Recompute: A Web Application for Recomputation

Chi-Jui Wu

September 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.

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 projects 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 broader 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 satisfy 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 virtual machine into a standalone box to be distributed and version-controlled.

Chapter 3

Installation

See README.md

Chapter 4

Usage

Read README.md about installing and running the server before proceeding to this section.

4.1 UI

The current user interface is a web application (see Figure 4.1). The server provides a RESTful service for its functionalities, therefore, it is possible to develop the application further to provide a command line user interface or a desktop application interface.

Figure 4.1: The Recompute application front end

4.1.1 Recomputation

When the user runs a recomputation, a new window will appear, showing the progress of the current recomputation (see Figure 4.2). If the user wants to view the progress after the window is closed, they can open the browser console to see it. The shortcut for opening the console on Google Chrom is Ctrl+Shift+J.

Figure 4.2: The Recompute progress window

A message box will appear when the recomputation is finished (see Figure 4.3).

Figure 4.3: The Recompute mesasge box

4.1.2 Recomputation page

Each recomputation has its own landing page (see Figure 4.4). The webpage contains essential information such as the software unique identifier, its GitHubl URL and description, and a history of different versions of the software.

Figure 4.4: The recomputation landing page

4.1.3 The play button

The play button (see Figure 4.5).

Figure 4.5: The browser terminal which gives user access to the virtual machine via the play button. The figure shows the virtual machine is running Idris, a functional programming language.

4.1.4 Recomputations list

The web application also allows the user the research a recomputed software by name (see Figure 4.6). It can be extended to support search by id and by key words.

Figure 4.6: The list of recomputations currently available on the application

4.2 Running

4.2.1 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.

4.2.2 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.

4.2.3 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.

Chapter 5

Use Cases

Recompute is only a MVP (minimal viable product) at its current form. However, we think there are real-life use cases with the current system.

5.1 Recomputation

Version control tools keep track of changes in code, but Recompute allows users to access the virtual machine, the development environment, that contains a specific version of the software. Virtual machines give people opportunties to share a piece of software without worrying about dependencies and compatibility. Perhaps there are specific features, or even bugs, that only exist in certain versions of the software.

5.2 Reproducible experiments

Software experiments, including testing, evaluation, and statistical analysis, can be run in self-contained virtual machines. There are limitations with the current system (see Chapter), but it provides a promising platform for making reproducible and accessible experiments.

5.3 Education and Research

Becausing running recomputation is almost effortless with Recompute, it may be useful for professors or researchers to package a piece of softwar for education or research purposes. Students may work on assignments in these virtual machines that contain a specfic environment or software.

Chapter 6

Design

6.1 Requirements

The software specification of the current system is as follows:

1. An intuitive, simple user interface to run recomputation, use a GitHub URL as input
2. The output of recomputation is a virtual machine
3. A landing page for every recomputation/software, which contains its persistent identifier
4. A play button to access the virtual machine
5. The functionalitity to recompute different versions of the software

6.2 System Architecture

The system consists of a RESTFul server. It handles recomputation requests and creates virtual machine terminals asynchronously, thus allowing it to respond to multiple requests at the same time. As mentioned earlier, the current system depends on VirtualBox and Vagrant for creating and accessing virtual machines. The system fetches the software from GitHub, and it heavily depends on the structure of Travis (continuous integration service) script to build the software inside a new virtual machine.

Chapter 7

Implementation

7.1 Server

The web application is a RESTful server written in Python, with Tornado the web framework. Tornado was chosen because of its ease of running asynchronous IO tasks and integrated websocket library.

7.2 Recomputation

The web application displays a HTML form which takes a name, GitHub URL, and the name of the base virtual machine as input and returns a virtual machine containing the specified software. The form validation is done by Tornado wtforms. The server will return the appropriate HTTP response whether the recomputation was successful, along with a text message briefly explaining the outcome. The front end will display the message in a draggable box when it receives a response. The application follows a sequence of steps to recompute a piece of software. Initially, it identifies the software dependencies. Then, it tries to build the software and the virtual machine. After building, it packages the virtual machine into a standalone VirtualBox image. The contents of the recomputation, along with a description file in JSON format, are stored in a separate directory.

7.2.1 Identifying

The system identifies the following components before recomputing the software:

- Name
- Base virtual machine
- Programming language
- Software dependencies

- Install scripts
- Build scripts
- Test scripts

The system accomplishes this by scraping the given GitHub webpage. The system also attempts to fetch a Travis script inside the GitHub repository. A Travis script is a recipe for building a software. The system will still attempt to build the software even if the software does not contain a TRavis script, but it is less likely to succeed.

The system has a default Vagrantfile for every programming language it recognizes. Currently, it supports Python, Node.js, C, and C++, Haskell, and Golang. Each default Vagrantfile contains the names of the packages required when building software written in that language. Vagrant will provision the new virtual machine with the dependencies when it starts creating it. Furthermore, the system will add contents to the Vagrantfile as it parses the Travis script, including adding additional apt-get repositories the specific software required.

7.2.2 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. As this is a long process depending on the size of the software, the server handles the recompute request inside a Tornado coroutine, or an asynchronous function. As a result, the recomputation process does not block the server, allowing it to handle other requests at the same time.

7.2.3 Packaging

Packaging the virtual machine is also trivial with Vagrant, via the command "vagrant package –output name". If the virtual machine were created successfully, the system will respond with a successful HTTP response, and the front page will be updated with a new recomputation. If the recomputation process was unsuccessful, the front end will display a error message and a link to download the log.

7.2.4 Versioning

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

7.3 Terminal

The in-browser terminal is based on term.js, and it communicates with the server via Tornado web sockets. Every terminal window spawns a pseudo terminal (ptyprocess) in the server. The input to the terminal on the browser is piped to the corresponding pseudo terminal, and vice versa for the output. The terminal on the server side is spawned with a "vagrant ssh" command, which gives access to the virtual machine. The terminal process is also used inside a Tornado coroutine, hence asynchronous.

7.4 Front end

The front end was built with Bootstrap, jQuery UI, and FontAwesome. The design and user interface aspects of the web application were not the primary concerns of the project, except for the ease of recomputation, which could be done in one click. The progress window which shows the progress of recomputation was implemented with web sockets, in much the same way as the in-browser terminal.

Chapter 8

Code

This chapter briefly describes the project code structure and comments on where improvements can be made.

Chapter 9

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 10

Future Work

There are multiple directions for future work.

- A evaluation on the current system
- A database for recomputations
- Support for more architectures and programming languages
- A learning algorithm for building software from source
- Support for reproducible experiments
- Recomputation of distributed systems
- Simulation of mobile devices and sensors and recomputation of experiments involving these software

Chapter 11

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.