Reproducible and User-Controlled Package Management in HPC with GNU Guix

Ludovic Courtès (ludovic.courtes@inria.fr)
Ricardo Wurmus (ricardo.wurmus@mdc-berlin.de)

Workshop on Reproducibility in Parallel Computing (RepPar)
25 August 2015
“Reproducibility”?
“Reproducibility”? 

1. bit-reproducible builds
“Reproducibility”?

1. **bit-reproducible** builds
2. **isolating** a software environment from changes
“Reproducibility”?

1. bit-reproducible builds
2. isolating a software environment from changes
3. sharing environments with others
“Reproducibility”? 

1. bit-reproducible builds 
2. isolating a software environment from changes 
3. sharing environments with others
“Reproducibility”? 

1. **bit-reproducible** builds
2. **isolating** a software environment from changes
3. **sharing** environments with others
“Reproducibility”?

1. bit-reproducible builds
2. isolating a software environment from changes
3. sharing environments with others
“Reproducibility”?

1. bit-reproducible builds
2. isolating a software environment from changes
3. sharing environments with others
Beyond Reproducibility

- User-controlled upgrades and roll-backs
- Change specific parts of the software stack
- Hackability: no black boxes
Beyond Reproducibility

- user-controlled upgrades and roll-backs
Beyond Reproducibility

- User-controlled upgrades and roll-backs
- Change specific parts of the software stack
- Hackability: no black boxes
Beyond Reproducibility

- user-controlled upgrades and roll-backs
- change specific parts of the software stack
- hackability: no black boxes

localhost
Beyond Reproducibility

- user-controlled upgrades and roll-backs
- change specific parts of the software stack
Beyond Reproducibility

▶ user-controlled upgrades and roll-backs
▶ change specific parts of the software stack
▶ hackability: no black boxes
Beyond Reproducibility

▶ user-controlled upgrades and roll-backs
▶ change specific parts of the software stack
▶ hackability: no black boxes
Beyond Reproducibility

- user-controlled upgrades and roll-backs
- change **specific parts** of the software stack
- **hackability**: no black boxes
VMs and Docker

Pros:
▶ "bit-reproducible"
▶ reproducible anywhere by anyone

Problems:
▶ VMs are heavyweight
▶ binary images are opaque
▶ not composable
VMs and Docker

Pros:

▶ "bit-reproducible"
▶ reproducible anywhere by anyone

Problems:

▶ VMs are heavyweight
▶ binary images are opaque
▶ not composable
VMs and Docker

Pros:
- “bit-reproducible”
- reproducible anywhere by anyone

Problems:
- VMs are heavyweight
- binary images are opaque
- not composable
Functional Package Management
Functional Package Management

Regarding the build & installation process of a package as a pure function.
openmpi = f(hwloc, gcc, make, coreutils)

where f = ./configure && make && make install
openmpi = \( f(\text{hwloc}, \text{gcc}, \text{make}, \text{coreutils}) \)

\( \text{hwloc} = g(\text{pciaccess}, \text{gcc}, \text{make}, \text{coreutils}) \)
openmpi = f(hwloc, gcc, make, coreutils)
hwloc = g(pciaccess, gcc, make, coreutils)
gcc = h(make, coreutils, gcc₀)
...

openmpi = f(hwloc, gcc, make, coreutils)
hwloc = g(pciaccess, gcc, make, coreutils)
gcc = h(make, coreutils, gcc₀)

the complete DAG is captured
 References

- *A Safe and Policy-Free System for Software Deployment* ("Nix"), Dolstra et al., 2003
1. functional package management (FPM) empowers cluster users
2. FPM is a solid foundation for reproducible software deployment
3. beyond reproducibility: Guix is programmable, supports experimentation
From the Architecture of Nix...

http://nixos.org/nix/

**build processes**
chroot, separate UIDs

**Nix tools**
- Nix language
- client lib

**Nix build daemon**
From the Architecture of Nix...

http://nixos.org/nix/

- build processes
  - chroot, separate UIDs

Nix tools
- Nix language
- client lib

Nix build daemon

RPCs
From the Architecture of Nix...

http://nixos.org/nix/

build processes
chroot, separate UIDs

Bash, make, etc.

Bash, make, etc.

Bash, make, etc.

Nix tools

Nix language

client lib

Nix build daemon

RPCs
From the Architecture of Nix...

http://nixos.org/nix/

Nix tools

- Nix language
- client lib

build processes
chroot, separate UIDs

Bash, make, etc.
Bash, make, etc.
Bash, make, etc.

Nix build daemon

RPCs
... to the Architecture of Guix

http://gnu.org/s/guix/

**build processes**
chroot, separate UIDs

Guile, make, etc.

Guile, make, etc.

Guile, make, etc.

**Guile Scheme**

(guix packages)

(guix store)

**Nix build daemon**

RPCs
1. Scheme is a “programmable programming language” → tailored EDSLs
2. general-purpose language with compiler, debugger, libraries, etc.
3. a single language → more code reuse, unified environment
4. complete package programming interface
Bit-Reproducible Builds*

* almost!

```
$ guix build petsc
```

**isolated build:** chroot, separate name spaces, etc.
Bit-Reproducible Builds*

* almost!

$ guix build petsc

-gnu/store/ h2g4sf72... -petsc-3.6.0

hash of all the dependencies
Bit-Reproducible Builds

* almost!

$ guix build petsc
/gnu/store/ h2g4sf72... -petsc-3.6.0

$ guix gc --references /gnu/store/...-petsc-3.6.0
/gnu/store/...-openmpi-1.8.5
/gnu/store/...-gfortran-4.9.3-lib
/gnu/store/...-superlu-4.3
/gnu/store/...-lapack-3.5.0
/gnu/store/...-glibc-2.21
...

Bit-Reproducible Builds*

* almost!

$ guix build petsc
/gnu/store/ h2g4sf72... -petsc-3.6.0

$ guix gc --references /gnu/store/...-petsc-3.6.0
/gnu/store/...-openmpi-1.8.5
/gnu/store/...-gfortran-4.9.3-lib
/gnu/store/...-superlu-4.3
/gnu/store/...-lapack-3.5.0
/gnu/store/...-glibc-2.21
...
(nearly) bit-identical for everyone
Reproducible Environments

- per-user “profiles”
- non-interference among users/profiles
- transactional upgrades & rollbacks
Reproducible Environments

$ guix package -i gcc-toolchain coreutils sed grep
...

$ eval 'guix package --search-paths'
...

$ guix package --manifest=my-software.scm
...

$ guix environment petsc
...

demo

$ guix environment --ad-hoc python-ipython python-numpy \ -E ipython
...

Experience at the Max Delbrück Center, Berlin

- Guix deployed on two clusters ($\approx$250 nodes) + workstations
- used by bioinformatics researchers
- 50+ bioinfo packages in use (C/C++, Python, etc.)
- replaces CentOS packages + sysadmin-managed software
Experience at the Max Delbrück Center, Berlin

- Guix deployed on two clusters (≈250 nodes) + workstations
- used by bioinformatics researchers
- 50+ bioinfo packages in use (C/C++, Python, etc.)
- replaces CentOS packages + sysadmin-managed software
- **advantages:** more user control, better resource usage, ...
Beyond Reproducibility: Supporting Experimentation
Fiddling with the HPC Stack

Example from Inria

- **linear algebra**
  - PaSTiX, Chameleon (solvers)

- **run-time**
  - StarPU (task scheduling)
  - hwloc (hardware topology)
  - MPI (message passing)
  - compiler’s run-time support, C library, etc.
Requirements for an Experimentation-Capable System

1. customize + non-ambiguously specify package DAG
2. reliably reproduce variants of the DAG
(define starpu

  (package

    (name "starpu")
    (version "1.1.4")
    (source (origin
              (method url-fetch)
              (uri "http://...")
              (sha256 (base32 "0zmkw...")))))

  (build-system gnu-build-system)

  (native-inputs '(
      "pkg-config" ,pkg-config))

  (inputs '(
      "fftw" ,fftw)
      ("hwloc" ,hwloc))

  (home-page "http://starpu.gforge.inria.fr/")
  (synopsis "Run-time system for heterogeneous computing")
  (description "Blah...")
  (license lgpl2.1+)))
(define starpu
  (package
    (name "starpu")
    (version "1.1.4")
    (source (origin
              (uri "http://...")
              (sha256 (base32 "0zmkw..."))))
    (build-system gnu-build-system)
    (native-inputs '(('pkg-config' ,pkg-config)))
    (inputs '(('fftw' ,fftw)
              ('"hwloc" ,hwloc))))
  (home-page "http://starpu.gforge.inria.fr/")
  (synopsis "Run-time system for heterogeneous computing")
  (description "Blah...")
  (license lgpl2.1+)))
guix graph --type=package starpu

10 nodes
DAG of Package Objects

starpu-1.1.4

fftw-3.3.4

perl-5.16.1

ncurses-5.9

numactl-2.0.9

expat-2.1.0

libpciaccess-0.13.2

zlib-1.2.7

Where are GCC, libc, etc.?
Same DAG, including implicit inputs

29 nodes  guix graph --type=bag-emerged starpu
Same DAG, including implicit inputs

What about the compiler’s compiler, etc.?  

29 nodes  
guix graph --type=bag-emerged starpu
Full DAG, including bootstrap

(Too big)

321 nodes

`guix graph --type=bag starpu`
(define starpu-1.2rc ;release candidate
  (package (inherit starpu)
    (version "1.2.0rc2")
    (source (origin
      (method url-fetch)
      (uri (string-append "http://.../
        "starpu-" version ".tar.gz"))
      (sha256 (base32 "0qgb6y..."))))))

Defining Package Variants
(define starpu-with-simgrid
 (package (inherit starpu)
 (name "starpu-with-simgrid")

 ;; Add SimGrid, an optional dependency.
 (inputs '(("simgrid" ,simgrid)
 ,@(package-inputs starpu))))
(define (make-chameleon starpu)
  ;; Return the Chameleon solver linked against
  ;; this particular variant of StarPU.
  (package
    ;; ...
    (inputs `(('"starpu" , starpu)
               ("blas" , atlas)
               ("lapack" , lapack)
               ("gfortran" , gfortran-4.8)
               ("python" , python-2))))

(define (make-chameleon starpu)
  ;; Return the Chameleon solver linked against
  ;; this particular variant of StarPU.
  (package
    ;; ...
    (inputs `(("starpu" , starpu)
      ("blas" ,atlas)
      ("lapack" ,lapack)
      ("gfortran" ,gfortran-4.8)
      ("python" ,python-2))))

(define chameleon
  (make-chameleon starpu))
(define chameleon/starpu-simgrid
  (make-chameleon starpu-with-simgrid))
Conclusion
Limitations

- Daemon must run as root to isolate builds
- Non-deterministic build systems
- Non-free software unavailable in Guix
- WIP: have daemon rely on user namespaces (Linux 3.8+)
- WIP thanks to http://reproducible.debian.net
- Would you do chemistry research out of magic potions?
- Reproducible research demands free software
Limitations

- daemon must run as root to isolate builds
  - WIP: have daemon rely on user name spaces (Linux 3.8+)

- non-deterministic build systems

- non-free software unavailable in Guix

- would you do chemistry research out of magic potions?

- reproducible research demands free software
Limitations

- daemon must run as root to isolate builds
  - WIP: have daemon rely on user name spaces (Linux 3.8+)

- non-deterministic build systems
  - must be identified & fixed upstream
  - WIP thanks to http://reproducible.debian.net

- non-free software unavailable in Guix
Limitations

- daemon must run as root to isolate builds
  - WIP: have daemon rely on **user name spaces** (Linux 3.8+)

- non-deterministic build systems
  - must be identified & **fixed upstream**
  - WIP thanks to [http://reproducible.debian.net](http://reproducible.debian.net)

- non-free software unavailable in Guix
  - would you do chemistry research out of magic potions?
Limitations

- daemon must run as root to isolate builds
  - WIP: have daemon rely on user name spaces (Linux 3.8+)

- non-deterministic build systems
  - must be identified & fixed upstream
  - WIP thanks to http://reproducible.debian.net

- non-free software unavailable in Guix
  - would you do chemistry research out of magic potions?
  - *reproducible research demands free software*
Guix allows cluster users to reproduce environments.
Guix allows *cluster users* to reproduce environments. It provides *the source of software environments*, not just the bits.
Summary

- Guix allows **cluster users** to reproduce environments
- it provides **the source of software environments**, not just the bits
- **composability, transparency, and hackability** of software stacks are key to reproducible research