PTC®

Performance Testing PTC Integrity Lifecycle Manager: A Primer

Author: **Scott Milton**

Revision: **1.0**

Change History

Date	Author	Version	Description
2012-06-29	Scott Milton	Draft 0.9	Draft for Review.
2013-06-25	Scott Milton	Draft 0.91	Second Draft for Review.

Table of Contents

1. Introduction

This document is intended to be a short introduction to performance testing strategies for PTC Integrity Lifecycle Manager implementations. That is to say, this is a document describing how to test the performance of the PTC Integrity Lifecycle Manager Application itself, rather than implement a load or stress testing workflow within Lifecycle Manager for testing other tools. A description of the functional areas in Lifecycle Manager and their respective areas for potential performance concerns will be presented. There are a variety of technologies available both from PTC and other vendors to perform performance testing; a brief summary of some options will be presented.

Load testing is generally a practice for determining acceptable response times, or acceptable server resource consumption, for operations performed by PTC Integrity Lifecycle Manager under specified loads. Stress Testing tends to exceed those loads to investigate the ultimate limits of the software. Another primary purpose of stress testing is often to determine the recoverability of the system by ensuring the system is pushed past its limits, sometimes to the point of system failure, and ensuring it will recover gracefully. Although certain suggestions will be made in this document regarding the areas to test, it is ultimately up to the reader to decide the necessary and sufficient amounts of testing for their own implementations.

Note: A brief update on naming. The product formerly known as "PTC Integrity" is now named "PTC Integrity Lifecycle Manager", since PTC Integrity now refers to a family of software and systems engineering products. For brevity and clarity, this guide uses "Lifecycle Manager" as an abbreviation for the full name, "PTC Integrity Lifecycle Manager".

The topics of "Lifecycle Manager Server Performance Tuning" and "Database Tuning" are both beyond the scope of this document. There are additional publications available from PTC that cover these topics in detail from http://support.ptc.com. It is highly recommended that the suggestions in these documents be applied to your implementation before proceeding with formal performance testing to ensure accurate results. It is also worth noting that this document only covers the basics of database performance monitoring. Enterprise-level Relational Database Management Systems have a wealth of monitoring tools available, and for more sophisticated monitoring the expertise of a DBA of the RDBMS is invaluable. It is highly encouraged that the reader should involve one in any performance tuning, monitoring and testing.

2. Performance Testing Basics

Test results will always be more applicable to a real production environment, and thus more useful, if the testing environment and operational scenarios are as similar as possible to the actual configuration and end user behavior of the reader's eventual production environment.

In general terms, an equation could be written to explain performance testing as follows:

<A> Configuration (<X> Users * <Y> Operations) = <Z> Response Time

This equation illustrates the three main factors influencing response time:

- the configuration (including Hardware and Application configuration)
- the # of users or concurrent operations
- the actual operations being executed

Before constructing the test scenarios to execute during performance testing, the reader should take into account how these variables should be realistically populated for their planned implementation.

Once best estimates have been established, the reader is also advised to plan for varying levels of server load (e.g., High Usage, Medium Usage, Light Usage) in their test scenarios and examine if the progression of response times is acceptable during load testing. The principle behind load testing is that varying the number of users and the frequency (or type) of operations may reasonably be expected to alter the system response times. The above equation could be extended to explain this as follows:

<A> Configuration (<X> Users * <Y> Operations) = <Z> Response Time

<A> Configuration (<X1> Users * <Y1> Operations) = <Z1> Response Time

<A> Configuration (<X2> Users * <Y2> Operations) = <Z2> Response Time

Simulation of these increasing load levels could be achieved with:

- Increased number of concurrent operations (e.g., the number of concurrent operations or users)
- Increased Frequency of operations (e.g., the delay between individual operations)
- The complexity and/or size of data, as this too will affect Lifecycle Manager Server load

The reader should only exceed a reasonable approximation of their production environment with any of these variables, if they are interested in stress testing their implementation to ascertain the upper limits of their particular configuration. In other words, using a realistic number of users, but an unrealistic quantity of data or frequency of operations will result in response times that do not approximate a real world scenario.

3. Areas of Functionality

Deciding which operations to include in your operational test scenarios while performance testing Lifecycle Manager can be complex due to the highly configurable nature of the functionality available in Lifecycle Manager. Individual configurations of Lifecycle Manager deployed by different customer locations may have vastly different response times due to varying hardware, triggers, computations, custom actions, amount and type of fields, number of content items in a document, etc. Different areas of functionality will require different server resources and may experience different performance bottlenecks. Although the Lifecycle Manager server will almost certainly draw resources from all of the following areas for many operations, a rough rule of thumb for the main resource required for various areas of functionality is as follows:

- DB Usage: Item Branching, Historical Operations (Item Views, Computations and Reports), Item Revisioning, Historical operations.
- Server Memory Usage: Run Reports, Charts & Dashboards, Web UI usage.
- Server Disk I/O Usage: Member Based Source Commands, Federated Server Architecture Caching.
- Network Usage: Large File Operations in Source (Checkouts or Checkins) , Federated Server Architecture Caching.
- LDAP Usage: Authentication, Refreshing User and Group Caches.

3.1. Configuration Management

Configuration management commands typically involve moving source code to or from the repository and/or capturing member or project revision information. Data is stored long term in the database and cached on the local file system to improve response times for frequently used data.

As always it is recommended to look at actual operations that users do in their current tools and test the comparable operations in Lifecycle Manager at a similar frequency. If you are unsure of the configuration management operations in use by your users, a good starting list is as follows:

- Checkpoint
- Add Label
- Create Sandbox
- Create Subproject
- Create Sandbox
- Check Out
- Check In

When testing operations that recurse throughout a project tree, it is recommended to have a realistic Source Project as the basis for testing. Important details include the depth of the project tree, i.e., the number of levels of subprojects and the width of any particular level, i.e., the number of members and subprojects at any particular level. When testing member operations, it is recommended to have both realistic file sizes and number of entries in the revision history.

Source testing operations tend to have a high correlation to DB performance, Server Memory and File IO. Large file transfers are heavily dependent on Network throughput and latency.

Note: When measuring, be sure to consider measuring Disk I/O for potential bottlenecks. The file system bulk data cache is recommended to be on a fast local hard drive. E.g., SSD Drive > 10000 RPM >7200 RPM > 5400 RPM

3.2. Workflows and Documents

Workflow and Document commands typically involve moving textual data from the server to the client or vice versa. Occasionally binary data is also moved, particularly in the case of attachments or images. Attachment data is stored long term in the database and cached on the local file system to improve response times for frequently used data, however the usage of this cache will typically be much lower than with Configuration Management Command. Workflow and Document operations tend to have a high correlation to DB performance and Server Memory.

As always it is recommended to look at actual operations that users do in their current tools and test the comparable operations in Lifecycle Manager at a similar frequency. If you are unsure of the Workflow and Document operations in use by your users, a good starting list is as follows:

- Create Item
- Edit Item
- View Item
- Create Document
- Edit Document
- View Document
- Baseline Document
- Run Report
- Run Chart
- Run Dashboard
- Run Query

When testing item based operations, it is recommended to use a realistic distribution of item types. Different types of items often have very different workflows, fields, relationships and triggers, each of which may have different impacts on server resources. In particular, triggers may have a large impact on the performance of particular create or edit operations, but they may not be fired upon every edit, so care will need to be taken to ensure they are tested accurately.

The same concerns also hold true for documents, although with documents there is a particular emphasis on relationships, mainly due to their nature as containers for content items. Generally speaking, it is expected that the number of items in a relationship, or the number of content items already contained within a document, will have a linear impact on the response time of inserting or removing additional content/related items, or viewing the document. This general rule will hold true with adding traces to content as well, so it is best to use a realistic set of relationships for your tests. If the reader's particular implementation has many long lived items (e.g., items that are active for multiple years), another area to potentially examine in test cases is the number of entries to the history of a single item caused by many subsequent edits.

Note: It is of critical importance that statistics are computed on your database regularly, including after a large load of data. Indexes will also need to be added to commonly used queries to ensure optimal performance.

4. Testing Tool Strategies

4.1. Command Line Interface

This strategy involves installing the Lifecycle Manager Client and using the Command Line Interface (CLI) to issue commands. The CLI allows for easy scripting of commands in a batch file or shell. This file should be distributed to multiple clients, who can then execute the file in either a serial or simultaneous fashion. If using a batch file on windows, it is recommended to combine these commands with a simple timing utility, such as timethis.exe, available free of charge from Microsoft. The timethis utility will print additional information to the output regarding when commands were started and the execution duration of commands.

E.g., Timethis "im createissue --type=<type>"

A representative set of commands should be used for the various areas of functionality that will be used in the Production environment. A sample subset of commands may be:

Command	**Purpose**
im baseline <segment id>	Baseline an existing document
im createcontent	Insert content into an existing document
im createissue --type=<type>	Create a new item of the specified type
im createsegment--type=<type>	Create a new Document of the specified type
im editissue <id>	Edit an existing item specified by id
im editissue --addrelationships=<id> <id>	Add related items to an existing item specified by id
im issues --query=<queryname>	Execute a query displaying results
im runreport <reportname>	Execute a report displaying results (note additional time may be required by the browser to render a report)
im viewissue <id>	View an existing issue

Due to individual configurations of Lifecycle Manager, especially the use of triggers, it may be necessary to repeat the commands on a variety of different item IDs and types to get a true representation of functionality and performance impact. A final list of commands should be agreed upon when the Lifecycle Manager configuration is as stable as possible.

The main drawback to the CLI technique is that it is not trivial to programmatically use results of previous commands in future commands. E.g., Use the ID created as the result of an 'im createissue' command to a subsequent 'im editissue <id>' command will require textual parsing of the 'im createissue'

Additional Notes:

- Commands should typically include all of the require connection arguments (--hostname, --port, --user, --password) and the --batch argument to prevent the batch file from requiring user interaction, which could skew results.
- When attempting to use a single client to simulate the load of multiple users, be sure to keep in mind that the individual Lifecycle Manager Client installation and its host hardware can be a bottle neck as well, e.g., Disk I/O, Network, CPU. A reasonable number of client machines are likely needed to simulate the full load on a server, but the exact number will vary based on the available hardware.
- Be sure to configure the java heap of the Lifecycle Manager Client upwards if it is doing heavy source operations or document operations. Generally speaking, it is appropriate to start at a reasonable number like 512MB or 1GB and add more only if needed (e.g., there are out of memory errors).

4.2. Application Programming Interface

A Load Test Harness has been field developed for stress testing Lifecycle Manager solutions at previous customer installations. The Load Test Harness provides a multithreaded application for running several iterations of predefined Lifecycle Manager API commands. The representative set of commands is the same as mentioned above. Running multiple threads through an instance of the Load Test Harness across multiple machines essentially simulates a large user load against the server. The primary objective of the Load Test Harness is to determine the maximum load an Lifecycle Manager Server is able to operate under and yet provide reasonable response times. Reasonable response time is subjective and analyzing the data from Load Test Harness is important to arrive at such conclusions. Since the Load Test Harness uses automated batch processing utilizing the MKS API, client-side user experience cannot be collected using this utility.

Transactions fed into Load Test Harness are xml based and the xml structure is closely modeled after the Lifecycle Manager API's 'Command', 'Response', and 'API Exception' data structures.

The primary benefits of using the Load test Harness over the CLI are:

- Easier use of prior command results in subsequent commands
- Easier launching of simultaneous, multi-threaded client scripts simulating multiple users

The primary drawbacks of the Load test Harness are it requires:

- More time for initial setup
- Detailed knowledge of XML
- Time for transcription of CLI commands into the XML equivalents

More detailed information is available in the Load Test Harness User Guide, which is available upon request. The reader could also easily use the API to construct their own Load Test Harness equivalent, should the particular version available from PTC not meet their needs.

4.3. Commercial off the Shelf (COTS) tools

At the current time, PTC does not recommend any particular COTS tools for performance testing. With capture and replay tools like LoadRunner, unfortunately, the playback often fails in our experience. There are also occasionally errors resulting from the Lifecycle Manager Server being unable to communicate with the simulated clients (resulting in terminated sessions), and the load generated is unrealistic since it invalidates all the caching done in the client.

Should the reader have any existing COTS tools they wish to use, it is recommended an analysis should be performed to ensure they provide effective results. The PTC Global Services Organization may be able to assist with this analysis, if needed.

4.4. Load Testing Tools used internally for Lifecycle Manager Product Releases

PTC has developed their own performance testing framework, used for all in house performance regression testing, load testing, scalability testing, and resilience/reliability testing. Unfortunately the framework isn't customer consumable at this time and is only intended for internal use. Essentially, the framework interfaces with the Lifecycle Manager Client via the standard/published API, as mentioned above. The API provides fairly broad coverage of the commands of typical usage patterns, although there will of course be variations in the server load generated via other interfaces.

5. Test Monitoring

5.1. Logs

The primary log output used for measuring performance testing will generally be the direct output of results from the CLI or API. In addition these output logs from various clients can be compared to determine different patterns. The server.log should be monitored for any issues arising on the Lifecycle Manager Server side. Additional categories of logging, such as SQL and LDAP logging can be enabled to diagnose performance issues, although enabling these categories may produce substantial amounts of logging which can in turn influence performance.

5.2. Diag Commands

Lifecycle Manager provides various diagnostics to monitor and diagnose issues that may arise when performing operations on the Lifecycle Manager Client and Lifecycle Manager Server. The Lifecycle Manager Administration Client, and the “si diag” and “im diag” CLI commands offer several diagnostics the reader may run on the “Workflow & Document Management” and “Configuration Management” components. The diagnostics are detailed fully in the "Running Lifecycle Manager Server Diagnostics" section of the Lifecycle Manager Installation and Configuration Guide.

Note: The AdminServer permission is required to execute these diagnostics, and they may be executed via the CLI or the Lifecycle Manager GUI.

For the purposes of Stress testing the following commands may be useful:

Command	Purpose
Im diag --diag=heap --target=server	Performs garbage collection and displays memoryin use, free memory, and total memory in bytes
Im diag --diag=metrics --target=server	Displays server metrics, such as the maximum change package entries per change package, and the average time entries per user
Im diag --diag=running --target=server	Displays list of charts, dashboards, queries, and reports that are currently running
Im diag --diag=statistics --target=server	Displays server statistics.

5.3. PTC System Monitor (PSM)

PTC provides system monitoring software, based upon dynaTrace, which can be used for sophisticated monitoring of the Lifecycle Manager System. In Production environments, use of the PSM is encouraged so that administrators can easily be notified of the current status of the Lifecycle Manager system. In production environments the PSM should be installed on a separate physical server to the Lifecycle Manager Server. In a non-production environment, if there are no separate physical servers available, it is possible, but not recommended to install the PSM concurrently with the Lifecycle Manager Server. In this case, the hardware must meet the combined system requirements of both Lifecycle Manager and PSM, and it should be noted that the PSM will introduce small overhead in CPU cycles and a larger overhead in memory consumption.

However, for small scale stress testing, or testing on non-production hardware, a decision will need to be made as to whether the extra time and hardware to install the PSM is worthwhile.

6. Limitations

PTC does not at this time provide a mechanism to simulate server load or multiple simultaneous users by using a single Lifecycle Manager Client installed on a single machine. Actual Lifecycle Manager client installs, either on physical machines or virtual machines are generally required for a satisfactory simulation. For this reason, performance testing will always be an approximation of a real world environment and cannot guarantee exact results.

7. Appendix A: Additional Guidance Checklist

The PTC Technical Support Team and Global Services Organization can provide further assistance when assessing the results of performance testing. For maximum efficiency, consider answering the questions on the below checklist when seeking assistance.

- Which business group or organization are you performing the testing for?
  - What are the business use cases that justify the test case?
    (E.g., users do this activity X times a day in order to accomplish Y.)
  - If there is no business case, what is the purpose of the test?
    (E.g., system functionality, stress testing)
- What test script is being executed?
  - Do the test results include the time to execute only integrity commands, or other application code as well?
  - What is the exact integrity command(s) that is being run including all arguments?
  - How many items is the command expected to return/work against.
    (E.g., query should return X items).
  - How many times is the test script executed?
  - What is the amount of time (pause) between executions?
  - How are the commands executed?
    (E.g., Bat file, Java api, C api, web services, portal, 3rd party tool, etc.)

- If many test cases are being executed at the same time (i.e., you are testing concurrency):
  - How many users is the test case execution designed to simulate? (E.g., 40 threads represent 4000 users)
  - How many threads are used?
  - Are all threads executing the same commands?
  - Are there significant performance differences between executing a single thread vs. multiple concurrent threads?
  - Has logging/diagnostic output been captured to show the differences between a single thread vs. multiple threads?
- What are the details of the environment used to execute the tests?
  - Are the hardware and application configurations available? (E.g., property files)
  - What dataset is used in the environment?
  - How is the environment/database reset between test executions?
- Were database statistics computed before and/or after the test was executed?
- Were the individual integrity commands used in the test script tuned for maximum performance?
  - E.g., for a query, look at the sql plans, creating database indices to support it, etc.
- What monitoring is available to go along with the test results?
  - Individual User SQL logging?
  - Debug Logging?
  - CPU?
  - Memory?
  - PTC System Monitor?
- If it is not already included, can you modify the test methodology so that
  - Before starting the test cases run “im stats --reset”
  - After executing the test cases run ”im stats” and collect the output