Cluster Computing with OpenHPC

Karl W. Schulz
Intel Corp.

C. Reese Baird
Intel Corp.

David Brayford
Leibniz Supercomputing Centre

Yiannis Georgiou
Atos

Gregory M. Kurtzer
Lawrence Berkeley National Laboratory

Derek Simmel
Pittsburgh Supercomputing Center

Thomas Sterling
Indiana University

Nirmala Sundararajan
Dell

Eric Van Hensbergen
ARM

ABSTRACT

OpenHPC is a newly formed, community-based project that is providing an integrated collection of HPC-centric software components that can be used to implement a full-featured reference HPC compute resource. Components span the entire HPC software ecosystem including provisioning and system administration tools, resource management, I/O services, development tools, numerical libraries, and performance analysis tools.

Common clustering tools and scientific libraries are distributed as pre-built and validated binaries and are meant to seamlessly layer on top of existing Linux distributions. The architecture of OpenHPC is intentionally modular to allow end users to pick and choose from the provided components, as well as to foster a community of open contribution. This paper presents an overview of the underlying community vision, governance structure, packaging conventions, build and release infrastructure and validation methodologies.

CCS Concepts

•**Social and professional topics** → **Software management;** *System management;* •**Software and its engineering** → **Software libraries and repositories;** *Development frameworks and environments;* •**Computer systems organization** → *Distributed architectures;*

1. INTRODUCTION

Launched initially in November 2015, and formalized as a collaborative Linux Foundation [6] project in June 2016, OpenHPC is a community driven project currently comprised of over 25 member organizations with representation from academia, research labs, and industry. To date, the OpenHPC software stack aggregates over 60 components ranging from tools for bare-metal provisioning, administration, and resource management to end-user development libraries that span a range of scientific/numerical uses. OpenHPC adopts a familiar repository delivery model with HPC-centric packaging in mind, and provides customizable recipes for installing and configuring reference designs of compute clusters. OpenHPC is intended both to make available current best practices and provide a framework for delivery of future innovation in cluster computing system software.

HPCSYSPROS '16 November 14, 2016, Salt Lake City, UT, USA

DOI: 10.1145/1235

2. COMMUNITY BUILDING BLOCKS FOR HPC SYSTEMS

2.1 Motivation

Many HPC sites spend considerable effort aggregating a large suite of open-source components to provide a capable HPC environment for their users. This is frequently motivated by the necessity to build and deploy HPC focused packages that are either absent or outdated in popular Linux distributions. Further, local packaging or customization typically tries to give software versioning access to users (e.g. via environment modules or similar equivalent). With this background motivation in mind, combined with a desire to minimize duplication and share best practices across sites, the OpenHPC community project was formed with the following mission and vision principles:

Mission: to provide a reference collection of open-source HPC software components and best practices, lowering barriers to deployment, advancement, and use of modern HPC methods and tools.
Vision: OpenHPC components and best practices will enable and accelerate innovation and discoveries by broadening access to state-of-the-art, open-source HPC methods and tools in a consistent environment, supported by a collaborative, worldwide community of HPC users, developers, researchers, administrators, and vendors.

The remaining sections provide a further overview of the community project by highlighting related work (§2.2), the technical governance structure (§2.3), repository enablement (§2.4), packaging conventions (§2.5), underlying build infrastructure (§2.6), and integration testing (§2.7).

2.2 Related Work

As highlighted in the previous section, the installation, deployment and configuration of HPC clusters can be tedious work. Consequently, a variety of open-source and proprietary cluster management tools have been created that provide different approaches to help address this problem. In the open-source arena, two widely known solutions have been Rocks and OSCAR. Rocks [25, 12] is an open-source Linux-based clustering solution that aims to reduce the complexity of building HPC clusters. It provides a combined bundle of an underlying CentOS distribution with additional software components covering the different administrative needs of a cluster, and offers a GUI to help administrators walk through different steps of the installation procedure. All cluster services and tools are installed and configured during the initial installation of the front-end with no need to download and configure other external packages. Furthermore, with the use of Rolls [14], the administrator can customize the base installation with additional optional software that integrates seamlessly into the management and packaging mechanisms of the base software. Rolls exist to deploy Rocks in alternative environments such as sensor networks [27] and clouds [24]. The most recent version of Rocks, version 6.2, was released in May 2015 and is based on CentOS (v6.6).

OSCAR [26, 10] (Open Source Cluster Application Resources) is a fully integrated and easy to install software bundle designed for high performance cluster computing. OSCAR follows a different methodology than Rocks; once the front-end is installed and booted, the cluster building components are downloaded and installed through tools that try to simplify the complexity of the different administrative tasks. There have been some variations of OSCAR based on the same framework to cover different types of cluster environments such as Thin-Oscar for diskless platforms, and HA-Oscar to support High Availability. OSCAR has previously been supported on multiple Linux distributions such as CentOS and Debian. However, the project is no longer actively maintained and the latest version 6.1.1 was released in 2011.

A common issue that arises in the design of system management tool kits is the inevitable trade-off between ease-of-use and customization flexibility. Rocks has adopted a more turn-key approach which necessitates the need for some level of embedded configuration management and Rocks leverages a hierarchical XML schema. As will be discussed further in §2.4, OpenHPC is providing an HPC focused software repository and adopts a more building-block approach. Consequently, it expects a certain level of expertise from the administrator, but is intended to offer a greater choice of software components, promote flexibility for use in a variety of system environments and scales, be compatible with multiple Linux distributions, and be interoperable with standalone configuration management systems. Furthermore, as a community effort, OpenHPC is supported and maintained by a group of vendors, research centers and laboratories that share common goals of minimizing duplicated effort and sharing of best practices.

Several end-user oriented projects also exist to mitigate the complexity of HPC and scientific software management including EasyBuild [19] and Spack [18]. Both of these systems provide convenient methods for building and installing many common HPC software packages. With similar goals, OpenHPC differs in scope and process. We aim to provide a complete cluster software stack capable of provisioning and administering a system in addition to user-space development libraries. OpenHPC also seeks to leverage standard Linux tools and practices to install and maintain software. Both EasyBuild and Spack are currently packaged in the OpenHPC distribution to allow users to further extend and customize their environment.

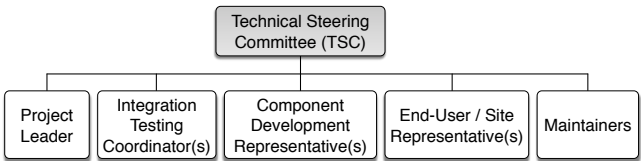

Figure 1: Identified roles within the OpenHPC Technical Steering Committee (TSC).

2.3 Governance & Community

Under the auspices of the Linux Foundation, OpenHPC has established a two pronged governance structure consisting of a governing board and a technical steering committee (TSC). The governing board is responsible for budgetary oversight, intellectual property policies, marketing, and long-term road map guidance. The TSC drives the technical aspects of the project including stack architecture, software component selection, builds and releases, and day-to-day project maintenance. Individual roles within the TSC are highlighted in Figure 1. These include common roles like maintainers and testing coordinators, but also include unique HPC roles designed to ensure influence, and capture points of view, from two key constituents. In particular, the *component development representative(s)* are included to represent the upstream development communities for software projects that might be integrated with the OpenHPC packaging collection. In contrast, the *end-user/site representative(s)* are downstream recipients of OpenHPC integration efforts and serve the interest of administrators and users of HPC systems that might leverage OpenHPC collateral. At present, there are nearly 20 community volunteers serving on the TSC [9] with representation from academia, industry, and government R&D laboratories.

2.4 Installation/Repository Overview

As mentioned previously, OpenHPC endeavors to adopt a repository-based delivery model similar to the underlying OS distributions commonly used as the basis for HPC Linux clusters. At present, OpenHPC is providing builds targeted against two supported OS distributions: CentOS7 and SLES12. The underlying package managers for these two distributions are **yum** and **zypper**, respectively, and OpenHPC provides public repositories that are compatible with these RPM-based package managers.

The installation procedure outlined in current OpenHPC recipes targets bare-metal systems and assumes that one of the supported base operating systems is first installed on a chosen *master* host. This is typically done leveraging bootable media from ISO images provided by the base OS and once installed, OpenHPC recipes highlight steps to install additional software and perform configurations to use the *master* host to provision the remaining cluster.

An overview of the physical infrastructure expected for use with current OpenHPC recipes is shown in Figure 2 and highlights the high-level networking configuration. The *master* host requires at least two Ethernet interfaces with *eth0* connected to the local data center network and *eth1* used to provision and manage the cluster backend (these interface names are examples and may be different depending on local settings and OS conventions). Two logical IP interfaces are expected to each compute node: the first is the standard Ethernet interface that will be used for provisioning and resource management. The second is used to connect to each host's baseboard management controller (BMC) and is used for power management and remote console access. Physical connectivity for these two logical IP networks is often accommodated via separate cabling and switching infrastructure; however, an alternate configuration can also be accommodated via the use of a shared NIC. For power management, we assume that the compute node BMCs are available via IPMI from the chosen master host. For file systems, the current recipe(s) document setting up the chosen master host as an NFS file system that is made available to the compute nodes. Installation information is also discussed to optionally include a Lustre [7] file system mount.

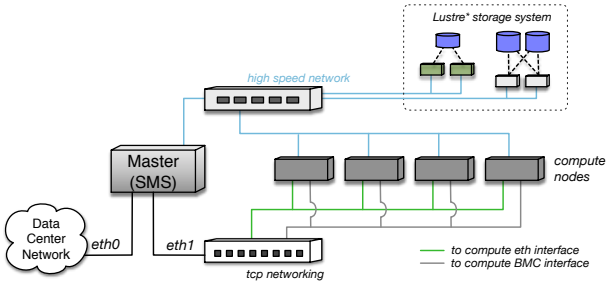

Figure 2: Overview of physical cluster infrastructure expected with OpenHPC installation recipes.

Community Repo: In cases where external network connectivity is available on the *master* host, OpenHPC provides an `ohpc-release` package that includes GPG keys for package signing and repository enablement. This package can be downloaded from the OpenHPC GitHub community site directly (https://github.com/openhpc/ohpc). Note that additional repositories may be required to resolve package dependencies and, in the case of CentOS, access to the EPEL [2] repo is currently required.

The most recent release branch for OpenHPC is version 1.1 and the output in Figure 3 highlights the typical repository setup after installation of the `openhpc-release-1.1` RPM in a CentOS environment. Following typical OS distro conventions, two OpenHPC repositories are enabled by default: a **base** repo corresponding to the original 1.1 release and an **updates** repo that provides rolling fixes and enhancements against the 1.1 tree.

2.5 Packaging

To highlight several aspects of the current packaging conventions, we next present several installation examples. Note that this discussion does not endeavor to replicate an entire install procedure, and interested readers are invited to consult the latest installation recipe(s) that are available on the community GitHub site (or via the installable `docs-ohpc`

```
# yum repolist
repo id            repo name
OpenHPC            OpenHPC-1.1 - Base
OpenHPC-updates    OpenHPC-1.1 - Updates
base               CentOS-7 - Base
epel               Extra Packages for Enterprise...
```

Figure 3: Typical package repository configuration after enabling OpenHPC (CentOS example).

RPM) for more detailed instructions.

Once the OpenHPC repository is enabled locally, a range of packages are available and a typical install on the *master* host begins with the installation of desired system administration services. In the example that follows, the Warewulf provisioning system [13] and SLURM resource manager is installed using available convenience groups:

```
[sms]# yum -i groupinstall ohpc-warewulf
[sms]# yum -i groupinstall ohpc-slurm-server
```

Figure 4: Example installations using convenience groups.

Convenience groups like the examples above are prefixed with the "ohpc-" tag and install a collection of related packages. As an example, the `ohpc-warewulf` group expands to include all the packages needed to enable a Warewulf provisioning server. Similarly, the `ohpc-slurm-server` group includes the packages needed to stand up a SLURM control daemon for resource management [21] across the cluster. Although not shown here, a related `ohpc-slurm-client` group is also available to allow for installation of a smaller set of packages needed to enable a SLURM client (typically installed in compute node images).

Note that individual packages provided via OpenHPC have their names appended with the "ohpc" suffix. The motivation for this convention was to allow for easy wild-carding queries with package managers, and to also provide the ability to install OpenHPC-packaged versions of software alongside of alternate distro versions of the same packages (if available). Finally, while the examples here continue to use the `yum` package manager, equivalent commands can be substituted using `zypper` when using SLES.

Development Libraries: In addition to providing tools primarily targeted at system administrators, OpenHPC also provides pre-packaged builds for a number of popular open-source tools and libraries used by HPC applications and developers. For example, OpenHPC includes a variety of builds for FFTW [15] and HDF5 [3] (including serial and parallel I/O support), and the GNU Scientific Library (GSL). A number of other development tools and libraries are included and the installation recipe(s) contain a detailed package manifest highlighting what is available for a given release. Note also that the list is expected to evolve and expand over time as additional software components are integrated within future releases.

General purpose HPC systems often rely on multiple compiler and MPI family toolchains [23] and OpenHPC supports this strategy via the adoption of a hierarchical build configuration that is cognizant of the need to deliver unique builds of a given software package for each desired compiler/MPI

permutation. The general naming convention for builds that have these toolchain dependencies is to append the compiler and MPI family name that the library was built against directly into the package name. For example, libraries that do not require MPI as part of the build process adopt the following RPM naming scheme:

```
package-<comp_fam>-ohpc-<ver>-<rel>.rpm
```

where `<comp_fam>` maps to the underlying compiler family and `<ver>` and `<rel>` correspond to the individual software version and build release number, respectively. Expanding on this convention, packages that also require MPI as part of the build additionally include the MPI family (`<mpi_fam>`) name as follows:

```
package-<comp_fam>-<mpi_fam>-ohpc-<ver>-<rel>.rpm
```

Given the large number of installation permutations possible for software supporting multiple compiler/MPI toolchains, combined with the fact that HPC sites also tend to make multiple versions of a particular component available to their users, there is a clear need to support a flexible development environment for end users. A popular historical choice in this space over the years has been the use of Environment Modules [16] to expose a `modules` command within a user's shell environment allowing them to load/unload desired software packages via management of key environment variables (e.g. `PATH` and `LD_LIBRARY_PATH`). Several implementations of the modules system have evolved and OpenHPC leverages a recent variant named *Lmod* [23, 4] which is Lua-based, and has embedded support for managing the hierarchical software matrix adopted in OpenHPC. In addition to providing a pre-packaged build of *Lmod*, development libraries and tools integrated within OpenHPC include the installation of companion module files. Consequently, once a desired package is installed, end users can then access and query the software through the underlying modules system. The packaging process includes a consistent set of environment variables for users to access a particular package's path for available header files and dynamic libraries. As an example, consider the following installation of the PETSc [11] scientific toolkit built using the GNU compiler and MVAPICH2 [20] MPI toolchain.

```
[sms]# yum install petsc-gnu-mvapich2-ohpc
```

Figure 5: Installation of PETSc for a particular compiler/MPI combination.

Next, assume an end user has a simple C code example they wish to build against the installed PETSc version. This can be accomplished as follows by leveraging the environment variables enabled through loading of the provided module file:

```
joeuser $ module load petsc
joeuser $ mpicc -I$PETSC_INC petsc_hello.c \
           -L$PETSC_LIB -lpetsc
```

Figure 6: Example compilation using variables provided by PETSc module file.

Owing to the hierarchical capabilities of *Lmod*, if multiple PETSc permutations were installed (e.g. for different MPI toolchains), the end user would also be able to swap toolchains and the underlying modules system will automatically update the user's environment accordingly to be consistent with the currently loaded MPI family.

2.6 Build Infrastructure

To provide the public package repositories highlighted previously in §2.4, OpenHPC utilizes a set of standalone resources running the Open Build Service (OBS) [8]. OBS is an open-source distribution development platform written in Perl that provides a transparent infrastructure for the development of Linux distributions and is the underlying build system for openSUSE. The public OBS instance for OpenHPC is available at https://build.openhpc.community.

While OpenHPC does not, by itself, provide a complete Linux distribution, it does have in common many of the same packaging requirements and targets a delivery mechanism that adopts Linux sysadmin familiarity. OBS aids in this process by driving simultaneous builds for multiple OS distributions (e.g. CentOS and SLES), multiple target architectures (e.g. x86_64 and aarch64), and by performing dependency analysis among components, triggering downstream builds as necessary based on upstream changes. Each build is carried out in a chroot or KVM environment for repeatability, and OBS manages publication of the resulting builds into package repositories compatible with `yum` and `zypper`. Both binary and source RPMs are made available as part of this process.

The primary inputs for OBS are the instructions necessary to build a particular package, typically housed in an RPM .spec file. These .spec files are version controlled in the community GitHub repository and are templated in a way to have a single input drive multiple compiler/MPI family combinations. To illustrate this approach, the code in Figure 7 highlights a small portion from the .spec file used to build METIS [22], a popular domain decomposition library. The primary item of note is the the use of a `compiler_family` macro which defaults to the gnu compiler family if not specified otherwise. This variable is then used to decide on underlying build and installation requirements chosen to match the desired runtime.

```
%{!?compiler_family: %define compiler_family gnu}

# Compiler dependencies
BuildRequires: lmod%{PROJ_DELIM}
%if %{compiler_family} == gnu
BuildRequires: gnu-compilers%{PROJ_DELIM}
Requires:      gnu-compilers%{PROJ_DELIM}
%endif
%if %{compiler_family} == intel
BuildRequires: gcc-c++ intel-compilers-devel%{PROJ_DELIM}
Requires:      gcc-c++ intel-compilers-devel%{PROJ_DELIM}
%endif
```

Figure 7: Snippet from METIS .spec file highlighting compiler hierarchy template used during the build process.

To link the underlying source and build infrastructure together, OpenHPC's public OBS instance is integrated with the associated GitHub repository. A benefit of this integra-

tion is that whenever commits are made on key git branches, OBS automatically triggers corresponding package rebuilds. OBS also analyzes inter-package dependencies and downstream packages are rebuilt as well with updated packages published after all builds are completed.

For builds that require MPI linkage, a companion .spec template is used which adds an `mpi_family` variable that defaults to the OpenMPI [17] stack unless specified otherwise. To maintain the concept of having a single maintainer commit drive multiple builds, our OBS configuration leverages the ability to *link* related software packages together. To illustrate this process, the text in Figure 8 contains the underlying OBS package configuration syntax for the PETSc toolkit built against MVAPICH2. The top line indicates that the MVAPICH2-based build is simply a link to the parent (default) package configuration that is OpenMPI based. What follows after that are stanzas that tell OBS to apply patches to the resulting .spec file prior to doing a build. In this case, the patches are trivial and simply redefine the `compiler_family` and `mpi_family` variables at the top of the .spec file. This linkage provides a convenient mechanism to extend the hierarchical runtime family approach to the underlying build system.

```
# cat petsc-gnu-mvapich2/_link
<link project='OpenHPC:1.1' package='petsc-gnu-openmpi'>
<patches>
<topadd>%define compiler_family gnu</topadd>
<topadd>%define mpi_family mvapich2</topadd>
</patches>
</link>
```

Figure 8: Underlying OBS package config highlighting linkage between builds using different runtime hierarchies.

2.7 Integration Testing

To facilitate validation of the OpenHPC distribution as a whole, we have devised a standalone integration test infrastructure. In order to exercise the entire scope of the distribution, we first provision a cluster from bare-metal using installation scripts provided as part of the OpenHPC documentation. Once the cluster is up and running, we launch a suite of tests targeting the functionality of each component. These tests are generally pulled from component source distributions and aim to insure development toolchains are functioning correctly and to ensure jobs perform under the resource manager. The intent is not to replicate a particular component's own validation tests, but rather to ensure all of OpenHPC is functionally integrated. The testing framework is publicly available in the OpenHPC GitHub repository.

A Jenkins continuous integration server [5] manages a set of physical servers in our test infrastructure. Jenkins periodically kickstarts a cluster master node using out-of-the-box base OS repositories, and this master is then customized according to the OpenHPC install guide. The LaTeX source for the install guide contains markup that is used to generate a `bash` script containing each command necessary to provision and configure the cluster and install OpenHPC components. Jenkins executes this script, then launches the component test suite.

The component test suite relies on a custom autotools-based framework. Individual runs of the test suite are customizable using familiar autoconf syntax, and `make check` does what one might expect. The framework also allows us to build and test multiple binaries of a particular component for each permutation of compiler toolchain and MPI runtime if applicable. We utilize the Bash Automated Testing System (BATS) [1] framework to run tests on the cluster and report results back to Jenkins. An example test driver shell script for the PETSc toolkit is highlighted in Figure 9. Recall that this package requires MPI linkage and the script highlights the fact that multiple tests are performed for each supported compiler and MPI toolchain.

```
#!/bin/bash

status=0
cd libs/petsc || exit 1
export BATS_JUNIT_CLASS=PETSc

# bootstrap the local autotools project
./bootstrap || exit 1

for compiler in $COMPILER_FAMILIES ; do
    for mpi in $MPI_FAMILIES ; do

    echo "---------------------------------------"
    echo "Libraries: PETSc tests: $compiler-$mpi"
    echo "---------------------------------------"

    module purge          || exit 1
    module load prun      || exit 1
    module load $compiler || exit 1
    module load $mpi      || exit 1
    module load petsc     || exit 1

    ./configure           || exit 1
    make clean            || exit 1
    make -k check         || status=1

    save_logs_mpi_family tests $compiler $mpi

    make distclean
    done
done

exit ${status}
```

Figure 9: Example test driver script for PETSc.

As the test suite has grown over time to accommodate a growing set of integrated components, the current test harness has both *short* and *long* configuration options. The short mode enables only a subset of tests in order to keep the total runtime to approximately 10 minutes or less for more frequent execution in our CI environment. For the most recent OpenHPC release, the long mode with all relevant tests enabled requires approximate 90 minutes to complete approximately 1,900 individually logged tests.

3. CONCLUSIONS & FUTURE WORK

This paper has presented an overview of OpenHPC, a collaborative Linux Foundation project with organizational participation from academia, research labs, and industry. The building-block nature of the OpenHPC repository was highlighted along with some basic packaging conventions and an overview of the underlying build and test infrastructure.

Future work by the OpenHPC Technical Steering Com-

mittee (TSC) is focused on formalizing and publishing a component selection process by which the community can request inclusion of additional software. Currently, OpenHPC provides simple configuration recipes for HPC clusters, but future efforts will focus on providing automation for more advanced configuration and tuning to address scalability, power management, and high availability concerns. We also hope to expand community cooperation between complementary efforts by developing package dependency conventions with EasyBuild and Spack.

4. ACKNOWLEDGMENTS

We would like to thank the Linux Foundation and associated members of the OpenHPC collaborative project for supporting this community effort. We are particularly grateful to the additional members of the Technical Steering Committee including Pavan Balaji, Todd Gamblin, Craig Gardner, Balazs Gerofi, Jennifer Green, Douglas Jacobsen, Chulho Kim, Thomas Moschny, Craig Stewart, and Scott Suchyta. We are also grateful to donations from Intel, Cavium, and Dell who have provided hardware to help support integration testing efforts, and the Texas Advanced Computing Center for hosting OpenHPC infrastructure.

5. REFERENCES

[1] Bash Automated Testing System (BATS). https://github.com/sstephenson/bats.

[2] Extra Packages for Enterprise Linux (EPEL). https://fedoraproject.org/wiki/EPEL.

[3] Hierarchical Data Format (HDF5). https://www.hdfgroup.org/HDF5/.

[4] https://github.com/tacc/lmod. Lmod: An Environment Module System based on Lua.

[5] Jenkins Automation Server for Continuous Integration. https://jenkins.io/.

[6] Linux Foundation. https://www.linuxfoundation.org.

[7] Lustre Parallel File System. http://lustre.org.

[8] Open Build Service. http://openbuildservice.org.

[9] OpenHPC Governance. https://github.com/openhpc/ohpc/wiki/Governance-Overview.

[10] OSCAR. http://svn.oscar.openclustergroup.org/.

[11] PETSc: Portable, Extensible Toolkit for Scientific Computation. https://www.mcs.anl.gov/petsc/.

[12] Rocks. http://www.rocksclusters.org/.

[13] Warewulf Project. http://warewulf.lbl.gov/trac.

[14] G. Bruno, M. J. Katz, F. D. Sacerdoti, and P. M. Papadopoulos. Rolls: modifying a standard system installer to support user-customizable cluster frontend appliances. In *2004 IEEE International Conference on Cluster Computing (CLUSTER 2004), September 20-23 2004, San Diego, California, USA*, pages 421–430, 2004.

[15] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation".

[16] J. L. Furlani and P. W. Osel. Abstract yourself with modules. In *Proceedings of the Tenth Large Installation Systems Administration Conference (LISA '96)*, pages 193–204, 1996.

[17] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[18] T. Gamblin, M. P. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, and S. Futral. The spack package manager: bringing order to HPC software chaos. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015, Austin, TX, USA, November 15-20, 2015*, pages 40:1–40:12, 2015.

[19] K. Hoste, J. Timmerman, A. Georges, and S. D. Weirdt. EasyBuild: Building Software with Ease. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis, Salt Lake City, UT, USA, November 10-16, 2012*, pages 572–582, 2012.

[20] W. Huang, G. Santhanaraman, H. Jin, Q. Gao, and D. K. Panda. Design of high performance MVAPICH2: MPI2 over infiniband. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2006), 16-19 May 2006, Singapore*, pages 43–48, 2006.

[21] M. A. Jette, A. B. Yoo, and M. Grondona. SLURM: Simple Linux Utility for Resource Management. In *Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*, pages 44–60. Springer-Verlag, 2002.

[22] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, Dec. 1998.

[23] R. McLay, K. W. Schulz, W. L. Barth, and T. Minyard. Best Practices for the Deployment and Management of Production HPC clusters. In *State of the Practice Reports*, SC '11, pages 9:1–9:11. ACM, Nov. 2011.

[24] P. M. Papadopoulos. Extending clusters to amazon EC2 using the rocks toolkit. *IJHPCA*, 25(3):317–327, 2011.

[25] P. M. Papadopoulos, M. J. Katz, and G. Bruno. NPACI rocks: tools and techniques for easily deploying manageable linux clusters. *Concurrency and Computation: Practice and Experience*, 15(7-8):707–725, 2003.

[26] S. L. Scott. OSCAR and the beowulf arms race for the "cluster standard". In *2001 IEEE International Conference on Cluster Computing (CLUSTER 2001), 8-11 October 2001, Newport Beach, CA, USA*, page 137, 2001.

[27] Y. Tanaka, N. Yamamoto, R. Takano, A. Ota, P. M. Papadopoulos, N. Williams, C. Zheng, W. Huang, Y. Pan, C. Wu, H. Yu, J. H. S. Shiao, K. Ichikawa, T. Tada, S. Date, and S. Shimojo. Building secure and transparent inter-cloud infrastructure for scientific applications. In *Transition of HPC Towards Exascale Computing - Selected Papers from the High Performance Computing Workshop, Cetraro, Italy, June 25-29, 2012.*, pages 35–52, 2012.