Consistent User Environment

Field Session 2010
Los Alamos National Laboratory

Garrett Motzner
Jared Duncan
Joe Martinez

Abstract:
Los Alamos National Laboratory (LANL) has a Module Display Application which tracks the state of installed software packages across their High Performance Computing (HPC) clusters. We need to increase the efficiency and effectiveness of this application by automating data gathering, adding data interfaces, adding notification capabilities and better reporting methods.

The current software does the needed identification of inconsistencies, however it takes more work than is necessary to run this software and keep the data up to date, and there is no automatic notifications of inconsistencies. We will essentially be making their current software more user-friendly and effective. We will need to replicate the software environment at least partially on a virtual machine (VM) at Mines in order to develop and implement these new features. We will also need to work with our contacts at LANL to make sure our software implementation also works on the HPC clusters on-site at LANL.

Introduction

Los Alamos National Laboratory currently has software that monitors the versions of software packages installed on their HPC clusters. This monitoring software consists of two main parts, data retrieval and data display. The data retrieval is achieved through various shell scripts that require frequent user interaction. The data display is a series of web pages that allow users to select data they wish to view. Our project is to make this software generally more user-friendly and efficient by automating data collection and adding more functionality to the web interface so the client can manage the cluster data better.

We need to write a shell script which will automate data collection on an HPC cluster. This script will run a second script on each back-end node of that cluster. Once all the data is collected the script will then call a third script which will tar all the data together for easier transport. The user only has to interact with our script, and does not need to even know about the other two scripts.

The web page needs administrative functions added. These functions will allow administrators easier access to database data for modification. Currently the only way to modify the database information is by manually interacting with the database tables. We are essentially creating an interface between the user and these tables. Because we are adding these administrative capabilities, we also need to implement some form of security, a roles and permissions listing, which will prevent certain users from gaining access to the administrative capabilities.

A notifications system is desired by the client to send out email notifications of software inconsistencies. The data that is gathered from the HPC clusters is loaded into a database, then when the user wants to view that data an algorithm checks for inconsistencies from the data. We need to find a way to automatically call this algorithm that checks for inconsistencies and augment it so that it can send out notifications.

Currently, the software has graphical history view capabilities; however this functionality contains display errors which prevent the history view from being displayed in some cases. It is also hard to use and understand. We need to fix the display errors and add labels or comments to make this functionality easier to understand.

Requirements:
Functional:
Our project is to increase the functionality and effectiveness of software that already exists and is used at Los Alamos. A previous field session team worked on setting up a basic interface and gathering data so that comparisons could be made between different clusters and nodes and compares the software installed on each. Two shell scripts are used to gather the data from the clusters, one to actually gather the data from the nodes of a cluster and one that tar's up all this data so the user can transport it easier. This data collection process requires frequent user interaction. The user has to log into each individual node of a cluster and run the shell script to gather data. To

make this process more efficient we are to add automation to the data collection so that there is less user interaction. The user should only have to log into one front-end node of a cluster and our script will collect data from all the other nodes. After the data is collected we also need to write a script for server-side data gathering, where the web server can go out and gather all the data collected by the previous scripts and correlate all the new data into a database.

Another job is to implement a history function into the program to graphically display current and previous software versions installed on clusters. The current software has some history graphing functionality already there, however it has errors in it that prevent the history from displaying in some cases. This implementation is also difficult to understand. We need to either come up with some other mechanism to view the history, or repair the current mechanism and make it easier to understand. We decided to just stick with the current history implementation and try to fix the errors and make it easier to understand by adding comments and a better description of what is being displayed.

Notification emails are desired by our client. These should be sent out when the cluster data has been loaded into the database and the server checks for inconsistencies in software in that new data. These notifications should be customizable, so a user can tell the system when he/she wants to be notified or to discontinue notifications of a particular problem.

Our final and largest task is to add role and permission controls to the program. At this point in time, anyone using the system, whether they are an administrator or just a general user, can do all of the same tasks with the software. Currently this is not a problem because there are no administrator portions to the program. However we are adding data interfaces into this software which should only be accessible to administrators. We have no set rules on how to implement this portion of the project provided by our client. We need to make it versatile enough to be able to be easily changed later on, both because of the security requirements our client will need to add into this software and because administrators will need to make necessary changes whenever needed.

Along with these large tasks, there are many different smaller jobs that could be implemented as well. An example of this would be adding a date calendar drop down for the user instead of making users manually input a date for which to view software installed on clusters for that date.

Non-functional:

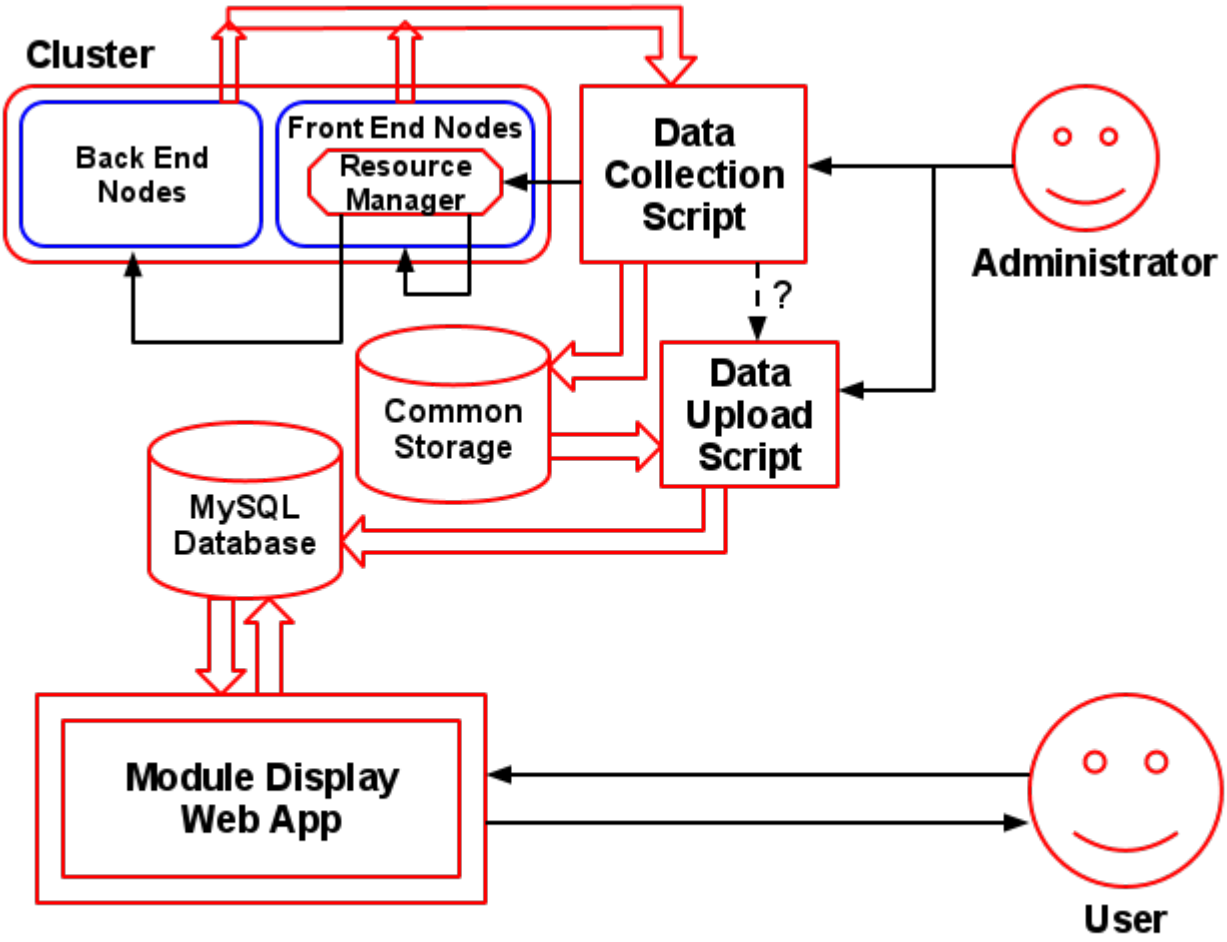
We have to set up a basic environment for this project as well as set up our system in a way that would be used in the same circumstances as if it were part of the system in LANL. That means setting up a system that has information on different clusters and nodes, provided by our client, so that we would be able to use actual data to develop with and not just test data. This all had to be done before we could start even basic programming. We first have to learn the existing software before we know how we can change it to be more usable. Another task we have to do is fill out the information and take the tutorials in order to get our crypto-cards, which are used to generate passwords

for LANL computer systems. We need to fill out paperwork and then give them time to get us access to their system which held us back from completing any of this early on in the project.

Design

The general flow of our program should follow along the general line of: A user will log onto a front end node of an HPC cluster. They will then run the automation script we have developed; this will gather all the data for that cluster in one place. The user will then have to load this data into a database. After that is done users can then log into the website and view the new data as well as the old data still in the database. Below is a diagram depicting the basic flow of our system.

Program Flow Diagram



The largest part of our project is the roles and permissions portion of the project. We implemented this with a CakePHP component called the ACL component. ACL stands for Access Control List. This list contains objects called Access Control Objects (ACO's) and Access Request Objects (ARO's). These objects interact with each other to determine if a user has access to view a certain page or resource.

ACO's in our case will be the pages of the website. ARO's will be the users in the system; each user will belong to a certain role. If a user wants to look at a page, the ACL will check to see if the user has access to view that page. This roles and permissions portion of the project is required because of the additional functionality we are adding to the system, which includes access to data only administrators should be able to access.

Users in this new system will be broken up into four roles per client wishes. A General User role will have access basically only to what can be viewed with the current system, which is the user environment on the HPC clusters. This includes a table of all the software installed in and across clusters, with inconsistencies marked, and history views for software that is and has been installed on a certain cluster.

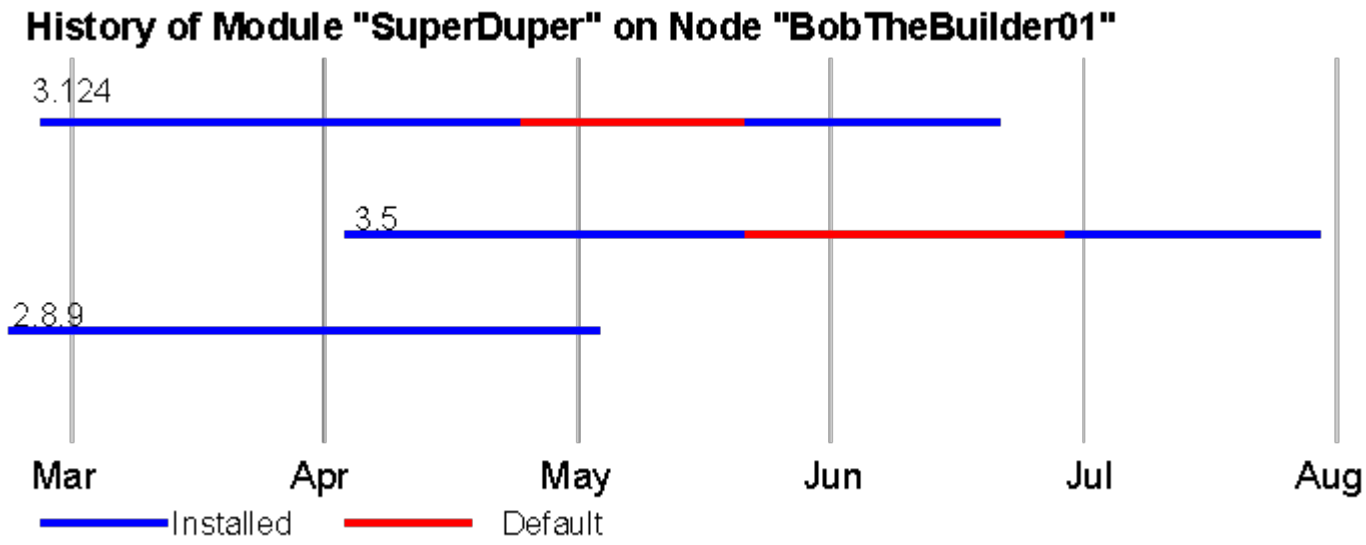
A Node Manager role should be able to have access to data concerning HPC clusters. They will be able to modify nodes in a cluster and also modify clusters themselves. They have the ability to add or remove nodes from a cluster, add or remove clusters, and to modify any data for a cluster or node.

A Software Manager role should be able to have access to data concerning software modules installed on HPC clusters. These users will be able to modify the software displayed by the system, most notably they will be able to define more categories of software for users to view.

Finally a Deity role should be able to have ultimate access to the system. They will have all the powers of General Users, Node Managers, Software Managers and then more. They should be able to modify users and roles and also define permissions for users and roles.

The history feature in the current software does not work correctly and is complicated to understand. We just corrected the errors in the current history display feature and added notes to make it easier to understand. Below is a portion of a sample output from this history display application. Blue lines represent the times that a particular version of software was installed on the system, while red lines mark the times when those versions of software were the default versions for that cluster. The history view has two layers, one that has a resolution of years and one that has a resolution of months. What you see below is a portion of the part of the graph with a resolution of months. These two different parts are mainly what confused users, so we added comments to the webpage trying to explain the different pieces of the graph better.

Example History View



Another important part of our project is the automation of data collection on clusters. We implemented this through a shell script that automated the execution of other already existing scripts on nodes of a cluster. One of these existing scripts gathers and formats the data while the other one puts it all together and tar's it up for the user to use when needed. Our script has to submit jobs to all the nodes in a cluster, run the script that gathers data on those clusters, and keep track of the progress of each of those submitted jobs. Then our script calls the last script that tar's up all the data that was collected. Clusters at LANL can be extremely busy at times, so there may be times when our script will not be able to run on back end nodes. To handle this problem we created a “timeout” feature in out shell script. The user can either specify a desired timeout or use the default five minute timeout. If the job for a back-end node does not finish before the timeout, it will be killed and any data gathered from other nodes will be bundled up like normal. The user is displayed a message that some of the jobs were not able to complete.

Our software interacts with many different databases. Most of these have to do with the roles and permissions portion of our project; they are databases that hold information concerning users, roles, and the various permissions. We also have databases to hold information gathered from all the HPC clusters. More information is provided in the Programmer's Guide concerning the databases and their layouts.

Design Language Decisions:
For this project, there was only one point where we had a choice on which language to choose. When automating the data gathering, we needed to write a shell script to complete this task. The already existing code was written in tcsh, however since we did not need to work with this, the only thing we needed to do was call it, we decided it would make much more sense to program in a language that we already knew.

This was the only choice in language that we really had throughout the project and it was a simple choice as it would have no affect on how the project ran whatsoever. Therefore we chose to program in bash because two of our three members had already had experience shell scripting in bash on a similar project making at least this part of the project much simpler. The website portion of this project already existed, so we could not choose a different language to design this in. We had to use PHP and the CakePHP framework. We also had to use the already existing mySQL database schemas.

For much of our project we used built-in CakePHP functionality. CakePHP has a lot of already existing tools to accomplish tasks we needed to do. We especially used cake auto-generated code to quickly complete items that we knew would have to be redone once the software was released to the client anyway. Los Alamos will need to rework the authentication portions of our code to comply with their own security standards, so we did not spend much time creating authorization code.

Design Code Choice Decisions:

For our project, we had to work directly with code already written and improve upon this. By that I mean that we had to add our own code as well as edit already existing code. For the majority of the project, we used CakePHP's code generation (cake bake) to implement the additions and features that the client wanted in the project. We did this because CakePHP would give us everything that we needed for the section. It also gave us a basic layout and consistent setup for the code that we were able to use throughout most of the project. We started off originally designing our own html layouts and implementing our code within that layout, but then we decided to auto-generate much of the code making it much more user friendly, consistent, and time effective.

Problems:

One of the issues we had to resolve was the basic setup of the project. We started off not having any access to the programs we needed to setup the project, or an environment in which to work. This includes server access at the school and at LANL. Because of this, we were set behind from the start on this large project; it took about a week to setup what we needed at the school.

Once this was set up we could work on the GUI while waiting for the other information we needed from LANL as well as our crypto-cards to get into their system. It took another 2 weeks for us to get the needed access to LANL's computing clusters to complete the automation portion of the project; however this portion of the project turned out to be much easier than we had first planned for.

Another snag we ran into was trying to send emails from our virtual machine. We needed to send emails for the notifications portion of this project, but since the email server couldn't be set up properly at school, we had no way of testing this functionality. As a result we postponed the notification portion of the project, which we did not have time to complete.

One of the biggest problems we encountered was time management. We did not plan this project out as well as we should have from the beginning. While we were waiting for the environment to be set up we could have been doing more design of portions of

our project, or doing more research on the automatically generated code that CakePHP could provide. When we were about halfway through the project we actually started to plan out portions of the project more. Looking back on the project now we realize how much time we could have saved if we would have planned out our work better.

The environment took us a lot longer to set up than we had first planned on. Even after the environment was up and running and we could start work on it, only one member of our group had access to the virtual machine on which our web server was running. It was not until halfway through the fifth week of the project that the rest of our group obtained access to this virtual machine. This slowed us down because a lot of the work we did all had to go through one member of our group.

Results:
Our data gathering script had to interface with two already existing scripts. This interface was pretty trivial however, all our script had to do was call these two other scripts. To check and demonstrate that our script performed correctly all we had to do was run our script and then manually gather data like our client would normally do and compare the results.

The website portion of the software had to be able to interface with the data that the shell scripts gather. However this interaction was already implemented in the original so we just had to make sure we did not break anything while adding on our additional functionality.

Project Constraints:
We were bound to the already existing versions of php and CakePHP, as well as the Apache web server they use on their system.

When working with CSS to make our web page look as good as possible, there are lots of different features that are available. The only problem with this is that there are also many different web browsers, some of which may not support the version of CSS we were using. Thus when making our GUI and adding different CSS features, we had to be careful that this would not affect the web page or crash it if it were used on a different web browser than the one we were using. This was not too much of a problem as we were given the specifics from the client that most personnel at LANL use either Firefox or Internet Explorer or any other of the top browsers, all of which should be up to date in this aspect at all times.

Scope and Project Progression:
When given our project, we were given three large projects to be completed. These three projects were the roles and permissions, the automation, and the notifications. Along with that we were given smaller projects/tasks to be completed when we had the opportunity. This left us with a very open ended and dense project as we saw it. The order of completion/importance was that the roles and permissions were most important. The automation was next and then the notifications last.

When running preliminary testing, we found out that we would not be able to test our email notifications due to not being able to send emails from our server. Thus we put the majority of our time into the roles and permissions, which was not a problem since it was our highest priority task. We set our goals according to this and thus we had planned to finish the GUI and roles and permissions about a week before the project was due and then finish the automation and notifications during that last week.

Because of the timing of this project, the automation and notifications became much smaller roles than were originally hoped for. This meant that the automation was still going to be completed the same way on the cluster side, but we did not have time to also automate collecting that data to the web server. Instead of implementing an entire new notification system, we planned on using the inconsistency system already implemented and just adding the capability to send emails to this. All of these decisions were discussed with the client and agreed upon.

Conclusion:

Our project was in essence a few different projects all bundled into one. Each of these had it's own goal as well as constraints and instructions. The automation of data collection from the HPC clusters was accomplished. We were able to add a roles and permissions system into the web pages, along with many administrative functions to access and control databases. We fixed all the errors we found in the history display functionality and added comments to hopefully make this easier to understand. We were not able to get to the notifications or the server-side automation of data gathering portions of this project due to time constraints.

Concerning the automation of data collection, the only thing that could have gone better is when the MOAB resource manager's command "msub" is called, which runs a command on a machine, it creates an output file in the directory of the script called "slurm-<processId>". At the end of the script, if every process completed, these output files are removed. However if every process didn't complete, we could not remove these files because there are many points where no server time would be given, thus trying to remove a file that did not exist would throw and error that would break the automation. All of these files will eventually get cleaned out during a successful run of the script though, so this was not classified as a large problem.

Since this was our first real-world project for many members of our team, we learned quite a few lessons on what we should and should not have done. After the first week of this project, we knew exactly what we needed to do. Not completely how we were going to do it, but we had all of our tasks and goals in front of us. One of the things we should have done right away was plan how we were going to do the project; determine which parts we were going to work on and which parts we were going to wait on at any particular time.

Another lesson we learned was again about planning: we needed to plan out our GUI to make sure that we built everything around a theme; so that it would be consistent. This project had a lot of wasted time working on parts of the GUI that we eventually changed or eliminated. We ended up wasting time because of this; we had to go back and change pages to the style we eventually settled on. Some pages went through multiple revisions before ending up with the finishing style. In a longer project these multiple

revisions could have been classified under a design phase of the project, but for this project it felt like wasted time.

One of the biggest lessons we learned was that you should learn any special features in the program that you will be using to implement your code. We spent some time building pages for a certain part of the project, and then looked at the CakePHP auto code generation capabilities to find that they provide code to do almost exactly what we needed. If we had researched this before writing any code then we could have saved a lot of valuable time, but the initial manual coding was good to get a better sense of what kind of a page we needed to build.

Learning from all of these lessons, we realized that when working with a large project like this, planning must be used to minimize wasted time. We were given a large project and ended up wasting time working on parts of the project that should have been trivial. Planning could have saved us all of this time, which could have been used to implement a notifications system.

Future Directions:

Our client will need to go back through our code and rework parts of it to meet any of their standards, such as security standards. Most importantly, Los Alamos will need to rework much of how we implemented the authorizations of our web pages to meet their security specifications.

We did not have time to address the notifications portion of this project. Instead of actually implementing this portion, we were only able to describe what we were planning on doing for the notifications. We put comments in the code and added sections to the Programmer's Guide describing what we were planning on doing.

We also did not have the time or access to the LANL server to address server-side gathering of all the data collected on the clusters. We also addressed this in our Programmer's Guide, describing how we had planned to implement this portion of the project as well.

Glossary:

- LANL/Los Alamos- Los Alamos National Laboratory
- HPC- High Performance Computing
- Node- Single computing element
- Cluster- Multiple nodes working together with shared resources.
- (User) Environment- The installed software versions and packages on a cluster or node
- ACL- Access Control List
- ARO- Access Request Object
- ACO- Access Control Object
- PHP- A web scripting language
- CSS- Cascading Style Sheet
- Web App- An interactive web page
- Cake/CakePHP- A framework for php with functions common web app tasks

References:

- lanl.gov - The sites we obtained info from having restricted access
- http://book.cakephp.org/
- http://www.w3schools.com/php/php_intro.asp

Database Interactions Diagram for the Clusters and Their Installed Software