Automated Software Testing and Release with Nix Build Farms

Eelco Dolstra
Universiteit Utrecht
Department of Information and Computing Sciences

Eelco Visser
Technische Universiteit Delft
Department of Software Technology

The Goal: Building and Testing Software
- It is good development practice to build and test a software system every time a developer commits a change to the project's version management repository.
- A build farm supports this: it's a set of machines that continuously builds and tests software components from a version management system, producing status reports and/or releases.
- It is called a build farm because the software typically must be tested in many hardware / operating system configurations (Windows, Linux, Mac OS X, 32 bits, 64 bits, etc.) so a large number of (virtual) machines is required.

Does it build correctly?  Does it pass the test suite?

The Solution: The Nix Deployment System
- Nix (http://nix.cs.uu.nl/) is a purely functional package management system.
- Packages are built from pure functions (Nix expressions), i.e., the build result only depends on the declared inputs and never changes after it has been built.
- Packages are stored in a Nix store under a name that contains a cryptographic hash of all package inputs:

Why is this useful for a build farm?
- As a component build language, the Nix expression language is ideal for describing the build tasks to be performed.
- As a functional language, the Nix expression language makes it easy to describe variants.
- Virtual machines can be built and used on the fly in a Nix expression from a declarative specification.
- Nix manages the storage of the dependencies.
- Nix supports distributed multi-platform builds transparently.
- The hashing scheme + complete dependencies allow builds to be reproduced reliably.
- Efficiency: due to the hashing scheme, we only rebuild things that have actually changed.

Run static/dynamic analyses
- PHP-SAT, the PHP static analysis tool release php-sat-1.0pre265

The Problem: Managing the Complexity of the Build Environment
- Non-trivial software packages typically have a large number of dependencies.
- So if we need to build a package with N dependencies on M configurations, then to install and manage those dependencies takes N x M effort!
- And what if there are conflicting dependencies? E.g., package A builds with GCC 3.3 but not GCC 4.1, while package B needs at least GCC 4.1.
- Finally, the (virtual) machines themselves need to be set up and maintained.

Status and Research Directions
- Nix-based build farm in use at UU, TUD.
- Used by various open source projects: Stratego/XT, MetaEnvironment, Nix, NixOS...
- Future work: automatic exploration of the configuration space – try to select configurations that are more likely to exhibit problems.
- Future work: use static analyses to find potentially troublesome configurations, e.g., interference between #ifdefs in a C program

More information
- Nix web site: http://nix.cs.uu.nl/
- Example Nix build farms: http://nix.cs.uu.nl/dist/, http://buildfarm.st.ewi.tudelft.nl/