Master Project Report

Teakwood: An Web Framework for Handling Many-task Computing

Submitted in partial fulfillment of
the requirements for the degree of

Master in System Science
in
Louisiana State University and Agricultural and Mechanical College
The School of Electrical Engineering and Computer Science
Computer Science and Engineering Division

by

Rui Guo

Under the guidance of
Jian Zhang

Fall Semester 2014

Preface

Four years ago, I know nothing about Linux. I then got a job to deal with HPC(High Performance Computing) systems. Suddenly, I jumped into an Linux environment. I found that all those GUIs(Graphic User Interfaces), which I used to when using Windows OS(Operating System), are gone, and I have to use Linux commands to manipulate those machines and manage my stuff. This makes me uncomfortable and cumbersome. With time goes by, I eventually can work with Linux comfortably. However, I still have the tendency to use GUI when proving me the option.

I found that I am not the only one that had such experience. In my working building, a lot of scientists and researchers are suffering this pain. Not only this, since these researchers usually have to work on cross-platforms, the OS conflict add them many wasteful jobs. For example, scientist A want to share a computing result to researcher B, as B needs it to do the visualization. Let's assume A uses Linux and B uses Windows. What should they do? Normally without tool's help, then need to type commands. Command, command, command... copy, paste, and then command, command ...

Why not create a GUI platform to reduce those repeated typing work and allow them to work cooperatively in one place? With this desire, Here comes the Teakwood framework.

Abstract

Using Linux commands to handle computing jobs can be a hurdle to those scientific researchers who don't have HPC related background. Teakwood provides a solution and beyond. Teakwood is a framework that migrates all the terminal typing work to a web console GUI, and provides user a total control of their jobs, data, computing resources and so on simply by clicking functional buttons. Teakwood is also an open platform that enables user to work cooperatively. Through Teakwood, user can share their models, results, and computing resources within their group and have discussions in Teakwood forum.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Using text-based interface(Unix-like Terminal, TBI) to manipulate computers can be very hard for normal people who don't have computer science related backgroud. Although TBI has some advantages, it have a steep learning curve for people to grasp in a short time. On the other hand, graphic user interface(GUI) provides a WYSIWYG feather, which makes people can directly work on their jobs without learning computer science knowledge first. This Teakwood framework is designed for that purpose.

1.2 Teakwood

Teakwood focuses on HPC jobs submission in computational science. Teakwood is a GUI framework that allows user to summit HPC jobs from Teakwood web console, and to have a full control of their projects. Teakwood migrates all the terminal typing work to Teakwood GUI, and wrap them with interactive web pages. This enables user to submit HPC jobs just by clicking functional buttons. No more terminal commands.
In Teakwood, all user's computing data are hosted in a file server. User can share their result, models and computing resources within their group. This greatly facilitated the collaborative work among users.
Teakwood also provides a project management system for user to organize their project data easily.
As Teakwood is a web framework, user can access from any where, any type of machine, as long as the machine has a browser and the Internet. no more hassle by the OS conflict.

1.3 Features

Functionally, Teakwood has the following features:

• Perfect documentation
Teakwood homepage provides diverse software documentations including installation guide, user manual, developer manual, video tutorial, etc. user can grasp Teakwood soon with document support.

• Neat web portal
A neat LSU style web console makes your job submission simple and easy. Drag, push and click, that's it. Let's say goodbye to terminal typing.

• Job monitor system
Job monitor system provides five labels: "uploading", "queued", "running", "finish", and "Data Ready" for user to read the job status. Job monitor system also periodically pulls the job running messages from computing server and displays them on working console, user may know more details.

• Project management system
All user's project data is well organized and web kept in a file server. user can compare, share, and download them as needed.

• Powerful admin
The powerful admin system is provided by Django itself. With tiny system configuration, user can activate their admin systems and have a top-down control of their accounts, machines, models and so on.

Chapter 2

Design Structure

2.1 Overview

Structurally, like most websites, Teakwood system has a three-layer layout: frontend, backend, and database. The difference is, Teakwood uses computing servers to run HPC jobs, so the eco Teakwood system actually has four layers: the up mentioned three plus a computing layer. the below figure shows how the four-layer structure looks like:

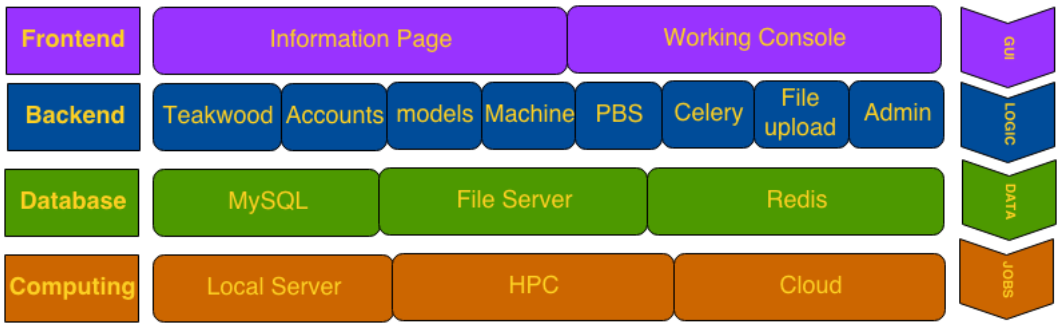

Figure 2.1: Teakwood System Overview

On the above figure, we can see the four straight layers bottom-up. Each layer from left to right: the left part is the layer name; the middle part is the layer content; and the right part is the layer reality. Let's go through these layers one by one.

2.2 Frontend

The frontend is a visible GUI that user interact with. Basically all what we see in the Teakwood website can be called the "frontend". In general, the frontend includes HTML files, JS, CSS, and the static images. For neat purpose, Teakwood separated the frontend into two parts: the **information pages** and the **working console**. See below:

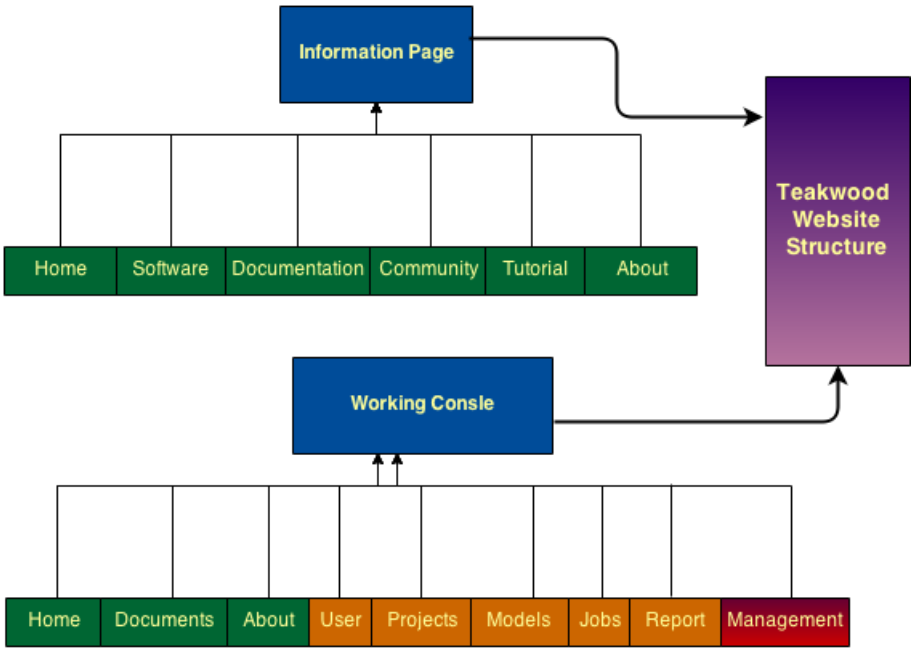

Figure 2.2: Website Strucutre

The **information pages** introduces you all the general information about Teakwood, including:

- What is Teakwood?
- What can Teakwood do?
- How to install Teakwood?
- How to use Teakwood?
- The user manual.
- Video tutorial.
- Teakwood forum.

The **working console** is the place where you actually tango with your projects. The functional buttons will guide you to different web pages for different purposes.

Let me explain these functional buttons one by one.

- **Home**: Directs you to the homepage.
- **Documents**: Directs you to the user manual.
- **About**: Teakood self introduction.
- **User**: Displays user information.
- **Projects**: Creates projects and provides project overview.
- **Models**: Create models and provides models overview.
- **Jobs**:Creates jobs, overview jobs and job monitoring.
- **Report**:Directs you to the output downloading page.
- **Management**:Access to the admin system.

Note the color differences in the bottom layer.

- The **Green**: all visitors can see and manipulate.
- The **orange**: only logging user can see and manipulate.
- The **red**: only superuser can see and manipulate.

For the frontend, I wrote all these HTML files using Teakwood template languages, which I will elaborate later. The easy-navigated Documentation in information pages is generated from reStrcutredText.

2.3 Backend

The backend contains all the logical design of data manipulating and HTML files producing. In detail, it includes verifying user's request, pulling requested data, generating HTML web pages and displaying the web pages. Teakwood separated all the system functions in to loosely coupled parts, each part is a well wrapped, independent application. These applications will work independently or cooperatively in order to achieve user's requests. We will explain the application details in chapter four. This section will only provides a general view.

- **Teakwood**: controls the frontend presentation.
- **Accounts**: provides the registration logic.
- **Models**: invoke and control the computing models.
- **Machine**: invoke and control the computing resources.
- **PBS**: guides the PBS script generation.
- **File upload**: gathers the input files to buffer for uploading.
- **Admin**: provies an overall control of user and data.

• **Celery**: third party app, handle synchronous processing.

Among the eight applications, Admin is provided by system itself; Account and Celery are third party wheels; the rest of the applications are designed by me.

2.4 Data Handling

Teakwood system handles three types of data: the website data, the computing data and the message queue data. For each type of data Teakwood provides a different storage. see this table:

Website data	**Computing data**	**Message queue data**
MySQL	**File server**	**redis server**
Teakwood data	**inputs and outputs**	**Asynchronous handling**

Teakwood uses MySQL database for storing the website data, e.g. the user accounts and the project labels.
For the computing data, Teakwood periodically synchronize them to the file server for backup and downloading purpose, e.g. input files and the output results.
Message queue data are generated when we use Celery to asynchronous process time consuming tasks. Since those data are ephemeral, we simply use a redis server to keep it.

This layer requires a lot of tools installation and system configuration, not too much coding work.

2.5 Connection Configuration

Before the first time we can run a job on HPC or cloud, we have to set up a connection between Teakwood and the remote servers. In detail, we have the following tasks to do.

Server side:
• Establish a password-less SSH login to the server.
• Compile the tools and packages we will use in the remote machine.
• Create system paths and working directories for Teakwood.

• Advance HPC system configuration for acknowledge Teakwood connecting.

Teakwood side:
• Fill the machine information in machine form.
• Fill the executable command in script form.

Once those steps are done, we will have successfully "plugged" Teakwood into remote machines. Now everything we interact with in the terminal will migrate to the Teakwood web portal.

Chapter 3

Models, Views and Templates

Models, views and templates consist the core of Teakwood System. The majority coding in Teakwood happens here. Models generate all the database tables; View provides all the web page functions; and templates generate all the HTML files. In an operational degree of view, models response for communicating with data; View decides which data to display; and templates decorate the data for displaying.

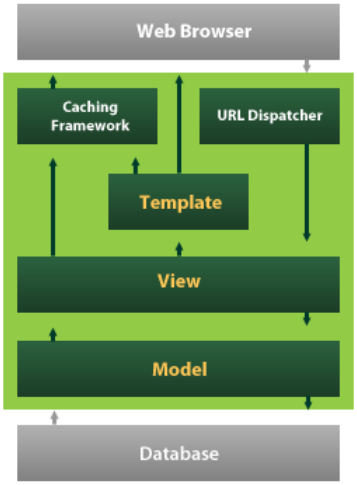

Figure 3.1: Teakwood MTV Framework

As we can see from the figure, Teakwood has a loose coupled modeling design. model, view, and template are separated from each other; each application(The light green square) is also separated from each other. This design makes the Teakwood system flexible to make changes and easy to extend feathers.

3.1 Models and Database

The models are Python objects that describes Teakwood data models/tables. Thanks to the system integrated object relational mapping (ORM) functions, I don't have work with SQL database in order to manipulate data; all we should do is manipulating the corresponding Python objects, ORM will do the mapping work.

For example this section of python code:

```
class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```

will generate a database table like this:

```
CREATE TABLE myapp_person (
    "id" serial NOT NULL PRIMARY KEY,
    "first_name" varchar(30) NOT NULL,
    "last_name" varchar(30) NOT NULL
);
```

So for the database designing what I did is create the "models.py" files and define the corresponding classes for mapping.

3.2 Views functions

The View layer are Python functions that defining Teakwood business logic. For example if I want to show my user information, I have to write a view function to tell the model layer pulling the User data from database I also have to decide which template.html be used for data decorating. As such, I will have to define two logic functions in my views.py file. In general, View layer can be seeing as a bridge between the model and the template.

Take a look at this example in Teakwood job view models.

```
@login_required
def job_view(request, job_id):
    job = get_object_or_404(Job, pk=job_id)

    if request.user == job.user or
    request.user.groups.filter(name=job.group):
```

```
        return render_to_response
        ('simfactory/job_detail.html',
        {'object': job},
        context_instance=RequestContext(request))
    else:
        return redirect(reverse("simfactory_joblist"))
```

This code will execute when I click your job-view button on my job console. It tells the model layer to pull the job information if the user is authenticated, otherwise redirect to job list.

3.3 Template Language

Teakwoods template language is designed to strike a balance between power and ease. It is designed to please both the HTML designer and the Python coder. Teakwood template language is not just HTML files or Python code. Here is an example:

```
{% extends "base_generic.html" %}
{% block title %}{{ section.title }}{% endblock %}
{% block content %}
<h1>{{ section.title }}</h1>
{% for story in story_list %}
<h2><a href="{{ story.get_absolute_url }}">
    {{ story.headline|upper }}
  </a></h2>
<p>{{ story.tease|truncatewords:"100" }}</p>
{% endfor %}
{% endblock %}
```

We can see there are some special symbols in the HTML code. Basically there are three types of special symbols in Teakwood template language: variables, tags and filters.

Variable: When the template engine encounters a variable, it evaluates that variable and replaces it with the result.
Tag : Tags are more complex than variables: Some create text in the output, some control flow by performing loops or logic, and some load external information into the template to be used by later variables. Some tags require beginning and ending tags.

filter:Filter is basically a restricted variable.

With the help of variables and tags, static HTML files become dynamic HTML files and can interact with python code.
Template files in template directory generate all the visible HTML content for Teakwood system. There are roughly 100 template files in Teakwood, all of them have to be predefined and wait for the calling from View.

Chapter 4

Operating Mechanism

4.1 Teakwood Working Flow

So far we've already known that Teakwood follows a "MTV" design pattern and has a loosely coupling design. We also know that Teakwood uses python object to work with database by mapping. Plus, Teakwood template language is powerful for generating HTML files. Now let's join all these feathers to see how Teakwood process a request from user.

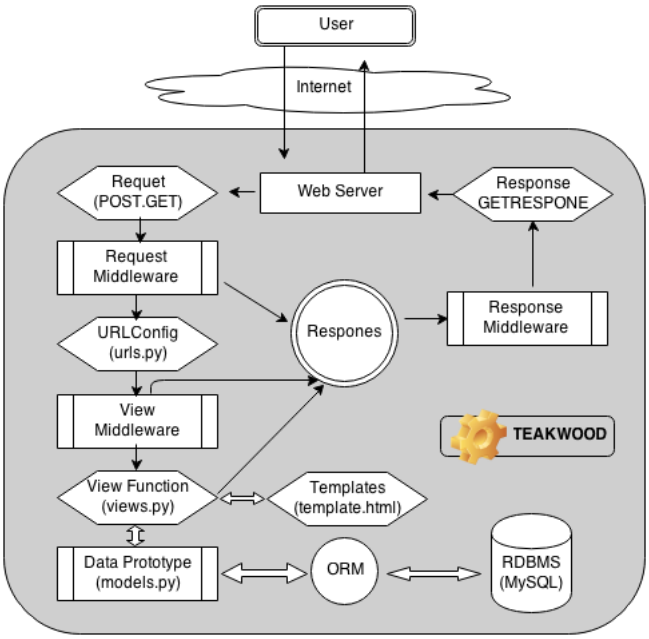

Figure 4.1: Teakwood Request-Response Working Flow

Then, let's put it in real world to see how this structure works. For example, a user want to view all the job status under his/her account.
• Click the job functional button.
• Button triggers a request.
• Request goes to request middleware for identity check, if legal? auth?
• if yes, goes to response and raised an error page;
• if no, goes to urlconfig to locate the right URL.
• URL goes to view middleware for legibility check, if wrong URL, auth?
• if yes, goes to response and raises an error page;
• if no, goes to view function.
• View function check if need data? if need template?
• if need data, trigger models to fetch data from database;
• if need template, find the right template, also fetch css, js and imgs.
• Send all files to response and rendering web page.
• Display the final page to user.

Chapter 5

Installation and Use Case

This chapter teaches user how to install and use Teakwood. Teakwood is designed for scientific researcher's daily use. A simple and compatible folw logic for submitting jobs is important. Teakwood created a maximally generalized working flow to fit most computing tools. Regarding the installation, Teakwood provides a one-execution set up. However, lake of required system components will lead Teakwood malfunctioning. Install system requirements is a necessary.

5.1 System requirements

Teakwood is a Django web framework which integrated a lot of third party packages and external tools; some of the packages or tools require extra libs and development packages to work. Before running Teakwood, we need to resolve all those dependencies as well as finish setting up all the packages and tools. Below is a list of the required installations during the Teakwood development.

• **Project Dependent libs**
As mentioned above, installing all dependent libs are the first step. Please refer to the installation guide in homepage for more information.

• **MySQL Database**
Teakwood uses MySQL database, so make sure mysql-server and mysql-devel is installed. If you want a GUI control of your database, I recommend an installation of phpMyAdmin.

• **Version Control**
Git is a very popular version control tool. Teakwood uses git as version control, and Teakwood source code is hosted in Github.

• **Virtual Environment**
It is highly recommended that we set up an independent working environment for Teakwood development. "Virtualenv" is a reliable solution.

• **Latex and Sphinx**
Teakwood uses sphinx package to generate diverse types of files such as HTML, Latex, and PDF. Hence, sphinx and Latex need to be installed.

• **Celery and RabbitMQ**
This is a solution for resolving synchronous process. install and start them before you run Teakwood.

• **Django 1.4.x**
No need to explain.

• **Third Party Packages**
Teakwood uses a lot of "wheels" to construct the structure. before runing Teakwood, those wheels need to be installed in the right path. Refer "requirements.txt" for the "wheels" list.

5.2 Job Submission

Let's take a look at the classical job submission flow in Teakwood.

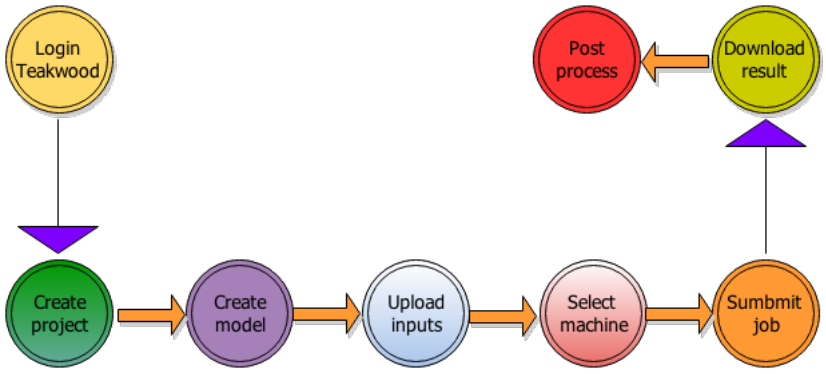

Figure 5.1: Job Submission Flow

On the above figure, each circle represents a different stage:

• **Login**: Only logged user can use the working console, visitors only can see the information page. Login is a necessary.
• **Project**: A project relates to a big problem resolving. A project may use a lot of computing tools(packages), a lot of models and a lot of machines in order to solve this big problem.
• **Model**: A configuration package that guides the computing tool how to use inputs and machines.
• **Inputs**: Data that will be used during the computing.
• **Machine**: Where your job runs.
• **Job**: A job is a single run on a single machine.
• **Result**: Final output of the job run.
• **Post-process**: Refine the output data, visualize the data, validate the data, etc.

This job submission figure gives you the general working flow on how Teakwood works, if you want a step by step hands on guide, please refer to the video tutorial in homepage.

5.3 Job Monitoring

In order to let user have a better view of their job status, Teakwood provides a five- stage monitoring.

• **Uploading**: means your job is sending to the remove machine.
• **Queued**: means your job is accepted and placed in the to-run list.
• **running**: means your job is running now.
• **Finished**: means you finished your job or your hour is used over.
• **Data Ready**: means your data is synchronized to file server and ready for download.

In project console, teakwood also provides a live job monitoring by displaying the running message on console. This can gives user more information about the job status.

5.4 Download Output

Navigate to the report page and download your output result. you've done all.

Chapter 6

Conclusion

The primary goal of this report is to give readers a general overview of Teakwood. The backend logic details is not expand greatly, user can refer to the Teakwood developer manual for details.
The general things like:
What is teakwood?
How is Teakwood like?
What can Teakwood do?
What are the features?
How can I tell that Teakwood is different from other web frameworks?
are the main guidance for writing this report.

This report has a moderate system structure explanation on how the frontend, the backend and the database communicate with each other. Also this report provides tips on how Teakwood connect to the remote servers for submitting jobs.

Teakwood is written in Python, the glue programming language. Python is very popular in mathematical and scientific area; while Teakwood is focus on user experience in scientific computing. This is a good match.

It is good that Teakwood follows a loose coupling design pattern. This makes adding or deleting feathers in Teakwood are simply and easy. with the application wrapping concept, Teakwood has a flexible and neat coding structure. There is no repeat coding, and deploying and detaching an application is easy.

All in all, I hope this report reveals Teakwood well to both scientific researchers and software professionals.

Chapter 7

Future Work

• Docker Hub

Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications. Any software can be wrapped as a docker for cross-platform use. Once a Docker is built, it can plugged into any machine for direct use without any system configuration or compiling or so. The only requirement is Docker is installed on that machine.
Teakwood aims to become a Docker hub, so that all users can contribute their Dockers to Teakwood platform. In the future, models deployment for Teakwood can also be a standard procedure.

• Visualization

Visualization is a challenge for Teakwood because output data are various, while visualization tools only accept structured data. In the future Teakwood will work on creating general data parsers for most popular output types.

• Computing on the go

If your job occasionally requires HPC computing and you don't have private computing resources, then "Computing on the go" is right for you.
This ideal is accounting on Amazon cloud. First deploy your models to an pure AMI(Amazon image), and save it as an new AMI. When you need your models for computation, just launch it from Teakwood. using the new AMI for initialization, then all your models can be used instantly without re-compiling. After your jobs are done, terminate your cloud service and save money.

Acknowledgments

I would like to say thanks to my committee for their support and encouragement: Dr. Jian Zhang, my committee chair; Dr. Jianhua Chen; and Dr. Konstantin Busch. Especially, I want to express my appreciation to Dr. Jian zhang for giving me useful comments on my project design and report written. I offer my sincere appreciation for the learning opportunities provided by my committee members.
My completion of this project could not have been accomplished without the support of my formal job's supervisor Dr.Tao at Center for Computation and Technology at LSU. Thanks for the project suggestions and for providing models and account for me to test my project.
Thanks for all the open sources providers: Django framwork, Celery, RabbitMQ, Bootstrap CSS, Bootstrap JavaScript, Studio MX, FAMFAMFAM icons, Google groups, Youtube, JQuery UI.

References

[1] Django framwork. https://www.djangoproject.com/

[2] Python programming language https://www.python.org/

[3] Celery: Distributed Task Queue. http://www.celeryproject.org/

[4] Sphinx Documentation generator. http://sphinx-doc.org/

[5] Message queue broker. http://www.rabbitmq.com/

[6] Science gateway https://www.xsede.org/

[7] Simulating wave nearshore http://swanmodel.sourceforge.net/

[8] LSU HPC: Supermike II http://www.hpc.lsu.edu/