Use of Grid Computing for Debian Quality Assurance *

Lucas Nussbaum

Laboratoire d'Informatique de Grenoble – LIG
lucas@debian.org – Lucas.Nussbaum@imag.fr

Abstract

Many quality assurance tasks require a lot of computing power, especially when applied to large GNU/Linux distributions such as Debian. We used the french Grid'5000 experimental computer grid to work on such tasks. After describing the QA tests we performed and the framework we used to distribute them over Grid'5000, we provide results showing that such tasks can be of interest for both communities. Possible improvements are also described.

1 Introduction

Automated Quality Assurance (QA) has proven very useful inside Debian, allowing the whole distribution to meet the high quality level that has made it so popular. Thanks to automated QA, one can ensure that all packages in a release meet at least the same basic criterias of quality. This is especially important in Debian, where there is a lot of *niche* packages that have only a few users: such packages might not necessarily get enough attention to make sure they work properly.

Many Quality Assurance (QA) tasks require a lot of computing power, especially when applied to the thousands of packages in Debian. For example, a rebuild of all packages in Debian on a modern desktop computer takes about 10 days. Such tasks are traditionally performed by motivated individuals, which makes it hard to ensure that Debian is throughoutly tested on a regular basis.

The Grid'5000 [1, 2] project aims at building an highly reconfigurable experimental Grid platform, to serve as an experimental testbed for research in Grid Computing. It currently includes 9 sites, hosting 15 clusters and a total of about 2500 CPUs. Since the grid is not being used for production work, it is relatively easy to find time slots where ressources are available, especially during nights and week-ends.

Grid'5000 was used to perform several large scale QA tasks during the months preceding the Debian Etch release, with two goals in mind: one was to help to improve the quality of Debian. The other one was to provide a good case study, intensively using a wide range of Grid'5000 resources, to help detect problems in the platform, and to serve as a basis for future work on grid experiments.

In this paper, we first introduce the two different types of QA tasks that were done on Grid'5000. Then, we shortly describe Grid'5000 from a user's point of view, and detail the framework that was used to perform those experiments. Finally, we present the results we obtained, and describe possible future improvements.

2 Quality Assurance tasks

While the Grid'5000 could be used to run many kinds of QA tasks, we concentrated on two different tasks: the regular rebuild of all packages, and the test of installation and removal of packages.

2.1 Rebuilding all packages

While packages are automatically built on all architectures when they enter the archive, they are not rebuilt regularly afterwards, even if their dependancies are updated, or if their build environment changes (new compiler, etc). This might result in packages that are part of a release, but can no longer be rebuilt from their sources, leading to problems with security sup-

*This work has been done within the LIG laboratory, jointly supported by CNRS, INPG, INRIA, and UJF. Computer resources are provided by the Grid'5000 platform (further information at `http://www.grid5000.fr/`).

port. Rebuilding packages also helps to find regressions in compilers or libraries (missing symbols, etc).

2.2 Testing the installation and removal of packages

Another test that can be performed automatically is the installation and removal of packages using *piuparts* [3]. *Piuparts* is a script that installs packages in a *chroot*, removes it with all the packages that were installed with it, and then "purges" them (removes the files that were left after removal). Piuparts allows to find a lot of bugs, but also generates a large number of false positives, for example caused by packages that require user input, or that depend on another packages being functional (e.g a MySQL server).

3 Distributing quality assurance tasks over Grid'5000

From the user point of view, Grid'5000 nodes are systems connected to a private high-speed network provided by RENATER [1]. Nodes are grouped in clusters, but all nodes are reachable directly from any Grid'5000 node. A typical Grid'5000 node is a bi-Opteron system with 2 or 4 GB of RAM, connected to a Gigabit Ethernet network, however some clusters include other types of nodes (Xeons, Itaniums...). A fast and scalable deployment system, *Kadeploy* [4], enables users to install the operating system they want on some or all nodes of a cluster. During our experiments, we installed Debian Sid on all the nodes we used.

Since our first goal was to reach a point where we would be able to run QA tasks, we chose a simple master/slave architecture, only targetting a "good enough" framework and leaving improvements for later. This infrastructure has obvious scalability problems, but proved sufficient for the experiments described in this paper. Our framework include three central points, as shown in figure 1:

- A "master" node is used to schedule the tasks on all the other participating nodes. It connects to the other nodes using *ssh* and maintains the connection until each job is over. This design hasn't proven to be a bottleneck.

[1] the French National Telecommunication Network for Technology, Education and Research.

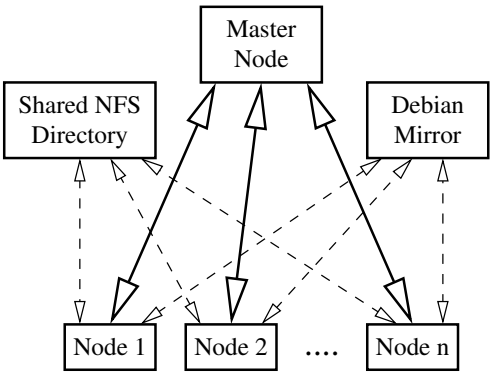

Figure 1: Infrastructure based on a master/slave model.

- An NFS server is used to share information common to all nodes: configuration files, scripts, and output logs. Despite the large amount of data written to logs, it doesn't seem to be the major bottleneck.

- An internal Debian mirror is used over HTTP to distribute source and binary packages to the nodes. Since *piuparts* tests consist mostly in package download, installation and removal, this central HTTP server is clearly a bottleneck for those tests. We plan to improve the situation by adding HTTP proxies on each participating node.

Here is an example execution of piuparts over the whole archive, on a cluster featuring 55 bi-dual-core Opteron systems, with 4 GB of RAM each.

- The 55 nodes are reserved for the duration of the experiment. The deployment of a Debian Sid environment on these nodes is started.

- After 12 minutes, the Debian Sid environment is successfully deployed using *Kadeploy* on 43 nodes. 12 nodes failed for various reasons (Kadeploy is still under development, and such failures are not uncommon since it relies on protocols which aren't known for their robustness such as PXE and TFTP). The first node is used as the "master" node, and the script responsible for controlling the other nodes is started on it. The master node prepares the other nodes by mounting the directory shared using NFS, by updating the packages list (`apt-get update`) and by installing the required packages (sbuild, piuparts, etc). It also, locally, updates the *chroots* stored in

the NFS directory, to make sure that we will run our tests in an up-to-date environnement. Since all tests happen inside those *chroots*, updating the host environment is not necessary: only the *chroot* is important.

- After two minutes, all this preparation is finished, and the master node starts to run tasks on the slave nodes (4 tasks are started concurrently on each node, to make as much use as possible of the 4 cores). Each task simply consists in a command line starting a script from the NFS directory: this is flexible enough to execute very different tasks using the same system. The output of each task is written to the NFS directory, so one can review it later.

- After 3 hours and 46 minutes, the 18153 packages in Debian Etch were tested using piuparts. The master node exits, and the nodes are made available for future tasks.

4 Results

Performing those tasks allowed to find a number of problems on the Grid'5000 clusters, including misconfigurations, a kernel bug, and random failures that were never noticed before. Such tasks, or very similar tasks, might be included in a test suite aimed at checking the platform.

From the Debian QA point of view, those tests resulted in more than 150 *release critical* bugs [5] reported and fixed in Etch (98 from rebuilds, and 65 from installation testing). This work has been generally welcomed by Debian developers, and some of them provided feedback and ideas for other tests, as seen in the bug reports [5].

While a full rebuild of all 10217 source packages in Debian takes about 10 days on a modern computer (for example, if using a single Grid'5000 node), we were able to rebuild it in less than 7 hours and 30 minutes, by distributing the package builds over the nodes (we chose not to build more than one package per node at a time). This is only caused by the fact that OpenOffice takes this time to build on a Grid'5000 node: it is therefore not possible to perform a shorter full rebuild (as shown on figure 2). Some other packages that take a long time to build are listed in table 1.

As shown in figure 3, most packages build in a very short time, and only a few packages take a long time to build. It should therefore be possible to achieve much faster complete builds by working on making those few packages faster to build. The most easy way to improve that would be to use several CPUs when building, but Debian doesn't have a standardized way to tell packages to use more than one CPU when building (see Debian bug #209008).

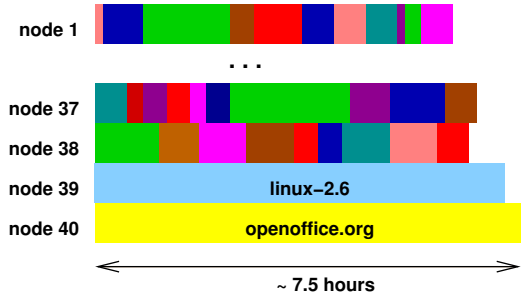

Figure 2: Example scheduling of package rebuilds. Longer builds are scheduled first, but with more than 37 nodes, the total time is limited by the time taken to build openoffice.org.

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 that take the longest time to build

Installation testing made it possible to benefit from a similar speed-up, going from about 5 days on a single computer to about 4 hours. By removing some bottlenecks in our infrastructure, it should be possible to reduce this time even further, since this test can be very easily distributed to a larger number of nodes.

5 Conclusion and future work

The use of Grid'5000 for Debian QA work has proven to be able to help significantly the search for bugs, allowing to find and report more a lot of *release critical* bugs in the months preceding the release of Debian

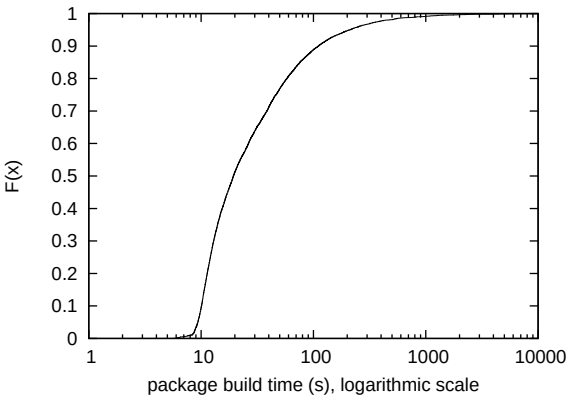

Figure 3: Cumulative distribution function of the packages build times. Only a few packages take a long time to build.

Etch. We plan to continue such work after the release of Etch, targetting other kind of issues (not necessarily serious issues).

Tasks such as rebuilding all packages in Debian or package installation testing are particularly well suited to this kind of resources, since it is possible to run them in parallel on a very large number of nodes. This allow a linear speed-up compared to running them on one computer.

However, there are still several areas where the process could be improved in the future:

- The infrastructure used to run such tasks inside Grid'5000 could be improved by making it more robust, more scalable and more manageable: the current architecture is clearly not a good example of what one should do. We are planning to move this infrastructure to a remote execution and control system developped in the LIG (Laboratoire d'Informatique de Grenoble) laboratory. This should benefit both parties by providing a challenging testbed.

- Piuparts testing is very easy to distribute over a large number of nodes. Package rebuilds are harder, since some packages take a very long time to build: the total build time is constrained by the build time of the longest package. We should try to optimize the build time of the packages that take the longest time. Another problem we encountered with Debian packages was false positives: making packages more piuparts-friendly would make testing much easier.

- Finally, the biggest problem we encountered was the lack of manpower for reviewing the logs. QA tasks are traditionally performed individually, which works very well when one rebuilds the archive on his own system. However, with Grid'5000, we are able to generate data much faster than we will ever be able to analyze it. We developped tools to do some parts of the analysis automatically, but this only reduced slightly the time taken by the reviewing of logs.

6 Acknowledgements

We would like to thank the French Ministry of Research, the ACI Grid and ACI Data Mass incentives, and the Grid'5000 staff.

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

[4] Yiannis Georgiou, Julien Leduc, Brice Videau, Johann Peyrard, and Olivier Richard. A tool for environment deployment in clusters and light grids. In *Second Workshop on System Management Tools for Large-Scale Parallel Systems (SMTPS'06)*, Rhodes Island, Greece, April 2006.

[5] Debian bug report logs: bugs tagged grid5000. http://bugs.debian.org/cgi-bin/pkgreport.cgi?tag=grid5000;users=lucas@lucas-nussbaum.net.