Los Alamos National Labs HPC Consistent User Environment Design Report

Chris Poupre   Ben Jones   Tim Porterfield

June 19, 2008

Abstract

The Los Alamos National Laboratory (LANL) has several clusters that receive software updates on a staggered schedule resulting in nonuniform software between clusters and more importantly among nodes on a cluster. The Software Support Team (Ptools) approached Mines with a field session project with the goal to gather data about the software on the clusters analyze the data and report the results on a webpage. This report describes the cooperation between Ptools and the Mines team. The Mines team has created a collection of scripts to process data, along with a database design and webpage to meet the needs of the Ptools team. The system has been heavily tested at Colorado School of Mines and been deployed at LANL with positive initial results.

Contents

List of Figures

1 Introduction to the Project

The Los Alamos National Laboratory (LANL) has several computing clusters with different environments in three different security partitions. Each cluster is made up of a number of nodes, which ideally should be updated simultaneously, but updates do not always work as expected. In addition, due to the the various architectures and security requirements of clusters, updates between clusters are staggered. For example, software must run properly on a turquoise (cooperative) cluster before it can be installed on a red (classified) cluster. These conditions can lead to inconsistencies between clusters, and even within clusters. LANL Software Support Team (Ptools) member Georgia Pedicini approached Mines with a field session project to design a system to collect data about the user environments on these clusters and display it on a web page. This will allow the Ptools team to easily find problems and users to quickly see which clusters have the software they need.

2 Requirements and Specifications

The original requirements for this project are as follows:

2.1 Functional Requirements

1. The system must be able to gather module versions from each cluster and indicate errors when they are detected
2. The website must display installed versions of all software on each cluster
3. The website must be able to display several views of the data including high-level, summary views and low-level, detailed views
4. The website must indicate an error when one or more nodes on a cluster do not have identical modules installed

2.2 Nonfunctional Requirements

1. The product must be a website that can reside in both the Secure and Protected networks at LANL
2. The website must be user friendly

2.3 Extensions

Once these requirements were met, the client requested that our project be able to:

1. Enable Ptools users to leave comments about cluster or software status that normal users can read, but not edit
2. Generate a "software history" display that shows which versions of a piece of software were installed on a cluster throughout its history
3. Send email notifications to the Ptools team when the data collector script encounters an error or finds an inconsistency
4. Display the status of clusters as of a user-inputted date

3 Solution Approach

Our team chose to use a set of scripts to collect data, a database to store the data, and a dynamic webpage to display the data.

3.1 Data Collection and Updating

This piece will be a set of scripts that use available system utilities to query each node, collect all data to a central location, and input it into a database. We examined several scripts for gathering information from the nodes, including a set of tools from Argonne National Labs [1] before deciding to modify a previously developed tool from LANL, get_mods, and write several new scripts. An overview of the process is shown in Fig. 1.

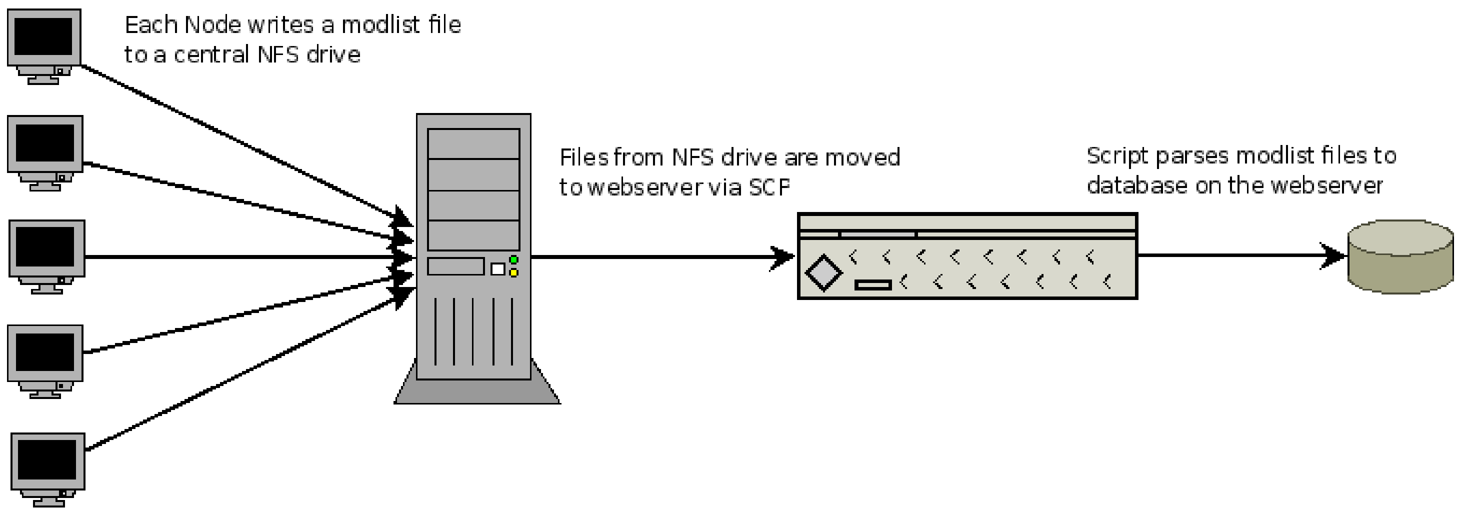

Each Node writes a modlist file to a central NFS drive
Files from NFS drive are moved to webserver via SCP
Script parses modlist files to database on the webserver

Figure 1: Overview of data collector

First, the get_mods script is run on each node by cron job, and generates a file for each node that lists all modules installed on that node and stores them on a shared NFS drive. These files are then copied to the project's web and database server, bear.lanl.gov. Next, the moduleDBUpdate script parses these files and updates the database, sending email notifications when there are errors or nodes are missing data.

There is also a script, nodeStatusUpdater.php that a Ptools member can run to change node names, which cluster a node belongs to, which partition a cluster resides in as well as several other properties of nodes and clusters. This tool can be used when data is either entered incorrectly with moduleDBUpdate.php or when there are changes to cluster configurations. It can also mark nodes as "different but OK," indicating that they are being used as test machines for "friendly users," or mark nodes as "inactive."

3.2 Data Storage

To store module data from each node, we chose to use a relational database. This met the requirements that our storage system be persistent and easily accessible. To protect against update anomalies and reduce data duplication, it was designed as third normal form. In order to keep a history of software installation and removal, the database keeps entries for each change to each node. A diagram of our design is shown in Fig. 2.

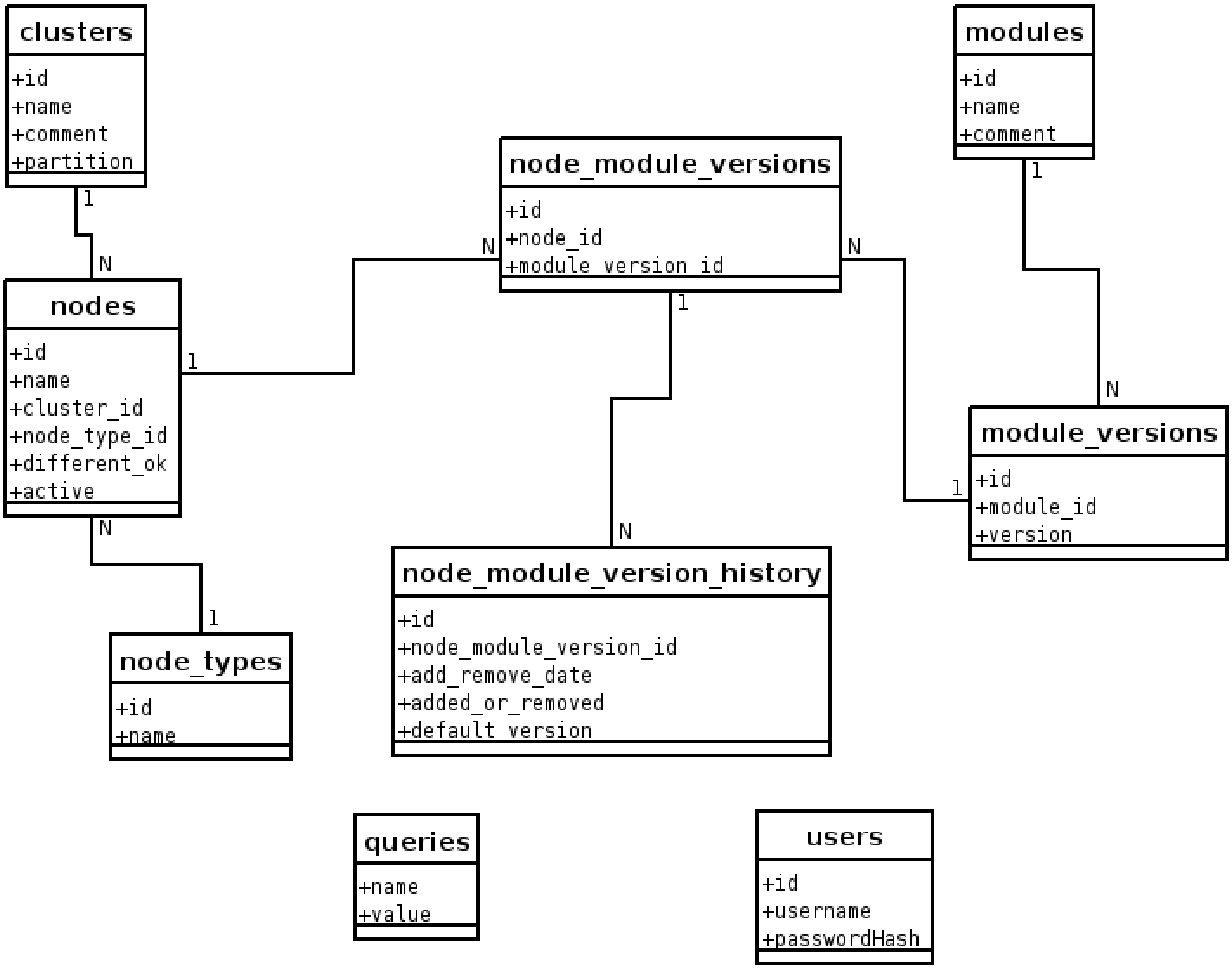


Figure 2: Database Organization

3.3 Data Display

For this piece, we chose to use PHP to query the database and format the results as a web page. We considered using the Eclipse BIRT package for this, but found that a PHP implementation would be easier to implement and more flexible [2]. The BIRT package's strengths are that it can generate charts, and output reports in a variety of formats. For this project though, we do not need charts, and the output format would be best as a web page. While BIRT can generate a webpage, we felt that a BIRT generated page would have the same cross-browser problems as one that we wrote from scratch. Also due to an additional learning curve and server-side requirements, we decided not to use BIRT.

In our implementation, PHP scripts query the database, analyze the data, and output the data in a table. They also generate styles that ensure it is displayed correctly on Unix, Mac, and Windows machines in Firefox, Safari, Internet Explorer, Opera, Konqueror and other browsers.

4 Implementation Details

4.1 Data Collection and Updating

Previous efforts at LANL regarding this project had produced several tools for analyzing module information from nodes in the clusters. We looked at these tools, and used the get_mods script that was provided after making some minor modifications. This tool formats the output of the `module avail` command which lists all installed modules on the node it is run on. get_mods generates a .modlist file that is used by modulesDBUpdate.php. Since the NFS drive where these can be stored cannot be directly accessed from our web and database server, they are copied via scp from the shared space to our server, bear.lanl.gov where they are processed further.

Although we did not implement it due to time constraints, we developed a possible method to copy modlists to bear.lanl.gov and run moduleDBUpdate.php on them. This is the proposed process:

1. A cron job on each node is set to run get_mods at a set time at night, 01:00 for this example, these are all collected on a shared NFS drive

2. Next a cron job on bear.lanl.gov runs some time later, 02:00 in this example:

    - Runs `scp` to copy all modlists to a temporary directory on bear
    - Runs `moduleDBUpdate.php` using the copied modlists
    - Runs `rm` over ssh to delete the modlists from the NFS drive
    - Runs `rm` locally to remove the modlists from bear

Once the files have been moved to bear by the method described above, or some other method, modulesDBUpdate.php reads in all of the modlist files in a directory, parses them and inserts changes in configuration to the database. It is a commandline php script that can be run either interactively or via cron job. Each time a new node is added, the script needs to find out which cluster it belongs to, as well as what type of node it is. Since these cannot be reliably found from the modlist filenames, the script must be run interactively for the user to input that data. If it is not run interactively, the script will skip the new node and send an email notification to the ptools team requesting an interactive run.

This script also checks for consistency within each cluster and can send email notifications when if finds an error. Since nodes can be marked as testing nodes, the script doesn't report errors when for inconsistencies with "different but OK" nodes.

The script, nodeModuleUpdater.php allows Ptools team members to update node and cluster status information. The user can deactivate a node, indicating that it is no longer in use. Also, the user can mark nodes as "different but OK," change node or cluster names, change cluster partition and change node machine types. The script is a menu based console application that will be run on bear.lanl.gov.

4.2 Data Storage

The database we chose to use for project is MySQL. It was chosen because it is free, commonly available, and interfaces well with PHP, which we used to write our scripts. The table design, shown in Fig 2, was chosen so that our most frequent queries, finding the current status of a module version on a particular node, would run fastest while keeping the database in third normal form. Third normal form was implemented to prevent repeated data and slower comparisons on long strings as opposed to quick comparisons on correlated id fields.

There is one field however, that prevents our database from being truly third normal form, the cluster "partition" field. This field could contain duplicate data for different entries in the clusters table, but since that table will be small (approximately 10 entries), and updates are predicted to be extremely infrequent, we chose to deviate from third normal form to simplify queries.

4.3 Data Display

To display data, we used several PHP generated webpages that query the database, process the data and display the results. The first page is a selection page, selection.php that allows the user to pick software, clusters and date that they want to see the status of. A screenshot is shown in Fig. 3.

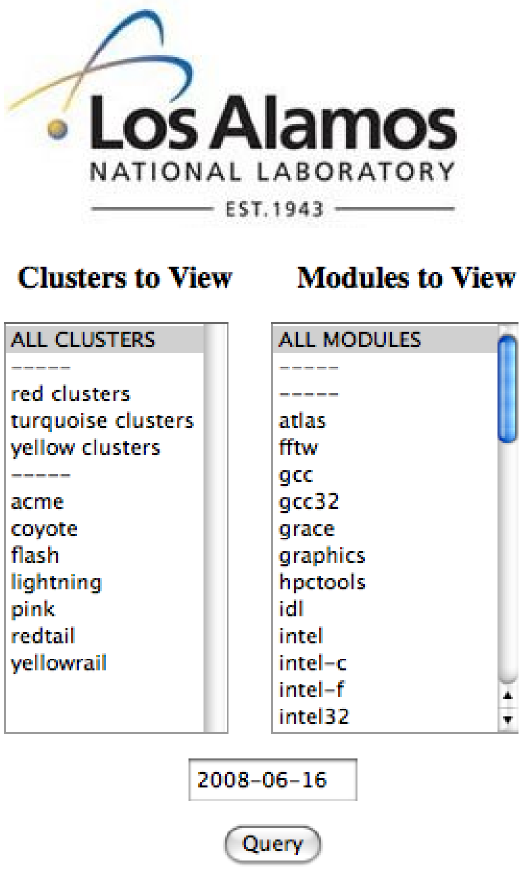
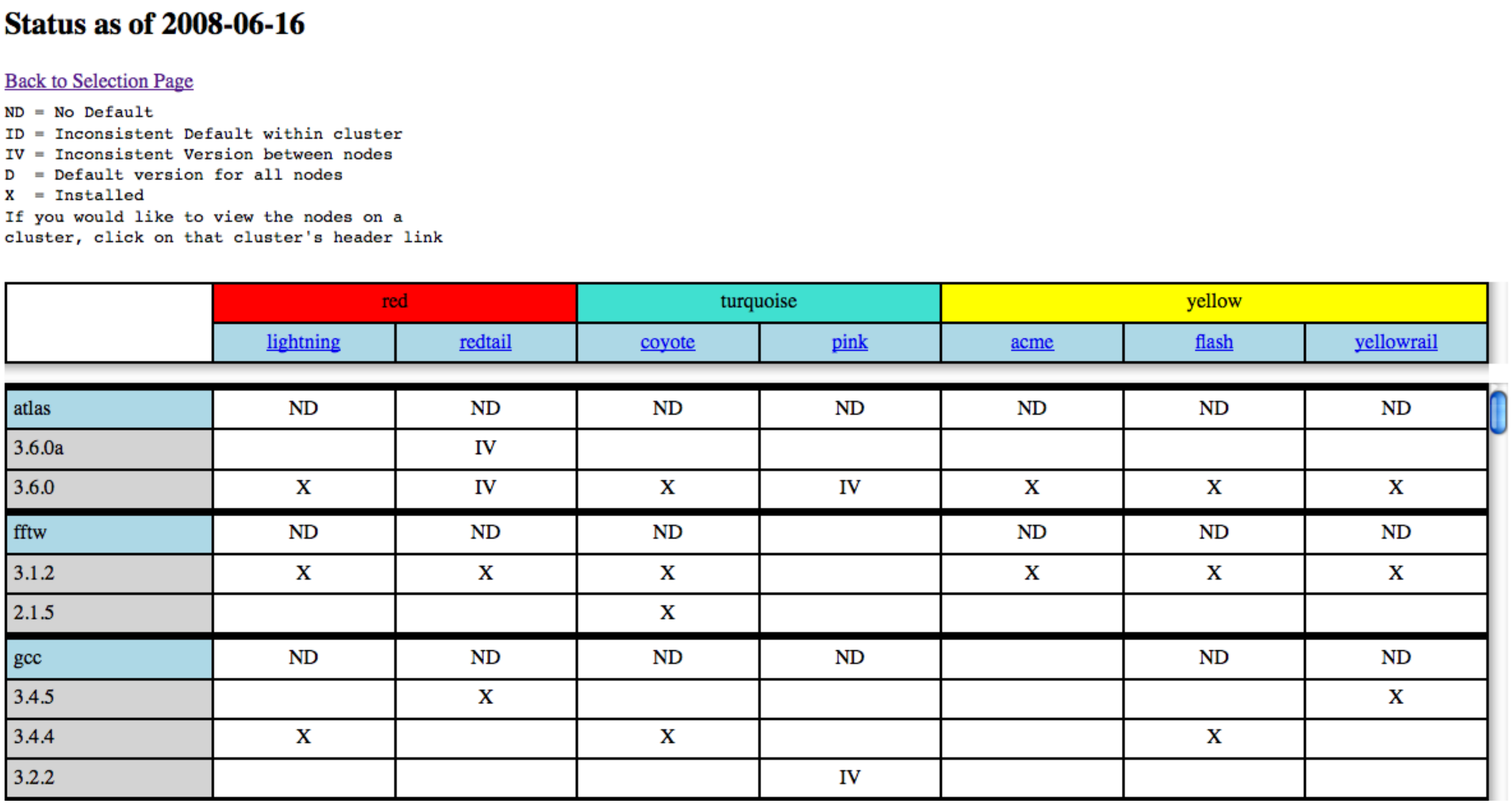


Figure 3: Selection Page

After the user chooses what clusters and software they want to see, they are taken to the clusters.php page which shows an overview of the selected clusters and software. This page indicates inconsistencies within clusters, default inconsistencies, default versions and installed (but not default) versions. It provides links to find out more information about a single cluster as well as a link back to the selection page. The header floats as table scrolls in all major browsers so that the user can see it as they scroll the table. A screenshot is shown in Fig. 4.



Status as of 2008-06-16

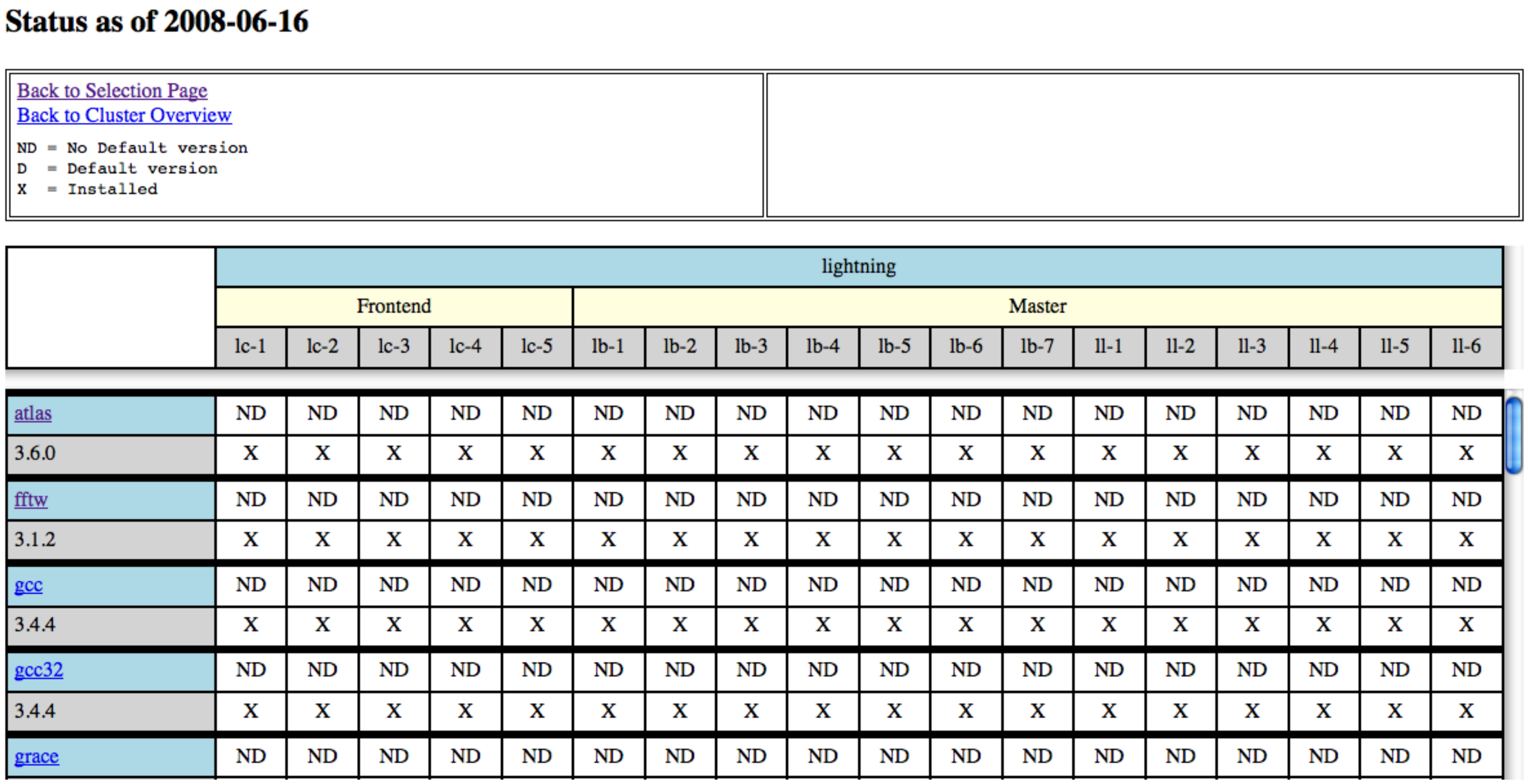
Back to Selection Page

```
ND = No Default
ID = Inconsistent Default within cluster
IV = Inconsistent Version between nodes
D  = Default version for all nodes
X  = Installed
If you would like to view the nodes on a
cluster, click on that cluster's header link
```

	red		turquoise		yellow		
	lightning	redtail	coyote	pink	acme	flash	yellowrail
atlas	ND	ND	ND	ND	ND	ND	ND
3.6.0a		IV					
3.6.0	X	IV	X	IV	X	X	X
fftw	ND	ND	ND		ND	ND	ND
3.1.2	X	X	X		X	X	X
2.1.5			X				
gcc	ND	ND	ND	ND		ND	ND
3.4.5		X					X
3.4.4	X		X			X	
3.2.2				IV			

Figure 4: Clusters.php with all modules and clusters selected

If a user follows one of the links in the header, they will be directed to nodes.php, which will displays information about their selected software on all the nodes in that cluster. This page will be used primarily by the Ptools team, which can use it to quickly find where the inconsistencies reported by clusters.php are, and by "friendly users" who want to try out a testing version of software on a cluster. A screenshot is shown in Fig. 5.



Status as of 2008-06-16

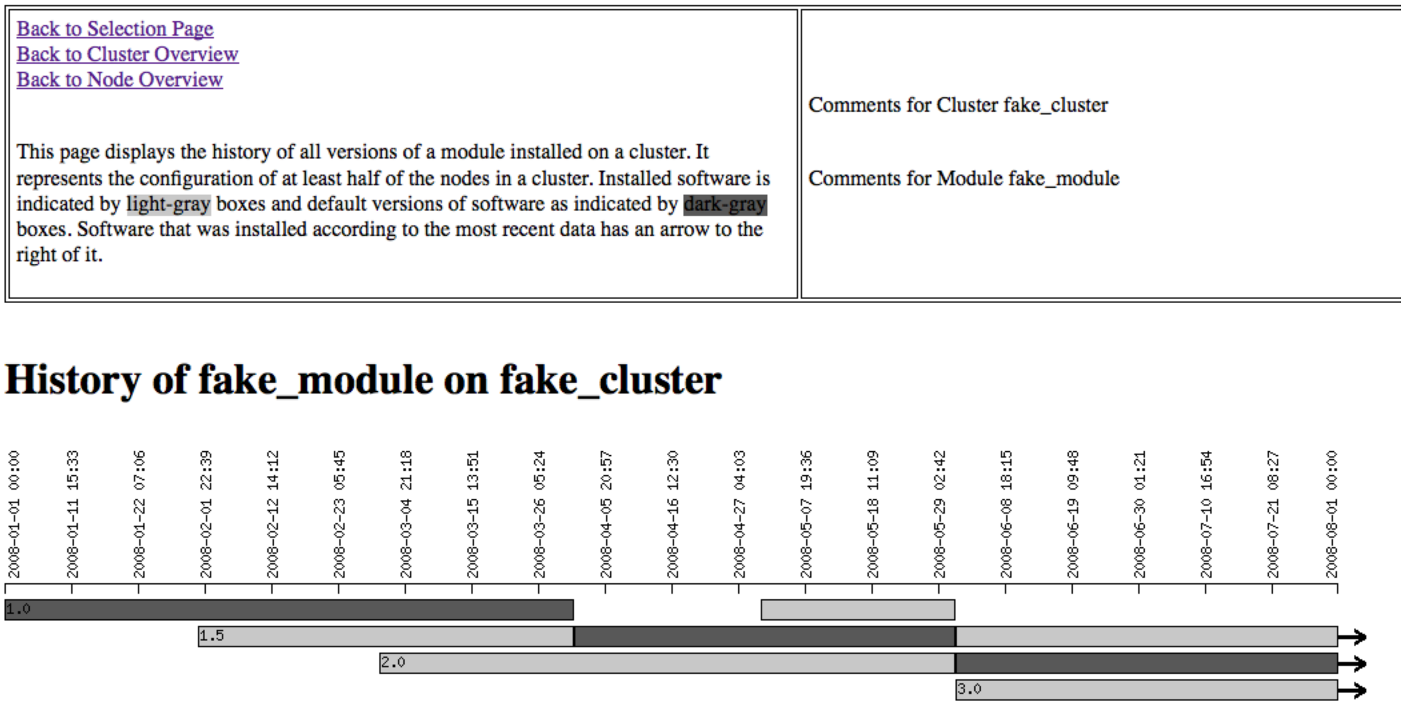
Back to Selection Page
Back to Cluster Overview

```
ND = No Default version
D  = Default version
X  = Installed
```

	lightning																	
	Frontend					Master												
	lc-1	lc-2	lc-3	lc-4	lc-5	lb-1	lb-2	lb-3	lb-4	lb-5	lb-6	lb-7	ll-1	ll-2	ll-3	ll-4	ll-5	ll-6
atlas	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
3.6.0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
fftw	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
3.1.2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
gcc	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
3.4.4	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
gcc32	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND
3.4.4	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
grace	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND	ND

Figure 5: Nodes.php with all software selected on Lightning cluster

If the user follows the links on a specific piece of software they are directed to softwareHistoryDisplay.php. This page displays the history of all versions of the software on the majority of nodes on that cluster as generated by softwareHistory.php. This script uses the PHP GD library to draw a timeline of installation history. It shades default versions and indicates when versions were installed or uninstalled. A screenshot is shown in Fig. 6.


Back to Selection Page
Back to Cluster Overview
Back to Node Overview

This page displays the history of all versions of a module installed on a cluster. It represents the configuration of at least half of the nodes in a cluster. Installed software is indicated by light-gray boxes and default versions of software as indicated by dark-gray boxes. Software that was installed according to the most recent data has an arrow to the right of it.

Comments for Cluster fake_cluster

Comments for Module fake_module

History of fake_module on fake_cluster

Figure 6: Possible history of a module on a cluster generated by softwareHistory.php

5 Project Progression

Throughout the project, our group experienced various difficulties ranging from cross browser incompatibility to slow, unoptimized code. In the beginning of our implementation we decided upon a specific html format for our tables to be displayed. Unfortunately, this table type as well as the css that formatted it were found only fully functional in one browser. Because of this, we tried various implementations until our webpages were fully functional in all browsers.

Another problem we had was with unoptimized nested loops of SQL queries. Our initial implementation was created only to get the general design of the web display implemented, without concern for performance. After testing with large datasets it was found that this implementation was slow and unwieldy. A new design was slowly developed that condensed loops and SQL queries down into a much quicker performance conscious implementation.

To simplify development, we installed our initial implementations on the alamode cluster at CSM, which was easier to access. We used old modlist data from LANL clusters as well as test files to test the functionality of our product. Once our implementations were debugged on alamode, we copied them to the LANL network to ensure that they worked properly there as well. Although we did not have time to test extensively at LANL, initial impressions were that all pieces were working as predicted. Currently, all pieces except for transfer of modlist files from the shared NFS partition to bear.lanl.gov are installed and have been briefly tested.

6 Conclusions and Future Work

One of the issues we experienced related to the network configuration at LANL, which affected our flow of data. Instead of having all the information collect in one spot where it is parsed into the database, we were forced to design an implementation where data was collected on an NFS drive on one network partition and is copied over to a webserver on a different partition. It was on this webserver that the data could be parsed into the database.

In the future there are various possible expansions to this project that due to time constraints, we were not able to implement. One of these expansions is including the libraries on slave nodes in the data collection and display. Due to its time consuming nature, it was decided that this feature would be left out of our design. This was also the situation with keeping track of RPM consistency. Our client shared plans for integrating the login for our product with the security model used throughout LANL.

We feel that the product we produced meets our client's goals, meeting the requirements we agreed on. We were able to modify some existing tools and develop new scripts to process data. Since data is stored in a relational database, so it can be efficiently queried and used by other applications if necessary. Finally, the web page we developed includes all the views the client requested and displays properly on a variety of browsers and operating systems.

References

[1] "The MCS Systems Administration Toolkit," MCS Systems Group Software, Argonne National Laboratory, http://www-new.mcs.anl.gov/systems/software/msys Updated, March, 2007.

[2] "Business Intelligence and Reporting Tools," The Eclipse Foundation, http://www.eclipse.org/birt/phoenix, 2008.