Automated Testing of Debian Packages

Holger Levsen – debian@layer-acht.org
Lucas Nussbaum – lucas@debian.org

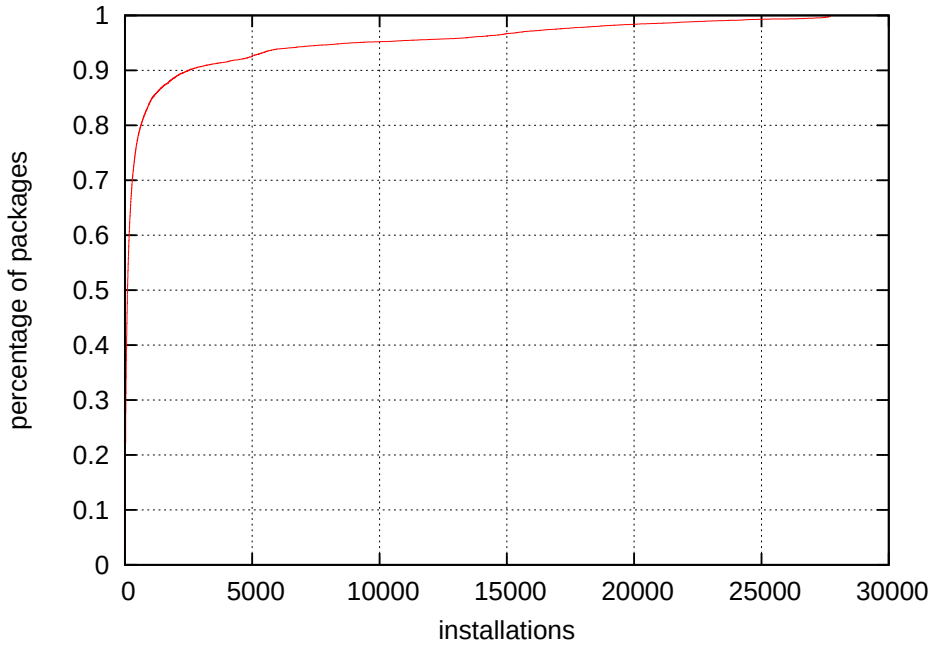
Summary

Summary

Debian's Quality

- Ask around : considered quite good compared to other distros
- A lot of packages, all supported in the same way :
  - 10316 source packages in etch/main
  - 18167 binary packages in etch/main

Packages installations according to popcon



- 18167 packages in etch/main (max : 27700 installations)
- 1591 packages have less than 10 installations
- 8985 packages have less than 100 installations
- 15321 packages have less than 1000 installations

$\Rightarrow$ Most packages don't have a lot of installations

Automated Testing

A way to :

- give the same level of attention to all packages in Debian
- not only rely on humans to find bugs
- avoid regressions
- keep maintainers busy :-)

Summary

Lintian and Linda

- Static checks on Debian packages
- Lintian : (mostly in) Perl, Linda : Python
- Easy to run them yourself
- Generates **lots** of warnings and errors (some false-positives, too)
- See `http://lintian.debian.org/` (not up to date)

Example Lintian output

belpic (maintainer : Wouter Verhelst) :

```
W: belpic source: diff-contains-substvars debian/substvars
E: libbeidlibopensc2: postinst-must-call-ldconfig usr/lib/libbeidpkcs11.so.2.1.0
E: libbeid2: postinst-must-call-ldconfig usr/lib/libbeidlibjni.so.2.7.2
W: beidgui: binary-without-manpage beidgui
W: beidgui: non-dev-pkg-with-shlib-symlink usr/lib/libbeidgui.so.1.5.0 usr/lib/libbeidgui.so
E: beidgui: no-shlibs-control-file usr/lib/libbeidgui.so.1.5.0
W: beidgui: postrm-should-call-ldconfig usr/lib/libbeidgui.so.1.5.0
W: beidgui: package-name-doesnt-match-sonames libbeidgui1
W: beid-tools: binary-without-manpage beidcrld
W: beid-tools: binary-without-manpage beidpcscd
W: beid-tools: init.d-script-missing-lsb-section /etc/init.d/beid
```

$\Rightarrow$ As I said, probably many false positives :-)

Future work

- Make maintainers use them ! (ideas ?)
- Fix bugs (or use overrides if false positives)
- And mostly infrastructure work :
  - Work on lintian.debian.org
  - Regular runs
  - File bugs ?

Summary

Rebuilding packages

- packages with "`Arch : all`" are only built on the developer's machine
- packages with "`Arch : any`" are only built automatically before they reach unstable (and only on $ARCH != Uploader's arch)

After that, the build environment changes :

- newer/older compiler and libraries
- build-dependencies not available anymore (b-deps are not considered for testing propagation)

Problems :

- Everyone should be able to build your package
- Stable releases must be self-contained (security upgrades !)

Rebuilding packages : tools

pbuilder :

- builds a package inside a chroot
- very easy to set up
- you should use it !
- use cowbuilder for faster builds (cowdancer package)

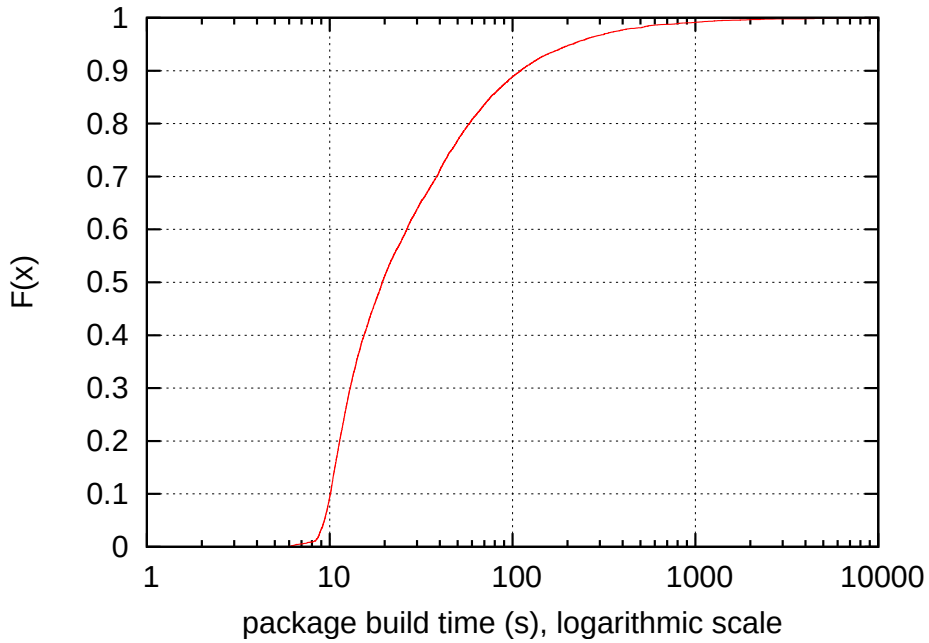
sbuild (the Debian package) :

- relies on schroot
- harder to set up, but more powerful

Rebuilding packages : resources usage

Rebuilding all packages in Debian Etch :
about 10 days on a single computer

Most packages are fast to build :



Rebuilding packages : resources usage (2)

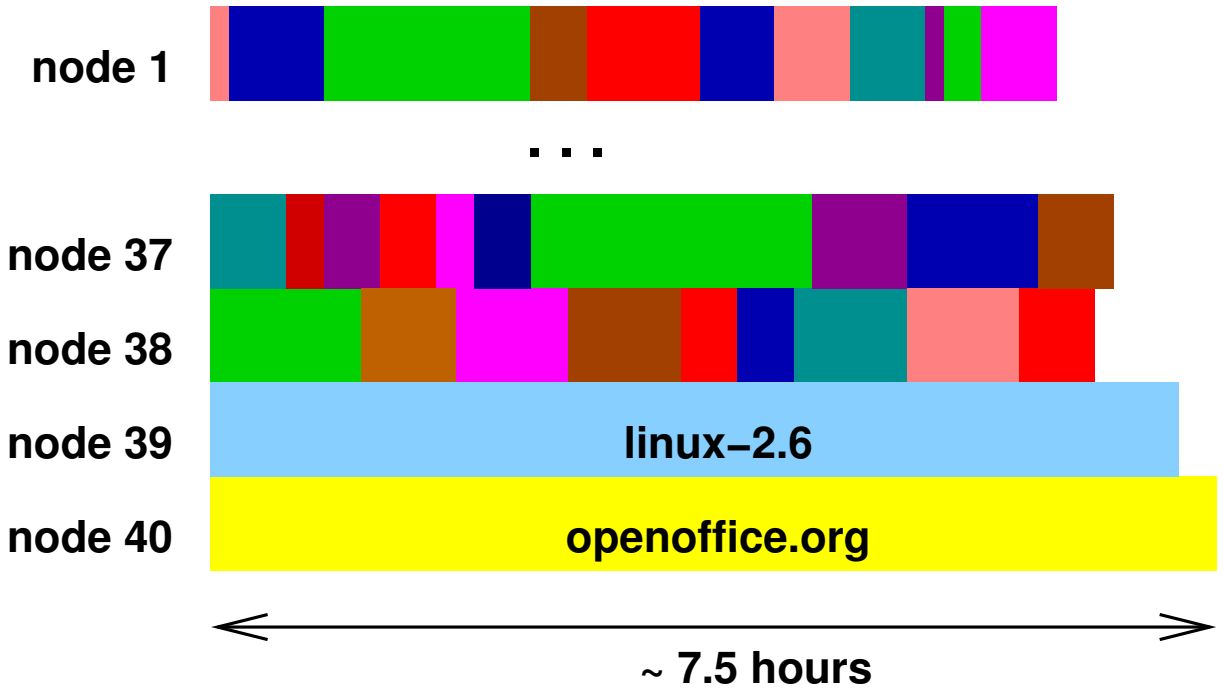
But some packages take a long time :

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

(On Dual-Opteron 2 GHz, 2 GB RAM)

Parallel Rebuilds

Easy to distribute over several nodes :
Since October, several rebuilds on Grid'5000
(french experimental grid platform, with 2500 CPUs)
$\Rightarrow$ Full rebuild in about 7.5 hours, on about 40 nodes



Parallel Rebuilds

- Using more nodes is useless
- Need to make a few packages build faster
- "make -j" :
  - no common interface (#209008)
  - Some packages fail to build when using several CPUs
- Solutions :
  - Work on #209008
  - Work on those few packages
  - Ignore those packages

Future Work

- Test build scripts (rebuild after change, `clean` rule, etc)
- Compare build results with what is in the archive
  - First results are quite depressing
- Build in "strange" environments and compare results

Summary

Piuparts

Tests installation and removal of packages

Process :

- cleans up a chroot (removes everything except apt)
- installs the package to test and its dependencies
- Removes everything, purge all dependencies
- Purges the package to test

$\Rightarrow$ test of the package maintainer scripts
(`preinst`, `postinst`, `prerm`, `postrm`)
under the most extreme conditions

Piuparts (2)

Also tests other things :

- upgrades
- running processes after removal
- dangling symlinks
- files left after removal/purge, files from other packages modified

Piuparts and false positives

Piuparts generates A LOT of false positives

To be tested, a package must be able to install non-interactively

- debconf is nice (`Noninteractive` frontend)
- but doesn't solve everything (e.g packages that need access a database)

$\Rightarrow$ Make all packages use debconf

$\Rightarrow$ After that, not much to do about false positives

Piuparts and set theory

Problem : how can one easily get a list of real failures, without false positives ?

$\Rightarrow$ Set theory !

Example : find packages that fail to install because of a missing depend on debconf.

- Run piuparts over all packages, without debconf installed
- Fetch the list of failures
- For each package that failed during the first run, re-run piuparts with debconf installed
- Packages that succeeded = our list of failures

Future work

- Other piuparts tests (not just installation/removal failures)
- Improve piuparts (now maintained collaboratively !)
  - Make it more flexible
- piatti.debian.org : dual Xeon in helsinki
  - Used by liw to run piuparts over the archive
  - Slower by Grid'5000 ;)
  - Idea : Xen instances for interested DD to reproduce/investigate results
  - More ideas ?

Summary

Structuring QA
Problems

QA mostly done by individuals

$\Rightarrow$ not a good solution on the long term :

- nobody knows what people are doing
- duplicated efforts
- things not tested, even in etch
- some resources could be shared, but are not

Better, collaborative QA

- use `debian-qa@l.d.o` for communication
- share information
  - documentation on processes
  - lists of false positives, bugs already filed, etc
  - use usertags

Example "good" process

- Bob wants to test a new compiler version (rebuild all packages with the new version)
- Bob tells about his plans on `debian-qa@l.d.o`
- Joe proposes to run the tests on his large computing cluster
- After discussing the details, Joe runs the tests
- Bob analyzes the logs and files bugs

"Collaborative QA" project @ alioth

collab-qa alioth project :

- share as much stuff as possible
- currently :
  - scripts to run rebuilds and piuparts on a cluster
  - scripts to analyze logfiles
  - data :
    - blacklists for rebuilds and piuparts
    - list of piuparts false positives
    - estimated build time for each package

$\Rightarrow$ Join us !

Summary

Conclusion

- We have a nice set of tools
  - could clearly be used a lot more
- Many tests to run and many bugs to fix with the current tools
- Main objective :
  - Be better at finding and fixing bugs using the current tools
  - Even if writing new tools is clearly sexier ;)