Introducing The Pardusman Project

A Google Summer of Code 2009 Entry

Today, almost all of us have tried our hands at building a custom OpenSuse distribution, using the latest SuseStudio. The experience of being able to customise a distribution installer to suit one's needs is stupendous. However, have you ever wondered if you could customise the installers for other distros too? Welcome to the Pardusman Project!

Google Summer of Code (GSoC) is one of Google's important programmes that encourages open source development and contribution. This event, first held from May to August 2005, is an annual programme. Through these few months every year, Google awards stipends to hundreds of students who successfully complete free and open source coding projects that have been requested for. The programme is open to students aged 18 and above.

Although it's dubbed the Summer of Code, I would say it's more about passion than code. A piece of code is just a written expression of the passion, anyway.

Being a student developer, it means a lot to us when speaking about our Summer of Code experience. Every year, Google selects around 1,000 student developers, from around the world, assigned to different open source projects. See *code.google.com/soc* for more.

This was my second GSoC. In 2008, I participated in a Fedora project. And in 2009, I worked with the Pardus Linux project. Pardus is a nicely built and user-friendly GNU/Linux distro from Turkey, developed and maintained by TUBITAK (The National Scientific and Technological Research Council of Turkey). My project was to create a Web-based ISO image creation tool for Pardus and I was mentored by Ekin Meroglu, core developer and project administrator, Pardus Linux Project.

A background on Pardus

Pardus, as a GNU/Linux distro, is known to be user-friendly, sleek, simple and spicy with lots of custom configuration tools written by the Pardus team. The distro is not based on any of the traditional base distros like Debian or Red Hat. It's completely written from scratch. The package management tool is PISI. The distro makes use of lzma compression for packages and also supports xdelta-like techniques to make fetching packages for installation, faster. Pardus has developed its own core libraries and APIs to develop different applications on the Pardus platform.

Personally, I have been fascinated by Pardus for the last two years. Its boot speed is what I found to be the most impressive. This had prompted me to dig into the initrd/init boot scripts. I found it interesting that all system configs and automation scripts (traditionally written in Bash) have been replaced by Python. Even before Ubuntu's fastboot technology Upstart came out, Pardus' init was much faster.

What's my project about?

We have numerous variants of GNU/Linux (popularly known as distributions or distros) for different purposes. When we go for a distro install, we receive a base system with a standard set of applications bundled along with them. To make the OS work to our likes and specs, we need to install another set of packages that cater to our working domain or interest. Also, it requires setting up themes, wallpapers, copying our own files to the home directory, etc.

My project aimed at bringing out a Web-based distro 'cooker'. Users should be able to create their own custom distro builds by providing numerous custom options ranging from wallpapers to package selections. By locating the URL of this distro cooker (let's call it Pardusman), we could do the following customisation, using a simple Web interface:

1. **Home Page**
   - Sign up for a user account
   - Sign in to Pardusman
2. **Distro type**
   - Select the distro type: Live CD or Install CD? If it's Live, specify the user name, password, and host name
   - Provide a build project name
3. **Repository:** Select the package repository from the Pardus servers, to be used for the build.
4. **Languages**: Select language support to be included, besides setting the default language.
5. **Upload**
   - Upload a *RELEASE* file (i.e., a text file that appears in the *root* of the CD-ROM that contains some notes on the build).
   - Upload the contents of the home directory.
6. **Packages:** A package tree widget will appear. Packages are classified as different components according to their group. For example, the GIMP and Inkscape are other graphic tools included in the package group 'multimedia-graphics'. By using a drop-down tree element, we access the packages under a package group and select the check boxes in order to include them. We can even include all packages in a package group by checking the group itself. Also, there is a search option available to look for packages. All the packages from the selected Pardus repository will be listed in the packages widget. There is also an option to calculate size—on clicking the Calculate button, you can get the total size for the build.
7. **Wallpaper:** You can either select the default option from the list of wallpapers, or upload your own.
8. **Media Selection:** You can select the required image output. For example, an ISO image, or virtualisation images like VMWare, Qemu, Virtualbox, etc.

Once you have completed the wizard, you can submit

Figure 1: Sign-in

Figure 2: Distro type

Figure 3: Repository selection

Figure 4: Package selection widget

the configuration for distro building. It will be processed by the *buildfarm* queue underlying in the Pardus server. You can check back after a few hours and look into the 'user log' page to get the status of the build you have requested, the link for download, the link to the project file and the Pardusman log file. Also, a history of all builds you have ever made.

Getting started

I started my work with UI (user interface) design. I worked out the basic pages and their structure. Then I designed these using Inkscape and published them on my blog for feedback. As it happens with FOSS projects in general, I too got a lot of feedback. In fact, one of my good friends (Hiran) helped me give the finishing touches to the final set of images I required for the front-end UI.

Typically, a project has several development versions of the same app while coding. Under the circumstances, the manual way of backing up the code makes everything complex and difficult. Sometimes this even leads to loss of data/code. So, it is always advisable to use some kind of version control system (VCS) to keep track of code.

Git is one of the best version control systems available. I decided to go with Git, synced with *github.com,* where several FOSS projects are housed. You can pull the latest copy of Pardusman, using:

```
$ git clone git://github.com/t3rm1n4l/pardusman.git
```

Coming back to the UI elements, my next job was to create the set of Web pages using HTML and CSS. I scripted each page to complete the Web wizard described earlier. My template was now ready.

However, since the wizard, whose purpose is to collect data from a user, consists of several pages, it required transition from page to page. Page-to-page transitions were in traditional styles and not of Web 2.0 standards. So, I decided to use *div* containers, which can dynamically load HTML content using AJAX. This made the requirement for refreshing and loading new pages through URLs, redundant.

JQuery is a rich open source JavaScript library that helps to implement AJAX methods and calls using its rich in-built functionalities. I could easily implement the dynamic loading of pages and simple animation effects, using JQuery.

Once the templates and basic AJAX loading were completed, I started off with Django, Python's own Web framework. Django is a rapid Web application development platform. It makes it easier to develop complex Web applications in a short span of time. The coding with Pardusman progressed very fast.

The database storage in Django is handled using its own data model object-oriented structures. Initially, I wrote the pages for the user account sign up and sign in. Django was pretty easy to follow and code.

I got stuck with coding while creating a complex package selection widget. A package selection widget consists of a drop-down of packages with components in the top level and check boxes attached to it. It was solved with JQuery hacks. Thanks to the #jquery IRC channel of irc.freenode.net for helping me out.

There will be more than one package repository, like *pardus-2008* and *pardus-2009*. By selecting any one of them, we can build either a Pardus 2009 or a Pardus 2008 distribution. Dynamically producing the package information and bringing it to the packages widget causes overheads and puts a heavy load on the CPU. There might be thousands of packages in the repo and hundreds of users simultaneously accessing Pardusman, which would surely bring the server down. So I decided to build a static repo package information creator that would be scheduled to update the packages information

every day. The Web application makes use of that static page to display package information.

The next difficult task with implementing the UI was regarding the option of size calculation around the package widget. Once users select the required packages through check boxes, they can calculate the total size for the custom build. The size must be calculated live.

I used the *memcached* server, which can cache data and program objects. But the size of the object to be cached is restricted to 1 MB. I maintained objects of all repositories with package lists and corresponding package sizes in the *memcached* as cache objects.

Once the user clicks the *Calculate* button, it sends the list of selected packages to a Django function in the backend, which accepts the post requests. From the received list of packages, it analyses the package names one by one, and its size is grabbed from the repository *memcached* object.

The backend handles a lot of complex tasks like dependency resolving, which is very interesting. The user selects a set of packages that are required. But they are not the only packages to be included in the distro build. Each package is dependent on some other package. This relation is called dependency.

Each *pisi* package (e.g., *pidgin.pisi*) consists of the data and binary files, along with a meta file that describes all the details about the package. This is an XML file. We can find its dependencies by parsing the XML file.

Each package requires some other package to work. For example, Pidgin requires the following packages to work, which in turn may require some other package to work, thus creating a chain of requirements.

Every time a user requests for a size calculation, it resolves dependencies before calculating the size each time and returns a set of information. Here's an extract from the Pidgin package metafile:

```
<Name>pidgin</Name>
<RuntimeDependencies>
<Dependency>audiofile</Dependency>
<Dependency>gtk2</Dependency>
<Dependency>gnutls</Dependency>
<Dependency>gstreamer</Dependency>
<Dependency>startup-notification</Dependency>
<Dependency>cyrus-sasl</Dependency>
<Dependency>gtkspell</Dependency>
<Dependency>avahi-glib</Dependency>
<RuntimeDependencies>
```

The method of solving this complex loop and calculating the list of net packages that should go with the distro build is known as dependency resolving. It is performed in the Pardusman backend. Take a look at the Pardusman code, which you will find interesting.

The wallpaper selection page relies on JQuery. When a user uploads a picture, it is to be stored in the server,

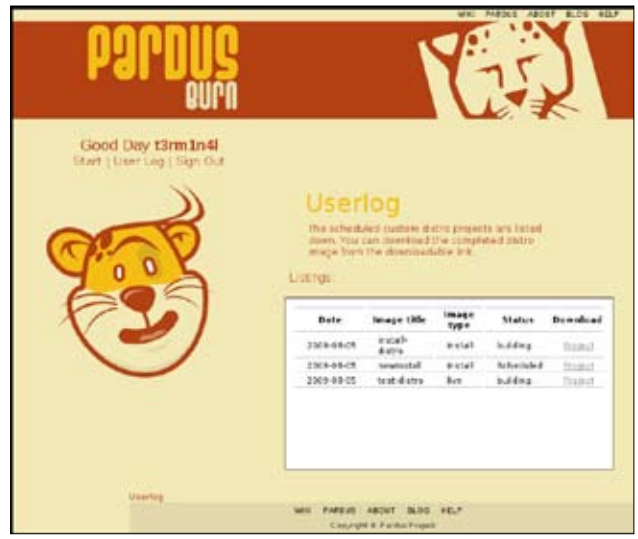

Figure 5: The 'user log' page lets you download the custom distro you created

and only a thumbnail is to be returned and shown in the list of wallpapers. For resizing the uploaded image to a thumbnail, I have used *imagemagick* in the backend.

The wizard involves seven basic steps. At each step, when the next button is clicked, it sends out the data collected from the current page to the server. This data is added to the browser session using the *request.session* dict object available in Django. Once all the wizard steps are completed, it has to generate a configuration file for the custom distro build making use of all the data provided by the user.

XML is used to represent the project configurations. The commonly used XML parser library in Python is ElementTree. But ElementTree is very slow to meet the requirements of this project. Hence, the Pardus team has ported *piksemel*, a XML parser used by the Jabber protocol, to the Pardus platform. It is a very fast XML parser. The project_config file is generated with the following tree structure.

```
<PardusmanProject type="install" media="iso">
<Title>Test-Project</Title>
<ReleaseFiles>RELEASE.txt</ReleaseFiles>
<Wallpaper>user_wallpapers/wallpaper_R6wL4i.jpg</Wallpaper>
<UserContents>user_contents.tar.gz</UserContents>
<PackageSelection repo="Pardus-2009">
<SelectedComponent>x11-server</SelectedComponent>
<SelectedComponent>desktop-kde-base</SelectedComponent>
<SelectedComponent>x11-util</SelectedComponent>
<SelectedPackage>glitz</SelectedPackage>
<SelectedPackage>libdmx</SelectedPackage>
<SelectedPackage>kdeedu-marble</SelectedPackage>
<Package>less</Package>
<Package>libX11</Package>
<Package>jpeg</Package>
<Package>sysvinit</Package>
<Package>piksemel</Package>
```

```
</PackageSelection>
<LanguageSelection default_language="en_US">
<Language>en_US</Language>
</LanguageSelection>
</PardusmanProject>
```

At the end of the wizard, the config file is generated. The project_file is designed as a tarball (.tar.gz) with the following structure:

```
Project_file.tar.gz
---+ user_contents.tar.gz ( user home contents )
--- user_presentation.odp
---- RELEASE.txt
---- project.xml
---+ user_wallpapers (wallpaper)
--- wallpaper1.png
```

On generating the project file, it is sent to the Buildfarm Process Queue. Buildfarm is a component of Pardusman that performs the distro builds from the given project configuration file. Buildfarm is run as a daemon that is run all the time. Process Queue maintains the list of distro build requests from users along with the link of the project file generated. There is another queue called the On_Progress queue, which maintains the list of distro projects for which the builds are in progress.

Distro building is a CPU-intensive process and it requires a heavy hardware configuration to build multiple distros at a time. On my laptop, performing more than two distro builds raises temperatures to very high levels and the laptop gets powered off. So simultaneous multiple distro builds are to be restricted according to the server capacity. Buildfarm is provided with a configuration file through which lots of parameters can be specified. It includes BUILD_LIMIT = No of builds to be performed at a time.

The Buildfarm_queue is capable of holding any number of user build requests. The On_Progress queue is restricted according to BUILD_LIMIT. Once the build of one project is completed, a project is fetched from the Buildfarm_queue to the On_Progress queue and the build will be started.

Users can keep track of the progress of their project build requests through the Userlog page. In the Userlog page, once the user requests for a build and the project is sent to the Buildfarm_Queue, it will be shown as 'scheduled' against the line that corresponds to a project build. Once the project is in the build state and is inside the On_Progress queue, it will be shown as 'in progress'. If the project build is successful, it will be shown as 'Completed' and if it fails, will show up as 'Failed'. Once the build is completed, a download link to the image will be provided. It also will list out a Logfile link, which can be used for debugging if the project build fails and can send a bug report or *dugg* with the reasons for it.

There are lots of complex things underlying the Buildfarm component. Explaining those implementation details is not within the scope of this article. Still, I would like to share some of the best features that come with the ***pisi*** package management tool, which helped me automate the build of the filesystem base easily.

For example, if you want to build a filesystem base in the current directory 'pardus-root', perform the following:

```
# pisi --yes-all -D"./pardus-root" ar pardus-install http://paketler.pardus.org.
tr/pardus-2009/pisi-index.xml.bz2
```

It initiates the directory 'pardus-root' as the root filesystem attached with a repository.

Now we can install packages to this filesystem as:

```
# pisi --yes-all --ignore-comar -D"./pardus-root" it -c system.base
# pisi --yes-all --ignore-comar -D"./pardus-root" it <package_name>
```

Did you find any package management tool that is capable of this? Apt, Yum, Emerge...? There are more. Suppose I have a bunch of ***.pisi*** package files in a directory, I can make that directory a ***pisi*** repository. What do you think?

Change the current directory to the above mentioned directory and execute the following:

```
# pisi ix
```

It will create the index file pisi-index.xml and pisi-index.xml.bz2

Now you can add the directory as the repository, using:

```
#pisi ar 'repo-name' ar <directory_path/pisi-index.xml.bz2>
```

That is all about the essentials of Pardusman. We are setting up Pardusman on the official Pardus server for all of you to master your own custom Pardus builds. Keep yourself updated with *http://pardusman.pardus.org.tr*.

I have already set up the Pardusman wizard right there. The Buildfarm component is not yet initiated as the daemon. The Pardus team will run the Buildfarm soon. Hence you can have your own distro versions of a custom-built Pardus, with your own wallpaper, packages, home folder contents and a lot more.

Grab the code at ***http://google-summer-of-code-2009-pardus.googlecode.com/files/Sarath_Lakshman.tar.gz***

Enjoy Pardusman. Get, set and burn!

Happy hacking! END

By: Sarath Lakshman

The author is a Hacktivist of Free and Open Source Software from Kerala. He loves working on the GNU/Linux environment and contributes to the PiTiVi video editor project. He is also the developer of SLYNUX, a distro for newbies. He blogs at *www.sarathlakshman.info*