Recompute: A Web Application for Recomputation

Chi-Jui Wu

August 2015

Abstract

Recomputation is the process that creates a working environment for a piece of running software, so that software can be referenced, maintained, and used in the future. Recompute is a tool which makes the recomputation process automatic.

Dedication

To Ian.

Declaration

This document is xxx,xxx words long.

Acknowledgements

I want to thank Ian Gent, John McDermott, Lars Kotthoff, and Christopher Jefferson.

Contents

Chapter 1

Introduction

Recompute is a summer project under the supervision of Professor Ian Gent, for `recomputation.org`. The aim is to develop a tool for software recomputation and citation. It will preserve both the state of the software and the environment for future evaluation and reference. The technology should allow any piece of software, along with the source code, to be discoverable and executable.

The system consists of a local web application, and its primary task is to build a piece of software from source inside a new, portable virtual machine. The application front-end will allow users to browse recomputed prpojects and use the vms through a web-based terminal. The project is publicly available on GitHub at `https://github.com/cjw-charleswu/Recompute`.

The project is founded on the concept of software citation and discoverability. Software citation and software discoverability suggest that any software should have a unique and persistent identifier, so that it can be referenced when required. Each version of the software will exist inside a separate virtual machine. Not only can developers maintain, run, and cite that version of the software, other users can discuss the software with respect to that virtual machine.

Recomputation, or in broder terms, reusability of software, is rather complex. A piece of software may be a part of a larger component system, have parallel executions, or use external devices. It is difficult to satisfiy all of requirements inside one or many virtual machines, at least not in the scope of the current project. The current system only considers self-contained software, such as a framework or library, and it aims to provide the recomputation functionality in a simple procedure.

The report will describe the steps to run Recompute the web application, along with some recomputation examples. It will provide a technical overview of the system design and implementation. Lastly, limitations and future work will be discussed.

Chapter 2

Background

The building block of recomputation is resolving dependencies and building the software from source. Many build tools, such as make, Maven, and Cabal, offer such feature. In addition, continuous integration tools like Jenkins and Travis CI provide automated builds to ensure that there is a working piece of software. The current system uses Travis script and other tools in the process of recomputation.

Virtual machine is the most promising medium to realize recomputation, according to the Recomputation Manifesto [1]. The solution is feasible due to cheap computer storage today. With virtual machines, one can be assured that there is a working machine which contains the specific software. The state of the environment is preserved, and the software is guaranteed to run every time in that environment.

The current system is the only tool that offers the service of packaging a piece of software into a virtual machine. However, it is still heavily dependent on two technologies, namely VirtualBox and Vagrant. Section 2.1 and Section 2.2 briefly describes each technology and how it is incorporated into the current system.

2.1 VirtualBox

VirtualBox is a free virtualization software that allows users to run virtual machines. Another paid alternative would be VMWare. Recompute uses VirtualBox as the default virtual machine provider, because it will allow anyone to try the current system without committing to a full paid license from other alternatives. In the future, the system will have the option to build virtual machines for other providers.

2.2 Vagrant

Vagrant is a free tool for creating and running virtual machines for various virtualization software, including VirtualBox. It provides users the ability to provision and access the virtual machine through the terminal with simple commands. To recompute a piece of software, the system will initially produce a Vagrantfile. Then, the system uses Vagrant to create a virtual machine with all the software requirements included in the recipe, or the Vagrantfile. Lastly, again with Vagrant, the system will package the virutal machine into a standalone box to be distributed and version-controlled.

Chapter 3

Installation

3.1 Prerequisites

The system requires Python 2.7x, Vagrant, and VirtualBox. Please download these software for your system. The application was developed on Fedora 21 with Python 2.7.8, Vagrant 1.7.2, and VirtualBox 4.3.28, and was tested most extensively on this setup. You are welcome to report any issues or bugs to the GitHub project (link shown below).

3.2 Downloading Recompute

You can clone the project from GitHub.

```
$ git clone https://github.com/cjw-charleswu/Recompute.git
```

3.3 Running the Server

1. Install pip.
   On Debian and Ubuntu:

   ```
   $ sudo apt-get install python-pip
   ```

   On Fedora:

   ```
   $ sudo yum install python-pip
   ```

   On Mac:

   ```
   $ sudo easy_install pip
   ```

2. Install virtualenv.

```
$ sudo pip install virtualenv
```

3. Create a virtual environment with virtualenv.

   ```
   $ cd /path/to/Recompute/
   $ virtualenv venv
   ```

4. Enter the virtual environment.

   In Bash:

   ```
   $ source venv/bin/activate
   ```

   In Windows Command Prompt (or Powershell):

   ```
   $ ./venv/Scripts/activate
   ```

5. Install all system dependencies.

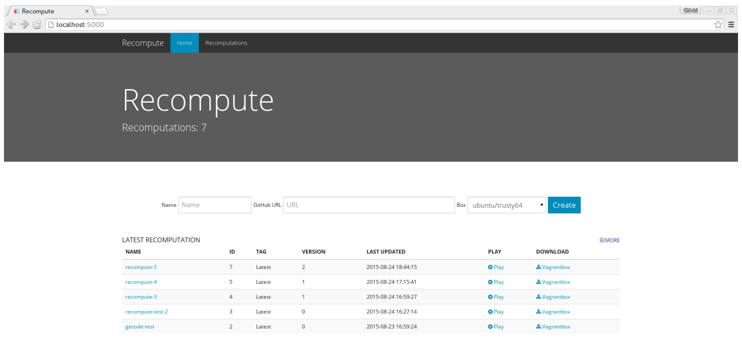
   ```
   (venv) $ pip install -r requirements.txt
   ```

6. Start the server.

   ```
   (venv) $ python run --port=[port]
   ```

3.4 Testing

The server is running correctly if your web browser can reach “localhost:[port]”. Figure 3.1 shows the web application’s homepage.



Recompute

Recomputations: 7

Box: ubuntu/trusty64

LATEST RECOMPUTATION

NAME	ID	TAG	VERSION	LAST UPDATED	PLAY	DOWNLOAD
recompute-5	7	Latest	2	2015-08-24 18:44:15	Play	Vagrantbox
recompute-4	5	Latest	1	2015-08-24 17:15:41	Play	Vagrantbox
recompute-3	4	Latest	1	2015-08-24 16:59:27	Play	Vagrantbox
recompute-test-2	3	Latest	0	2015-08-24 16:27:14	Play	Vagrantbox
gecode-test	2	Latest	0	2015-08-23 16:59:24	Play	Vagrantbox

Figure 3.1: A screenshot of Recompute's homepage

Chapter 4

Usage

Read Section 3.3 about running the server before proceeding to the examples.

4.1 Example: Recompute

Recomptue is the current system written in Python. Visit `localhost:[port]`. Enter the name “recompute” and url `https://github.com/cjw-charleswu/Recompute`. Click the “Create” button.

4.2 Example: gecode

gecode is a constraint programming solver written in C++. Visit `localhost:[port]`. Enter the name “gecode” and the url `https://github.com/ampl/gecode`. Click the “Create” button.

4.3 Example: Idris

Idris is a dependently typed functional programming language. Visit `localhost:[port]`. Enter the name “idris” and the url `https://github.com/idris-lang/Idris-dev`. Click the “Create” button.

4.4 Browse Recomputations

Visit `localhost:[port]/recomputations`. You may search a recomputation by name.

4.5 Individual Recomputations

Visit `localhost:[port]/<recomputationname>`. You may update (rebuild) the current recomputation and view the build history.

Chapter 5

Design

5.1 Requirements

The software specification of the current system is as follows:

1. Intuitive, simple user interface to run recomputation
2. To recompute a project given its GitHub URL, in the form of a virtual machine
3. A landing page for every recomputation/software
4. A persistent identifier for every recomputation/software
5. A play button to access the virtual machine as well as to run programs and experiments

5.2 System Architecture

The system is a Python web application with a front-end interface and RESTful API for running recomputation. The current system is a minimal viable product with a single-thread execution. It is dependent on two external software, namely VirtualBox and Vagrant.

Chapter 6

Implementation

6.1 Server

The server is a integrated Python Tornado and Flask server. The system uses Flask, becauses it is a lightweight web framework. Tornado is introduced to the software stack for its asynchronous I/O and socket library.

6.2 Terminal

The in-browser terminal is based on term.js and Tornado sockets. Every terminal window spawns a pseudo terminal in the server. The input to the terminal is piped to that pseudo terminal, and vice versa. The terminal uses the shell that Vagrant creates when making a ssh connection to the virtual machine.

6.3 Recomputation

The web application takes a name, GitHub URL, and the name of the base virtual machine as input and returns a virtual machine containing the required software, if recomputation were successful. The application follows a sequence of steps to achieve the goal. Initially, it identifies the components required to build the software and the virtual machine. Then, it builds the software and the virtual machine. After building, it packages the virtual machine. Eventually, it writes a summary of the recomputation to a file.

6.3.1 Identifying

The program needs to identify the following components of a software:

- Name
- Base virtual machine

- Programming language
- Software dependencies
- Install scripts
- Build scripts
- Test scripts

Identifying the above components requires web scraping of the given GitHub webpage. However, sometimes it is not enough. The system assumes that there is a Travis script inside the GitHub repository. A Travis script is a recipe to build a software. If there is no Travis script, the system will still attempt to build the software, but the likelihood of success will be much smaller.

The system has a default Vagrantfile for every programming language it recognizes. Currently, it supports Python, Node.js, C, and C++. Each default Vagrantfile contains the names of the packages that are required when building software based on that language. When Vagrant uses the Vagrantfile to create a virtual machine, it will install all the included packages. Furthermore, the program will add extra information to the Vagrantfile as it parses the Travis script. For instance, it will add additional apt-get repositories that the default Vagrantfile lacks.

When the system has gathered all the useful information from the GitHub webpage., or eseentially the Travis script, it will create a new directory along with the newly generated Vagrantfile.

6.3.2 Building

Vagrant makes creating virtual machines a trivial task. Inside the recomputation directory containing the Vagrantfile, the command "vagrant up –provision" will start to create a new virtual machine.

6.3.3 Packaging

Packaging the virtual machine is also trivial with the command "vagrant package –output name". If the virtual machine were created successfully, the system would add the box to the Vagrant repository and move it to a new directory as the storage location for a version of that software. The web application front-end will be updated to display the virtual machine.

6.3.4 Versioning

The user may build different versions of a software through the web application. As mentioned earlier, each version is contained within a directory, consisting the virtual machine and the Vagrantfile used to creat that machine. The main recomputation directory also contains a JSON file defining the software's id, name, GitHub URL, and summary of each build history.

6.3.5 Downloading

At last, the web application will prompt the browser to download the recomputed box. It will also display error messages if recomputation was unsuccessful.

Chapter 7

Limitations

There is no guarantee that the system will recompute any project due to unknown dependencies or build instructions. The current system only recognizes the following programming languages: Python, C, C++, Haskell, Go, and Node.js. It also does not check whether the software is working as intended and whether the tests have passed successfully.

A major issue with the current system is the lack of testing and evaluation. There are no metrics to show how well the system performs, in terms of recomputability, under different scenarios.

Chapter 8

Future Work

There are multiple directions for future work.

- The server would scale to run multiple recomputations at the same time
- The system would learn to build complex projects
- The system would make it easy to reproduce experiments and display the results
- The system would reecompute and run distributed systems
- The system would simulate mobile devices and sensors
- A database for recomputations

Chapter 9

Conclusion

The current system recomputes an existing project into a virtual machine. The recomputation history of that project is preserved. The web application front-end provides a convenient way to run recomputation and browse past recomputations. It also allows the user the access the virtual machine through a web-based terminal.

Bibliography

[1] Ian Gent. The recomputation manifesto, July 2013. http://www.software.ac.uk/blog/2013-07-09-recomputation-manifesto.