A Practical, SCVM-based Approach to Enhance Portability and Adaptability of HPC Application Build Systems

Magdalena Slawinska, Jaroslaw Slawinski, Vaidy Sunderam

***Abstract*—We describe a novel approach, based on the concept of a "system-call virtual machine" (SCVM), to enhancing portability of the HPC application deployment process across heterogeneous high-end machines. The SCVM approach to portable builds is based on the insertion of toolkit-interpretable directives into original application build scripts. Modifications resulting from these directives preserve the semantics of the original build instruction flow. The execution of the build script is controlled by our toolkit that intercepts build script commands in a manner transparent to the end-user. In order to intercept script commands we utilize *strace* system calls. We have applied this approach to a scientific production code (Gamess-US) on the Cray-XT5 machine.**

***Index Terms*—application-build, high-performance computing, portability**

I. Introduction

Rapid advances in capability-class computing enable scientific breakthroughs but pose multiple challenges for scientific software. Most high-end applications may be characterized as *continually evolving legacy codes* originally written, and updated over time, for platforms several generations removed from contemporary hardware. Changes include scaling to thousands of PEs, interconnect networks faster by many orders of magnitude, and most recently, heterogeneous multi- and many core architectures that demand new programming approaches. Adapting large application codes to execute efficiently on these emerging platforms has therefore become increasingly challenging [1].

In this paper we focus on the "build" aspect of this adaptation process and propose a new approach to enhancing portability of application build systems across different platforms. This will help improve the adaptability of build systems to frequent software updates and hardware upgrades in large HPC settings. This research is part of the *Harness Workbench Toolkit* (HWT) and *Enhancing Cyber-Infrastructure Usability* (ADAPT) projects that aim to support end-users through the entire HPC application life cycle by streamlining the build, providing software assistance at each build stage and simplifying deployment on varied target platforms.

Addressing the build aspect is important for several reasons. Firstly, in order to enable end-users to build highly optimized and efficient executables for a particular target machine, large HPC settings often offer a number of different compilers (each in a few versions) and a set of libraries for a given architecture (often in many flavors). Available compilers' options, applications' requirements, and the target machine characteristics lead to the explosion of possible combinations of build factors and make building scientific software challenging. Secondly, in high-end computing centers, cross-compilation becomes an issue as the service nodes where the compilation takes place often differ from the compute nodes where the binaries are executed. We note that common build systems such as GNU Autotools do not support cross-compilation [2]. Thirdly, building efficient, optimized, and correct executables on cutting-edge machines requires substantial cross-domain expert knowledge. Currently this knowledge is difficult to share and maintain, posing problems related to a knowledge reusage, especially, for non-expert end-users who are often distracted by the necessity of dealing with build-related issues.

Although there are efforts to relieve both application scientists and site administrators from the build burden (e.g., minimizing differences between service and compute nodes in terms of hardware and system software), such efforts address the build problem only partially, leaving many build issues unresolved.

The HWT/ADAPT approach to enhance application build systems' portability and adaptability is based on toolkit-interpretable directives embedded into original application build systems. Directives allow retrieval of situation-specific build-related knowledge from our ontology-driven *profiles* that organize expert knowledge in a semantic manner. The application build systems are executed under control of our toolkit that intercepts build script commands at run time to obtain target-specific settings for the particular application build. We note that since the directives are inserted as comments in original build scripts, it is possible to execute the build scripts in the exact same manner as the original scripts (adding comments does not influence the execution flow).

In this paper we describe the (1) *static* modification of the original build systems (scripts, source codes, etc) to insert the relevant directives, and (2) *dynamic* modification to apply build-specific values during build scripts execution. For the dynamic modification we exploit *system call virtual machine* (SCVM), specifically Umview [3], that allows to influence the behavior of the executed process transparently to the user. As a proof-of-concept we present our initial experiences with a production molecular dynamics code, Gamess (US) [4], on LCF ORNL's Jaguar. Encapsulating the expert HPC knowledge is an interesting and complex research topic that requires a separate discussion not provided in detail in this paper and can be found elsewhere [5].

Manuscript received November 30, 2011; revised December 27, 2011.
Research supported in part by US National Science Foundation Grant OCI-1124418.

Math and Computer Science, Emory University; Atlanta, GA 30322, USA; magg@gatech.edu, jaross@mathcs.emory.edu, vss@emory.edu

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

II. Related Work

A few tools and projects aim to address the build issues related to environment management, packaging, and automating the build process.

The Environment Modules project [6], [7] helps manage the configuration of build environments. The end-users configure their 'build' machine by *loading* or *unloading* previously defined *modules* that set or unset relevant environment variables. Modules can operate in tandem with script *wrappers*, prepared by site administrators or vendors, that invoke the actual compiler with its respective options. Wrappers and modules relieve end-users from the 'best-compiler-best-options-selection' dilemma by providing the 'abstract' compilers (e.g., cc, ftn). Since the actual compilers are invoked indirectly through wrappers, the responsibility for providing relevant options for a given compiler is at the wrappers' developer's side. We propose an ontology-driven profile-based approach to encapsulate and use the build-specific knowledge. Our *profiles* allow to encapsulate knowledge related to target computational platforms, application settings, compiler options, and provide mechanisms to retrieve *semantically* relevant build-case data for a specific target platform such as environment variables, compiler options, optimization levels, compatible libraries or system software, etc. The proposed, profile-driven approach reduces the necessity of switching build contexts (as it is in the case of Modules), and enables precise compilation tuning at the file- or even function-level. It also allows to deal with situations when the build process needs to produce executables for both compute nodes (target machine) and service nodes (build machine) as it is for instance in the Gamess case [4].

In order to relieve end-users from the build burden related to resolving dependencies, compilation, installation, and maintenance issues, a few projects proposed a package-based approach by automating those tasks, if possible. The ReST project [8] allows to create a binary or source software package with all necessary installation and deployment data such as dependencies, options or a configuration process. NetBuild [9] aims to help with the selection of appropriate library dependencies on various target machines. The NetBuild client obtains a target architecture-specific library package from a well-known location and links it into the application compiled for the target machine. The Repository in a Box (RIB) [10], a complementary project to ReST and NetBuild, allows to catalog software and present software metadata to end-users as web pages. The CheckInstall [11] project helps keep track of software installed from source codes by automating the 'packaging' process during the routine installation. When the compilation is done, CheckInstall automatically creates and installs a package (Slackware, RPM, or Debian) with an appropriate package manager as a regular binary package. Our approach is complementary to the package-based projects as it helps create target-specific packages.

Ideally, a build system should support compilation (including cross-compilation), build, deployment, optimization for target machines. To some extent this is provided by current build systems such as GNU Autotools [12] or SCons [13]. Unfortunately, GNU Autotools introduces its own compatibility issues (e.g., requiring compatible versions at the user's and developer's sides [2]) and does not address well cross-compilation that is common at large HPC settings. One approach to address the 'build' problem would be proposing a new build system. This, however, would require rewriting many build systems for legacy software, extensively utilized in large HPC settings. We note that our prior analysis of build systems of HPC applications indicates that build systems are highly diversified and range from proprietary shell scripts, through makefiles, to GNU Autotools. Rewriting those build systems would likely involve a lot of effort. In this work, we propose an evolutionary step toward new build systems. It is based on extracting the build-related knowledge from the build systems and encapsulating it into profiles. We modify original build systems by inserting toolkit-controlled directives that allow to guide the build system via the information retrieved from profiles to act appropriately to the actual situation-specific build case.

We note that the proposed profile-driven approach enables sharing the expert build-related knowledge. For instance, it can be used in the dashboard systems such as eSimMon [14] to provide data about a particular build case (e.g., used compilers and specific compiling options, optimization parameters for a particular libraries, etc).

III. The HWT Approach to Portable Builds

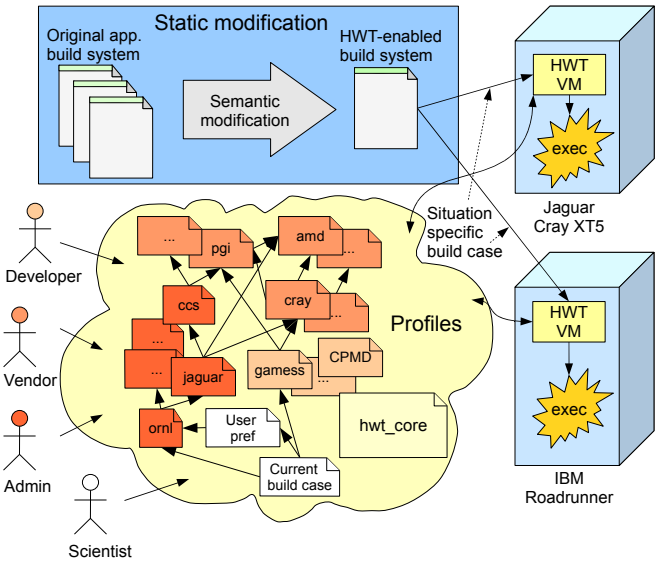

Fig. 1. The HWT approach to portable builds across different target architectures

The HWT/ADAPT project aims to support end-users through the entire build life cycle. In this section we describe the key concepts of the HWT that enable cross-platform portability of builds.

Figure 1 schematically presents the HWT approach to portable builds. Our approach to portability of builds across various HPC platforms is driven by ontology-based profiles that encapsulate build-related knowledge at different levels (application, system, user), and are provided independently by site administrators, developers, and users. They contain build-related knowledge with respect to target computational platforms, application requirements, system software (e.g.,

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

settings for environment variables, library dependencies, compiler options, optimization levels).

Current build systems allow to define *meaningless* values in forms of strings, numbers, expressions, objects, etc, that can be interpreted in ambiguous ways by different end-users (vendors, developers, site administrators, scientists). Ontologies not only support assigning *meanings* to concepts but go beyond that and provide mechanisms for their unambiguous interpretation by different subjects [15]. In order to implement profiles, HWT utilizes a few semantic standards and tools: OWL [16] for profiles, SPARQL [17] and Jena-Pellet reasoners [18], [19] for querying profiles. During the course of the HWT project we developed preliminary profiles for the Gamess application (application-level profile), the ORNL Jaguar XT5 platform (system-level profile), and the Gamess-Jaguar build case profile (user-level profile) containing data related to the customized build of Gamess for the Jaguar system. We used those profiles for our experiments described in Section IV. The interested reader can find more details about ontology-based profiles in our previous work [5].

As illustrated in Figure 1, in order to utilize the knowledge stored in profiles, the original build system requires a *static* modification at the source code level and *dynamic* modification at the build execution time. Static modifications regard inserting HWT directives into the source codes of the build scripts and/or applications. The dynamic modification is performed at the time of processing directives by the HWT virtual machine, called *hwtvm*. The HWT obtains actual target-specific values for a particular build case by querying *profiles*. The example fragments of the toolkit-enabled build system are presented in Listing 2.

The HWT approach to portable builds is motivated by the 'transparent adaptation' of the original build system to allow for execution both in the HWT-controlled environment as well as in the non-HWT environment. To achieve this, the toolkit-interpretable directives are inserted as build script comments, as shown in Listing 2. The directives are implemented in the same programming dialect as the original build system, and are preceded by #HWT to indicate that they should be interpreted by *hwtvm*.

Inserting HWT directives as comments helps document build scripts in a semantic manner as HWT queries are semantically-oriented. It also allows to execute the HWT-enabled build system in the non-HWT controlled environment, since inserted HWT directives are ordinary shell comments and therefore ignored by the executing shell. In order to enable the interpretation of HWT directives in a user-transparent manner, the HWT implements a system call virtual machine (SCVM). SCVM enables redefinition of syscalls to virtualize the execution environment of a process and its subprocesses.

In the following sections we will describe the static and dynamic modifications.

A. Static Modifications

In our previous work [5], we proposed the modification of the original build system in order to make it generic. Necessary changes regarded modifying hard coded values such as `CCOMP=cc` to respective semantic queries such as `CCOMP=`$QUERY "$PREFIXES select ?p where {$CC hc:path ?p}"`. This usually resulted in maintaining two separate versions of a build system: original and HWT-enabled. In order to reduce the maintenance effort, in this work we propose another approach that allows to maintain one HWT-interpretable build system. The relevant build script fragments that lack meanings are changed to their semantic counterpart versions in exactly the same programming dialect as the original build script. The changed fragments are preceded by HWT 'markers' (i.e., #HWT). We refer to the '#HWT' marked statements as HWT directives. The HWT directives are comments in the original build system and in fact, they document meanings of respective values in original build scripts. The fragments of original build files and the corresponding example modification are presented in Listing 1 and Listing 2, respectively.

Listing 1. Fragments of the original build system (Gamess-US)

```
1  set TARGET=cray-xt
2
3  ...
4
5  if ($TARGET == cray-x1) then
6     sed -e "s/\*UNX/    /"
   actvte.code > actvte.tmp
7     sed -e "s/\*CRY/    /"
   actvte.tmp > actvte.f
8     rm actvte.tmp
9     ftn -Ocommand -o actvte.x actvte.f
10    rm actvte.f
11 endif
12 if ($TARGET == cray-xd1) then
13    sed -e "s/\*UNX/    /"
   actvte.code > actvte.f
14    pgf90 -o actvte.x actvte.f
15    rm actvte.f
16 endif
17 #  For the cray-xt
18 if ($TARGET == cray-xt) then
19    sed -e "s/\*UNX/    /"
   actvte.code > actvte.f
20    ftn -o actvte.x actvte.f
21    rm actvte.f
22 endif
```

In order to retrieve data which are stored in profiles, the HWT resolves SQL-like queries (specifically SPARQL [17] that permits formulation of queries close to natural language and is a natural option for our OWL-based profile implementation), and applies the obtained results as appropriate values in build scripts. In particular, to support querying profiles the HWT predefines a few commands such as ont_prefix, ont_query, and files to interface query's results (e.g., HWT_QRESULT, HWT_QRESULT_COL1, HWT_QRESULT_COL2, etc).

The static modification alters unportable fragments of original build scripts, and leaves portable fragments unmodified. The unportable fragments and their portable HWT-enabled counterparts coexist in the same modified build script. In order to exclude original unportable fragments from the build script during the HWT-controlled execution, and enable the execution of portable HWT counterparts instead, a *here document* notation is used (lines 12–20 in Listing 2). A here document is a common way to specify a string literal in command line shells and scripting languages.

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Listing 2. Enabling the original build system for the HWT interpretation. The modifications correspond to fragments in Listing 1

```
#HWT ont_prefix default http://dcl.emory/hwt/ont/gamess-us.owl
#HWT ont_query "select ?bc where {?bc ac:builds [a :Gamess-US]}" ||\
#HWT   (echo "query error"; exit 1)
#HWT [ -s HWT_QRESULT ] && set bc=`cat !#:2` ||\
#HWT   (echo "no answer"; exit 1)

set TARGET=cray-xt
#HWT ont_query "select ?target where {$bc ac:buildTarget ?target}"
#HWT set TARGET=`cat HWT_QRESULT`
...
#HWT #mute all following ifs
#HWT cat > /dev/null <<MUTE_IFS
if ($TARGET == cray-x1) then
   ...
endif
...
if ($TARGET == cray-xt) then
  ...
endif
#HWT MUTE_IFS

#HWT #developer knows for which platforms source needs activation
#HWT ont_query "ask {$TARGET a :SrcToActivation}"
#HWT if (`cat HWT_QRESULT`) than
#HWT   #is target (according to developer) UNIX compatible?
#HWT   ont_query "ask {$TARGET a :UnixCompatible}"
#HWT   if (`cat HWT_QRESULT`) \
#HWT     sed -e "s/\*UNX/    /" actvte.code > actvte.f
#HWT   ont_query "ask {$TARGET a ac:Cray, ac:Vectorized}"
#HWT   if (`cat HWT_QRESULT`) \
#HWT     sed -e "s/\*CRY/    /" actvte.code > actvte.f
#HWT   #get Fortran 77 path for Service Node (not for the target!)
#HWT   ont_query "select ?snFtn where {$bc ac:buildEnvironment ?env .\
#HWT     ?env ac:installedSoftware [a ac:F77Compiler; \
#HWT       ac:compatibleWith ?env; ac:path ?snFtn] }"
#HWT   [ -s HWT_QRESULT ] && set snFtn=`cat !#:2` \
#HWT     || (echo "no local Fortran 77 found"; exit 1)
#HWT   $snFtn-o actvte.x actvte.f
#HWT   rm actvte.f
#HWT endif
```

Enhancing portability of original build systems with the HWT approach requires a one-time effort to reimplement fragments that break cross-platform portability. Once this is done, adding a new architecture requires configuring appropriate settings at the profile level, instead of the commonly practiced copy-paste-modify.

Although modified build scripts can be executed in a traditional manner (prior to modifications) since modifications are implemented as comments and therefore ignored during ordinary execution, they are intended to be executed under HWT control. One approach to interpret HWT-enabled build scripts is to develop an HWT build script interpreter. However, this might be complicated due to a variety of build systems utilized by HPC applications (different command line shell flavors, scripting and programming languages, etc). We propose to address HWT-directives interpretation by intercepting syscalls.

B. Dynamic Modifications

The HWT-controlled execution of modified build scripts assumes suppressing HWT markers (i.e., '#HWT' in Listing 2) in order to enable execution of both portable original build script statements and HWT semantic statements (commented by HWT markers). Said differently, the HWT executes the modified build script but the modifications need to be revealed before or 'just-in-time' of the actual execution.

To accomplish this in an automatic manner, we considered two approaches: (1) a *link-based* approach that takes advantage of links in Unix-like systems, and (2) implementing an HWT virtual machine based on the exploitation of the syscall interception. In the link-based approach, first the HWT creates a separate directory with symbolic links to source files, next preprocesses the modified build system, i.e., deletes HWT markers, and finally executes the preprocessed build system. The HWT commands (ont_prefix, ont_query, etc) in this context can be implemented as scripts. The link-based approach is a simple yet effective solution. However, the HWT aims to support comprehensive virtual build environments to facilitate isolated user-space installations. In this context, more advanced mechanisms are required that enable command and file virtualization. Aside from that, the link-based approach does not provide mechanisms to enable cross-compilation that is plausible in the virtualization-based approach. Therefore, we focused on the HWT virtual machine approach that utilizes syscall interception mechanisms.

Developing the HWT virtual machine allows to implement 'just-in-time' modification of a file content. The naive

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

implementation requires intercepting one filesystem syscall, namely read, and changing the read content in place (without changing its size). Intercepting syscalls such as read, open, and execve allows for precise control of the build execution and dynamic redefinition of its behavior.

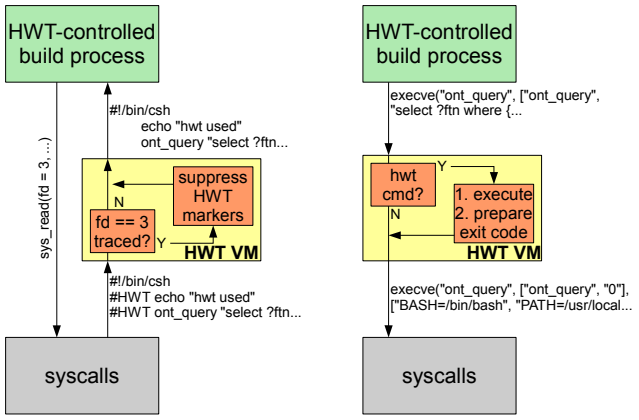

Fig. 2. The selective interception and selective overriding syscalls in HWT VM (the syscall execve does not return unless it ends with an error)

From the process' standpoint, overriding syscalls takes a *global* effect in the perception of the entire operating system. In order to increase performance and permit overriding syscalls selectively for specific (e.g., build-related) files, the HWT requires tracing pathnames of files used by the process as shown in Figure 2. Implementing this requires intercepting four other filesystem syscalls (apart from read), namely, open, close, dup, and dup2.

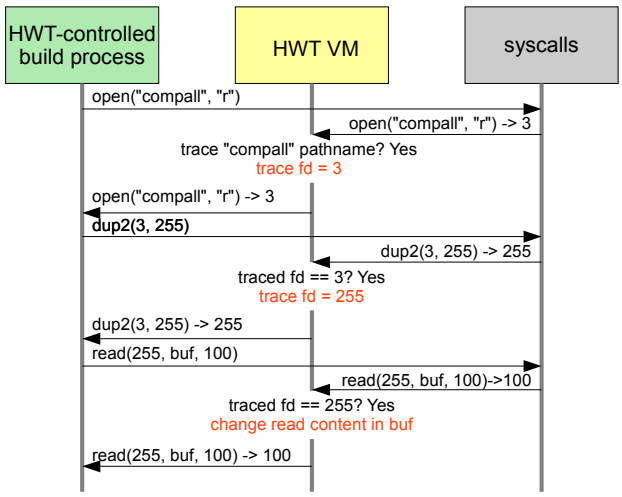

Fig. 3. Tracing relations between file descriptors and their pathnames in HWT VM

These calls are needed to trace the relation between the file pathname and its descriptors (Figure 3). Tracing that relation is not supported by the operating system and different syscalls take either the file descriptor or the pathname as an input parameter (specifically, the syscall read takes a file descriptor as an input parameter while the syscall open takes a pathname).

We examined a few available techniques allowing to intercept syscalls at large HPC systems such as ORNL's Jaguar, namely Fuse [20], DLL injection, Umview [3], and ptrace [21].

a) Fuse: Fuse [20] is a Filesystem in Userspace and allows to mount userspace virtual filesystems. The userspace Fuse daemon can redefine filesystem operations. However, it requires installation of a kernel module that is an issue on service nodes at large HPC sites where Fuse is not supported for the sake of stability and maintenance.

b) DLL injection: The DLL injection technique (in Unix-like systems implemented by the environment variable LD_PRELOAD) is used to force loading a dynamic-link library to modify program behavior. However, in order to benefit from LD_PRELOAD, a program has to be dynamically linked. In order to utilize the DLL injection technique in the HWT, all system tools involved in the actual build needed to be dynamically linked. Clearly, relying on this assumption is problematic. Moreover, the variable LD_PRELOAD can be controlled by subprocesses resulting in disabling HWT-intended virtualization.

c) Umview: Umview [3] is an implementation of the system call virtual machine (SCVM) that allows to virtualize the execution environment of a process and enables redefinition of syscalls. It does not require root privileges to be installed; Umview can be installed in userspace. Aside from *filesystem* syscalls, Umview allows to redefine the process *execution-related* syscalls including execve. This allows to also address cross-compilation issues in build systems such as GNU Autotools that perform micro-tests to gather platform-specific knowledge for the build. Instead of performing those tests locally, the micro-tests execution can be intercepted and forced to execute on compute nodes. Although Umview, for the above reasons, initially seemed promising, it turned out that its usage encounters substantial obstacles on ORNL's Jaguar service nodes (the 64-bit AMD architecture with Suse Linux OS).

d) Ptrace: The interception functionality required by the HWT (five filesystem calls and execve) is offered by the syscall ptrace [21]. In fact, ptrace is at the core of Umview. The syscall allows to control processes and is primarily used in debuggers such as GNU GDB [22] and system call tracers, e.g., strace [23]. Ptrace and its derivatives can be used in userspace and do not require root privileges. However, programming the syscall ptrace can pose a challenge since the syscall manipulates at the register level. Therefore, we approach this challenge at the higher abstraction level by adapting sources of strace to implement the HWT virtual machine.

IV. Experiments

In order to verify the feasibility of the SCVM approach to portable builds we tested this approach against the Gamess-US build system on ORNL's Jaguar XT5. The reason for such a selection was the availability of respective application, system, and build case profiles—as we developed them earlier during the course of the HWT project. Gamess-US [4] is general computational chemistry software. Its build system consists of three essential csh scripts *comp*, *compall*, and *lked*, and the additional file *compddi* to compile the DDI library. The Gamess build scripts contain guidelines (scattered among all these files) for building the application for 26 target architectures.

To perform the experiment we modified the Gamess build system by manually inserting HWT directives as described

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

in Section III-A and executed it under control of the HWT syscall virtual machine called *hwtvm*. The prototype implementation of *hwtvm* is based on strace's source codes and allows to intercept six syscalls (open, read, close, dup, dup2, and execve) in order to appropriately execute the modified Gamess build system, i.e., it takes care of retrieving target-specific build related information from respective profiles. The HWT SCVM-based approach allows to build Gamess in two modes: traditionally and under *hwtvm* control. It starts to show its potential when the application needs to be compiled for a new target machine. Instead of tedious examination of the entire build system and adapting it to the new target architecture, the HWT user configures the build by creating the relevant build case profile and provides an URI of this profile to *hwtvm* as an input parameter.

V. Summary and Future Work

This paper describes the HWT SCVM-based approach to enhance portability of the scientific applications' build systems across various HPC platforms. The approach is based on the one-time modification of the original build system that requires the insertion of toolkit-interpretable directives and their interpretation by the HWT SCVM. HWT directives take advantage of the expert knowledge stored in HWT ontology-driven profiles. The modification preserves the original instruction flow and is transparent to the original build system in terms that it takes effect only when the modified build system is executed by *hwtvm*. The *hwtvm* is based on the ptrace syscall that allows to virtualize the environment of the process execution. The *hwtvm* intercepts syscalls to change the behavior of the executed build script at the script execution time. This allows to retrieve target-specific values from profiles in a manner transparent to the user. More importantly, the presented approach does not require root privileges as we demonstrated this by building the Gamess application for ORNL's Jaguar.

Our future work will focus on further exploration of the presented SCVM approach towards improving portability of builds. The ability to intercept the syscall execve opens an interesting research on addressing cross-compilation issues. For instance, GNU Autotools execution of micro-tests could be intercepted and performed on the target machine. The other open research issue is to investigate possibilities of automating the build system modification process that currently is performed manually, e.g., classification of patterns breaking portability or designing query patterns for retrieval data from profiles.

References

[1] L. Hochstein and V. R. Basili, "The ASC-Alliance Projects: A Case Study of Large-Scale Parallel Scientific Code Development," *IEEE Comp.*, vol. 41, no. 3, pp. 50–58, 2008.

[2] M. B. Doar, *Practical Development Environments, Chapter 5*. O'Reilly, Oct 2005.

[3] L. Gardenghi, M. Goldweber, and R. Davoli, "View-OS: A New Unifying Approach Against the Global View Assumption," in *ICCS '08: Proceedings of the 8th international conference on Computational Science, Part I*, (Berlin, Heidelberg), pp. 287–296, Springer-Verlag, 2008.

[4] Mark Gordon's Quantum Theory Group. Ames Laboratory/Iowa State University, "The General Atomic and Molecular Electronic Structure System (GAMESS)." http://www.msg.ameslab.gov/GAMESS/GAMESS.html, 2009.

[5] M. Slawinska, J. Slawinski, and V. Sunderam, "Enhancing Build-Portability for Scientific Applications Across Heterogeneous Platforms," in *Parallel and Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pp. 1–8, May 2009.

[6] NERSC, "Modules Approach to Software Management," 2008. http://www.nersc.gov/nusers/resources/software/os/modules.php.

[7] J. L. Furlani and P. W. Osel, "Environment Modules Project," 2005. http://modules.sourceforge.net/.

[8] E. Meek, J. Larkin, and J. Dongarra, "Remote Software Toolkit Installer," Tech. Rep. ICL-UT-05-04, ICL UT, June 2005.

[9] K. Moore and J. Dongarra, "NetBuild: Transparent Cross-Platform Access to Computational Software Libraries," *Concurrency and Computation: Practice and Experience, Special Issue: Grid Computing Environments*, vol. 14, pp. 1445–1456, Nov/Dec 2002.

[10] S. Moore, A. Baker, J. Dongarra, C. Halloy, and C. Ng, "Active Netlib: An Active Mathematical Software Collection for Inquiry-based Computational Science and Engineering Education," *Journal of Digital Information special issue on Interactivity in Digital Libraries*, vol. 2, no. 4, 2002.

[11] Felipe Eduardo Sanchez Diaz Duran, "The checkinstall project," 2009. http://www.asic-linux.com.mx/~izto/checkinstall/.

[12] G. V. Vaughan, B. Elliston, T. Tromey, and I. L. Taylor, *GNU Autoconf, Automake and Libtool*. New Riders publishing, 2000. http://sources.redhat.com/autobook/.

[13] S. Knight, "SCons User Guide 1.1.0," 2008. http://www.scons.org/doc/1.1.0/HTML/scons-user/book1.html.

[14] S. Klasky, R. Barreto, A. Kahn, M. Parashar, N. Podhorszki, S. Parker, D. Silver, and M. Vouk, "Collaborative visualization spaces for petascale simulations," *Collaborative Technologies and Systems, 2008. CTS 2008. International Symposium on*, May 2008.

[15] D. Allemang and J. Hendler, *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2008.

[16] W3C, "OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax, W3C Working Draft," Dec 2008. http://www.w3.org/TR/owl2-syntax/.

[17] W3C, "SPARQL Query Language for RDF," Jan 2008. http://www.w3.org/TR/rdf-sparql-query/.

[18] "Jena – A Semantic Web Framework for Java," Dec 2008. http://jena.sourceforge.net/.

[19] Clark & Parsia, LLC, "Pellet: The Open Source OWL DL Reasoner," Apr 2009. http://clarkparsia.com/pellet.

[20] "Filesystem in Userspace," 2010. http://fuse.sourceforge.net/.

[21] M. J. Rochkind, *Advanced UNIX programming*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1985.

[22] "GDB: The GNU Project Debugger," 2010. http://www.gnu.org/software/gdb/.

[23] "The strace project page," 2010. http://sourceforge.net/projects/strace/.

ISBN: 978-988-19251-1-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)