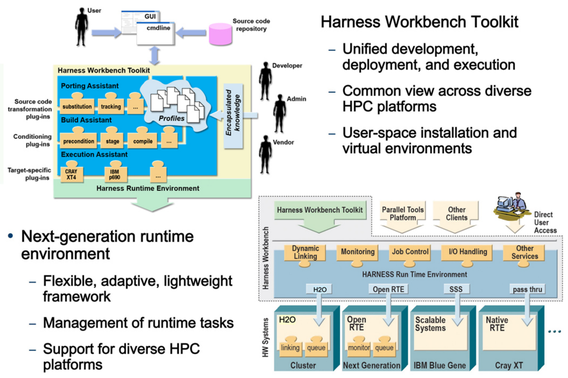
The Harness Workbench:
Unified and Adaptive Access to Diverse High-Performance Computing Platforms

Al Geist, Christian Engelmann (Oak Ridge National Laboratory), Jack Dongarra, George Bosilca (University of Tennessee, Knoxville)
Vaidy Sunderam, Magdalena Sławińska, Jarosław Sławiński (Emory University)

Harness workbench core components

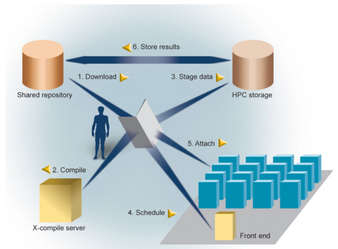


Harness Workbench Toolkit
- Unified development, deployment, and execution
- Common view across diverse HPC platforms
- User-space installation and virtual environments

- Next-generation runtime environment
  - Flexible, adaptive, lightweight framework
  - Management of runtime tasks
  - Support for diverse HPC platforms

Research and development goals

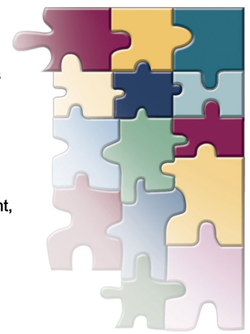
- Increasing the overall productivity of developing and executing computational codes
- Optimizing the development and deployment processes of scientific applications
- Simplifying the activities of application scientists, using uniform and adaptive solutions
- "Automagically" supporting the diversity of existing and emerging HPC architectures



Typical scientific application development, deployment, and execution activities

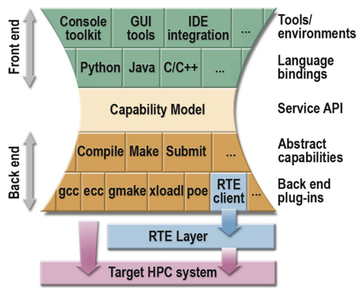
Harness workbench core technologies

- Automatic adaptation using pluggable modules
  - Harness Workbench Toolkit plug-ins
  - Runtime environment plug-ins
- Development environment and toolkit interfaces
  - Easy-to-use interfaces for scientific application development, deployment, and execution



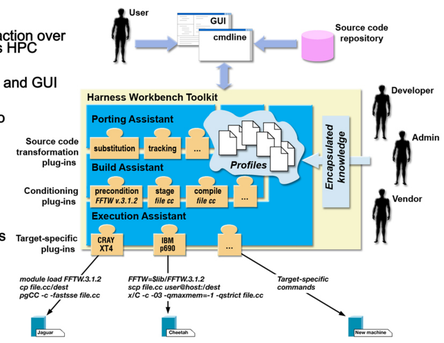
Common view across diverse platforms

- Various interfaces and bindings to external development and deployment tools and environments
- Generalized model for unified access to common development and deployment activities
- Mapping of generalized activities onto platform-specific toolkits and runtime environments (RTEs) via pluggable modules



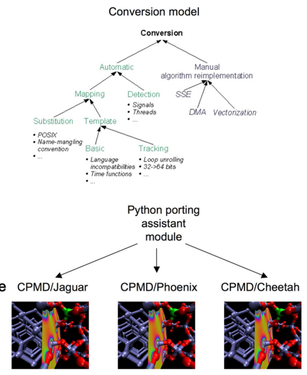
Harness workbench toolkit (HWT)

- Unifying abstraction over heterogeneous HPC resources
- Command line and GUI tools
- Translation into fine-tuned invocations of native toolkits
- Behavior encapsulated in plug-ins
- Configurable through profiles
- Tunable by end users



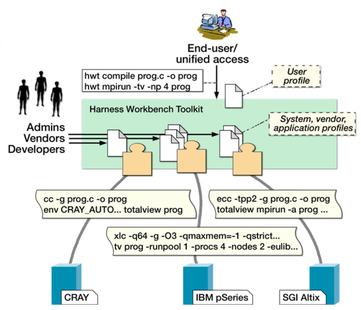
HWT porting assistant

- Facilitates source code adaptation through specialized plug-ins
  - Suggests safe conversions
  - Highlights manual code replacement areas
  - Guided by situation-specific profiles
- Prototype
  - Plug-ins as Python scripts
  - Ported CPMD across Jaguar (Cray XT3), Phoenix (Cray X1), Cheetah (IBM p690)
  - Example conversions: detection, function mappings, data type size changes



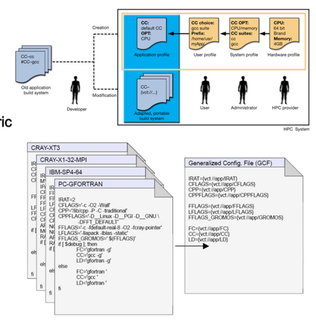
HWT virtualized command toolkit (VCT)
HWT build and execution layer

- Abstraction layer between users and platform-specific compilers, linkers, libraries, testing and debugging software, launching systems, etc
- End-users issue generic build commands that are processed to produce a target-specific set of commands
- Pluggable modules to deliver back-end functionality

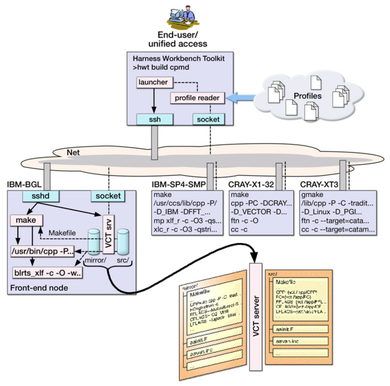


VCT prototype approach

- Target-specific knowledge encapsulated in profiles
- Generic build-related files
- *Late binding* – concretizing generic build files at runtime



VCT prototype architecture



VCT prototype use-case

Scenario

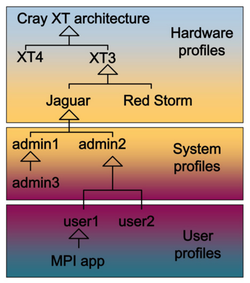
1. End-users stage-in application source codes with application's generic build system on the front-end node to 'src/'
2. HWT starts VCT server that mirrors 'src/'
3. The build system is launched on the front-end node in 'mirror/'
4. OS redirects filesystem operations to VCT server (e.g. reading Makefile)
5. To concretize generic files VCT server interacts with HWT and, based on profiles, resolves *vct references* at runtime through *late binding*

Implementation details

- VCT server exploits FUSE (Filesystem in USErspace)
- Python scripts
- Tested on CPMD

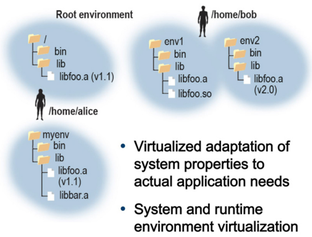
HWT profiles

- Declarative descriptions created by vendors, administrators, developers and adjusted by users
- Encapsulate target-specific knowledge at the system and application levels
- Support mechanisms
  - *Inheritance*
  - *Dynamic recursive resolution* to allow switching among predefined settings



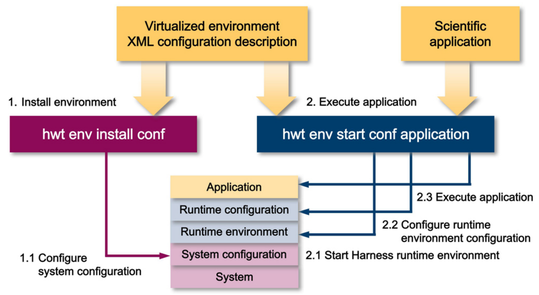
HWT virtualized environments

- Problem:
  - Application dependencies may cause conflicts with system-wide installed libraries.
- Solution:
  - Use co-existing, alternative user-space installations.
- Approach:
  - Provide isolated installation environments ("sandboxes").
  - These can inherit from one another to build nested hierarchies.

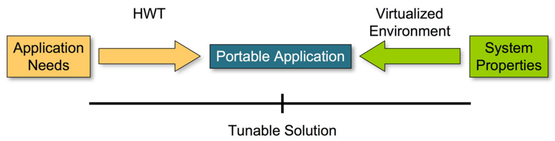


- Virtualized adaptation of system properties to actual application needs
- System and runtime environment virtualization

Configurable "sandboxes" for scientific applications



Tunable scientific application portability for day-one operation



- HWT for adaptation of application to HPC system properties
- Virtualized environment for adaptation of HPC system to application needs
- Tunable solution, where applications can move from one HPC system to another with initial minimal changes enabling day-one operation
- Continued adaptation to actual target platform properties assures eventual performance gains by gradually taking advantage of new system features

Point of Contact: Al Geist (geistgaii@ornl.gov)
www.csm.ornl.gov/harness | icl.cs.utk.edu/harness | dcl.mathcs.emory.edu

Office of Science
U.S. DEPARTMENT OF ENERGY

OAK RIDGE National Laboratory

EMORY UNIVERSITY