The Introduction of the OSCAR Database API (ODA)

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 [14] cluster installation toolkit was created by the Open Cluster Group (OCG) for one particular type of High Performance Computing (HPC) cluster. OSCAR is currently one of the widely used cluster installation toolkits; it boasts hundreds of thousands of downloads and active mailing lists. OSCAR has expanded its area with several sub-projects targeting other types of HPC clusters. 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 [15]). Because OSCAR and its sub-projects are targeted at new, innovative environments (including non-HPC environments), there are significant issues with managing various configurations of each project. For example, as we previously showed [8], "previous versions of ODA were unable to represent the complex, ever-growing set of data required to accurately describe the clusters that it manages. Further, its API was extremely complex, requiring a steep learning curve for OSCAR developers". Therefore, we have designed and implemented a new database schema to deal with these issues.

This new version of ODA has not only resolved the above problems but also, as proposed in our previous paper, enabled storage and retrieval of various configuration information, and encouraged data re-use between the main OSCAR project and its derivative projects. In addition, the new version of ODA has sped up the OSCAR installation process. This document presents a simpler, highly flexible design and implementation of ODA slated to be included in OSCAR v5.0. It also suggests a blueprint for maintaining the database modules of ODA in a systematic, organized way.

1 Introduction

ODA is an OSCAR Database API to make it easy for users to use the OSCAR database. When using ODA, there is no need to know how to connect the database or determine what its schema look like. ODA deployed on the OSCAR Subversion trunk uses Perl modules to connect, update, and query the database. Also, all the database subroutines for the end users are defined in a single Perl module, which is a collection of database subroutines and does the intermediate work between back-end database and OSCAR installation. As the previous paper showed, the old ODA has three problems. First, it can not fully support new features of OSCAR. Second, its implementation is overly complicated and takes a long time to learn and modify. Finally, its schema was also not well organized: 11 tables among 30 OSCAR tables are not used at all after OSCAR installation creates all the tables and numerous redundant shortcuts make developers confused in deciding what shortcuts should be used.

OSCAR, therefore, needed more flexibility and a better organized database schema. The new version of ODA was developed not only to resolve the above chronic problems but also to establish a bridgehead for supporting OSCAR. It allows the OSCAR installer to add new features of the OSCAR sub-projects without modifying the whole OSCAR framework. The new version of ODA also makes it easy to participate in improving the database modules. The OSCAR sub-projects include HA-OSCAR [10] (High Availability, for mission-critical clusters), SSS-OSCAR [13] (Scalable System Software, a US Department of Energy research project investigating terrascale computational resources), SSI-OSCAR [2] (Single System Image, for clusters that behave like a single large symmetric multiprocessor machine) and Debian-OSCAR [1] (OSCAR porting Debian distro). The new version of ODA is deployed on the OSCAR Subversion trunk and is planned to be fully implemented and supported in OSCAR v5.0.

The rest of the document is organized as follows: Section 2 presents a design, database schema, implementation and higher performance of ODA for OSCAR v5.0. Sec-

tion 3 provides detailed examples to show how ODA works and contains a brief explanation of subroutines that are used in the OSCAR main frame. This section also presents some ideas to make the new version of ODA manage all the database modules systematically. Finally, Section 4 gives some conclusions and future work.

2 Database Architecture

As the previous paper showed, the new database architecture was required to resolve the three problems with the old ODA:

1. The ODA code was too complicated; it consisted of thousands of lines of complex Perl code. It was extremely difficult for OSCAR developers to maintain ODA or to add new functionality. Eventually, this detered new development in OSCAR.

2. There were too many ODA shortcuts. The ODA shortcuts were composed of more than two basic ODA commands in order to make it easy for developers to implement database queries. Over time, hundreds of shortcuts emerged and most of them were redundant and/or of no use.

3. The database schema was not well organized. There were 30 database tables, 11 of which were not touched at all after the cluster was setup. Additionally, some fields of the different tables contained duplicate data. Even worse, the relationships between tables were neither documented nor clear.

Based on solving the problems of the old ODA, we developed the new database architecture with the following new features:

- The scheme is fundamentally simple, enabling easy maintenance. The old ODA used the complicated Perl codes for the command line interface, which is not really necessary for development in OSCAR. The new version of ODA removes the command line interface and has provided a Perl module interface for all interactions between the OSCAR installer and ODA.

- The Perl module for the new version of ODA allows the OSCAR derived projects to add their new functionalities to the main ODA modules so that they can directly interact with database without fear of breaking any main frame of OSCAR. If necessary, each of these projects can make their own ODA module for the specialized database functionality.

The database schema was revamped with the above new features. The database tables and fields are newly designed to make the schema simple and well organized by cleaning up all the old unnecessary tables and fields.

Considering these aspects, the new version of the OSCAR database architecture is shown in Figure. 1.

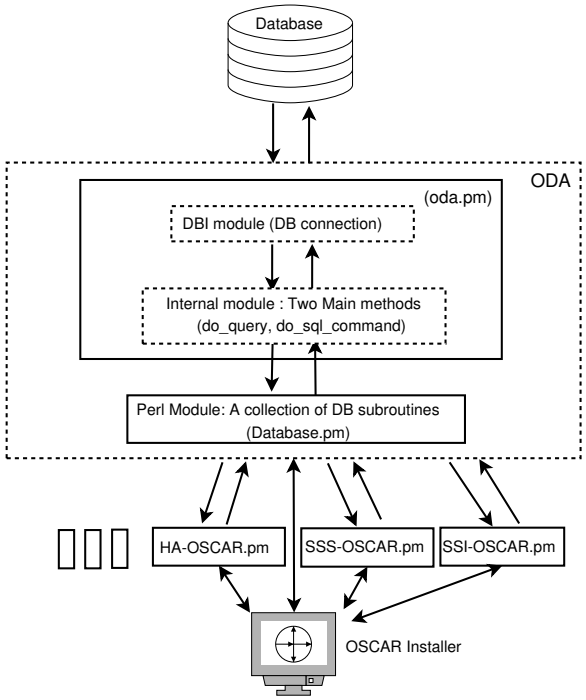

Figure 1. OSCAR database architecture.

2.1 ODA Perl Module

The new version of ODA is composed of two Perl modules (`oda.pm` and `Database.pm`) and a script to convert XML information to the SQL strings. As the OSCAR database architecture shows in Figure. 1, `oda.pm` hides all aspects of connectivity from the caller; the database name, username, password, and all the other database connection technology are behind the abstraction layer because there is no need for users to access it directly to do their database queries. `oda.pm` takes care of the database connection and provides the two main SQL queries: "do_query" and "do_sql_command". `Database.pm` is a collection of all the subroutines that the OSCAR installer or a user uses to implement the database SQL queries. The SQL queries generated by a subroutine in `Database.pm` are handed over to "do_query" for querying or to "do_sql_command" for updating, inserting, and creating. They are implemented at `oda.pm`. `oda.pm` then returns the output to `Database.pm`. All the OSCAR packages have their particular `config.xml` file not only for defining packages

but also for specifying their information to populate into the "Packages" table. ODA also has its own `config.xml` file and it includes additional XML information for creating all the OSCAR tables. The Perl XML Simple parser, "XML::Simple" [12] takes the `config.xml` file of each of the packages and converts it to the data format, Hash, `Database.pm` can understand. The current XML parser for ODA is much simpler than the old one because it limits rules about what data packages can place in the `config.xml` files.

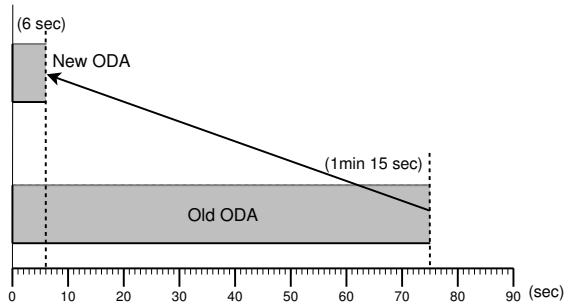

Figure 2. ODA Performance Test

When all the complicated Perl codes are taken out of old ODA, the new version of ODA shows better performance in the database operations. A simple test has been done to prove how fast the new version of ODA implements the Perl script, `package_config_xmls_to_database` , which creates the OSCAR database and all the tables and also populates the information from `config.xml` of each package to the Packages table. The test was implemented by running the script on the old ODA and the new version of ODA in the same computer with Intel(R) Pentium 4 CPU 1.50 GHz and 1GB memory. As Figure. 2 shows, the new version of ODA has radically improved the performance.

2.2 Database Schema

The OSCAR database tables are created based on the entity-relation (ER) diagram shown in Figure. 3.

The central entities of the ER diagram are Nodes, Packages, and Groups. These three entities and their relations describe the heart of the database schema.

Groups essentially provides categorizations of nodes and packages. For grouping of nodes, Groups typically includes 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). On the other hand, the Group_Packages relation contains the package groups to represent what packages are installed in a group. A default package group is setup and

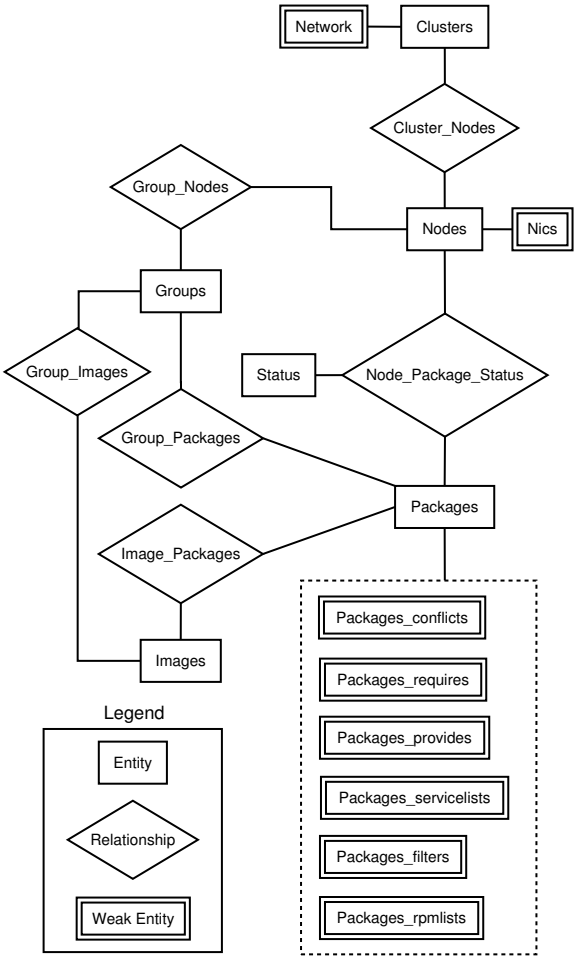

Figure 3. The entity-relation diagram for the OSCAR database schema. The entities and relations of this diagram shows OSCAR database tables and their relationship. "Weak" entities are entities that do not have a primary key.

the packages that belong to this group are installed in all the nodes.

The Packages entity is related to the Nodes and Groups entity. The Node_Package_Status relation which comes from the relation of Nodes, Packages, and Status entity displays the status of OSCAR packages that are related to a given node, such as which packages are installed, will be installed, or should be uninstalled on a node. The Nodes entity contains not only all the basic node information but also the keys to connect with the Groups, Packages, and Clusters entities.

For example, a Node named "`oscarnode1`" may belong to a certain group, "`OSCAR server`" as well as the default cluster, "`OSCAR`". The packages installed on "`oscarnode1`" are determined by the relation Group_Packages which has the entries displaying that "`Default`" group contains these packages. The "`oscarnode1`" node may also have entries showing that the Ganglia [11] and PVM [5] packages are successfully installed, that the Torque [3] package will be installed in the future, and that the PBS [7] package should be uninstalled.

As described above, the relation among Groups, Nodes, and Packages is designed to describe the installation status of packages on a given node, what packages belong to what groups and which nodes are associated with a 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 among Nodes, Groups, and Packages. Previously left over, the Status entity and Node_Package_Status relation which connects Nodes, Packages, and Status entity are added to the current ER. After discussion of OSCAR Packages Manager (OPM), the Status entity and Node_Package_Status are needed to keep track of the status of installation of each of the OSCAR packages on the specific node.

3 Implementation

OSCAR installation is implemented by performing numerous configurations to setup the order of installation and to install the OSCAR packages.

3.1 What can ODA do?

These configurations are not only for setting up the OSCAR framework, including a base library of internal functionality, but also for the installation of OSCAR packages which are well known as an HPC tool. The OSCAR framework is the main process of the OSCAR installation, which makes the installer proceed from one step to the next. During installation of OSCAR, the configurations need to be stored, queried, and updated to manage the full OSCAR installation. ODA offers a place to store the configurations and has been designed to query and update OSCAR data, including the configurations resulting from installation. In particular, ODA does the following:

- Connects to the database with Perl DBI [4], which is the primary interface for database programming in Perl
- Parses `config.xml` to convert into the SQL commands
- Executes SQL query (select, create, update, delete, and so on)
- Stores configurations
- Stores installation status
- Simplifies database queries

ODA is implemented with the two Perl modules: `oda.pm` and `Database.pm`.

The highest level of ODA hierarchy is `oda.pm`, which implements the direct database connection and main database queries for the OSCAR database. The database connection is performed by Perl DBI. DBI requires data source, username, password, and option arguments for the DBI module, "connect". So, the database handler is created by calling the "connect" module of DBI. For example,

```
$dbh = DBI->connect( $data_source, $username,
                                   $password, %attr );
```

The main database queries consist of the two Perl subroutines on `oda.pm`: "do_query" and "do_sql_command". These subroutines can only be executed with the database connection of DBI described above. The "do_query" takes any SQL string with several other arguments, executes the SQL commands with given arguments, and then returns the query results. It is designed to return all the possible format values by generating general data format. Any statement calling the subroutine "do_query" takes return values with a consistent data format and can set up the same routine of codes to handle the return values. The "do_sql_command" takes any SQL string and executes it with the DBI database connection. The command does not have any return values to transfer but returns the flag to determine whether the SQL string has been successfully executed or not. The "do_query" routine does not call "do_sql_command" to execute its SQL command. Instead, it executes the command by its own internal codes since "do_sql_command" can not treat the return values that "do_query" needs to transfer.

`Database.pm`, located at the next level of the ODA hierarchy, is an abstract Perl module to handle directly all the database operations under the control of `oda.pm`. Many

Perl subroutines defined at `Database.pm` are exported so that non-database codes of OSCAR can use its subroutines as if they are defined by importing `Database.pm` (e.g., "use `Database.pm`").

For example, "get_packages" in `Database.pm` is a subroutine that the OSCAR installer can call to get the list of OSCAR packages. The Perl module, `Database.pm` including the subroutine, "get_packages", looks like this:

```
package OSCAR::Database
use oda

@export = (get_packages,
           .... );

sub get_packages{
    my $ref_result = shift;
    my $sql = ''SELECT package FROM Packages'';
    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;
}
```

The subroutines have been created to query or update data from or to the specific database tables. For the query of data, the name of the subroutines starts with "get" and if there is an argument to set the "WHERE" clause, "with_" and the argument name are added as a suffix. On the other hand, the subroutines to update data into the tables are named starting with "set". Like the query subroutines, the updated subroutines can also add "with_" and the argument for the "WHERE" clause as a suffix. There is a conventional rule to set the arguments to the subroutines. The required arguments should be set as the first and next argument and the optional arguments are supposed to be set from the last, in the reverse order, so that the optional arguments can be disregarded even though they are not given to the subroutines. The usual optional arguments in `Database.pm` are "$options_ref" and "$error_strings_ref". The subroutines most frequently used in `Database.pm` are "do_select" for querying, "do_update" for updating, and "do_insert" for inserting. The subroutine "do_select", internally calls the "do_query" subroutine of `oda.pm` and transfers the return values of "do_query" to the caller or another subroutine. Like "do_select", "do_update" and "do_insert" use the "do_sql_command" subroutine of `oda.pm` to run the database works at the back-end. Then, they return the signal to determine if the SQL query is successfully implemented to the caller or another subroutine.

3.2 Maintenance of ODA

More functionality needs to be added to `Database.pm`: the OSCAR installer requires more complex database queries and needs to store more detailed configuration parameters. ODA facilitates good management of subroutines to make them simpler, to enable better search of subroutines, and to allow easier maintenance. Some ideas to manage the database modules (including `oda.pm` and `Database.pm`) in a systematic and organized way follow:

- Naming rule
  As the current database modules use, having "get" prefix for query and "set" prefix for update makes it easy to differentiate the two database functionalities: query and update. The suffix with "with_" and the arguments for the "WHERE" clause enables the expansion of the basic getter or setter subroutines. It would be more convenient for developers if subroutines were placed consistently.

- Convert `config.xml` of ODA to list of SQL commands
  The `config.xml` file of the oda package was created to define the oda package in general and to provide information to create the OSCAR database tables as well. ODA has converted `config.xml` of the oda package to the list of SQL commands via the xml parser in the `packages_config_xml_to_database` script. The xml parser will not be needed any more if the OSCAR installer uses the text file, oda.sql, to contain the list of SQL commands for the creation of the OSCAR database tables. In addition, the `packages_config_xml_to_database` script will be simpler and be able to avoid the process time to convert config.xml to the SQL queries.

- Specialized module
  As OSCAR expands its area with various OSCAR subprojects, the modules of ODA will have to take care of more functionality and configurations of the OSCAR main frame, including additional features of the OSCAR sub-projects. The ODA modules will be more complicated and confusing if all the additional funcationality for the sub-projects is added to the main ODA modules. So, the new features for the sub-projects would need to be in a different module file in order to differentiate from the main ODA modules. On the other hand, having another directory for all the new module files for the sub-projects would make the ODA module more organized and keep it simpler. For instance, OSCAR Package Manager (OPM) is one of

the new features of OSCAR and uses the immense database queries to manage the installation of OSCAR package more flexibly according to the user's needs. If OPM is ready to go to the current OSCAR framework, the database module for only OPM would be named as `OPM_Database.pm` and be located under the new directory, Database, for only OSCAR sub-projects or additional ODA modules.

- Documentation
  Documentation is as essential to a software project as any screen or module. A piece of software can be brilliantly written, but if another developer can not understand and modify it, it may be re-written from scratch because re-writing takes less time than understanding undocumented code. In these repects, a piece of software is only as good as its documentation. If the ODA documentation is maintained by the Perlpod [9], which is a simple-to-use markup language used for writing documentation for Perl, Perl programs, and Perl modules, it can be translated to the various formats like plain text, HTML, man pages, and more.

4 Conclusions

The database architecture and database schema described in this paper are fully implemented at the current ODA not only to resolve the three main problems shown previously but also to allow for easier development and maintenance. The current database architecture is built with only two Perl modules: `oda.pm` and `Database.pm`. They are taking care of all the database queries. The database schema displayed at the entity-relation (ER) diagram, (Fig. 3) is designed to make the OSCAR database tables as concise as possible. In addition, the current ODA showed dramatic improvement in the query performance due to the renovated database architecture and schema.

Formalization of the database is required for two reasons: 1) to allow any derivative OSCAR projects and third parties to use the current ODA without fear of modifying the main OSCAR database codes and 2) to interact with their own specialized database modules. As OSCAR expands its area with OSCAR sub-projects, we plan to allow for expansion into new types of clusters, guarantee data integrity by importing the new database table type, InnoDB[1], and enable the use of various database programs (e.g., postgreSQL [6]). This year we will setup OSCAR Package Manager (OPM), which is a project to build an intelligent tool to manage the installation of OSCAR packages. Data integrity among the various data transactions is essential to OPM. This integrity requires ODA to utilize the InnoDB table type. OPM is expected to be included in the version of OSCAR targeting SC06.

[1] InnoDB provides MySQL with a transaction-safe (ACID compliant) storage engine that has commit, rollback, and crash recovery capabilities.

5 Acknowledgments

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

References

[1] OSCAR on Debian. `http://oscarondebian.gforge.inria.fr/`.

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

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

[4] A. D. . T. Bunce. *Programming the Perl DBI*. Feb 2000.

[5] 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.

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

[7] 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.

[8] D. Kim, J. M. Squyres, and A. Lumsdaine. Revamping the OSCAR database: A flexible approach to cluster configuration data management. In I. Kotsireas and D. Stacey, editors, *19th International Symposium on High Performance Computing Systems and Applications*, pages 326–332, Guelph, Ontario, Canada, May 2005. IEEE Computer Society.

[9] S. M. B. Larry Wall. *PerlPOD - perldoc.perl.org*.

[10] 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.

[11] 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.

[12] G. McLean. *XML::Simple Easy API to maintain XML*, 2004.

[13] 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.

[14] 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 6îh World Multi-Conference on Systemic, Cybernetics and Informatics (SCI 2002)*, Invited Session of SCI'02, Commodity, High Performance Cluster Computing Technologies and Applications, Orlando, FL, USA, 2002.

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