On Ramp to Parallel Computing

Zackory Erickson
Computer Science
Univ. of Wisconsin-La Crosse
La Crosse, WI 54601
erickson.zack@uwlax.edu

Samantha S. Foley
Computer Science
Univ. of Wisconsin-La Crosse
La Crosse, WI 54601
sfoley@uwlax.edu

Abstract

Today parallel computers are used in almost every field of study. However they are difficult to learn to use. Parallel computers are built for performance and must be used via the command-line. Our project aims to increase the ease of use of parallel computers. Through a web interface users will be able to interactively launch jobs on a parallel computer hosted at their university. Users will gradually learn the tools needed to use parallel systems directly, while still being able to do meaningful computation, right from the beginning. As users feel more comfortable with parallel computing, they will be given more control of the detailed build, configuration and launch settings. Throughout this paper we will discuss the high level design and goals of this project, as well as the implementation details of the prototype system we have developed. We would also like to invite users to evaluate the system.

1 Introduction

Parallel computing has become the solution to increasing performance along the curve described by Moore's Law and thus parallel architectures are everywhere. Multiple processor cores are common on commodity computers, from desktops to cellphones. Despite the widespread availability of these architectures, many CS programs do not dedicate significant time to parallel computing topics throughout their curricula. There are many reasons for this, however the problem of access to parallel computers is significantly easier to overcome today.

In addition to almost all commodity multicore hardware that students and faculty are using for work and pleasure, distributed systems have also become cheaper and easier to obtain. A cluster of computers can be easily created out of commodity hardware, by using older machines [13] or Raspberry Pis [14]. Custom solutions, specially designed for higher education, may be purchased through various vendors [7, 5, 2]. Each of these solutions make parallel systems affordable and accessible on even the most modest budgets; however, they do not solve the usability problem: parallel system software remains complex to use and requires significant knowledge to set up.

The system software required to make parallel computers work efficiently is complex and follows a very different philosophy than the multitasking personal computing environments we are used to using on a daily basis. Parallel systems are primarily designed for performance, and thus are designed to have the system software stay out of the way of the application. To do this, a user must set up the application to run, and then ask the system to launch it on dedicated resources when the required number of resources are available. This means that the system will typically only allow one application to use a processor core or node (logical unit of multicore processors) at one time. System software called the *resource manager* is responsible for managing this access and facilitating the launch of jobs. Many times there is more work than there are resources to fulfill this work, so another layer of system software is needed to manage the ordering and scheduling of these jobs (requests to run an application), which is called the *batch scheduler*. A user of a parallel system must be adept at using these systems, in addition to the technology to write, build and execute parallel programs. To make matters worse, the only way to interact with these systems is through the command-line. New users to parallel computing must learn all of these software tools, parallel programming, parallel architecture and how to navigate the command-line all before being able to use a cluster. This steep learning curve prevents many new users from learning to use parallel systems and many educators from teaching parallel computing to students easily.

The goal of this project, On Ramp to Parallel Computing, is to develop a web-based environment that will ease the user into using parallel systems. We do this by creating a familiar point-and-click, web-based environment, with supporting documentation, that allows new users to be able to use existing parallel systems. This approach allows users to run parallel jobs immediately and safely, while learning the parallel computing environment gradually. The target audience for this work is primarily students in undergraduate computer science

programs, but it could easily be adapted for use by students at lower and higher levels, as well as in other disciplines where parallel computation is used.

This paper describes our initial design and prototype implementation. First, we will describe the current efforts to make parallel computing more accessible and easier to teach (section 2). Next, we will describe the requirements for the project in section 3. The design and corresponding implementation are described in sections 4 and 5. Finally, we conclude with a discussion of the current capabilities and future work in section 6.

2 Related Work

There have been many fantastic efforts to make parallel computing easier to learn at various educational levels. There are a few efforts that have served as the inspiration for this work. Members of these projects collaborate to strengthen their offerings and we believe our efforts work in concert with these efforts.

The hardware component of the trio of parallel computing education projects is the LittleFe [17]. LittleFe is a small, portable cluster that can be built for less than $3,000 or for free if you contribute educational materials back to the project [18]. The cluster comes with all necessary software components, and is fun and easy to build for faculty and students. The system software for the LittleFe is provided by the Bootable Cluster CD (BCCD) Project [16]. This project develops and maintains a full system software stack that is portable and contains all the necessary software to boot and run parallel applications in a parallel environment, including some example applications. Lastly, the Computational Science Education Reference Desk (CSERD) [10] provides educators with computational science applications and associated education materials. These curricular modules are designed for high school and college settings. Several of these modules are included on the BCCD which comes with a LittleFe at a buildout event. Taken together they form a complete package of educational materials, system software, and hardware to teach parallel computing. However, one must still be able to navigate the system at the command-line and be adept at parallel computing in order to effectively manage the cluster and teach the material. Students will still be thrown into the unfamiliar environment of command-line Unix tools, batch scripts, and parallel programming. The On Ramp to Parallel Computing project aims to make the transition from no parallel computing experience to running on a cluster a bit smoother through a web interface, which is complementary to these and other efforts to make parallel computing widely available.

Other hardware approaches to making parallel computing widely available include creating clusters out of Raspberry Pis [14], condor pools in computer labs [19], and traditional beowulf clusters [13]. System software for clusters and more specific architectures can be found for free, including the ROCKS software suite and package management tool for educational clusters [8]. These efforts provide affordable and relatively easy to manage cluster solutions, for someone who is familiar with parallel computing. However, the problem remains that one needs to be comfortable at the command-line and understand parallel

architectures and system software in order to even use the machine.

When examining the vast array of tools available for learning to program, many graphical and web-based projects can often be found. Code.org [4] is a prime example of how a website can be designed to help students (or really anyone) learn to program in a fun and graphical way. As a user progresses through the tutorials, the complexity increases and new information that is relevant for the user is presented. We aim to follow this model and apply it to parallel computing.

3 Requirements

The purpose of this project is to provide an easy to use web interface for learning and using parallel computers. The software requirements for the project revolve around the ease of use and connection between a web interface and a target cluster. We assume that there is a cluster available at the institution and someone who is familiar with parallel computing to set up the necessary software for the system.

The web interface portion of the system must be able to accommodate multiple levels of users, including new users, intermediate and advanced users. New users require information about parallel computing, parallel architectures, and the particular application that is being run. The web interface is a perfect place to be able to provide this information and give the user a small number of execution parameters to launch a job, all on one screen. For new users, prebuilt applications will be made available. Intermediate users should be able to upload their own parallel applications and associated Makefiles. The web interface should provide the necessary information to help intermediate users run their code on the cluster effectively and allow them to edit the Makefile. Lastly, advanced users should be able to also create and edit shell scripts to launch jobs, all from the web interface.

Cluster-side software is also needed to establish connections to the web interface allowing users to interact with it remotely. First and foremost, this involves being able to launch jobs on the cluster, via the existing batch scheduler and resource manager. The cluster-side software must be able to handle multiple users and multiple jobs being launched simultaneously in a scalable fashion. Additionally, it should include the ability for users to monitor jobs and for administrators to perform some basic administrative tasks from the web interface. The cluster-side software must also be easy to install on any cluster.

4 Design

The initial design consists of three building blocks: a web interface, a web server and a cluster. The web interface is the graphical portion of this project, which allows a human user to interact with the entire system. The web server hosts the web interface for various users. Lastly, the cluster is the target parallel computer that the user will interact with through the web interface. Figure 1 shows the initial design with these three components.

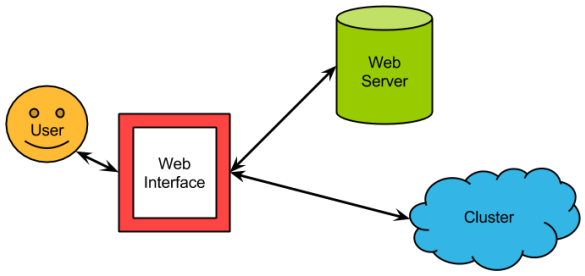

Figure 1: The initial design diagram.

4.1 Web Service Design

In figure 1 we assume that a web interface is able to connect directly with a cluster, however this is not the case. Rather the web interface works best when connecting to other web-like technologies, and thus our cluster needs a way to communicate with the World Wide Web. Therefore, our cluster needs to run a web service to facilitate the communication between the cluster and the web interface. A web service is, as Wikipedia defines, "a method of communications between two electronic devices over the World Wide Web" [12]. The web service enables the web interface to send data directly to a cluster. The web service runs as a daemon process on the cluster and acts as a middleman between data coming from the World Wide Web (via our web interface), and the parallel computer itself. Further, a visual representation of the web service integrated into our current design is displayed in figure 2.

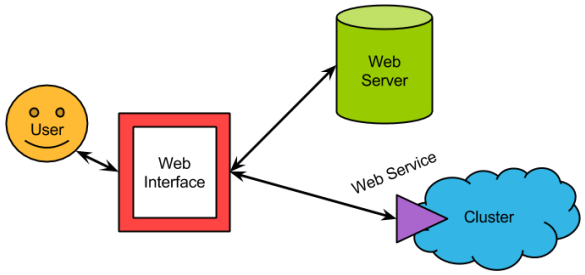

Figure 2: Allowing a cluster to communicate with the World Wide Web through the web service.

A major design feature of the web service is the ability to handle multiple connections, or multiple users, at the same time. Figure 3 displays exactly how this should look. Furthermore, the web service uses multiple threads and processes to service multiple connections and job launches.

The web service acts as a single entry point into the cluster from our web interface, and thus is an ideal place to implement security protocols. The overall design for security on the web service is straight forward. If the received data doesn't first pass through a security check, using a username and password, the data and files sent to the web service are dropped. This prevents unauthorized data from being processed or allowed access onto the cluster. This simple security model is similar to the existing command-line security model

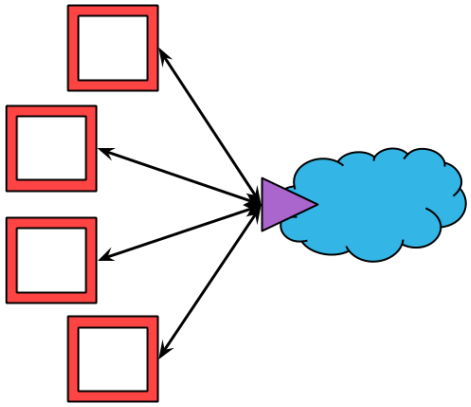

Figure 3: The web service handling multiple web interface connections.

on most clusters and should be sufficient for educational settings.

4.2 Web Server Design

The main task of the web server is to host the web interface and make it available to multiple users at the same time. Although not displayed on the above diagrams, the web server is also used to initially setup the web service on the cluster. One feature we wanted to design for was that a user can setup and install the web service on a cluster, using the web interface. When a user requests to setup the web service on a cluster, we rely on the web server to send all of the necessary setup files to the specified cluster, and then initiate the setup process for the user. When all is said and done, the user's cluster should have a running web service, that was completely setup through the web interface.

Initially, the design used the web server as the intermediary between the web interface and the cluster, as shown in figure 4. This meant that all communication would go through the web server; however, we found that this would become a bottle neck and cause unnecessary overheads in communication. Due to this, almost all communication goes directly from the web interface to the web service on the cluster.

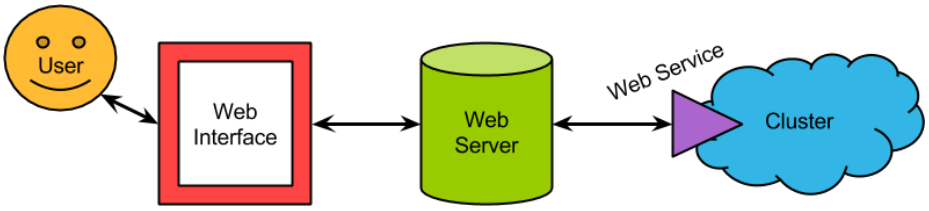

Figure 4: Using the web server as a middleman between the web interface and web service.

4.3 Web Interface Design

The web interface is the most important piece of the design, as it's how we propose to improve the usability of parallel computers. The primary goal of the web interface is to replace the command-line with a graphical interface when launching a job on a parallel computer.

Job launch is the primary activity for which users will be using the web interface. Launching jobs should be as simple for a new user, as it is for an experienced parallel computing user. Thus, we have designed the web interface to support multiple user modes for various levels of parallel computing experience. The only difference between one user mode and the next is what launch options are exposed to the user, with those not exposed to the user set to some default settings.

Viewing the results of the job is a secondary, yet critical, activity that users will perform. When a launched job completes, a user is able to view the results of that job from within the web interface. Users will only be able to see results from jobs they launched, not jobs launched by others.

In order to make the web interface a self-contained learning and parallel computing environment, documentation on how to use the web interface options, specific cluster details, application details and parallel computing basics are provided. Without this system, a new user learning to launch jobs on a cluster will often bounce between a variety of websites and tutorials to learn all of the needed material. The goal of the web interface is to present just the necessary information needed to launch a job using the exposed launch options for each user mode. This information should also include in-depth information about every option and feature available within the web interface in separate documentation built into the web interface.

The web interface is also designed to allow users to connect with multiple clusters. In the case that a user has access to multiple clusters, the web interface is able to establish connections with multiple web services, as shown in figure 5.

Lastly, figure 6 displays the entire design for our project, incorporating the design features described above.

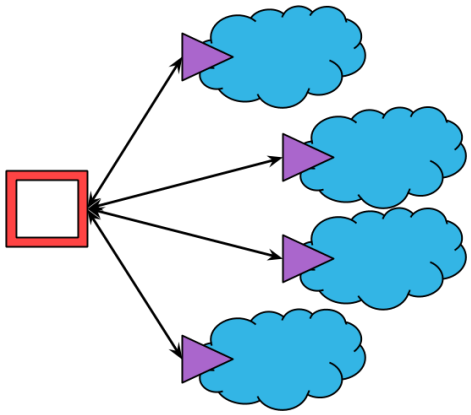

Figure 5: The web interface connecting to multiple clusters, or web services, at the same time.

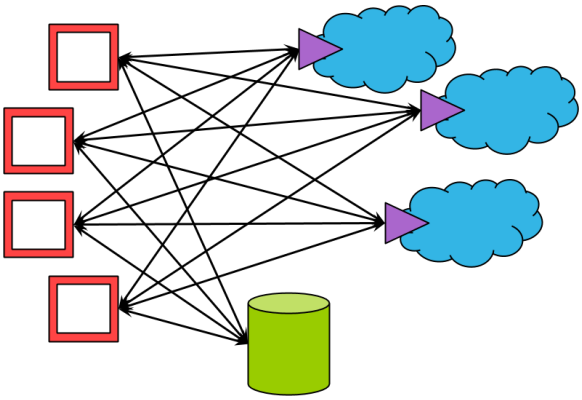

Figure 6: A complete overview of the entire design for our project. For visibility, only a single web server is displayed, however, it is quite plausible that users within the diagram could actually be connecting to various different web servers.

5 Prototype Implementation

5.1 Web Service

When looking for a framework to use for our web service, we aimed for a framework that had good support for REST (Representational State Transfer) calls [15]. We chose CherryPy, a python, object-oriented web framework, that has excellent support for handling REST calls [3]. Since CherryPy is a python framework, we implemented the web service using python 2.7. Python is widely supported and available on most clusters and parallel computers, meaning our web service can be easily setup and run on many clusters with few extra dependencies.

As described in the design, the user will send a request to the web server to setup a web service on a given cluster. Setting up the web service on a cluster can be done through the web interface, and requires only three pieces of information: a valid URL to access the

cluster, a username and password for an existing Unix user account on the cluster. Once a user begins the setup process, the web server will send all necessary setup files to the cluster, and then start the setup. After the setup is complete, the web service is started, and will be running under the provided user account on the head node of the cluster. A virtual user is also created to provide access into the web service.

When looking at using user accounts for authentication we had two main options: use the Unix user accounts that already exist on a cluster, or implement virtual user accounts. At the moment the web service uses a virtual user structure; however, as the project progresses, the use of Unix users will also be implemented. A virtual user is implemented simply as a directory structure, created within the Unix user account space that is hosting the web service. With this, a virtual user can gain access through the web service's authentication; however, does not have access to log directly onto a cluster through SSH. There are a variety of benefits that come with using a virtual user structure, including:

- Simple to setup, minimal maintenance, and increased flexibility.
- Virtual user accounts can be created or deleted on the fly.
- Usernames and passwords can be easily changed.
- Ability to add permissions to specific users.
- Easier for the web service to access a virtual user directory, versus a Unix user's directory.

Each user in the virtual user structure has their own security file which the web service uses for authentication. This security file is created along with a user, and holds only two pieces of information: the user's username, and whether or not the user has admin permissions. Once the security file is created, it is then AES encrypted with the user's password. Since every REST call to the web service must provide a valid username and password that matches an existing virtual user, the security file provides a way to verify this information. Upon receiving a call, the web service opens up the specified user's security file and decrypts it with the provided password. If the password is incorrect, then the decryption will fail and the username from the decrypted file will not match the user's actual username. If the password is correct, then the usernames will match, and the decrypted file will also show if the virtual user has admin permissions. Admin permissions are necessary for performing tasks like creating or deleting virtual users, changing a user's password, or upgrading a web service to a newer version.

A virtual user structure is great for an educational or learning environment, such as with only a single user, or a classroom of students. However, when users start running more sophisticated applications, security becomes an issue, and a virtual user structure is no longer the answer. The virtual user structure is designed to be secure when users launch prebuilt parallel programs that we provide. Yet, when a user launches their own source code on a cluster, it is conceivable that their program could access the entire virtual user directory structure. At this point using Unix user accounts on the cluster is the proper solution as it provides the file system permissions of real Unix user accounts.

5.2 Handling Multiple Users

Just as described in the design, the web service must be able to handle multiple connecting users at the same time. One benefit of the CherryPy framework is that it already handles multiple users with multithreading. When the web service receives data through a REST call, it automatically creates a copy of itself, as a new thread, and then pushes the REST call to this new thread. This way when five users all call the web service at the same time, the web service will create five new threads, one to handle each request.

We get a single level of concurrency automatically with CherryPy's threading to handle multiple users; however, we've implemented a second level of concurrency within the web service as well. The web service creates a new process when it handles computationally significant tasks, such as launching a job. A job launch can take some time as it may involve building an executable and generating a batch script; thus, a separate process is created to fulfill the request so it can run in parallel with the web service. Once the job is launched, the process exits and the user is notified that the job has been launched. Users can then query the web service for the status of the job through the web interface.

5.3 Launching a Job

When a web service receives a REST call to launch a job, a variety of information must be included with the call such as authentication information; project files; and various build and launch settings. Once called, the web service begins by verifying the provided username and password, using the security steps provided earlier. If authentication fails, then the call is instantly dropped and returned to the user with an error message. If authentication succeeds, then we create a second process, and redirect the launch settings and files to this new process. From here the web service thread can return to the web interface, while our second process handles the launch request. If the user did not submit their own Makefile, then the new process will begin by creating a default Makefile for them. A launch script (shell script) will then be created using the launch settings that the user provided. Once the launch script is complete, the source code will be built using the Makefile, and the job will be submitted to the job queue.

The web service actually has two primary functions that it must be able to complete, and launching a job is only half the story. A web service must also be able to send data, including job results and cluster information, back to the web interface. The main difference between launching a job and querying the status of a job is that the web service does not need a second level of concurrency. With or without a new process, the web service must wait until all data has been collected before it can return the requested information to the user.

At the moment we have only tested launching jobs on a Rocks [8] cluster running the Sun Grid Engine batch-queuing system. Despite this, the web service has been designed to be platform independent and handle multiple system software configurations. A few of the

upcoming systems available in the department we will be targeting are: LittleFe which PBS/Torque [11] for job scheduling and resource management respectively, and another cluster that uses SLURM [9] for scheduling and resource management. Once this work is complete, the web service software should be compatible with nearly all commonly used cluster software configurations.

5.4 The Web Interface

The web interface is built with three languages that are commonly found together: HTML5, CSS3, and JavaScript. Together these languages have a huge support base online, they make implementing the web interface extremely simple, and there are plenty of open source libraries and frameworks available for all of these languages. HTML5 and CSS3 are both used for the layout and design of the interface; whereas, JavaScript provides the backend functionality for the interface, and allows us to make REST calls directly to the web service.

5.5 Cluster Connections

The web interface has been designed so that a single user can connect to a variety of clusters at the same time. Every time a user wants to interact with a cluster, a few things are needed to send the REST call to the web service: the cluster's URL, as well as a username and password for a virtual user on the cluster. Rather than forcing users to enter this information in every time they communicate with a cluster, users can save this information by adding a 'connection' to their cluster, as shown in figure 7. Additionally, users can save multiple connections, each of which are stored as a session using HTML5's Web Storage [6]. Once a connection has been saved, a user can effortlessly select which connection (cluster) they would like to use when launching a job or viewing results.

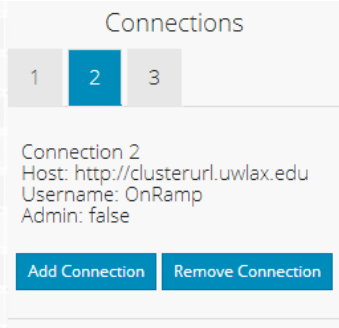

Figure 7: Multiple connections displayed within the interface.

5.6 User Modes

A major highlight of the On Ramp to Parallel Computing project is that it aims to make parallel computing easier, and we make this possible through multiple levels of launching

capabilities, called user modes, with corresponding documentation and options. Launching a parallel program should be no more difficult for a brand new user as it is for someone with years of parallel computing experience. To prove this, we built three different user modes, each for varying levels of parallel computing experience. The first mode, allows novice users to launch prebuilt applications with a few launch options that allows for exploration of parallelism without having to deal with all of the details of batch schedulers, resource managers and parallel programming libraries. Users brand new to parallel computing will find this mode to be the most comfortable. Figure 8 shows how simple launching a prebuilt project is. The prebuilt applications have been adapted from the CSERD [10]. They include educational materials and all necessary source code to run them. As the project progresses, we plan to include even more projects from the CSERD.

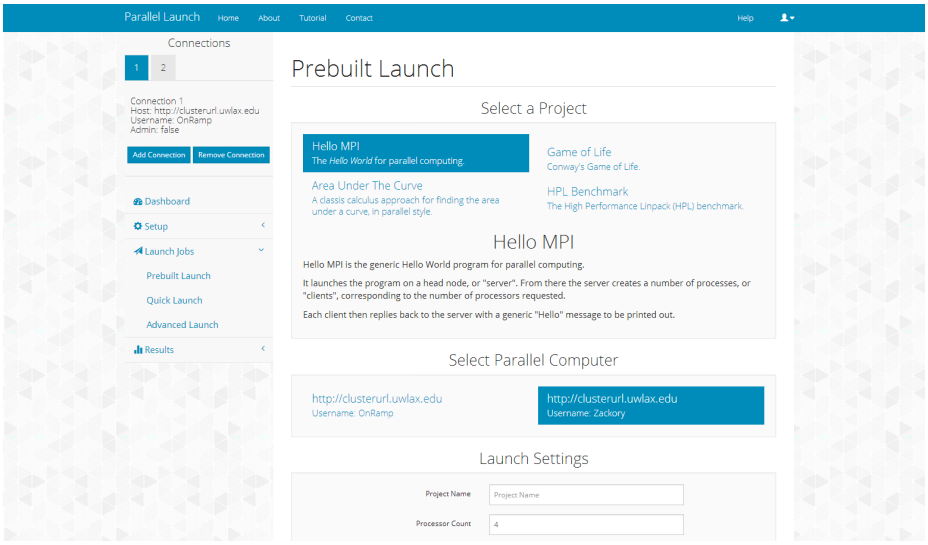

Figure 8: Launching a prebuilt project can be done with just a few steps. First select a prebuilt project and an available connection (parallel computer). Next give the project a name, enter in how many processors the job should use and click Launch.

Once a user requests to launch a prebuilt project on a cluster, the web service will begin by copying the prebuilt project into the virtual user's directory. No source code or Makefiles are needed from the user to run them, making these applications perfect for new users to explore parallelism. From here the project and source code will be built and then subsequently launched. Overall prebuilt projects are a non-scary introduction into launching jobs that provide real results back within seconds.

The next user mode is for intermediate users who have some experience in parallel computing. In this mode, the web interface allows users to upload their own project files and source code. In addition, the intermediate mode provides significantly more launch settings, so that users can begin to experience how various settings affect a launched job. Users also have the choice between using the web service's default Makefile, or uploading their own Makefile. When users select to upload their own Makefile, we provide an

embedded code editor in the web interface using a JavaScript library, Ace [1]. Using an embedded code editor allows us to provide a default Makefile which works with all of the prebuilt projects, and provides users with a starting point when uploading their own Makefile. If users have no experience using a Makefile, the embedded code editor allows them to learn from and make modifications to a working generic Makefile. Figure 9 shows this embedded code editor in action.

```
1  # $\Id$
2  # "ProjectName" Should match the Project Name provided below.
3  PROGRAM = ProjectName
4  CSRCS = hello-mpi.c
5  INCS =
6  CLEAN = $(PROGRAM).c-mpi
7  all: c-mpi
8  c-mpi: $(CSRCS) $(INCS) $(PROGRAM).c-mpi
9  $(PROGRAM).c-mpi: $(CSRCS) $(INCS)
10 include ../../../Makefile
```

Figure 9: Embedded code editor with the default Makefile.

Although not yet implemented, the last mode will be for advanced users. Users who are comfortable with parallel computing will find this stage to be the most robust and helpful. The first benefit of this mode is that all launch and build settings will be available. Furthermore, it will provide users the option to upload and edit their own launch script. This will again be implemented through an embedded code editor with a default or generic launch script to give users a place to start from.

5.7 Viewing Results

After a user has launched a job, they can view the job's results and output directly within the web interface. Viewing results is designed to be as simple as one click. Once a user has navigated to the results page within the interface, they can select an active connection to view the results for the specified virtual user on a cluster. When a connection has been selected the web interface will send a REST call to the web service, requesting all of the user's launch results. Once the web interface receives the results, we will display them to the user as shown in figure 10.

6 Conclusions

The prototype implementation described in this paper demonstrates an exciting start to this project. It proves that it is possible to build such a web interface to make parallel computing easier to use. The current design and implementation focuses on the main functionality that is needed for the whole project to work, namely the remote launch capability.

Figure 10: Viewing job results within the web interface.

Now that the prototype is implemented, there are several areas for improvement, including: finishing the implementation for the different user modes, including the documentation and launch options; extend the web interface to be able to launch other prebuilt applications; extending the web service tools to be able to run in different cluster environments including those mentioned in section 5.3; and, finally testing the system in an educational setting. In addition to these near-term goals, we also plan to implement more administrative and cluster monitoring tools that will allow users and administrators to watch the usage of the cluster as a whole.

Currently, this work has been funded by a small undergraduate research grant and we plan to apply for external funding by the end of the year. With that funding, we are hoping to attract new students to work on these features and more. We are also hoping to attract some early adopters to test the system and provide feedback.

References

[1] Ace. `http://ace.c9.io/`.

[2] Aspen systems. `https://www.aspsys.com/sectors-served/higher-education`.

[3] Cherrypy. `http://www.cherrypy.org/`.

[4] Code.org. `code.org`.

[5] Cray Supercomputers: Solutions for Higher Education. `http://www.cray.com/IndustrySolutions/HigherEducation.aspx`.

[6] HTML5 Web Storage - W3Schools. `http://www.w3schools.com/html/html5_webstorage.asp`.

[7] Penguin Computing. `http://www.penguincomputing.com/solutions/education`.

[8] Rocks Open-Source Toolkit for Real and Virtual Clusters. http://www.rocksclusters.org/wordpress/.

[9] Simple Linux Utility for Resource Management (SLURM). https://computing.llnl.gov/linux/slurm/.

[10] The Computational Science Education Reference Desk (CSERD) - The Shodor Education Foundation. http://shodor.org/refdesk/.

[11] TORQUE Resource Manager - Adaptive Computing. http://www.adaptivecomputing.com/products/open-source/torque/.

[12] Wikipedia: "web service". http://en.wikipedia.org/wiki/Web_service.

[13] J. Adams and D. Vos. Small-college Supercomputing: Building a Beowulf Cluster at a Comprehensive College. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '02, pages 411–415, New York, NY, USA, 2002. ACM.

[14] S. J. Cox, J. T. Cox, R. P. Boardman, S. J. Johnston, M. Scott, and N. S. O'Brien. Iridis-pi: a low-cost, compact demonstration cluster. *Cluster Computing*, pages 1–10, 2013.

[15] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, 2000.

[16] A. Fitz Gibbon, D. A. Joiner, H. Neeman, C. Peck, and S. Thompson. Teaching High Performance Computing to Undergraduate Faculty and Undergraduate Students. In *Proceedings of the 2010 TeraGrid Conference*, TG '10, pages 7:1–7:7, New York, NY, USA, 2010. ACM.

[17] C. Peck, T. Murphy, P. Gray, S. Thompson, J. Houchins, A. Weeden, A. F. Gibbon, D. Joiner, I. Babic, M. Ludin, L. DeJong, K. Muterspaw, S. C-P, and I. Traxler. Little Fe Project. http://littlefe.net/, March 2014.

[18] C. Peck, T. Murphy, S. Thompson, and A. Weeden. LittleFe Buildout Workshop (Parts 1 & 2): Hardware, Software, and Curriculum for Parallel and Distributed Education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 763–763, New York, NY, USA, 2013. ACM.

[19] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.