University POLITEHNICA of Bucharest

Automatic Control and Computers Faculty,
Computer Science and Engineering Department

BACHELOR THESIS

vmchecker - enhancement and scalability

Scientific Adviser:
As. Dr. Ing. Răzvan Deaconescu

Author:
Valentin Gosu

Bucharest, 2012

Abstract

The testing and grading of students' programming assignments is a process that requires a lot of time, but also a similar testing environment and an objective individual to do the testing. Vmchecker is a system that allows the automated testing and grading of programming assignments with little human involvement.

This project is an attempt to improve the system, by expanding the types of testing environments and making the process scalable by distributing tasks between several workstations.

Contents

Abstract ii

1 Introduction 1
- 1.1 Brief history of vmchecker . . . 1
  - 1.1.1 Motivation . . . 1
  - 1.1.2 Development and support . . . 2
  - 1.1.3 Features . . . 2
- 1.2 The architecture of vmchecker . . . 2
  - 1.2.1 The storer . . . 3
  - 1.2.2 The tester . . . 4
  - 1.2.3 Storer - Tester interaction . . . 5
  - 1.2.4 Installation . . . 5
- 1.3 Analysis of the shortcomings of the former architecture . . . 5
  - 1.3.1 The virtualization environments . . . 6
  - 1.3.2 SSH keys and persmissions . . . 6
  - 1.3.3 Scalability . . . 6
- 1.4 Improving the existing model . . . 6

2 Extending the types of virtualization 8
- 2.1 LXC . . . 8
- 2.2 KVM . . . 8
- 2.3 VMware . . . 8
- 2.4 vmchecker's executor . . . 8

3 Redesigning the communication model 9
- 3.1 Overview . . . 9
- 3.2 Authentication and security . . . 9
- 3.3 Daemon's implementation . . . 9

4 Load balancing and self-management 10
- 4.1 Estimating the evaluation time . . . 10
- 4.2 Choosing the best machine . . . 10

5 Testing and evaluation 11

6 Conclusion 12

7 Further development 13

A Project Build System Makefiles 14
- A.1 Makefile.test . . . 14

List of Figures

1.1 Simple structure 2
1.2 Storer structure 3
1.3 Repo structure 4
1.4 Tester structure 4

Notations and Abbreviations

procfs – proc filesystem

Chapter 1

Introduction

1.1 Brief history of vmchecker

1.1.1 Motivation

For students in Computer Science classes, programming assignments are the main way in which they gain practical experience regarding certain concepts and technologies. These assignments are also an important opportunity to receive feedback and might also count for the student's final grade.

Student assignments are generally easy to assess. The process consists of downloading the student's submission, running a number of tests to check that it compiles and gives the right output and finally reviewing the source code to make sure that no obvious mistakes were made.

While the whole process is not a difficult one, it quickly becomes a tedious task when you need to assess more than 20 student assignments. Also, when a large number of assignments need to be tested, they are generally divided between teaching assistants who do perform the tasks individually. Considering each of them might test the assignments in a different environment, grading fairness and consistency is eventually lost.

Another issue with the classical aproach to grading programming assignment is that while providing the student with feedback upon the work he has done is one of the main objectives, that feedback often arrives too late. Homework is usually graded days or weeks after completion, so what would have been valuable advice during the development phase, is now too little too late.

Apart from the code-review stage of assesing a student submission, the process is fairly easy to automate. Also, while an automated system may not provide feedback on the quality of the submitted code, it could provide the student with immediate information regarding the behaviour of his homework given certain input data.

In order to automate the testing process as much as possible, the **vmchecker** project was initiated by the Computer Science department at the Politehnica University of Bucharest.

This project aims to improve the performance of the vmchecker automated grading system by adding support for multiple virtualization types, simplifying its design and by insuring its scalability.

1.1.2 Development and support

TODO:
Info

1.1.3 Features

TODO:
Features

1.2 The architecture of vmchecker

The vmchecker automated grading system consists of two separate subsystems:

1. The storer
2. The tester

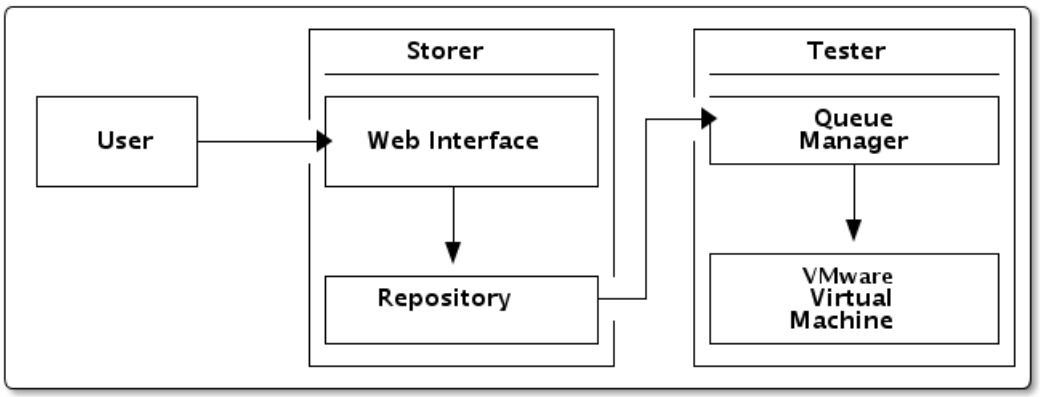

Figure 1.1: Simple structure

The storer subsystem runs an webserver that allow the user to upload an archive containing the user's homework assignment. This archive is stored in a repository, along with the results of the homework's evaluation.

The tester runs a queue manager, which awaits tasks from the storer and upon receiving them proceeds to powering on a virtual machine, compiling the submitted source code, running certain tests and uploading those tests to the storer.

1.2.1 The storer

The storer is basically a Linux running machine that runs an Apache webserver in order to present the student with a web-interface so he can upload his assignment. As well as running the said webserver, the storer also hosts a repository of containing all of the student submissions, organized by course, assignment, student name and submission date. This is done in order to prevent the accidental overwriting of a submission.

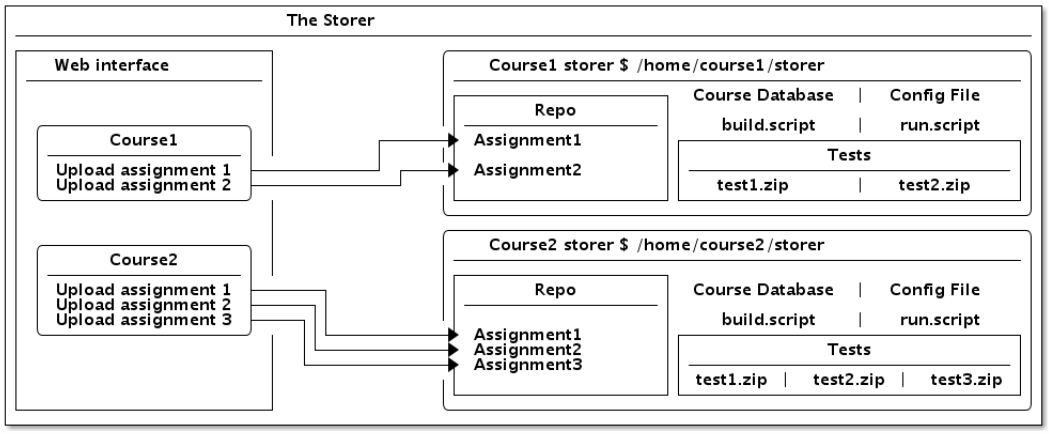

Figure 1.2: Storer structure

The storer machine may host repositories for multiple courses. The storer must insure that only the authorized individuals have access to each repository. This is done by creating a separate linux user account to each course, and placing the repository inside the user's home.

Access to the machine is provided by SSH and authorization is done by placing the public keys of the account's users in the authorized_keys file.

The course's storer folder contains a config file, which lists the parameters for each assignment (such as deadline, title, virtual machine name), the configuration for each available tester (ip or url, username, queue path) and the configuration for each virtual machine.

Also in course's storer you may find a tests/ folder, which contains an archive with each assignment's tests, several scripts to build and run each assignment, a database file, and a repo/ folder.

The repo/ folder contains every submission made to this course. It is organized by assignment, student name and subission date. A symbolic link is made to the most recent submission. The submission folder holds the original archive uploaded by the student, a git/ folder with the unzipped contents of the archive and a file named submission-config which lists the submission's description, and a results folder which holds the output from the testing process.

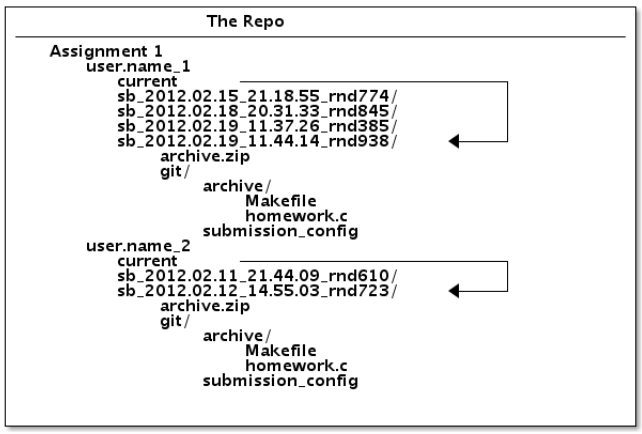

Figure 1.3: Repo structure

1.2.2 The tester

The tester is a separate Linux running machine which hass, just like the storer, a separate user account for each course. A process called queue-manager waits for file system events in the queue/ folder. The storer uploads a new archive to the folder which consists of the student's submission, the tests, and the necessary scripts to build and run the files. When a new archive is uploaded, the queue-manager process extracts the contents to a folder and runs the executor.

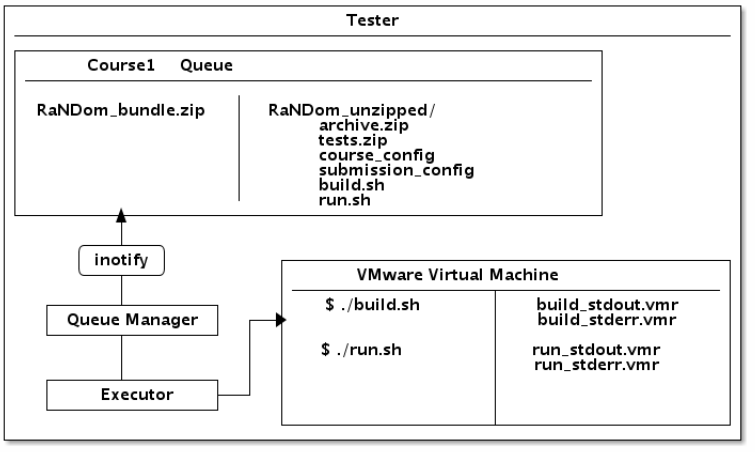

Figure 1.4: Tester structure

The executor is a script that handles the virtual machine specific interactions. It powers on the virtual machine, it uploads the files, runs the build script and the necessary tests. The test results are contained in .vmr files. The executor then copies the result files to the bundle folder, shuts down the virtual machine, then uploads the results to the storer, in the folder designated by the submission-config file.

1.2.3 Storer - Tester interaction

Communication and file transfer between the storer and the tester is done through the SSH protocol.

Since the archived bundle is uploaded from the storer by the www-data user (which runs the apache server) it needs to have permission to read every course's private key. Permission is given by using the Posix ACL file permission system.

After the testing is completed, the queue-manager uploads the results to the storer using the path listed in the submission-config file. queue-manager is run by the course's tester account so it uses by default the account's private key to upload the files.

1.2.4 Installation

Installing the vmchecker system is a pretty complicated process. On the storer system, one simply needs to run the setup.py script, and then the vmchecker-init-course script, with the parameter storer. The last script initializes a new storer folder, with a default config file and an empty database.

The next step is generating the public/private keys and set the appropriate persmissions so that the www-data account can read them. Then add the appropriate settings for the assignments, tester and virtual machines to the config file. Keep in mind that the tester and virtual machines have not been set up yet.

The web interface is set up by creating a new user accound called vmchecker, setting up a public_-html folder in its home and placing all the necessary files to this location. The next step is creating a new website configuration at /etc/apache2/sites-available/

Finally, you need to edit the global configuration file at /etc/vmchecker/config.list so the course will be listed in the web interface.

To install the tester a few extra steps are needed. vmchecker uses the VMware-VIX API to interact with the virtual machines. So you first need to install VMware server or WMware wokstation and the VMware-VIX package, then install the pyVix wrapper created by Lucian Grijincu.

The next step is to run the setup.py script, then the vmchecker-init-course script, with the parameter tester. The script will initialize the folder with a config file, and queue/ and tmpunzip/ folders.

The queue-manager script is then run.

1.3 Analysis of the shortcomings of the former architecture

While it is a very useful and powerful tool, vmchecker has its share of drawbacks. These are caused by the way it was developed - in an incremental way, tackling each issue as it came up.

While rebooting the entire project might be the right course of action, backwards compatibility would be lost and the workload would proove monumental. We arrive to the conclusion that for the time being it's best to improve upon the existing model, while focusing on the major areas of concern.

1.3.1 The virtualization environments

From its inception the only virtualization environment supported by vmchecker has been VMware. This aspect leads to several flaws within the architecture.

First of all VMware-VIX API is needed to interact with the virtual machines. While there is only official support for the C language, a python wrapper is used to allow its integration with vmchecker's modules. It's needless to say that any modification to the API might break existing functionality and maintaining the python wrapper becomes an additional concern.

Beside the fact that it's very difficult to install on a Linux operating system using VMware-VIX API also has some requirements for the hosted virtual machines: VMware tools must be installed and running inside the guest in order to communicate with the virtual machine.

Another issue with VMware is the long startup time and the fact that it isn't immediately responsive following the restauration of a snapshot.

1.3.2 SSH keys and persmissions

An important issue with vmchecker is the ease of setting up the tester and storer for a new course. At the moment the process is impeded by the fact that communication is done through the SSH protocol using public key authentication.

A new pair of keys needs to be generated, for both the storer and the tester account, then the appropiate permissions need to be set so that www-data has access to the keys.

1.3.3 Scalability

The existing support for a distribuited model in vmchecker is limited. While you may have multiple testers associated with the same storer, an assignment can only use one of those testers. While this is useful, since many courses may share the same storer, it also means that at times you'll have machines that are sitting idle, and machines that have a queue of over 20 submissions to be tested.

Also, in cases where the tester is a multicore machine, multiple virtual machines may be run at the same time. However, because of the current architecture of vmchecker, the extra performance of the host goes to waste.

In these cases, a dynamical way of distributing the workload between testers is needed.

1.4 Improving the existing model

This project aims to improve the performance of vmchecker.

All of the concerns listed in the previous section will be tackled as follows:

* Support for new virtualization platforms will be added, while keeping support for VMware. This will ensure a smooth transition and the new version of vmchecker will be backwards compatible.

* In order to eliminate the need for SSH keys for each account the communication model will be modified, such that a daemon process running on the storer will deal with the communication with the tester machines. The daemon process will also be the one to fetch the results from the tester machine, thus eliminating the need for the tester to know the storer's public keys.

* The queue-manager service will be able to provide status updates to the daemon, so it will be aware of the number and capabilities of all the available virtual machines. A load balancing service will be provided by the storer daemon, which will be able to properly place the submitted assignment in the least used queue.

Chapter 2

Extending the types of virtualization

TODO:
Write Intro chapter

2.1 LXC

2.2 KVM

2.3 VMware

2.4 vmchecker's executor

Chapter 3

Redesigning the communication model

TODO:
Write Intro chapter

3.1 Overview

3.2 Authentication and security

3.3 Daemon's implementation

Chapter 4

Load balancing and self-management

TODO:
Write Intro chapter

4.1 Estimating the evaluation time

4.2 Choosing the best machine

Chapter 5

Testing and evaluation

TODO:
Write Intro chapter

Chapter 6

Conclusion

TODO:
Write Intro chapter

Chapter 7

Further development

TODO:
Write Intro chapter

Appendix A

Project Build System Makefiles

A.1 Makefile.test

```
1  # Makefile containing targets specific to testing
2
3  TEST_CASE_SPEC_FILE=full_test_spec.odt
4  API_COVERAGE_FILE=api_coverage.csv
5  REQUIREMENTS_COVERAGE_FILE=requirements_coverage.csv
6  TEST_REPORT_FILE=test_report.odt
7
8
9  # Test Case Specification targets
10
11 .PHONY: full_spec
12 full_spec: $(TEST_CASE_SPEC_FILE)
13         @echo
14         @echo "Generated␣full␣Test␣Case␣Specification␣into␣\"$^\""
15         @echo "Please␣remove␣manually␣the␣generated␣file."
16
17 .PHONY: $(TEST_CASE_SPEC_FILE)
18 $(TEST_CASE_SPEC_FILE):
19         $(TEST_ROOT)/common/tools/generate_all_spec.py --format=odt
               -o $@ $(TEST_ROOT)/functional-tests $(TEST_ROOT)/
               performance-tests $(TEST_ROOT)/robustness-tests
20 #
21
22 # ...
```

Listing A.1: Testing Targets Makefile (Makefile.test)

Bibliography

[1] Argparse tutorial - python. http://docs.python.org/dev/howto/argparse.html#id1. [Online; accessed 13-06-2012].

[2] Creating a read/write proc entry. http://tuxthink.blogspot.ro/2011/02/creating-readwrite-proc-entry.html. [Online; accessed 22-05-2012].

[3] Creating and using proc files. http://www.rt-embedded.com/blog/archives/creating-and-using-proc-files/. [Online; accessed 22-05-2012].

[4] How to compile linux kernel. http://elf.cs.pub.ro/so2/wiki/laboratoare/lab_compilare. [Online; accessed 20-03-2012].

[5] How to use stacktrace. http://stackoverflow.com/questions/5863122/how-to-include-c-backtrace-in-a-kernel-module-code. [Online; accessed 29-04-2012].

[6] The kernel newbie corner: Kernel debugging with proc "sequence" files–part 1. https://www.linux.com/learn/linux-career-center/37985-the-kernel-newbie-corner-kernel-debugging-using-proc-qsequenceq-files-part-1. [Online; accessed 30-05-2012].

[7] The kernel newbie corner: Kernel debugging with proc "sequence" files–part 2. https://www.linux.com/learn/linux-career-center/39972-kernel-debugging-with-proc-qsequenceq-files-part-2-of-3. [Online; accessed 30-05-2012].

[8] The kernel newbie corner: Kernel debugging with proc "sequence" files–part 3. https://www.linux.com/learn/linux-career-center/44184-the-kernel-newbie-corner-kernel-debugging-with-proc-qsequenceq-files-part-3. [Online; accessed 30-05-2012].

[9] Kobjecs, ksets and subsystem. http://www.makelinux.net/ldd3/chp-14-sect-1. [Online; accessed 28-02-2012].

[10] Sysfs and kobjects. http://www.win.tue.nl/~aeb/linux/lk/lk-13.html. [Online; accessed 27-02-2012].

[11] The zen of kobjects. http://lwn.net/Articles/51437/. [Online; accessed 26-02-2012].

[12] Greg Kroah-Hartman. kobjects and krefs. http://www.kroah.com/linux/talks/ols_2004_kref_paper/Reprint-Kroah-Hartman-OLS2004.pdf. [Online; accessed 27-02-2012].

[13] Paul E. McKenney. Overview of linux-kernel reference counting. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2007/n2167.pdf. [Online; accessed 28-02-2012].

[14] Paul E. McKenney. What is rcu? part2: Usage. http://lwn.net/Articles/263130/. [Online; accessed 20-05-2012].

[15] Corey Minyard. How to use kref structure. http://marcbug.scc-dc.com/svn/repository/trunk/linuxkernel/linux-2.6.16-mcemu/Documentation/kref.txt. [Online; accessed 28-02-2012].

[16] Jonathan Walpole Paul E. McKenney. What is rcu, fundamentally? http://lwn.net/Articles/262464/. [Online; accessed 20-05-2012].

[17] Gilles Muller Suman Saha, Julia Lawall. Finding resource-release omission faults in linux. http://pagesperso-systeme.lip6.fr/Suman.Saha/src/plos11.pdf. [Online; accessed 27-02-2012].