Automated Testing of Debian Packages: Status Update

Lucas Nussbaum
lucas@debian.org

Abstract

During the months preceding the etch release, I worked on several large-scale QA tasks using an computing grid, including rebuilds of all packages in Etch, piuparts runs on all packages in Etch, etc. This talk will briefly describe what was done and how, and then will discuss various directions for future improvements, including changes needed to run such tasks on a regular basis, and to make the processes more efficient and scalable.

1 Introduction

With 10316 source packages in Debian *Etch* (18167 binary packages), it is getting increasingly difficult to ensure a good overall quality. While most other Linux distributions have chosen to split their packages into two groups of *supported* and *unsupported* packages, all packages in Debian are supported in the same way.

Being able to check that all packages meet a given quality criteria is very important. But it is of course impossible to check all packages manually: only automating such tests can make that possible. Some tests don't require a lot of resources, even when run over 10000+ packages. This is the case of static tests like *Lintian* and *Linda*. Others, however, are more expensive to run, making it hard for a non-profit organization such as Debian to run them on a regular basis. As an example, rebuilding all source packages in Debian on a modern AMD64 computer takes about 10 days.

During the months preceding the release of Debian *Etch*, I had the opportunity to perform several rebuilds and piuparts tests of all packages in *Etch* on an high-performance computing grid called Grid5000 [1, 2, 3], making it possible to rebuild all packages in Debian in about 7.5 hours. This resulted in a lot of issues reported and fixed in *Etch*. The goal of this paper is to discuss some technical and social issues that arose during that work.

This paper will be divided as follows: first, we will give a short overview of piuparts, and the test consisting of rebuilding packages from source. Next, technical issues will be discussed, then social issues. Finally, some goals for the *Lenny* release cycle will be discussed, aiming at improving the process.

2 Overview of QA tasks

In this section, a short overview of two QA tasks will be given.

2.1 Rebuilding packages from source

Debian packages are only built when they are first uploaded. Also, at that time, they are not built on the developer's architecture (if a developer is using an AMD64 system, the package that was built on the developer's system will go to the AMD64 archive without being rebuilt on the official build daemons). Binary packages that apply to all architectures (packages with `Architecture: all`) are not rebuilt at all.

One source of problems with that is that developers sometimes build packages in the environment they use daily, with lots of packages installed, possibly from unofficial repositories, instead of building them in a clean environment using a *chroot* or tools like `pbuilder`. This leads to packages with missing `Build-Depends` (especially `Architecture: all` packages), that won't build again without modification.

Another source of problems are changes in the build environment. Versions of packages in `Build-Depends` are provided manually by the developer, who usually doesn't investigate exactly which version of the package's dependancy is required. Also, it's usually not possible to predict the future: the developer cannot determine that the package will fail to build with a newer version of a compiler, for example.

The principle of this task is therefore simple: rebuild packages in a specific build environment, and see if it fails. The environment can be *testing*, *unstable*, or a custom environment, for example *unstable* with a newer compiler version installed.

2.2 Testing package installation with Piuparts

Piuparts [4] is a tool used to test installation, upgrade and removal of packages. It does so by installing packages in an absolutely minimal environment (only containing `apt` and all its dependancies).

Its main use is to find problems in *maintainer scripts*. *Maintainer scripts* are scripts executed during package installation, removal or upgrade. Package developers often forget that packages using such scripts must `Depend` (for `postinst` scripts, or `Pre-depends` for `preinst` scripts) on the packages needed to run the script (a package using `debconf` in its `postinst` must `Depend` on `debconf`).

Interested readers can refer to [5] for more details about piuparts.

3 Technical issues

Those tasks raise several technical issues, which will be discussed in this section.

3.1 Optimizing makespan of rebuilds on a computer grid

When building packages on a computer grid, several packages can be built simultaneously (building one package per compute node). However, using about 40 nodes, and scheduling builds in a smart way (starting with the longest build first), the time taken to build the whole archive cannot be improved by adding more nodes: the total time, needed to build all packages, becomes the time taken by the longest build (see figure 1 for an example).

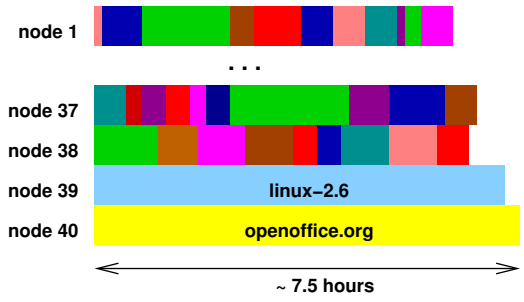

Figure 1: Example scheduling of package rebuilds. Longer builds are scheduled first, but the total time is limited by the time taken to build openoffice.org.

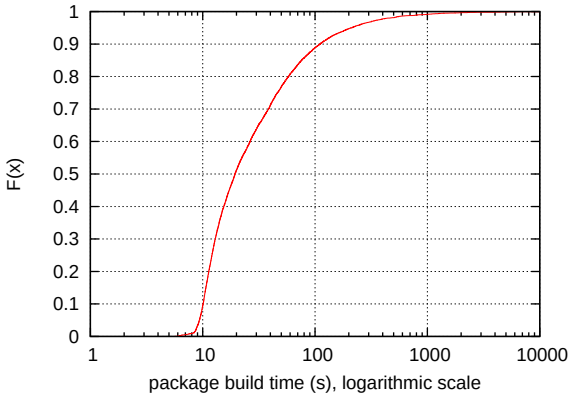

Figure 2: Cumulative distribution function of the packages build times. Only a few packages take a long time to build.

Source package	Time
openoffice.org	7 h 14 min
latex-cjk-chinese-arphic	6 h 18 min
linux-2.6	5 h 43 min
gcc-4.1	2 h 52 min
gcj-4.1	2 h 44 min
gnat-4.1	1 h 52 min
gcc-3.4	1 h 50 min
installation-guide	1 h 45 min
axiom	1 h 44 m
k3d	1 h 39 min

Table 1: Packages from *Etch* that take the longest time to build (on a dual-Opteron 2 GHz)

It is interesting to note that most packages take a very short time to build, as shown by figure 2: 90% of the packages need less than 100 seconds, and only 52 packages need more than half an hour. So, if those 52 packages were not built, it would theoretically possible, using enough compute nodes, to rebuild all of Debian in half an hour !

It is probably possible to improve the build time of most packages. The easiest way to

do that is to use several CPUs when building. Unfortunately, there is no standardized way (in policy) to ask for a build using several CPUs. Bug #209008 proposes to solve that by using `DEB_BUILD_OPTIONS="parallel=n"`. This would also benefit all the developers using dual-core system, which are becoming more common nowadays.

3.2 False positives during Piuparts tests

The main issue when doing piuparts tests is false positives (tests that fail despite a correct package).

Some packages fail to install during piuparts tests because they rely on another package being installed and configured, but this other package might require manual configuration. Such failures look impossible to avoid.

Another more important class of false positives is composed of packages which need to ask questions to the user during their installation. `debconf` allows packages to be installed non-interactively, but there are still some packages prompting manually in the archive (prompting using `debconf` is only *recommended* by the policy). Bug #206684 discusses making the use of `debconf` for prompting mandatory, which would clearly help.

4 Social issues

Social issues were also encountered during this work, and as often, they are much harder to solve than technical issues.

4.1 Lack of manpower

<Lunar> I think I'm becoming a perverse...
I enjoy reporting FTBFS.

A common problem in the Free Software community is the lack of manpower, especially for tasks not really considered fun. While it is "fun" to write tools to find problems, it is less "fun" to report those problems. This results in a number of very good tools to find issues in Debian packages, but those issues are never fixed in the packages, or even reported to the maintainers.

A similar problem exists with rebuilds of the archive. It is "fun" to run a build daemon, but this can easily be left to people with more computing resources. A back-of-the-envelope calculation shows that building all packages in Debian (which takes about 10 days) on a computer consuming 250 W costs about 30 euros.

Reviewing logs of tests and reporting bugs is a tedious task, but it can easily be shared amongst several developers, making it possible to review more failures, and even making it more "fun".

4.2 State of the archive

The Debian archive contains a lot of packages in a poor state, like packages with release-critical bugs opened for a long time, without anyone interested in fixing them. This makes running large-scale tests difficult, because of the number of failures to analyze. It might be a good idea to remove some of packages that have been RC-buggy for a long time, and are not going to be fixed. But since the decision to remove a package from Debian can only be taken by the package maintainer, this is usually a long process, especially when inactive maintainers are involved.

Also, some maintainers aren't well informed of the status of their packages. After the *Etch* release, the list of packages in unstable that were not included in *Etch* was reviewed, and it was clear that many maintainers did not know that their package was not part of *Etch*. For example, some maintainers fixed RC bugs, but forgot to ask for an "unblock".

5 Goals for *Lenny*

This section will describe a few goals for the *Lenny* release cycle, besides those already described before.

5.1 Improving organization of the collab-qa team

The collab-qa team is an alioth project aimed at sharing data about QA activities (using the `qa` repository for that is problematic, since non-DD cannot have commit rights on it). Communication is done on the `debian-qa` mailing list.

After a few months of activity, the project is slowly getting more active, with more people contributing. But it is still to be determined whether enough people will participate to allow this process to continue during the whole *Lenny* release cycle.

5.2 Regular builds of the whole archive

Ideally, QA tasks should be performed during the whole release cycle, on a regular basis. Builds are currently being done every 2 or 3 weeks, but it is not clear yet whether it will be possible to continue to review the failures.

Also, performing some piuparts runs to catch new failures from time to time would be a good idea as well.

5.3 Mail notification of maintainers

It might be a good idea to develop an opt-out system (so maintainers would be subscribed by default) to notify maintainers by email. This could be used to remind them of serious issues (a package not being in *testing*, or a package with an RC bug). And it could also be used as an easier way to report issues, such as results from automated tests that weren't analyzed carefully enough to file an RC bug.

But sending automated mails to maintainers is usually considered a bad idea, and the system will have to be carefully designed to minimize the annoyance and maximize the usefulness.

5.4 Additional tests

A few other tests could be interesting as well. After building packages, the resulting binaries could be compared with what is already in the archive. Since some packages were built several years ago, differences are likely to appear.

Piuparts could also be used more extensively, to report other issues, such as left-over files, dangling symbolic links, etc.

6 Conclusion and future work

Automated Quality Assurance is a good way to ensure that the same quality level is reached by all packages in Debian. This paper gave an short overview of QA tasks that were performed during the last months of the *Etch* release cycle. It also described in more details the technical and social issues that arose during the process.

Some ideas and goals for the *Lenny* release cycle were also given. Implementing them might allow to noticeably improve the overall quality of Debian.

7 Acknowledgements

This work has partly been done within the LIG laboratory, jointly supported by CNRS, INPG, INRIA, and UJF. Some computer resources were provided by the Grid'5000 platform (further information at `http://www.grid5000.fr/`). We would like to thank the French Ministry of Research, the ACI Grid and ACI Data Mass incentives, the Grid'5000 staff, as well as all the members of the Debian community who provided feedback.

References

[1] Franck Cappello, Eddy Caron, Michel Dayde, Frederic Desprez, Emmanuel Jeannot, Yvon Jegou, Stephane Lanteri, Julien Leduc, Nouredine Melab, Guillaume Mornet, Raymond Namyst, Pascale Primet, and Olivier Richard. Grid'5000: a large scale, reconfigurable, controlable and monitorable Grid platform. In *Grid'2005 Workshop*, Seattle, USA, November 13-14 2005. IEEE/ACM.

[2] Grid'5000 website. https://www.grid5000.fr/.

[3] Lucas Nussbaum. Use of grid computing for debian quality assurance. In *Research Room @ FOSDEM 2007*, Brussels, Belgium, 02 2007.

[4] Piuparts: .deb package installation, upgrading and removal testing tool. http://packages.debian.org/unstable/devel/piuparts.

[5] Nobody expects the finnish inquisition or: confessions of a debian package torturer. http://liw.iki.fi/liw/talks/finnish-inquisition.pdf.