are most frequently used. Spending effort on the remaining 80% of scenarios does not typically yield considerable performance improvement. The goal of the performance test script for a use case is to be representative of the most common user flows, not to press every button or to activate every option, as might be the case with functional testing.

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

Structure logons and logoffs in the tests so that they match the way that they are used in a production environment. If the end user typically logs on once in the morning and maintains that connection throughout the day, the scripts should reflect that behavior. If the end user typically logs on, performs some work, and then logs off, the script should reflect that behavior. If you want a specific logon and logoff use case, add a script specifically for that case, so those transactions can be controlled and measured by the workload mix.

4.2 Workloads

Developing a valid workload can be a challenge. The workload can usually be determined from a combination of historical data from production systems, database queries of live systems, and interviews with application support personnel. 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 create work orders. You also need to know the rate at which work orders are created. For example, are 20 work orders created per hour, or are 100 created per hour?

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

4.2.1 Concurrent Users and Simultaneous Users

Knowing the correct figures for concurrent users and simultaneous users is vital to performance testing an application. For testing, you need to distinguish between concurrent users and simultaneous users. Concurrent users is the number of users that have an active, but possibly idle, session with the application. Simultaneous users are those that are actually doing work at a given time. The distinction is important when you analyze results.

The terms "concurrent users" and "simultaneous users" are not rigidly defined. If you are told that a Web site has 150 concurrent users, you cannot assume that means 150 requests are being processed at once. That would mean that 150 users were logged on and actively interacting with pages on that Web site. In fact, many of them might be reading the page, thinking, or typing responses, and thus would not be loading the server simultaneously.

4.2.2 Real-time versus 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 real-time usage history (for example, if 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 with the server load created on the server during the performance test. This helps to confirm that the load simulated during the performance test is representative of the real-world load. Adjust the think times in the test scripts to create the appropriate load on the server. For more information, see Think Time and Delay Time. The two things to consider when you analyze server load are requests per second and number of active users.

Depending on how the server and application handle resources related to connections, keep-alive settings, and other workload criteria, generating the same rate of requests with *n* users rather than *2n* users might result in a significantly different performance load on the server, affecting total server capacity and client response times. You can account for this by monitoring the number of open connections, and perhaps other related application resources like number of sessions, in the real world and comparing those with the server load simulated during the performance test. You can perform experimental runs to see if the server and the application are sensitive to the number of users for a given request rate, and if not, it can be ignored.

Requests per second is a more granular and appropriate unit to measure the load that is handled by the server. The number of users that is handled by the server is a metric to be reported to the business stakeholders. Performance testers should focus on the number of requests handled by the server.

4.2.3 Think Time and Delay Time

“Think time” is the time that users take to think or to navigate to different pages in an application. Consider think time when you are deciding on a test strategy.

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

Rational Performance Tester has concepts of both “think time” and “delay time.” Both are added to the tests when they are recorded. Delay time is a fixed amount of time between various page elements. Think time is the user pause between actions. The IBM Tivoli Performance and Scalability teams manipulate both of these variables.

Another way to introduce randomization is to randomize each iteration of the test by setting loop control features. The IBM Tivoli Performance and Scalability teams use this so that the transactions per second per user can be controlled to provide the projected transaction rates over the test time. When you use the loop control features, you might want to set the schedule to use the recorded think time, because the loop control is providing the randomization. However, it is typically better to use both to better model real users.

Think time affects the server load. Decreasing think time increases the server load. Increasing think time reduces server load. To calculate the appropriate think times, you must understand the desired transactions per second per user.

Ensure that think times are placed in the script outside of the start and end of the transactions that are being measured. If they are not, the think times are included in the transaction-response-time results. Rational Performance Tester has two metrics for measuring the time that is associated with a transaction: elapsed time (wall clock time) and net server time. You can think of net server time as the sum of all the page response times, excluding think times and other processing time such as custom code execution time and reference harvesting that is not associated with normal client-server response time.

4.3 Test Scenarios

Design your test scenarios to start by using relatively few system resources, then steadily increasing resource use to reach a stable peak level, then decreasing resource use. During the stable period, the target user load needs to perform various operations on the system with realistic think times. Metrics should be taken during the stable peak period, not during the ramp-up and ramp-down periods. Ensure that enough data samples have been gathered before you make conclusions about the response time metrics; there might be extraneous reasons for faster or slower response times at any single point in 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. A mirror image of the production system is ideal, but is often not possible. Therefore, pay special attention 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 following features:

- The same overall architecture as the production environment:
    - The same operating system and middleware platforms. For example, if the production system uses VMs, AIX, DB2, and so on, the test configuration should use those as well.
    - Similar tiers, hardware proportions, and topology. For example, use 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 as the JVMs that are being used for the user interface.
- Matching software stack versions; the same versions of the operating system, 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, both have the same maximo.properties file, and both have the same security layers (for example, 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 might be different between the two environments.

Always identify the system configuration details of the server machines in the production and test environments. Identify the number of CPUs, CPU capacity (clock speed), RAM capacity, disk capacity, free space available on the disk, NIC card capacity, and network bandwidth. Identify these details before you schedule the performance test, and document them in the test plan document.

Successful performance testing requires a dedicated testing environment. Random application traffic or rogue processes can skew performance results and create confusion and variances in the tests. Both the server and the network components should be among the dedicated resources. 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 for system performance guidelines to the hardware and software. Document any modifications that you make during the testing process.

5.1 Application Servers

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

For the latest tuning information for IBM Tivoli service management products, see the *Best Practices for System Performance* 7x white paper.

5.2 Database Servers

When you prepare for a performance test, you must have sufficient data in the test database. The execution time of a SQL query might be significantly less when a database contains 1000 records rather than 100,000 records. Database performance in the production environment might be significantly worse than in the test environment if the production database has a higher volume of data than what is tested.

Ensure that a representative amount of data exists in the test database to allow for data usage during the tests. How table records are arranged and indexed affects performance. Run database maintenance utilities before you create a database backup. Use the backup to restore to the same state before each test run to allow test repeatability.

For the latest tuning information for IBM Tivoli service management products, see the *Best Practices for System Performance* 7x white paper.

5.3 IBM Tivoli Service Management Products 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 that 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 misses potential bottleneck situations. It is a good practice to compare results with monitoring on and with monitoring off to gauge the impact of monitoring. It might be necessary to increase the level of monitoring when you are searching for performance bottlenecks. But for official results, it is a good idea to reduce monitoring levels to only what is needed to ensure that the results are valid.

6 Test Development Essentials for Rational Performance Tester

6.1 Load Simulation

Load testing tools such as Rational Performance Tester are typically thought of as having two parts:

- The load test controller, which you use to create and drive the required load.
- The load test generators, which send the actual load to the server.

Depending on the number of virtual users that you plan to simulate, the load test controller and the load test generator might be able to reside on the same hardware. The Rational Performance Tester documentation might provide guidance about when hardware should be dedicated and when it can be shared.

6.1.1 Load Test Controller

The load test controller can be referred to as the workbench or workspace. Rational Performance Tester refers to the computer that is running the load as the workbench, and the container for the tests and schedules as the workspace.

6.1.2 Load Test Generators

A load test generator, typically called an agent or driver, can be installed on several systems to spread the load. To perform a 1000-user test, you can install agents on multiple computers. Each agent might be configured to create a percentage of the virtual users. Keep the resource utilization of each agent computer under 70%. Keep the throughput of the agent computer reasonably below the capacity of the network interface. If either of these guidelines is exceeded, results might be affected. The user load, created on each different IP address is then sent to the server under test. From the load test controller, the load generated by each load test generator is aggregated when you view it. By default the individual results collected by each generator are discarded after aggregation for efficiency. To see individual results, clear the Only Collect All Hosts Data statistics flag for the schedule. Individual results are then retained and you can view them.

6.1.3 Load Test Data

Depending on the complexity of the test data, either the performance test engineer or a database administrator can populate the data. When possible, use a database dump from a production system to populate the test database. It might be necessary to write queries to generate data to create your Rational Performance Tester datapools, which contain the data used by the virtual users that execute the Rational Performance Tester tests.

6.2 Recording a Test

6.2.1 Before Recording

Before you record a test, document the steps to take. For example, after you enter text into a filter, do you tab to the next field, press the Return key, or click the binoculars icon? Define the names of your transactions so that when you execute a schedule containing multiple tests, the page/transaction names are displayed in the correct order. A useful format is `<ScriptName>_<StepNumber>_<StepDescription>`. For example, `AM01_01_Login`, or `AM01_05_EnterStatusWAPPR`.

6.2.2 Workspace Settings

To reduce unnecessary automatic data correlation, modify the workspace preferences. After you open Rational Performance Tester, select *Windows > Preferences*. From the left pane, expand *Test*, then *Test Generation* and *HTTP Test Generation*. In the right pane, select the *Data Correlation* tab. Set *Optimize automatic data correlation for execution* to *Efficiency.*

Also set the preference to display heap information on the status line by selecting *Windows > Preferences > General* and selecting *Show heap status.*

6.2.3 Recording

6.2.3.1 Caching

Rational Performance Tester Release 8.2.1 has introduced a new feature of page cache emulation. All static content that the server transfers to cache (last-Modified or Age) is handled by the emulation. The feature that emulates page caching considers only the caching within a test. Once the user exits the test and re-executes it, the cache is re-loaded. Note that in previous performance test papers, IBM Tivoli service management products recommended disabling the MaxAge filter for performance testing; however, leaving it enabled lends itself well to the caching now available in Rational Performance Tester, so the recommendation is now to leave the MaxAge filter enabled, which is the default setting when installing IBM Tivoli service management products.

When you are ready to begin recording, click on the recording icon and select *HTTP test*. Clear the browser cache before the recording. If you have a need to re-record the test, clear the cache before the next recording also.

Note down the number of pages recorded and compare that with the expected number. Some actions record into multiple pages. If necessary, re-record the script, overwriting the previous recording and place a comment in the recording where you would expect multiple pages.

6.2.3.2 Transaction names

Name some pages while recording. The name applies to the page just recorded. Do not name any pages that recorded into multiple pages at this time. It might cause confusion when you clean up the test. You can name the remaining pages in the editor after recording is complete.

6.2.3.3 Comments

Enter comments while recording. The comment precedes the next action that is recorded. Because some steps record into multiple pages, comments are essential to understanding the recorded test. Another way to annotate the test and improve its understandability is to use the Rational Performance Tester screen shot feature. A picture can be the best way to convey the state of the application at the time operations are performed.

6.2.3.4 Entering search values

If you record a search with a specific value, such as an item number or an asset number, precede the value with an equals sign (=) in the filter. If you do a wildcard search, append or precede the string with a percent sign (%).

6.2.3.5 Saving two versions of the recording

After all the steps have been recorded, close the browser to terminate the recording. Use Save As to save the script with another name and make all modifications on the copy. It might be necessary to return to the original unmodified script to view the test as recorded.

6.3 Editing Your New Recording

6.3.1 Test Flow

If there is a step that recorded into multiple pages and there is no obvious way to distinguish the actions, you can merge the pages together. Highlight the pages, then right-click and select *Merge Pages*. Select the first of the pages as the destination. If you have already named the first page, you might have to do so again.

It might also be necessary to split recorded pages into multiple pages.

To name any pages that were not named while recording, position the cursor on the page and edit the *Page Title* field in the *Test Element Details* section.

Add additional comments as needed so that anyone else viewing the script knows what actions were taken. Click *Add* then select *Comment* to add the comment after the last page element on a highlighted page. Click *Insert* then select *Comment* to insert the comment above the highlighted page.

Because a typical user does not log in and log out between each major action performed, put a loop surrounding the go_to_application through to the return_to_start_center. You might also want to put a loop around other repeated actions, such as any next page/screen pages.

6.3.2 Parameters

Rational Performance Tester takes the host and port numbers from the URLs and has most of them parameterized to the test variables HOST and PORT. If no port number was specified during recording, the default port number 80 is used. If you have multiple tests in your schedule and plan to run the test against a different host than you recorded against, create a datapool file with two columns, one for each value. The port column in the datapool cannot be empty.

If your script performs an action against a specific item that was not the result of a search, that item number should be parameterized in either a test variable or a datapool. If multiple users will run the script, create a datapool for the user ID and password.

When you add the datapools to the test, consider how they will be used when you set their properties. Search the online help for datapool options.

After you add the datapools to the test, modify the test variables to point to the datapool fields.

6.3.3 Correlating Values

To make Rational Performance Tester scripting of IBM Tivoli service management products successful, it is important to correlate the uisessionids and to correlate the dynamic UI values (the MXid values).

Although Rational Performance Tester does a good job of correlating many values, there are usually many other values to correlate.

Setting the recording preference to *Efficiency* instead of *Accuracy* is very important for IBM Tivoli service management products sessions-ids. If the preference is set to *Accuracy* and for any reason a page fails, the next session-id correlation usually fails, causing the remaining pages to fail, as one failure leads to another. If the preference is set to *Efficiency*, it will use the initial session-id found in the login.

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

In recent versions of both Rational Performance Tester and IBM Tivoli service management products, some of the MXid values are automatically correlated.

Click on the top of the test, and then click on the view button to *Display Data Correlations*. Review each item that is labeled *Datapool Candidate* (the font color is green by default). Examine the names and values to determine which ones might need correlating. Most values named *targetid* and *value* need to be correlated if they are in the format *mx####. When* using static *MXids,* the format of the values to correlate is longer, and contains both alpha and numeric values after the initial mx. Values that are names of menus should be correlated if all users do not have identical access permissions and application permissions. Although *currentfocus* candidates do not need to be correlated, those values are often used later as *targetids* and will need to be correlated.

Values in the format *mx###[R:#]* are selections in an array. If you always plan to use the same element, this can be correlated like any other value. Otherwise, correlate the base portion *mx####* and create custom code to select a random value.

Give the newly created reference a meaningful name. After creating the reference, find any other consumers of that value, and substitute them with the newly referenced item.

6.3.4 Modifying Regular Expressions

After all necessary correlations have been made, switch the view back to *Display all test Contents*. Click on the top of the test then right-click on *Display References*.

Review the counts in the *Occurrence* column. It might be necessary to edit the regular expression that was created so that it is more stringent. Click the *Properties* button to modify the regular expression. If necessary, add more text to the regular expression and click *Verify*. If verification passes, Rational Performance Tester was able to locate the string. It is not necessary to update the *Specific occurrence number* because Rational Performance Tester does that. There are times when you cannot make the occurrence count 1. Make a serious effort to get the number down to a single digit when uniqueness cannot be achieved.

Following are some hints for making unique results from the regular expressions:

Append to regular expression after (.*?)"

```
[^>]*?title="<Matching_Text>"
[^>]*?accesskey='<Matching_Text>'
```

Preface the regular expression with:

```
input
table
<Matching_Text>[^<]*?
<Matching_Text>.*
```

If the IBM Tivoli service management products build/release level has changed since the test was created, it is possible that the test no longer executes correctly. It might not be necessary to re-record the test. You probably need to modify the regular expressions for your correlated MXid values. The expressions might need to be either more stringent or less stringent.

Due to a defect in Rational Performance Tester, there might be occasions when Rational Performance Tester creates a regular expression that uses the host name. If the test will potentially be executed against a different host, modify each of these regular experssions. Go to the top of the test, right-click, and select *Display references*. Look at the *Regular expression* column. If any expressions contain the host name, modify them. Depending on the current regular expression, one of the following expressions might be an appropriate substitution:

```
http.{0,1}://.*?(/[^?]*)/
http.{0,1}://.*?(/.*?)'
http.{0,1}://.*?(/.*?)"
http.{0,1}://.*?(/.*?)\]
```

6.3.5 Data Correlation References in Conditional Code

If there is a loop in the code that is executed conditionally, ensure that correlations that occur after that loop are not correlated to pages within the loop. Correlate to a page that is always executed.

6.3.6 Removing Unused References

Data correlation causes slight overhead in executing a test. If there are many correlations that are not used, you might want to remove them. From the *Display References* window, review the counts in the *Occurrence* column. If the occurrence count is blank, the corresponding reference is not used and it can be removed. Select the reference and click the *Disable* icon.

6.3.7 Text Checks

To help ensure that Rational Performance Tester scripting is successful, include content verification points. Without verification points, you can get a false sense of confidence that your tests are executing correctly.

Create a content verification point for each page. Select a page element that has a *Response 200 – OK* message. Click *Response 200 – OK*. Click the *Add* button and select *Content Verification Point*. A page might contain more than one page element with a *Response 200 – OK* message*.* Select the most appropriate one. Use the *Response Content* or *Browser* tabs under *Protocol data* to help decide which response to use. Look in the content window on the right and search for some text that might be unique to that page. Ensure that the pull-down selection for the conditions that pass or fail the verification point is set correctly.

6.3.8 Think Time and Delay Time

Remove all think time that was recorded in the scripts on each page. Add realistic think time (five or ten seconds) prior to each action during which a user would normally read the screen or enter text on the screen.

In addition to the think time that Rational Performance Tester applies just prior to executing the requests associated with each page, you might want to adjust the playback speed of the client (browser) delays within the page. From the root of the test, click on the *HTTP Options* tab in the *Test Element* section. Adjust the playback speed. The slider adjusts the speed at which the HTTP requests are sent to reflect slower or faster client (browser) processing. Move the slider all the way to the left for no delay. This scale is applied to all requests in the test.

6.3.9 Secondary Images

Rational Performance Tester Release 8.2.1 includes a page cache emulation feature. If using an earlier version, depending on the expected amount of caching on your working environment, you might consider disabling the loading of images. From the root of the test, click on the *HTTP Options* tab of the *Test Element* section. Click *Modify* next to *Secondary request behavior*, and then check *Images*. Click *Disable*. You do not need to use this method when using a current version of Rational Performance Tester.

6.3.10 Custom Code

Custom code can be used for many different purposes. Some of the common uses include printing out a value, picking a random element, checking the value of a variable, and exiting a loop.

You do not need to be a Java™ programmer to create custom code. You just need an understanding of basic programming and a few examples to start with. Position the cursor on the page that you want the code inserted before. Click *Insert* and select *Custom Code*. Give the code a meaningful name in the *Class name* field. It is a good practice to put all your custom code in its own package, separate from the test package that Rational Performance Tester uses. When you click *Generate code*, Rational Performance Tester creates a skeleton for you.

The following code is an example to print out a passed value.

```
package mypkg;
```

```
import com.ibm.rational.test.lt.kernel.services.ITestExecutionServices;

/**
 * @author unknown
 */

public class PrintValue implements

com.ibm.rational.test.lt.kernel.custom.ICustomCode2 {

        /**
         * Instances of this will be created using the no-arg constructor.
         */

        public PrintValue() {

        }

        /**
         * For javadoc of ICustomCode2 and ITestExecutionServices interfaces, select
        'Help Contents' in the
         * Help menu and select 'Extending Rational Performance Tester
        functionality' -> 'Extending test execution with custom code'
         */

        public String exec(ITestExecutionServices tes, String[] args) {
                return null;
        }

}
```

The *ITestLogManager* interface logs messages and verification points to the TestLog (execution history) from actions in custom code. Use the following import line in your custom code:

`import com.ibm.rational.test.lt.kernel.services.ITestLogManager;`

Add the following lines between **`public`** `String exec(ITestExecutionServices tes, String[] args) {` and **`return null`**`;`

```
ITestLogManager tlm = tes.getTestLogManager();
String ValueToPrint = args[0];
tlm.reportMessage("Passed Value: "+ ValueToPrint);
```

You now have custom code that can print out a value that is passed to it.

For more information, search Rational Performance Tester documentation for *Extending test execution with custom code.*

6.3.11 Creating References

If you have custom code in your test, you might need to create a reference to an item to pass to the custom code.

Select the page element that has the response that contains the value that you want to capture. Go to the server response (*Response 200 – OK*) of that page. Enter *CTRL+Left Click* to display the server response in the content window. In the content window, right-click, select *Find*, then *In Content* (CTRL+F often does not work in a content window). Search in the content pane on the right for the text that you want. After you find the value, highlight the text, right-click, and select *Create reference*.

Sometimes it might be necessary to pass the entire response content to custom code. To create that reference, position the cursor in the content window and right-click, and select *Create Field Reference*.

6.3.12 Handling Maximo Page Sequence Numbers

Tivoli Process Automation Engine Release 7.5.0.1 can send multiple requests asynchronously. In order to ensure that all the requests are received, two sequence numbers have been added to the server requests on every *maximo.jsp*: *pageseqnumber* and *xhrseqnum*. These sequence numbers are specific to a user and tracked by session number. When looping within a test, you must manipulate these sequence numbers. Each new page must increment *pageseqnum* and reset *xhrseqnum*. Every request within a page must increment *xhrseqnum*.

Data correlation rules are created to handle some of the details. Create a rule that searches the html headers to see if *pageseqnum* is present. If found, create a test variable and create a substitution from the *pageseqnum* header value to the variable. A similar rule should be created for *xhrseqnum*.

To be able to update the variables from the user data area, the variables must be defined to span across the life of the virtual user. Each time the script is invoked, the variables must also be initialized to 0 (zero).

You can manipulate the variables with custom code. One routine should increment *pageseqnum* and reset *xhrseqnum*. The other should increment *xhrseqnum*. Prior to each *maximo.jsp*, one of the two custom code modules must be inserted.

The following snippet of the custom code handles *pageseqnum:*

```
public String exec(ITestExecutionServices tes, String[] args) {
       ITestLogManager tlm = tes.getTestLogManager();
       IDataArea userDataArea = tes.findDataArea(IDataArea.VIRTUALUSER);
//     Get the variables value field.
       String pageseqnum = userDataArea.get("pageseqnum").toString();
//     tlm.reportMessage("pageseqnum " +pageseqnum);
       int PageSeqNum ;
       int NewPageSeqNum ;
       PageSeqNum = Integer.parseInt(pageseqnum);
//     tlm.reportMessage("pageseqnum " +PageSeqNum);
       NewPageSeqNum = PageSeqNum + 1;
       tlm.reportMessage("New pageseqnum: "+ NewPageSeqNum);
       userDataArea.put("pageseqnum", Integer.toString(NewPageSeqNum));
       userDataArea.put("xhrseqnum", "1");
       return null;
}
```

The following snippet of custom code handles xhrseqnum:

```
public String exec(ITestExecutionServices tes, String[] args) {
       ITestLogManager tlm = tes.getTestLogManager();
       IDataArea userDataArea = tes.findDataArea(IDataArea.VIRTUALUSER);
//     Get the variables value field.
       String xhrseqnum = userDataArea.get("xhrseqnum").toString();
//     tlm.reportMessage("xhrseqnum " +xhrseqnum);
       int XhrSeqNum ;
       int NewXhrSeqNum ;
       XhrSeqNum = Integer.parseInt(xhrseqnum);
//     tlm.reportMessage("xhrseqnum " +XhrSeqNum);
       NewXhrSeqNum = XhrSeqNum + 1;
       tlm.reportMessage("New xhrseqnum: "+ NewXhrSeqNum);
```

```
         userDataArea.put("xhrseqnum", Integer.toString(NewXhrSeqNum));
        return null;
}
```

6.3.13 Special Situations

A known problem with automated testing is that IBM Tivoli service management products can display an unexpected pop-up window. There are specific conditions to consider:

- When the pop-up always occurs and is in the recorded test.
- When the pop-up occurs while waiting for an event.
- When an error message box appears.

6.3.13.1 When the pop-up window is recorded

One case of a pop-up window being recorded is when a Please Wait window appears on the screen when the status of a work order, purchase requisition, or purchase order is changed. In the IBM Tivoli service management products code, there is a loop with a three-second sleep timer, then a check for the expected response.

In the change-status case, the test records the Please Wait pop-up window. There should be a single page element with a Response 200 status. The Test Data field should show an event of `longopcheck` with a targetid of `longopwait`. Before the first page element, specify a delay of three seconds. Create a reference to the entire response from the page element. After the page element, insert custom code that checks for text in the response that would indicate a failure. If any of that text is found, insert code to break out of a loop,

`tes.getLoopControl().breakLoop();`.

Set the return status so that you know that a problem occurred. Insert additional custom code that checks the response for any text that would indicate that the request was successful. If such text is found, insert code to break out of a loop. Within the page, create a loop to include the delay time, the page element, and both pieces of custom code. Determine the maximum amount of time that you think it could possibly take for the expected response to be returned under load. Set the loop iteration count to one third of that time. Remember that the loop takes a minimum of three seconds to complete.

The response time for the page will not be correct. The initial three-second delay is excluded. Rational Performance Tester starts the timing of a page from the send of the first page element to the end of the response from the last page element (with adjustments to account for overhead, such as regular expression-processing time to extract references from responses, or the execution time of any embedded custom code within the page). You need to manually adjust your results to include an additional three seconds.

6.3.13.2 When the occurrence of a pop-up is inconsistent

One case of inconsistent occurrence of a pop-up window is when a Please Wait window appears following a filtered search. A Please Wait pop-up window does not appear on all searches.

Depending on the version of IBM Tivoli service management products that you are running, this problem can be avoided by setting one or more of the following properties:

- webclient.disablelongopquery=true
- webclient.longopquerydialogwaitetime=300000 (was 3000)

If these properties are not available in the version of IBM Tivoli service management products that you are running, coding for this situation is more difficult. You must force a situation in which you can record the steps and always have the pop-up occur. You need to code the Please Wait page as described, but also put a single iteration loop around the search and the Please Wait code. After your search, you can check the response to see if you have your list of items. If so, break out of the loop; otherwise, continue and execute the Please Wait loop.

Early versions of IBM Tivoli service management products 7.x required a fix from development for a databean.class file that had the same effect as the specified properties statements.

6.3.13.3 When an error message box appears

This case is the most challenging. If you can anticipate when the server might possibly return an error, you can create a reference to the entire response field and pass it off to custom code that takes the appropriate action. If the error cannot be anticipated, you need to have enough logging enabled to debug the situation and handle it appropriately.

6.4 Executing a Schedule

6.4.1 System Clocks

Synchronize the system time of all computers in the system that is under test so that resource monitoring uses the correct timestamps. Rational Performance Tester can handle small differences in system clocks.

6.4.2 Eclipse Settings

A staged schedule, by default, terminates at the end of a stage if the expected number of users in the stage is not active at the end of the stage. You can set a tolerance value to allow the schedule to continue. The value is a percentage of users that are allowed to have not completed, and the default is 0. Increasing this value allows you to debug the test if there are errors.

```
-DrptStopTolerance=10
```

The Full Eclipse .ini file can be found at <Install_Directory>\SDP\eclipse.ini. The Streamlined Eclipse version can be found at <Install_Directory>\SDP\rptse\eclipse.ini. Make the specified changes to both of the Rational Performance Tester eclipse.ini files.

6.4.3 Agent Settings

6.4.3.1 Multiple Agents on the Same Computer

Because the volume of users that an agent can handle is limited by the heap size, it might be necessary to put multiple agents on a single computer and differentiate between the agents by an entry in the host file of the workbench computer. Be sure not to put more agents on the computer than the CPU and memory resources allow.

6.4.3.2 Settings Within the Workspace

After creating an agent location in your workspace, modify the properties of the agent to include the following line:

```
RPT_DEFAULT_MEMORY_SIZE = 1500
```

For a Linux™ agent, you can use the value `3000`.

In general, set `RPT_VMARGS = -DKEEP_ALIVE_ACROSS_TEST=TRUE` when looping over tests in the schedule so that Rational Performance Tester attempts to reuse connections across loop iteration boundaries, that is, test boundaries. (By default Rational Performance Tester closes connections at the end of the test.) This behavior means that fewer connections are created, which can be significantly less work for both the agent and the system under test. However, if your test represents a browser session (which closes its connections upon exit), then the Rational Performance Tester default behavior is correct and you should not override it with this setting.

If the virtual user does not reuse the connection within the keep-alive timeout, the server might close it and then the virtual user has to create a new connection.

If the virtual user exceeds the server keep-alive reuse limit, the server closes the connection and the virtual user must create a new connection.

With at least one IBM Tivoli service management products customer, fewer resources were used creating new connections for each iteration. The `RPT_VMARGS =` value was substituted with `-DKEEP_ALIVE_ACROSS_LOOPS_WITHIN_TEST=FALSE` (the default). The situation was not investigated further, but perhaps the time cycle of the loop and the particular server keep-alive settings combined to create a situation in which virtual users were continually trying to reuse connections that were always closed by the server.

6.4.3.3 Settings on a Windows Agent Computer

Ensure that any Windows® agent computers have sufficient TCP/IP ports. Set the following values in the system registry:

```
TcpTimedWaitDelay          dword:0000001e (30)
StrictTimeWaitSeqCheck     dword:00000001 (1)
MaxFreeTcbs                dword:00011940 (72000)
MaxHashTableSize           dword:0000ffff (65535)
TcpWindowSize              dword:0000ffff (65535)
EnableDynamicBacklog       dword:00000001 (1)
MinimumDynamicBacklog      dword:00000032 (20)
MaximumDynamicBacklog      dword:000003eb (1000)
DynamicBacklogGrowthDelta  dword:0000000a (10)
Interfaces\TcpAckFrequency dword:00000001 (1)
```

For Windows XP and Windows Server 2003, set:

```
MaxUserPort                dword:0000ffff (65535)
```

These settings can be found in
`HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services/Tcpip/Parameters`

For Windows 7, Windows Server 2008, and Windows Vista, the default dynamic port range has changed. The new default start port is 49152 and the default end port is 65535. There are 16,384 ports available by default, rather than 5,000. To view the dynamic port range, start the Command Prompt window and use the netsh command:

```
netsh int ipv4 show dynamicport tcp
```

The following command changes the dynamic port range for the maximum number of ports allowed:

```
netsh int ipv4 set dynamicport tcp start=1025 num=64510
```

The minimum start port is 1025 and the maximum end port cannot exceed 65535.

6.4.3.4 Settings on a Linux or AIX Agent Computer

For Linux or UNIX® or AIX® agents, modify ulimit -n 30000 or unlimited to change the per-process file limit from the default of 1024.

Linux

Edit */etc/sysctl.conf* with the following parameters and save the file. Run the following command:

```
sysctl –p
```

```
net.ipv4.ip_local_port_range      2048 65000
net.core.rmem_default             262144
net.core.wmem_default             262144
net.core.wmem_max                 33554432
net.core.rmem_max                 33554432
net.core.netdev_max_backlog       5000
net.ipv4.tcp_no_metrics_save      1
net.ipv4.tcp_rmem                 4096  16777216      33554432
net.ipv4.tcp_wmem                 4096  16777216      33554432
net.core.optmem_max               40960
```

AIX

Execute the following commands with the values below.

```
no –p –o <parameter> = <value>
chdev -l sys0 -a maxuproc=<value>
```

```
sb_max clean_partial_conns 1
tcp_timewait               30
tcp_finwait2               60
tcp_keepidle               600
tcp_keepinit               40
tcp_nodelayack             1
tcp_keepintvl              10
tcp_ephemeral_low          1024
tcp_sendspace              4096000
tcp_recvspace              4096000
rfc1323                    1
Maxuproc                   8192
```

6.4.3.5 Agent Selection

When you select the location on which to execute a schedule group, consider both load balance and how your datapools are configured: shared, private, or segmented.

6.4.4 Resource Monitoring

With Rational Performance Tester, you can collect Windows PerfMon data, Unix/Linux/AIX rstatd data, DB2 data, Apache Tomcat data, JVM data, WebSphere® PMI data ,as well as any other performance metrics that are available from Tivoli Monitoring, if installed.

By default, the rstat daemon is not configured to start automatically on most systems. To configure this, perform the following steps as the root user:

- Edit /etc/inetd.conf and uncomment or add an entry for rstatd; for example,`rstatd sunrpc_udp udp wait root /usr/sbin/rpc.rstatd rstatd 100001 1-3 2`
- Edit /etc/services and uncomment or add an entry for rstatd; for example, `rstatd 100001/udp 3`
- Refresh services: `refresh -s inetd`
- Start rstatd: `/usr/sbin/rpc.rstatd`

When you create your resource monitors, put the various types of monitors on their own location asset. Multiple PMI ports cannot be placed in the same location; each must be in its own location.

When you select metrics to collect, do not collect more data than you plan to analyze, and do not collect data too frequently.

6.4.5 Test Log

Collecting too much data can negatively affect performance, but collecting insufficient data can make debugging virtually impossible. In a production environment or with a large number of users, limit logging. Always set *What to Log* to *Show errors and failures* and set *Also show warnings* to *All*.

When debugging tests, set the log level for all areas to *All*, and monitor a significant number or percentage of users. These settings show all page responses, even for pages that did not fail.

After the tests have been debugged, set the log level for *And also show all other types* to *Primary Test Actions.* Decrease the percentage of monitored users to a relatively small number. With these settings, you can see the details for the page that failed for the subset of users that are logged.

6.4.6 Statistics and Resource Monitoring Interval

The sampling interval on the schedule *Statistics* tab sets the sampling interval for reports. Increase the statistics sample interval to 30 or 60 seconds for long-running tests. Rational Performance Tester might prompt you to increase that interval when invoking an extremely long test. The suggested time in that prompt should be the minimum time that you specify. However, if you think that you need a shorter sample interval than is suggested and you think that you have sufficient CPU and memory resources available in the workbench, you can try a shorter interval. Observe the workbench performance to see if it is acceptable.

The polling interval for resource monitoring is set on the *Options* tab in the location document for the monitor. For long runs, increase the Windows Performance Monitor and UNIX rstatd monitor interval to at least 30 or 60 seconds. The default is five seconds. You might want to set the polling interval to the same value as the statistics sampling interval. Set the interval for WebSphere PMI to 300 or 600 seconds. Set the DB2 interval to 300 or 600 seconds. Increase other monitored values appropriately.

6.4.7 Performance Requirements

Each application that is tested should have a set of criteria that determines if the test passes or fails. At a minimum, set values for response time, pass/fail percentage, and CPU usage on both the application server and the database server. Many more elements can be set as performance requirements. Information about whether or not the requirements are met is available in the performance report results.

6.4.8 Run Configurations

The use of run configuration minimizes the possibility of result corruption when you move results from the default location to a subfolder. A run configuration file allows you to name the results to something more meaningful than the schedule name appended with the date and time of the run.

From the *Run* menu, select *Run Configuration*. Click the left-most icon in the left pane: *New launch configuration*. In the right pane, name the configuration. On the *Schedule* tab, select the schedule to run. On the *Test Log* tab, clear the check box next to *Use defaults*. Specify the test results name in the *Name* field. The date and time of the run is appended to the name. Specify the location where you want the results to be placed. If you want a new directory, create it before you create the run configuration. Click the *Apply* button. The first time you use the run configuration, you need to click the *Run* button. After that, the configuration name is available on a drop-down menu to the right of the run icon (a green circle with an arrow inside it).

6.4.9 User Comments

After the schedule is invoked or is completed, there is a *User Comments* field in the *Performance Test Results* window. Use the field to document the environment that the test was executed in.This field is available to performance report results as of release 8.2.1 of Rational Performance Tester.

6.5 Performance Results

Reports are displayed automatically during a run. Preference settings (*Window > Preferences > Test > Performance Test Reports)* control whether you see the results for the active time range or for the entire run.

When you close a report, it is not saved. Rational Performance Tester reports are like stored queries. You can open a report (run a query) on a given result as many times as you want and the result is not affected. If you modify the report (add a counter, add a filter, and so on), you have effectively modified the query. The user is prompted to save or discard those changes when the report is closed. To redisplay the report, in the *Test Navigator*, expand the project until you locate the run. To display the default report, doubleclick the results. To display another report, right-click the test results, click *Display Report*, and then select the report to display.

6.5.1 Custom Reports

The default report is the HTTP Performance report, which contains 11 tabs. You might want to create your own report. By default, the verification report and the percentile report are separate reports. You might want to create new tabs in your report for that information. You would need to modify the preferences to make your report the new default. From the *Windows* menu, select *Preferences* > *Performance Test Report* > *Default Report.*

For more information on creating a custom report, see Customize, export, and compare reports.

6.5.2 Analyzing Performance Test Results

While the test is executing, any errors that occur are displayed in the *Execution Event Console* tab. If possible, review the errors as the test executes and terminate the test if a significantly large percentage of errors are occurring. Once the test completes, you can review the test log by right clicking on the performance results and selecting *Display Test Log*.

Review the errors on the *Overview* tab and, if necessary, review the log details on the *Events* tab.

If your load tests get connection errors after applying the TCP/IP tuning recommendations for the agent computers, you might experiment with applying those settings on the application server as well. In particular, the values that affect the number of available ports (MaxUserPort and dymanicport settings in Windows or the `tcp_ephemeral_low` setting in AIX or Linux), as well as the TCP time wait delay settings.

6.5.3 Comparison of All Time Ranges

When you run a schedule that contains stages, time ranges are automatically created for each stage. You can display a report that compares these stages. You also can set preferences to display the report automatically at the end of a staged run.

The Compare report compares the time ranges of each stage. The report provides a quick side-by-side analysis of how the system under test performs under various user loads. After the test is complete, right-click on the results file and select *Compare All Time Ranges*. Depending on whether or not your preferences are set to display the active time range or the default time range, this might take several minutes to complete. You can also select specific time ranges to compare in the *Performance Test Run* window.

6.6 Other Workspace Issues

6.6.1 Project Cleanup

On occasion, when a project has been imported into an existing workspace, there might be hundreds of entries displayed on the *Problems* tab.

Two steps might be needed to remedy the situation. In the *Navigator* window, expand the tree until you see *src / test*. (To display the Navigator tab, click *Windows* > *Show view* >

Navigator.) Remove all the items that end in *java* and that have names that match your test and schedule names and contain a large numeric string in the name. Do not remove any that have a name that matches custom code that might have been put in that class.

In the *Test Navigator* window, right-click on the project and select *Properties*. Click on *Java Build Path*. Click on the *Libraries* tab. Highlight all entries except *JRE System Library [jdk]* and any externally added libraries. (The entries to remove should have a red X next to them, which indicates that they cannot be found.) Click *Remove*. From the *Project* menu, select *Clean*. Click the radio button next to *Clean projects selected below*. Select the project. Clear the check box next to *Start a build immediately*. Click *Build only selected projects*.

7 Troubleshooting Performance Problems

Use the troubleshooting procedures in a development or test environment for performance analysis and debugging. In general, you should not use the procedures in a production environment. Use the 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

Review WebSphere logs for any errors. For load-balanced implementations, pay attention 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, are in these logs.

Native_system_err.log—Verbose garbage collection information is in this log if verbosegc is enabled.

http_plugin.log—Review this log 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 that connection leaks are not occurring by setting `log4j.logger.maximo.dbconnection` to `INFO`.

Web Server

Review Web server logs 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 figures for memory and processor use to determine whether a problem is caused by a connection, and not some other component.

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

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

Database Server

Analyze the *maxsession* table and the number of database connections to verify session activity is as expected.

Also monitor database server memory and instance memory. You can gather database traces and snapshots to assist with database tuning issues. See DB2 Query Tuning and Index Creation and Tuning and Indexing Oracle Databases for more information.

Network

You can monitor the bytes per second processed by the network interface to gain insight into potential network overload.

If more detailed network information is required to understand bandwidth requirements, you can use bandwidth-monitoring tools 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

You can use tools to help determine performance problems in IBM Tivoli service management products.

IBM Tivoli Service Management Products Tools

IBM Tivoli service management products provide the following tools to help with problem determination:

- DB2 Snapshot Format Tool—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 help idenfity slow-performing SQL queries.

- Activity Dashboard (also called PerfMon)—For instructions on enabling and using the activity dashboard, see 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

You can use the following tools to debug Java code:

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

  For information about collecting thread dump files for Websphere Application Server, see *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, see 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—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—A free SFTP and FTP client for Windows that can be used for file transfers to and from UNIX platforms.

- psftp—An SFTP client included in the PuTTY utilities that can be used for file transfers to and from UNIX platforms that are secured with SSH.

Application Profiling Utilities

Use the following tools to profile and debug Java code:

- Performance Inspector—Contains suites of performance analysis tools that help you understand the performance of applications and the resources they consume.

- YourKit —A CPU and memory Java Profiler that supports J2EE/J2ME.
- Eclipse Memory Analyzer—Helps you find memory leaks and reduce memory consumption.
- OProfile—A system-wide profiler for Linux systems that can profile any running code with low overhead.

Other Utilities

- ActivePerl—Provides a Perl programming environment that can be useful for writing scripts to help determine integration framework problems.
- Database SQL Query Tools—Each of the database platforms contain tools that you can use to analyze SQL queries to help with 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 you can use the Tivoli monitoring portfolio to monitor your environment.

7.2.1 Monitoring Tools

You can also use the following tools to monitor your IBM Tivoli service management products.

Application Monitoring Tools

- Tivoli Monitoring Agent for Maximo—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 about using IBM Tivoli Monitoring with IBM Tivoli service management products, see 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 about using the ITCAM family with IBM Tivoli service management products, see System Performance Monitoring and Diagnosis using ITCAM Products.

Middleware Monitoring Tools

Use the following monitoring tools to monitor the middleware components that are 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 you can use to monitor the database and provide useful information.

System Resource Monitoring Tools

Use the following tools to monitor system resources while you perform tests:

- PerfMon—Use to gather performance metrics on Windows-based systems. It is part of Windows and provides access to all of the Windows performance counters.

- nmon—Use 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. You can find versions of nmon for previous versions of AIX and more information about using the tool 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—A package of utilities that contains the sar, sadf, iostat, mpstat, and pidstat commands for AIX and 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

Use the following monitoring tools to monitor network and HTTP bandwidth while you perform tests:

- Wireshark—A network protocol analyzer that you can use to capture network traffic for bandwidth analysis.

- WinPcap—A packet capture and filtering engine that is used by many network protocol analyzers, including Wireshark.

- HttpWatch—Logs and allows inspection of HTTP and HTTPS traffic. You can use it to understand the requests and responses between a browser client and a Web server for bandwidth analysis.

8 Analyzing Results

When you analyze the results of a performance test, remember the goal. The typical goal of a performance test is to determine an optimal active concurrent user load for an average implementation, for a specific hardware and software configuration, where end-user response times meet the criteria specified by the business requirements.

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

Because your workload mix is based on transaction rates, calculate the actually executed transaction rates to determine if you have a valid test. Consider the transaction pass-and-fail rate to determine if you should run your test again.

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

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

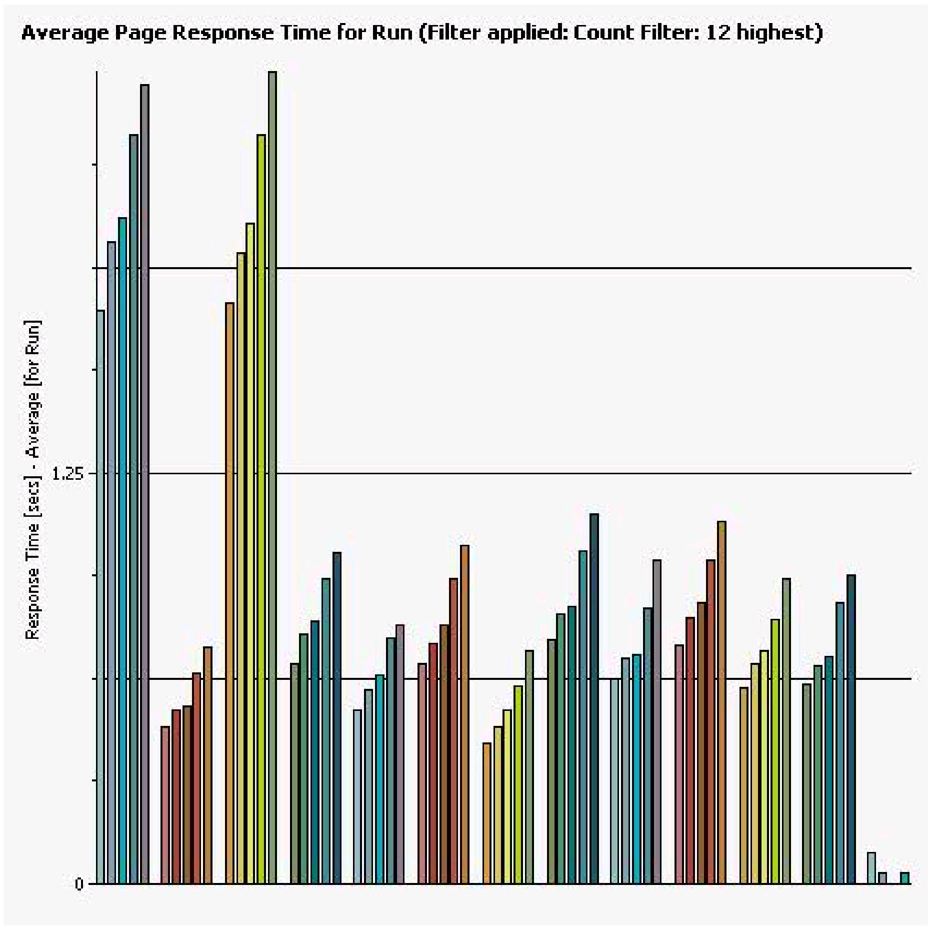

Figure 1: Example Results Graph

You should include graphical results in a final report that refers to the business requirements. The final report should document whether performance requirements were met. If the requirements were not met, the report should outline the potential risks and discuss future actions that might be taken.

9 Rational Resources

9.1 General

Rational Performance Tester Overview

Rational Performance Tester: A resources roadmap for all users

Rational Performance Tester: Find technical developer content and resources for Rational® Performance Tester.

Redbooks publication: Using Rational Performance Tester Version 7

Rational support: Licensing

9.2 Forum

developerWorks > Rational > Forums > Rational Performance Testing

9.3 IBM Support

Rational Performance Tester Support (You must register.)

Software support - *Open service request* (You must register.)

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