BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

PERFORMANCE OPTIMIZATION OF TESTING AUTOMATION FRAMEWORK BASED ON BEAKERLIB

OPTIMALIZACE VÝKONU AUTOMATIZOVANÉ TESTOVACÍ PLATFORMY ZALOŽENÉ NA BEAKERLIBU

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR JAKUB HEGER
AUTOR PRÁCE

SUPERVISOR Mgr. Bc. HANA PLUHÁČKOVÁ,
VEDOUCÍ PRÁCE

BRNO 2017

Abstract

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v anglickém jazyce.

Abstrakt

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v českém (slovenském) jazyce.

Keywords

Sem budou zapsána jednotlivá klíčová slova v anglickém jazyce, oddělená čárkami.

Klíčová slova

Sem budou zapsána jednotlivá klíčová slova v českém (slovenském) jazyce, oddělená čárkami.

Reference

HEGER, Jakub. *Performance Optimization of Testing Automation Framework Based on Beakerlib*. Brno, 2017. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Pluháčková Hana.

Performance Optimization of Testing Automation Framework Based on Beakerlib

Declaration

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana X... Další informace mi poskytli... Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

....
Jakub Heger
May 10, 2017

Acknowledgements

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant, apod.).

Contents

Chapter 1

Introduction

...
...
...

Chapter 2

Relevant projects

In this chapter describes BeakerLib and projects relevant to it.

2.1 BeakerLib

BeakerLib is a shell-level integration testing library, providing convenience functions which simplify writing, running and analysis of integration and blackbox tests. It is developed and maintained by Red Hat and operates under GNU General Public License.

- Main features of BeakerLib include:
- Journal - uniform logging mechanism.
- Phases - logical grouping of test actions, clear separation of setup / test / cleanup
- Asserts - common checks affecting the overall results of the individual phases (checking for exit codes, file existence & content...)
- Helpers - convenience functions for common operations such as managing services, backup & restore

[citace beakerlib github wiki]

This thesis focuses on BeakerLib Journal and problem it causes with long tests.

2.2 Beaker

Beaker is a full stack software and hardware integration testing system, with the ability to manage a globally distributed network of test labs. [citace beaker doc] It is Red Hat community project under GNU General Public License version 2.

Main functionality includes management of hardware inventory, on which Beaker can install wide variety of operating systems from Red Hat Linux family. Another notable part is Task library which contains rpm packages of individual tests which can be run on provided machines. Users then can specify which hardware they require with which OS and tests they want to run on it through either command-line tools or web interface both of which are part of Beaker install package [link to beaker download page?]. If Beaker meets given criteria in its inventory it installs Test harness to which it gives list of tests to be run. After Test Harness finishes running the tests, results are sent back to Beaker where they are stored for specified period of time.

2.2.1 beaker-wizard

Part of Beaker package. Interactive command-line tool which automates creation of BeakerLib tests. Using predefined or user-defined templates it creates all files that are needed to run BeakerLib test.

2.3 Test Harness

Test harness is a software framework that automates test execution. It contains tests to be run, executes them and reports results. <expand>

Beaker's harnesses prepare environment for BeakerLib by setting variables to proper values.

2.3.1 Beah harness

Default Beaker harness . <expand> [link to doc]

2.3.2 Restraint harness

Alternative Beaker harness which can, unlike Beah, run with Beaker or standalone without it. <expand> [link to doc]

Relations between Beaker, Harness and BeakerLib is shown in figure 2.1

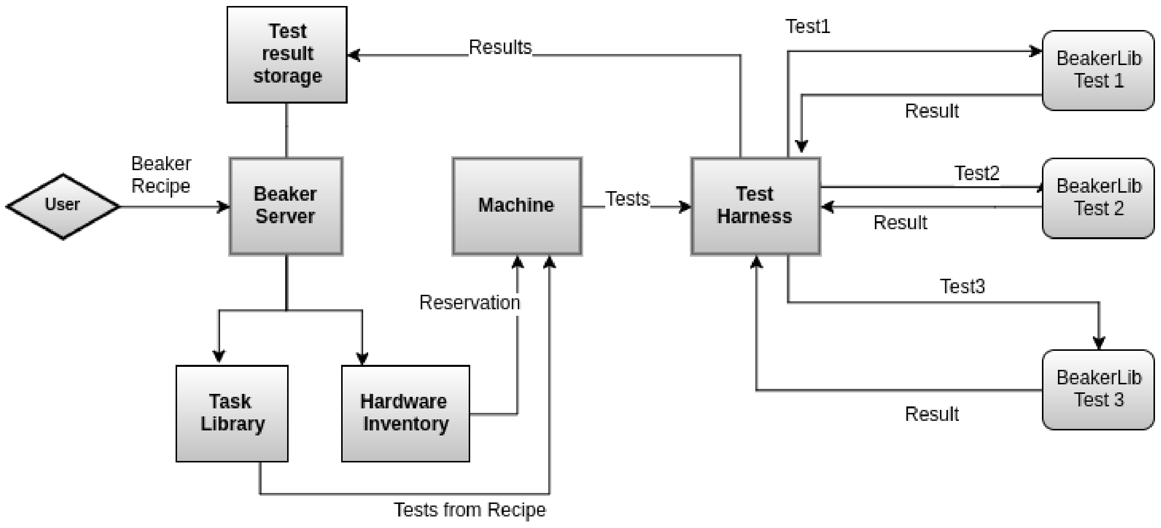

Figure 2.1: Beaker relation to BeakerLib

Chapter 3

BeakerLib

3.1 More in-depth description

3.2 Journal in-depth

3.3 Proposed solutions

3.3.1 Change of xml parser

3.3.2 Change in accessing journalling.py

Queue file solution

Daemon-like solution

Chapter 4

Performance measuring

<Definition of performance measuring>

For performance measuring of BeakerLib I chose two kinds of tests in in two kinds of testing environments.

4.1 Tests

4.1.1 Artificial tests

First type of tests are artificial tests created by me with beaker-wizard* to specifically target and measure performance of journalling modification I made. They consist mostly of commands that directly work with journalling.xml. For example commands rlLog or rlPhaseStart and rlPhaseEnd. That way we can observe clear difference in performance without being affected by operations unrelated to journalling (executing actions that verify functionality of components in real tests).

<description of artificial tests with links to Appendix>

4.1.2 Real tests

Second type are real tests used in Red Hat. These are examples of tests that have been reported to have bad performance with BeakerLib so I am testing them to see if my modifications have real life impact on performance.

<description of real tests>

4.2 Testing Environment

4.2.1 Local

First environment is local laptop for convenience and speed of execution. I measured two parameters, time of execution of each test and memory usage during execution. Tests were run directly, without any harness and with these technical specifications.

<table of tech specs>

4.2.2 Remote in beaker

Second round of testing was done to emulate real testing conditions. I created Beaker recipe containing all performance tests which I submitted to Red Hat beaker server. Only time of execution was measured in remote testing.

<table of tech specs>

4.3 Baseline measurements

<results>

4.4 Implemented optimizations

<results>

Chapter 5

Conclusion

Recap of results
Future work

Appendices