# Revamping the OSCAR Database: A Flexible Approach to Cluster Configuration Data Management

DongInn Kim, Jeffrey M. Squyres, Andrew Lumsdaine
Open Systems Laboratory, Pervasive Technology Labs,
Indiana University, Bloomington, IN 47404
Email: dikim,jsquyres,lums@osl.iu.edu

## Abstract

*The OSCAR [17] cluster installation toolkit started life as the result of an ad-hoc working group attempting to bundle a set of "best practices" for building clusters into a single software solution. Although mainly developed as "skunk works" projects in each of the participating institutions, the OSCAR toolkit has gained a large following, boasting hundreds of thousands of downloads and active mailing lists. The original OSCAR toolkit was aimed at one particular type of High Performance Computing (HPC) cluster. Since then, several sub-projects targeting other types of HPC clusters have been spun off the main working group's efforts. Each of these projects share a core set of OSCAR code, including the OSCAR Database and its access API, "ODA" (OSCAR Database API). The ODA abstraction layer – consisting of a database schema and corresponding API – hides a commodity back-end database (e.g., MySQL).*

*As OSCAR and its derivatives are targeted at new, innovative environments (to include non-HPC environments), the configuration management issues that OSCAR must handle have grown exponentially. As such, its current database schema for holding the cluster configuration is starting to show its age – it is simply unable to represent the complex, ever-growing set of data required to accurately describe the clusters that it manages. ODA's API is overly complex, requiring a steep learning curve for OSCAR developers. This paper proposes a simpler, highly flexible design and implementation that will allow ODA to not only handle all the data that ODA currently manages, but also allow expansion into new types of clusters, enable storage and retrieval of configuration information in a variety of different formats, and encourage data re-use between the main OSCAR project and its derivative packages.*

## 1. Introduction

The installation of a high performance computing (HPC) cluster requires a lot of work, including: procuring funding to purchase the equipment and relevant software licenses, physically obtaining and installing all the hardware (including all the infrastructure required to power and cool the machinery), and deciding policies on which to operate the cluster. Once this initial setup is complete, the process of installing software – the heart of the cluster – can begin.

Unfortunately, this task is no easier than any of the others. Although a wealth of high-grade commercial and free software exists for HPC clusters, installing and maintaining it in a consistent manner is a difficult undertaking even for experienced system administrators. It is nearly an impossible task for those new to Unix environments and/or system administration.

For these purposes, the Open Source Cluster Application Resources (OSCAR) [17] working group was formed in 2000 to create a software package for the installation of HPC clusters: the OSCAR toolkit. The OSCAR toolkit guides a user through the basic software installation, configuration, and maintenance of an HPC cluster. This software enables complex, large-scale cluster installations to be performed by novices (an important, yet frequently overlooked, demographic in today's HPC community).

Over the past few years, the OSCAR toolkit has continued to evolve. It now supports multiple different Linux distributions (and continues to be updated to support recent versions), allows the user to perform more automated complex maintenance tasks, and so on. OSCAR itself has become a framework for a cluster installer, and can be customized to perform specialized tasks.

As such, the main OSCAR working group has spun off several sub-project groups: HA-OSCAR

[14] (High Availability, for mission-critical clusters), Thin-OSCAR [2] (Diskless, for clusters with a minimum number of moving parts), SSS-OSCAR [16] (Scalable System Software, a US Department of Energy research project investigating terrascale computational resources), and SSI-OSCAR [1] (Single System Image, for clusters that behave like a single large symmetric multiprocessor machine). Each of these projects expands the OSCAR framework for its own specific, dedicated purposes.

OSCAR v5.0, scheduled to be released at SC2005, will be the most comprehensive OSCAR yet, including a much more modular framework and enhanced features for the maintenance of OSCAR clusters. Toolsets enabling the partitioning of a cluster, selective software installs across the cluster, and fault recovery are among the features that will be supported.

At the heart of OSCAR is the cluster configuration data – how each node is configured, what software is installed where, etc. All of this information is stored in the OSCAR database and accessed via the OSCAR Database API (ODA). ODA is an abstraction between the main OSCAR framework and a commodity back-end database. The current version of ODA uses a sophisticated database schema and has been the useful tool for simplifying access to the back-end database. It allows queries both on the shell command line and through a Perl interface. Command line queries utilize a simple language (even simpler than SQL) and need not have knowledge of where the database resides, nor how to login to it. A variety of "ODA shortcuts" provide trivial access to commonly used information, sometimes combining complex SQL queries into a deceivingly short shell command line.

The current version of ODA is not without its share of problems, however. This paper describes the current status of ODA in OSCAR v4.0 – to include all of its benefits and shortcomings – and then proposes a a simpler, highly flexible design and implementation that will allow ODA to not only handle all the data that it currently manages, but also allow expansion into new types of clusters, enable storage and retrieval of configuration information in a variety of different formats, and encourage data re-use between the main OSCAR project and its derivative packages. The new version of ODA is planed to be fully implemented and supported in OSCAR v5.0.

The rest of the paper is organized as follows: Section 2 provides an overview of the main components in the OSCAR toolkit and the current status of ODA. Section 3 proposes a new design, database schema, and implementation of ODA for OSCAR v5.0. Finally, Section 4 gives some conclusions and future work.

## 2. Current Implementation

OSCAR v4.0 is divided up into several distinct subsystems: the overall framework, OSCAR packages, the testing framework, and the database.

### 2.1. The OSCAR Framework

The overall framework is the "core" of OSCAR; it has a base library of internal functionality, provides the graphics panels that comprise the OSCAR installation wizard, and contains the logic and sanity checks that cause the installer to flow from one step to the next. In many cases, the logic and work performed at each step are not in the OSCAR framework itself, but are rather invoked via "hooks" in the framework to external functionality.

These external functionality hooks are precisely why OSCAR has been used as a vehicle for the sub-projects mentioned in Section 1 – OSCAR itself can be customized for specific kinds of cluster deployments simply by providing add-on scripts and functionality. While this framework is far from perfect, it does provide a large degree of flexibility for derivative projects without requiring changes to the OSCAR code base.

### 2.2. Packages

OSCAR packages are self-contained bundles of software that are installed across a cluster. In short, an OSCAR package is a plugin to the OSCAR framework that tells the installation process how to install and configure it across the cluster. In its simplest form, an OSCAR package is a single binary package, such as an RPM [4]. [1] More complex configurations are also possible, to include XML to describe the functionality of the package, scripts for pre- and post-configuration, and built-in testing capabilities. The following are several common components of OSCAR packages.

*A* `config.xml` *file:* The `config.xml` file is used to provide textual names and descriptions of the functionality provided in the OSCAR package, explicit version numbering and dependency information, and a list of the binary packages to install (and uninstall). This file is read into the OSCAR database.

*The* `RPMS` *folder:* This folder contains the binary packages that are to be installed.[2] Its contents may be preprocessed to select only the RPMs that are to be in-

1 OSCAR v4.0 currently only supports RPM-based packages, but work is ongoing to expand this to include support for other binary packages such as Debian packages, OS X packages, etc.

2 This directory may be renamed and its contents revamped once more binary package types are formally supported by OSCAR.

stalled on a user's particular configuration, which may depend on the particular distribution and version of Linux being used.

*The* `SRPMS` *folder:* If present, this folder contains "source" binary packages, such as source RPMs. Since, by definition, OSCAR is open source software, the OSCAR working group encourage packages (even third-party packages) to include the SRPMs folder and include the source code for their software.

*The* `doc` *folder:* If present, this folder can contain any of the following files:

- `install.tex`: A LaTeX file containing documentation regarding the installation of the package that will automatically be included in the OSCAR installation guide.
- `user.tex`: A LaTeX file containing user-level documentation about the package that will automatically be included in the OSCAR user guide.
- `license.tex`: A LaTeX file containing the package's copyright and / or license information that will automatically be included in the OSCAR installation guide.

*The* `scripts` *folder:* This folder contains "hook" scripts that are invoked by the OSCAR framework at various phases during the cluster installation. This allows the package to customize itself during key points during the definition and configuration of the overall system.

*The* `testing` *folder:* If present, this folder can include scripts (and any necessary supporting material) that will be executed by the OSCAR testing framework.

The main OSCAR distribution includes several well-known clustering software tools that are bundled up as OSCAR packages: LAM/MPI [5, 19], SIS [6], C3 [7], Ganglia [15], HDF5 [18], MPICH [9, 10], PVM [8], Torque [3], and Maui [12, 13].

## 2.3. Tester

OSCAR contains an integrated testing harness for testing the functionality of each OSCAR package at the end of the installation process. As noted above, a package can provide a `testing` folder that contains scripts and supporting material that will be invoked by the OSCAR testing harness. The harness will evaluate the success and failure of each test and display prettyprint messages summarizing the results.

## 2.4. OSCAR Database and ODA

The OSCAR database is the heart of the configuration management scheme. Originally designed and implemented by researchers at the National Center for Supercomputer Applications (NCSA) at the University of Illinois, the OSCAR Database API (ODA) is an abstraction layer between the main OSCAR framework and the back-end database. The original rationale for this layer included supporting a variety of back-end databases and providing simple, OSCAR-specific access methods to read and write to the database (i.e., obviate the need for any direct SQL access anywhere in OSCAR). The architecture of ODA is shown in Fig. 1.

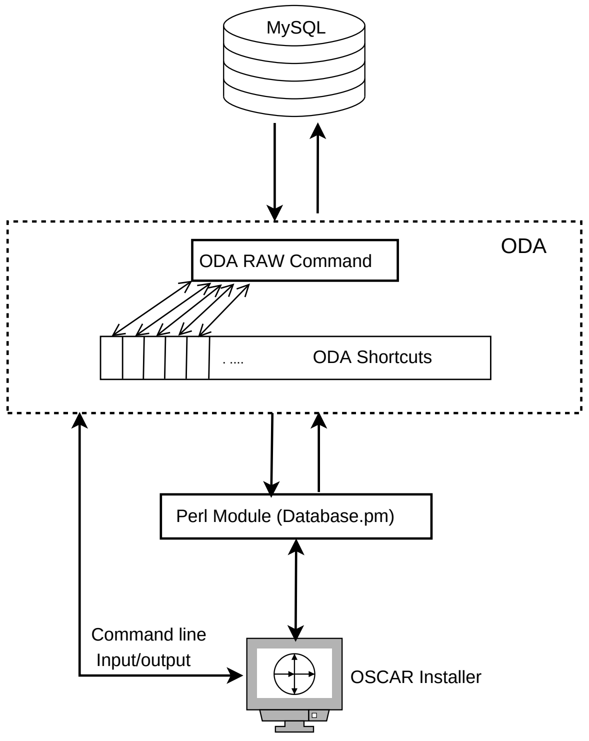

Figure 1: The database architecture used in OSCAR v4.0.

ODA also included functionality to read in each package's `config.xml` file and store it in the database. All of the data is then able to be queried via ODA. Similarly, all cluster-related information is read in via ODA (node names, hardware makeup, disk partitions, etc.) and is therefore available via simple ODA queries to the rest of the OSCAR framework.

ODA provides both command line and Perl-based access methods. The following is an example of a simple ODA command line query:

```
shell# oda packages_that_provide mpi
mpich
lam
```

The query "`packages_that_provide`" is called an ODA "shortcut" – it is an abbreviation for a more complex, back-end SQL query. In this case, it returns a list of all OSCAR packages that provide MPI functionality. Shortcuts are resolved into a series of ODA raw commands (i.e., SQL) which are then sent to the back-end database. The results are returned via normal SQL semantics, and then converted to the form desired by the caller (i.e., either Perl lists or written to the standard output). Most of the OSCAR framework uses the Perl ODA module, but a few places use the command line interface.

The ODA in OSCAR v4.0 has done a good job so far and has played a key role in the current database architecture. But it has several problems:

1. The ODA code is far too complicated; it is composed of thousands of lines of complex Perl code. It is extremely difficult for OSCAR developers to add new functionality or modify ODA – the learning curve for its internals is simply far too steep. This actively deters new development in OSCAR.
2. There are too many ODA shortcuts. Over time, hundreds of shortcuts have emerged, many of which are redundant and/or are no longer needed.
3. The database schema is not well organized. The back-end schema contains 30 tables, 11 of which are unused after the cluster is setup. Additionally, some fields of one table are re-named in another table with exactly same data. Still worse, the relationship between tables is neither documented nor clear. Finally, some tables were just created for the temporary solutions and are never used again.

## 3. Proposed Database Architecture

The new database architecture is based around solving the three main problems with the current ODA. The new architecture will differ from the old in the following key aspects:

- The entire scheme will be fundamentally simpler, enabling a smaller, less complicated code base in which to implement ODA. The command line interface will be removed (it is extremely complex code, creating great difficulties for expansion and continued maintenance); all interactions will be through a Perl module interface.[3]
- This directly allows the OSCAR derived projects to consolidate their code and directly interact with the database without fear of it changing over the release chronology of OSCAR. If necessary, supplemental Perl modules can be used to provide specialized database functionality in each project.
- The number of shortcuts will be dramatically reduced. All redundant shortcuts will be eliminated, and only those shortcuts which are actively used by the OSCAR framework will be maintained.
- The database schema will be revamped. Essentially, the current schema will be scrapped and a new one will take its place.
- By formally utilizing the Perl DBI, ODA will be able to support multiple different back-end databases such as MySQL[20] and PostgreSQL [11].
- Be able to support more than one OSCAR cluster in a given OSCAR database instance. This is entirely new functionality.

Reflecting these points, the proposed OSCAR database architecture structure is shown in Fig. 2.

### 3.1. ODA Perl Module

The ODA module will hide all aspects of connectivity from the caller; details such as which database technology, the database name, and the username and password are all below the abstraction layer. Indeed, since ODA may be invoked from a cluster node, ODA will take care of opening network connections to the back end database, sending commands, receiving results, etc.

The ODA Perl module will be split into four main functional units: the old ODA raw commands, a subset of the old shortcuts, a `select()` function (for reading from the database), and a `update()` function (for writing to the database). Most of the raw commands and shortcuts will be implemented in terms of `select()` and `query()`. The following listing shows a potential implementation of the `list_of_tables()` shortcut – it maps directly to the SHOW TABLES SQL command:

```
package OSCAR::ODA

sub list_of_tables{
    my $ref_result = shift;
    my $sql = "SHOW TABLES";
```

3 Although it is not anticipated, if the command line interface is still required, it can be re-implemented as a simplified, thin wrapper around the Perl module interface.

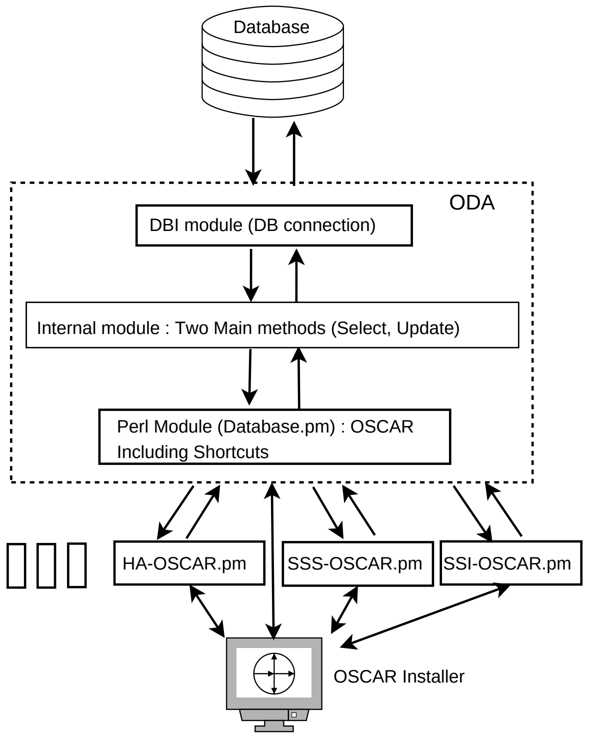
Database
ODA
DBI module (DB connection)
Internal module : Two Main methods (Select, Update)
Perl Module (Database.pm) : OSCAR Including Shortcuts
HA-OSCAR.pm
SSS-OSCAR.pm
SSI-OSCAR.pm
OSCAR Installer

Figure 2: Proposed OSCAR database architecture.

```
    my $error;
    my $local_result;
    my $status = ODA::query(\$sql, \$local_result,
        \$error);
    # ...translate $local_result into common
    # form and store in $ref_results...
    $status;
}
```

`select()` and `query()` will be straightforward – they mainly translate between the back-end database (via the Perl DBI) and the standardized data structures that the OSCAR framework expects. For example, the `select()` function will translate the input into valid SQL and feed it back to the database. It will then obtain the results from the database and put them into common data structures used in OSCAR (e.g., lists of hashes with well-defined keys and layouts). The `update()` command will be similar – implementing SQL CREATE, UPDATE, and DELETE – but only needs to return indications of success or failure of the update.

## 3.2. `config.xml` Parsing

The functionality for parsing `config.xml` files will be separate, but will utilize the ODA module for writing the information back to the database. This functionality is fairly complex and it may be difficult to simplify it. However, it was originally designed to be flexible enough to be able to handle any kind of data in `config.xml` files. By creating limiting rules (and assumptions) about what data packages may place in their `config.xml` files (perhaps allowing any data in `config.xml`, but only guaranteeing that specific subsets will be stored in the database), at least some of the complexity of the parsing code can be eliminated.

## 3.3. Back-End Database Schema

The new database tables will be created based on the following entity-relation (ER) diagram. There will be no unused tables or fields; each field will have a defined purpose. The relation between tables will be clear and have a complete integration. The ER diagram for the new database tables is shown in Fig. 3.

The following is a selected list of tables from the database schema and a summary of some of the more important fields in each:

- Clusters: the name of the cluster, the distribution used on the head node, and the architecture of the head node.
- Cluster_Nodes: cross reference between the Clusters and Nodes tables; create a mapping of nodes in a cluster.
- Networks: for a given cluster, the IP broadcast and gateway addresses.
- Nodes: the name and hostname(s) of a given node.
- Node_Groups: a cross reference between the Nodes and Groups tables; allow for arbitrary groupings of nodes.
- Groups: a name and description of a node group.
- Packages_Groups: a cross reference between the Packages and Groups tables; allow for arbitrary groupings of OSCAR packages.
- Packages: data about OSCAR packages; selected information from each package's `config.xml` file.
- Node_Packages: a cross reference between the Nodes and Packages tables; a listing of which OSCAR packages are installed on each node.

The Clusters table contains all the information about an OSCAR cluster (recall that a goal for the new

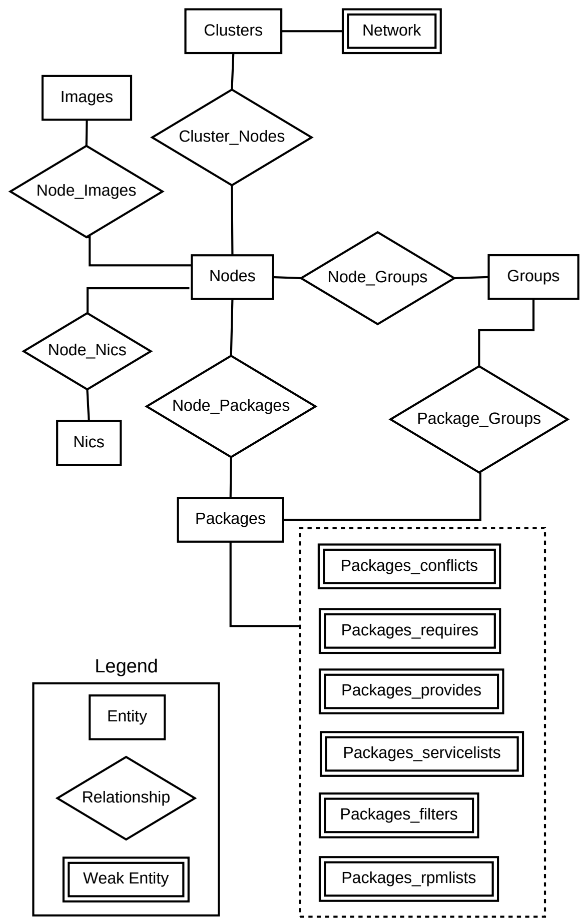

Figure 3: The entity-relation diagram for the proposed OSCAR database schema. The entities and relations shown here essentially map to tables in the back-end database; this diagram mainly shows the main data element tables and how they are related to each other. "Weak" entities are entities that do not have a primary key.

database schema is to enable support for multiple OSCAR clusters in a single database) such as the cluster's name, the server's distribution, and server's architecture. The Clusters table is related to both the Networks table (which may simply be fields of the Clusters table) and the Nodes table. The Nodes table describes each node, such as the node's name and its hardware information (CPU type, amount of RAM, disk partition information, etc.). The Nodes table is associated with both the Groups and Packages tables.

The Groups table is a "meta-grouping" table for providing lists of both nodes and OSCAR packages, and is described below. The Packages table describes all information about OSCAR packages, such as its name, version, verbose description, name and e-mail address of its maintainer, etc.

## 3.4. ER Analysis

The analysis of the ER shows how the proposed database schema is designed. The central entities of the ER diagram are Nodes, Packages, and Groups. These three entities and their relations describe the heart of the proposed database schema.

Groups essentially provides categorizations of nodes. Typical node groups include the OSCAR server (a group simply containing the OSCAR head node of a given cluster), OSCAR clients (all the client nodes in a cluster), and images (one or more disk images that, for management purposes, are treated like real nodes).

The Packages entity is related to the Nodes entity in order to show the status of OSCAR packages that are related to a given node. The Node_Packages relation displays the state of the package installation on the node, such as which packages are installed, will be installed, or failed to be installed on a node. For instance, a Node named "`oscarnode1`" may have entries showing that the Maui and Torque packages are successfully installed, that the LAM/MPI package will be installed in the future, and that the Ganglia package failed to install.

As mentioned above, the relation between Groups and both Nodes and Packages is designed to describe what packages belong to what groups and which nodes are associated with the certain group. This meta grouping allows the configuration of one or more nodes. For example, installing/uninstalling packages to certain nodes can be controlled precisely by the relation between Nodes, Groups, and Packages.

The Packages entity contains many attributes. For example, basic information from `config.xml` fills more than ten fields. To avoid complexity, Packages does not attempt to contain every piece of information about an OSCAR package itself (indeed, not all packages have the same kinds of attributes, which could lead to a sparsely-filled table). The Packages entity is therefore designed to only have the only basic information about each package, but maintain relationships with a collection of package-related entities. Since there is no corelation among package-related entities, they can be in the one collection and the relation between Packages entity and this group can explain every possible situation between packages (or even between OSCAR packages and binary packages such as RPMs). If another package-related entity is found, it can also be in the same collection and have the same relationship with Packages entity.

## 4. Conclusions

The new database architecture and high-level database schema proposed in this paper will renovate the current database schema by precisely defining entities and relations in a new, smaller set of tables. The amount of code in the ODA abstraction layer will also shrink dramatically, allowing for easier development and maintenance. Also, the number of ODA shortcuts will be reduced to a small, useful core set that will be fully documented and easily accessible to OSCAR developers. Perhaps most importantly, the formalization of the database allows third parties and derivative OSCAR projects to interact with the OSCAR database in a stable, predictable manner. This minimizes (and potentially eliminates) the patches required for the spin-off projects whenever a new version of the OSCAR core code base is released.

Note that this paper reflects a design proposal; its exact details are both likely to change over time and as is implemented. Future work includes integration of the new database architecture with other milestones in OSCAR's long-term roadmap. This OSCAR database re-structuring is a first step towards integration with a fully distributed node synchronization and configuration management system. This tool, called the Node Event Synchronization Tool (NEST), is a project that started with the original ODA developers at NCSA and has continued development by the larger OSCAR development team. It is expected to be released as part of OSCAR v5.0.

## 5. Acknowledgments

This work was supported by a grant from the Lily Endowment.

## References

[1] SSI-OSCAR. `http://ssi-oscar.irisa.fr/`.

[2] Thin-OSCAR. `http://thin-oscar.sourceforge.net/`.

[3] Torque portable batch system. `http//www.supercluster.org/projects/torque/`.

[4] E. C. Bailey. *Maximum RPM.* Red Hat Software, Inc., North Carolina, Feb 1997.

[5] G. Burns, R. Daoud, and J. Vaigl. LAM: An Open Cluster Environment for MPI. In *Proceedings of Supercomputing Symposium*, pages 379–386, 1994.

[6] S. Dague. System installation suite: Massive installation for linux. In *Proceedings, Ottowa Linux Symposium*, 2002.

[7] R. Flanery, A. Geist, B. Luthke, and S. L. Scott. Cluster command and control (C3) tool suite. In *Proceedings, 3rd Austrian-Hungarian Workshop on Distributed and Parallel Systems (DAPSYS)*, Balatonfured, Lake Balaton, Hungary, September 2000.

[8] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine – A Users' Guide and Tutorial for Networked Parallel Computing.* Scientific and Engineering Computation Series. MIT Press, November 1994.

[9] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, Sept. 1996.

[10] W. D. Gropp and E. Lusk. *User's Guide for* `mpich`*, a Portable Implementation of MPI.* Mathematics and Computer Science Division, Argonne National Laboratory, 1996. ANL-96/6.

[11] T. P. G. D. Group. *PostgreSQL 8.0.1 Documentation*, 1996 2005.

[12] D. Jackson, B. Bode, D. M. Halstead, R. Kendall, and Z. Lei. The portable batch scheduler and the Maui scheduler on linux clusters. In *Proceedings, 4th Annual Linux Showcase and Conference*, October 2000.

[13] D. Jackson, Q. Snell, and M. Clement. Core algorithms of the Maui scheduler. In *Proceedings, 7th Workshop on Job Scheduling Strategies for Parallel Processing*, 2001.

[14] C. Leangsuksun, T. Liu, S. L. Scott, R. Libby, and I. Haddad. HA-OSCAR Release 1.0 Beta: Unleashing HA-Beowulf. In *Proceeding of* $2^{nd}$ *Annual OSCAR Symposium (OSCAR 2004)*, Winnipeg, Manitoba Canada, May 16-19 2004.

[15] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing*, 30(7), July 2004.

[16] J. Mugler, T. Naughton, and S. L. Scott. The Integration of Scalable Systems Software with the OSCAR Clustering Toolkit. In *Proceeding of* $2^{nd}$ *Annual OSCAR Symposium (OSCAR 2004)*, Winnipeg, Manitoba Canada, May 16-19 2004.

[17] T. Naughton, S. L. Scott, B. Barret, J. M. Squyres, A. Lumsdaine, and Y.-C. Fant. The penguin in the pail – OSCAR cluster installation tool. In *The 6t̂h World MultiConference on Systemic, Cybernetics and Informatics (SCI 2002)*, Invited Session of SCI'02, Commodity, High Performance Cluster Computing Technologies and Applications, Orlando, FL, USA, 2002.

[18] E. Pourmal. *HDF5 Abstract Data Model*, Sep 1999.

[19] J. M. Squyres and A. Lumsdaine. A Component Architecture for LAM/MPI. In *Proceedings, 10th European PVM/MPI Users' Group Meeting*, number 2840 in Lecture Notes in Computer Science, pages 379–387, Venice, Italy, September / October 2003. Springer-Verlag.

[20] L. Welling and L. Thomson. *MySQL Tutorial*, Nov. 2003.