Guideline

Designing a Successful Performance and Scalability Test

Product(s): IBM Cognos 8 BI

Area of Interest: Performance

Copyright and Trademarks

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to help guide people with designing a successful Performance and Scalability test with Cognos 8. It is a very high level approach outlining some key principles of Performance and Scalability.

1.2 Applicability

This guide will specifically refer to Cognos 8 BI but the concepts can be applied to many multi-user, multi-server software applications. It should be taken from the perspective of a potential customer who may be evaluating Cognos 8 as the vendor of choice for their Business Intelligence (BI) needs.

1.3 Exclusions and Exceptions

This is only meant to be a guide and factors such as time constraints, coverage "creep", hardware issues, etc can, and likely will, affect the scope of what is actually tested. The assumption made while writing this document is that software and hardware stability is not an issue.

2 Validating Software Performance

For many organizations consuming Business Intelligence, Software Performance can be thought of as validating that the Cognos 8 BI product meets certain performance criteria. Performance criteria may include things such as execute 1000 specific scheduled reports in less than an hour, or maintain an average transaction time of under 5 seconds at 100 simultaneous users, or something as simple as be as fast as the legacy software, for example.

The most prominent criteria are typically dependent on the function a person performs within an organization:

1. A Business Analyst will typically be concerned with how much time certain tasks take.
2. A Server Administrator will typically be concerned with how much server resources (ex CPU and memory utilization) the software requires.

2.1 One-user Performance testing

From the Business Analyst perspective, this test is typically the easiest to test and requires the least amount of external tools to execute. The point of this test is to measure (manually, if necessary) the software responsiveness of end-user actions.

The Analyst running the test should have the following tools available to them:

1. A test plan outlining the user gestures to measure and the expected result or success criteria.
2. A method to accurately measure the user gestures in seconds. See Appendix A.
3. A place to accurately record the test results.

A Server Administrator will typically use any one-user testing to gather a baseline of the server resources the application consumes when it is relatively quiet. The Server Administrator should have server resource monitoring running on the servers while the Business Analysts test is running. Server resources of most interest are the Big 3 of memory, CPU, and network utilization. See Appendix B for a short list of useful tools for monitoring server resources.

2.2 Multi-user Performance testing

Multi-user testing requires a decent amount of planning and having the right set of tools for the job. The tester running the test should have the following tools available to them:

1. A test plan outlining the user gestures to measure and the expected result or success criteria.
2. A method to accurately measure the user gestures in seconds. This will require a user load generating tool. See Appendix A.
3. A place to accurately record the test results.

Server Administrators should have server resource monitoring running on the servers while the Business Analyst multi-user tests are running. Server resources of most interest are the Big 3 of memory, CPU, and network utilization. For multi-user tests the Server Administrator should work directly with the Performance tester to ensure the correct commands for monitoring server resources are executed while the tests are running. The resource monitoring should be started and stopped concurrently with the multi-user tests otherwise correlating the multi-user tests with resource usage can become difficult. See Appendix B for a short list of useful tools for monitoring server resources.

The question may be asked "That's fine, monitor memory, CPU, and network utilization but what should I be looking for". Generally speaking, the server resources can be approaching the limit but not surpassing it such that product performance is suffering. For example:

1. Say the CPU is approaching 99-100% utilization. This may not be an issue if the server is dedicated to the software currently being tested and the user end experience is well within the success criteria.
2. Say the physical memory is totally exhausted and virtual memory usage is extremely high. This can point to a severe issue if at the same time memory is exhausted, end user transaction times suffer. If extra memory is available, adding it to the system can often solve this problem. Also, shutting off unnecessary programs on the server can free up extra memory resources as well.
3. Say the network utilization is at 100%. This is an issue since the network has now become a bottleneck in the system.

As a final note, the computers generating the load can also run out of memory, CPU cycles, and network. Keeping a lightweight resource monitor on them is useful as well to ensure the load generator does not become a bottleneck.

3 Validating Software Scalability

An often forgotten aspect of testing is Scalability Testing. Scalability tests often answer the questions people ask after an initial Performance test has been completed. Things such as:

1. What happens when I add more users to this system? How does that affect the performance of my current users?
2. What happens if I add more servers to the system to host the software? Do my users see any improvement?
3. If I want to double my users, can I expect they will experience the same performance if I simply double my hardware?

These questions can be summed up in the following three sections.

3.1 Predictable Scalability

This can be defined as the ability to add additional users to a static hardware environment and experience a predictable, proportional, and linear change in performance.

For example, if a customer doubles the amount of users on the system they may expect user end performance to degrade by 50%. Inversely, if they reduce the amount of users on the system by half they may expect user end performance to improve by 50%.

General instructions to test Predictable Scalability:

1. Configure the software application on specific hardware.
2. Run a load test at a “small” load for the system for a given time frame. Say 20-users for a smaller server. Record the end-user responsiveness and server resources information.
3. Run another load test at a “medium” load for the system for a given time frame. Say 40-users for a smaller server. Record the end-user responsiveness and server resources information.
4. Run a third set of data points at a “high” load for the system for a given time frame. Say 80-users for a smaller server. Record the end-user responsiveness and server resources information.
5. Review and collate the data. The end-user responsiveness and server resources should grow in a predictable fashion.

3.2 Horizontal Scalability

This can be defined as the ability to add hardware to a production environment, keep the amount of system users constant, and expect to see improved performance.

For example, if a customer doubles the available hardware while keeping the amount of system users constant they could expect a 50% improvement in performance.

General instructions to test Horizontal Scalability:

1. Configure the software application on specific “small” hardware configuration. Say a single server. Have 2 more identical servers configured but sitting idle.
2. Run a load test on the single server system for a given time frame. Say 80-users for a single server. Record the end-user responsiveness and server resources information.
3. Run the same load test (from a number of users perspective) on a two server system for a given time frame. Say 80-users for two identical servers. Record the end-user responsiveness and server resources information.
4. Run the same load test (from a number of users perspective) on a three server system for a given time frame. Say 80-users for three identical servers. Record the end-user responsiveness and server resources information.
5. Review and collate the data. The end-user responsiveness and server resources should decrease in a predictable fashion.

3.3 Vertical Scalability

This can be defined as the ability to add hardware to a production environment, add a proportionate amount of users to the environment, and expect the same level of performance. For example, if a customer doubles the hardware they might expect to double the users and maintain the same performance.

General instructions to test Vertical Scalability:

1. Configure the software application on specific “small” hardware configuration. Say a single server. Have 2 more identical servers configured but sitting idle.
2. Run a load test on the single server system for a given time frame. Say 40-users for a single server. Record the end-user responsiveness and server resources information.
3. Run another load test but increase the user load and servers proportionally. Say 80-users for two identical servers. Record the end-user responsiveness and server resources information.
4. Run a final load test and again increase the user load and servers proportionally. Say 120-users for three identical servers. Record the end-user responsiveness and server resources information.
5. Review and collate the data. The end-user responsiveness and server resources should remain relatively flat and stable.

4 General Tips

4.1 Change one variable at a time.

Try to be as scientific as possible. While it may seem time consuming to change one variable at a time in a large software environment it will save time in the long run. Nothing is worse than changing 5 things, watch the performance take a nosedive, and then have to backtrack through all the changes to find the offending problem. Make sure every change is clearly documented and include the reason for the change.

After a change, it is recommended that at least 2 data points be revisited if all cannot be given the time frame. Take a high user and low user test for a specific test case. The reasoning being that some changes may assist users on a very busy system but inversely slow down users on a less busy system or vice versa. This is good information to note prior to recommending any changes, especially to a production environment.

4.2 Reset as many things as possible prior to executing a test

Since tests results are compared to one another ensure that each test starts at the same point. For example, it is recommended that each performance or scalability test be run after the entire software system (hardware can usually be left running) has been restarted. Otherwise, the previous test can influence the current test since certain database connections may already be established, data may be cached, and processes may be using more memory since they have yet to clean up all the information from the previous test. Cognos 8 products can easily be started and stopped from the command line so as long as the tester has the correct privileges it can be trivial to reset Cognos products prior to each test.

4.3 Execute Baseline tests in the same timeframe as the current test

What sometimes occurs is testers will use results from 6 months ago as the baseline for a current test. The hardware and configuration are the same so what could be the problem? What can happen is that in that time frame the server may have been patched, the disk may have become heavily fragmented, or new software may be running on the server that is consuming CPU cycles and memory that wasn't present 6 months ago. All can affect the scalability and performance numbers.

To minimize this risk; always re-run the baseline right before the current test so that the baseline and current test are equally affected by any factors on the server.

4.4 Automate as much of the test as possible

Automation will solve two problems:

1. The test can run relatively unattended so off-hours can be utilized for testing. Working hours can then be spent analyzing results rather than running tests.
2. Automation will ensure each test or subsequent re-test is run in the exact same fashion.

Typically, all the software tools used for Performance testing come will a full set of command line tools so only simple batch programming is required.

4.5 Monopolize the hardware the test is running on

Nothing is worse than having an unexpected result and later finding out someone else was on the server while the test was running and executed some nasty query thinking it wouldn't affect the ongoing test. In many environments, Performance and Scalability testers need to share with user acceptance testers. A schedule should be worked out with these other groups so that the Performance tests don't interfere with User Acceptance tests and vice versa.

4.6 Do not wait until the test is over to analyze the results

Plan to analyze test results on a daily basis. Like a doctor, testers will want to find problems early so that they can be solved. Problems are easier to resolve at the beginning of the test cycle than near or at the end. This will also allow the tester to provide preliminary results to management types who are very interested in the progress of the tests. With these results, plan to send out daily status updates so all parties involved are aware of the current status of the test.

5 Creating a Test Plan for the Performance and Scalability Engagement

Testing without a plan is a recipe for trouble. These are common guidelines but they are useful to review. Ensure that all the stakeholders read and sign that they agree to the test plan prior to testing.

Ensure the following sections are in the test plan.

1. The purpose of the test. It should be direct and short such as "The purpose of this Performance and Scalability test is to measure specific deltas between Cognos 8.3 and the current production running version of Cognos 8.4. Specific end-user experience cases and server resource consumption during those cases will be measured."
2. Define the performance success criteria for the test. Be as specific as possible. It is extremely helpful to know the success criteria before testing.
3. A full description of the specific test cases for the performance test. Write out step-by-step what each test does. For example, for Business Analyst test cases write out each click the user or Performance Test tool will perform. If reports are run in Cognos 8 BI, have a description of the type of report run, the amount of rows returned, etc. These should relate to the success criteria for the test.
4. Take the general test descriptions from Steps 2-3 above and create specific test cases where the test case, the number of users, and the hardware resources used for that test case are clearly defined. These should relate to the success criteria for the test.
5. A clear description of what deliverables will result from the test and their due dates.
6. Generate a schedule that can fit into the time frame of the test. Include script development, automation development, software configuration, etc into the schedule. Save *at least* one day for every week of the test cycle for documentation and final analysis of the test results. Have all parties agree to the schedule. Place a caveat into the schedule that states that if issues arise that negatively affect the schedule that a meeting should be called as soon as possible to either extend the test cycle or remove some test cases, for example.

Finally, after the test is completed. Review the test plan to see what could be improved. Make note of any issues and incorporate the solutions into subsequent tests.

6 Delivering the Results

While it is more fun just to jump in and start testing, take the time in the initial part of the test planning phase to determine the deliverables for the test. Sometimes going back later to gather missing information is not an option.

Here is a short list of things to discuss when it comes to deliverables.

1. How will the data be delivered to various audiences? For example, executives may prefer a formal PowerPoint type presentation while the Business Analysts and Server Admins may prefer to see granular data in an Excel format or a Cognos 8 report.
2. What information is required for each audience? For example, executives may want to see summaries most times. Analysts may want to see how much time certain tasks can take. Server Admins may want to see how much server resources are consumed by various tests. Discuss this ahead of time so nothing major is missed.
3. Learn how to correlate the end-user experience with the server resources in the same report or graph. These are very powerful graphs in which data can be derived to see how the end-user experience correlates to the resource consumption on the server. For example, one test may show big performance degradation in end-user times. However, when this graph is correlated to a graph showing memory consumption it may show that the server has insufficient memory to run this many users.

Appendix A: Useful Software for Measuring End-User Transactions and Generating Load for the Performance and Scalability Test

The following software, while not an exhaustive list by any stretch, are strong tools for driving load and measuring certain aspects of Performance and Scalability. They are merely suggestions. Investigate these tools and evaluate them for use in your organization's test suite. Fiddler is the only free tool from this list.

IBM Rational Performance Tool (Single and Multi-user)

Rational Performance Tool or RPT is a performance testing tool offered by IBM (http://www-01.ibm.com/software/awdtools/tester/performance/).

LoadRunner (Single and Multi-user)

LoadRunner is a performance testing tool offered by HP.

Fiddler (Single user)

Fiddler is a free, browser based, single user performance tool available online (http://www.fiddler2.com/fiddler2/). It works with Internet Explorer, FireFox, and Opera among others.

Appendix B: Useful Software for Measuring Server Resources for the Performance and Scalability Test

The following software, while not an exhaustive list by any stretch, are strong tools for measuring server resource usage during Performance and Scalability tests. These are merely suggestions. While there are a multitude of tools for purchase that measure server resources, most operating systems come with a useful toolkit by default. Investigate these free tools and evaluate them for use in your organization's test suite.

Nigel's Monitor (NMON for AIX and Linux)

http://www.ibm.com/developerworks/wikis/display/WikiPtype/nmon

This is a nice tool since its file output can be analyzed automatically with an Excel based application called 'nmonAnalyser' (http://www.ibm.com/developerworks/wikis/display/WikiPtype/nmonanalyser).

Microsoft's SysInternals Tools (Windows)

http://technet.microsoft.com/en-us/sysinternals/default.aspx

There are a multitude of useful tools such as psList that can output process and server related information to a file. That file then can be parsed to retrieve relevant information.

Microsoft's Performance Monitor (Windows)

This is Windows default Performance Monitoring tool and comes with the operating system.

'top' and 'ps' (Solaris and HP)

These tools come with the operating system. Review the man pages for both commands for the correct arguments to gather the necessary server and process information for products running on these operating systems.