Use of Grid Computing for Debian Quality Assurance

Lucas Nussbaum
lucas@debian.org – lucas.nussbaum@imag.fr

Laboratoire d’Informatique de Grenoble - Projet MESCAL
Summary

1. Introduction
2. QA tasks
3. Infrastructure
4. Results
5. Future Work
6. Conclusion
Summary

1. Introduction
   - Quality Assurance in Debian
   - Grid’5000

2. QA tasks

3. Infrastructure

4. Results

5. Future Work

6. Conclusion
Quality Assurance in Debian

Debian:
- the largest volunteer-based GNU/Linux distribution
- renowned for its quality

QA in general plays an crucial role:
- to ensure a minimal quality level for all packages
- to track not-so-well maintained packages
- ...
But some QA tasks require a lot of computing power

- e.g. rebuilding all packages in Debian:
  about 10 days on a single computer

Difficult to perform by volunteers who pay their electricity bills, especially on a regular basis.
Grid’5000

- aims at building an highly reconfigurable, controlable and monitorable experimental grid
- dedicated to computer science research
- funded by french ministry of research, INRIA, CNRS, ACI Grid, and other public organizations
- gathers 1200 compute nodes (2500 CPUs) in 13 clusters
- typical node: Dual-Opteron 2 Ghz, 2 Gb of RAM
- high speed network (10GbE)
- free time-slots during nights and week-ends
Grid’5000 (2)
Grid’5000 (3)
(Obvious) idea: use Grid’5000 to work on Debian QA

- Which tests are suitable?
- With which infrastructure?
Summary

1. Introduction

2. QA tasks
   - Overview
   - Rebuilding packages
   - Installation testing using piuparts

3. Infrastructure

4. Results

5. Future Work

6. Conclusion
QA tasks performed on Grid’5000

Ideal task:
- consumes a lot of time
- can be distributed over a lot of nodes
- doesn’t generate too many false positives
- would improve Debian quality

Two different tasks performed on Grid’5000:
- Rebuild of all packages in Debian
- Installation and removal testing using Piuparts
Rebuilding all packages in Debian

- **Arch : all** packages are only built on the developer’s machine
- **Arch : any** packages are only built automatically before they reach unstable

After that, the build environment changes:
- newer/older compiler and libraries
- build-dependencies removed

Not tested automatically, but important for the release:
Etch must be *self-contained* (think of security upgrades!)

Easy to distribute (build in parallel)
Installation and Removal testing

installability can be tested statically (see debcheck, edos-debcheck) But packages have maintainer scripts:

- executed during package installation and removal
- to configure stuff, start services
- helper scripts exist (debconf, update-{rc.d,modules/inetd})
- lots of bugs: missing dependencies, shell scripting mistakes, etc
Installation and Removal testing (2)

**piuparts automatically:**
- installs packages in a near-empty chroot
- remove it
- remove as many packages as possible
- purges it

⇒ most extreme test for maintainer scripts

But quite a lot of false positives:
- packages that prompt without debconf
- packages that depend on a DBMS (mysql,...)

Easy to distribute (test packages in parallel)
Summary

1. Introduction
2. QA tasks
3. Infrastructure
   - Principles
   - Architecture
   - Typical job
4. Results
5. Future Work
6. Conclusion
Infrastructure for QA tests on Grid’5000

Principles

- connection to Grid’5000 nodes via SSH
- one task per node (easier to manage)
- simple master/slave architecture
Infrastructure for QA tests on Grid’5000

Architecture

3 central points:

- *Master* node that schedules jobs
- Shared NFS directory to write results
- Internal Debian mirror
Infrastructure for QA tests on Grid’5000

Typical job (piuparts test)

- 55 nodes are reserved; deployment of a Debian Sid environment using Kadeploy is started.
- After 12 minutes: environment deployed on 43 nodes. First node is used as master node:
  - Prepares the other nodes (install required packages, etc)
  - Locally updates the chroots
  - Script responsible for controlling the other nodes is started
- After 2 minutes, preparation is finished: master nodes starts to schedule jobs on the other nodes.
- After 3 hours and 46 minutes, the 18156 packages in etch have been tested
Summary

1. Introduction

2. QA tasks

3. Infrastructure

4. Results
   - Grid’5000 bugs
   - Debian Bug reports
   - Speed-up

5. Future Work

6. Conclusion
Results - Grid’5000 bugs

Those experiments allowed to find a few important problems on Grid’5000: misconfigurations, performance problems, etc.

In the future, it will serve as a testcase to validate extensions to the platform.
Results - Debian Bug Reports

About **200 RC bugs found** (and fixed) in Debian Etch

- about 100 from rebuilds
- about 100 from piuparts testing

Efforts welcomed by a majority of developers (but not all :-)
Results - speed-up

Rebuilding the 10217 packages in Debian Etch:
about 10 days on a single computer

⇒ about 7.5 hours on Grid’5000

Testing the 18153 binary packages in etch:
about 5 days on a single computer

⇒ about 3 hours and 46 minutes on Grid’5000
Summary

1. Introduction
2. QA tasks
3. Infrastructure
4. Results
5. Future Work
   - Overview
   - Rebuild speed-up
   - Improving the log reviewing
6. Conclusion
Future Work

- Improve the infrastructure:
  - Jobs using several Grid’5000 clusters at the same time
  - Central Debian mirror is a bottleneck ⇒ local cache on the nodes
  - Shared NFS directory for logs is a bottleneck ⇒ try other solutions
- Other QA tasks (less critical ones)
- Increase the rebuild speed-up
Increasing the rebuild speed-up

Most packages take a very short time to build, but a few packages take a very long time (hours)
### Increasing the rebuild speed-up (2)

**Top ten packages**

<table>
<thead>
<tr>
<th>Source package</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>openoffice.org</td>
<td>7 h 14 min</td>
</tr>
<tr>
<td>latex-cjk-chinese-arphic</td>
<td>6 h 18 min</td>
</tr>
<tr>
<td>linux-2.6</td>
<td>5 h 43 min</td>
</tr>
<tr>
<td>gcc-4.1</td>
<td>2 h 52 min</td>
</tr>
<tr>
<td>gcj-4.1</td>
<td>2 h 44 min</td>
</tr>
<tr>
<td>gnat-4.1</td>
<td>1 h 52 min</td>
</tr>
<tr>
<td>gcc-3.4</td>
<td>1 h 50 min</td>
</tr>
<tr>
<td>installation-guide</td>
<td>1 h 45 min</td>
</tr>
<tr>
<td>axiom</td>
<td>1 h 44 m</td>
</tr>
<tr>
<td>k3d</td>
<td>1 h 39 min</td>
</tr>
</tbody>
</table>
Increasing the rebuild speed-up (3)
Using more nodes is useless

- Already scheduling longest builds first

```
<table>
<thead>
<tr>
<th>node 1</th>
<th>node 37</th>
<th>node 38</th>
<th>node 39</th>
<th>node 40</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>linux-2.6</td>
<td>openoffice.org</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>~ 7.5 hours</td>
<td></td>
</tr>
</tbody>
</table>
```

Lucas Nussbaum  Use of Grid Computing for Debian QA
Increasing the rebuild speed-up (4)
Possible solution : "make -j"

- Grid’5000 nodes have several CPUs, but only one is used during build
- No standard way to tell "use more than one CPU" (Debian bug #209008)
- Some packages fail to build when told to use several CPUs
  ⇒ Possible solution:
  only work on the few packages that annoy us...
or just ignore them.
Real bottleneck: manpower for log reviewing

So many logs, so little time...

Such QA tasks were traditionally *solitaire* games

Sharing the load is necessary to continue on the long term
Summary

1. Introduction
2. QA tasks
3. Infrastructure
4. Results
5. Future Work
6. Conclusion
Conclusion

Grid’5000:
- a really nice tool
- well suited to running such tasks

Quality Assurance in Free Software projects:
- could really benefit from using such a tool
- needs improvement, both
  - technically: better testing tools, less false positives
  - also human problem: needs collaboration on reviewing generated data