Incorporating Interactive Compute Environments into Web-Based Training Materials using the Cornell Job Runner Service

Susan Mehringer
Cornell University Center for Advanced Computing
530 Rhodes Hall
Ithaca, NY 14853
1.607.254.8777
shm7@cornell.edu

Aaron Birkland
Cornell University Center for Advanced Computing
535 Rhodes Hall
Ithaca, NY 14853
1.607.255.3431
apb18@cornell.edu

ABSTRACT

Online training materials, such as the *Cornell Virtual Workshop*[SM] have many advantages, the foremost being that they are always available as a 24x7 option for learners who want to study a topic on demand and at their own pace. It can be challenging to create online materials that are engaging and provide a realistic learning environment. Traditionally, training materials and compute environments have been separate entities. Even in the HPC environment, students learn from online materials in one window, then log into a new machine or session to try out new skills or concepts. Accessing this second environment can impose obstacles such as gaining access to the appropriate computer and learning to navigate a computer-specific login environment and file system. In an effort to circumvent these obstacles, the Cornell University Center for Advanced Computing (CAC) developed the *Cornell Job Runner Service*[SM] (CJRS), along with a general-purpose toolkit for using the CJRS to embed a computing environment directly into web pages, backed by real or virtual compute resources. This implementation provides the learner immediate access to a compute environment that looks and feels like a typical HPC login node or batch job, allowing incorporation of on-demand learning experiences interspersed with general training content. With CJRS, students can try out commands and run jobs without obtaining an account or leaving the learning environment to sign in to a remote machine. This paper explores the use of the CJRS toolkit to provide three different interactive modes for learners: a Linux console configured as a general login node, a form element that launches a pre-defined SLURM job, and a guided session which allows the user to walk through pre-planned steps of compiling, fixing, and running MPI code.

Categories and Subject Descriptors

D.2.11 [**Software Architectures**] *Patterns*
D.4.7 [**Organization and Design**] *Interactive Systems*
H.3.5 [**Online Information Services**] *Web-based services*
H.5.2 [**User Interfaces**]: *Interaction styles, Training, help, and documentation*
K.3.1 [**Computer Uses in Education**]: *Distance learning*

General Terms

Management, Design, Human Factors.

Keywords

Online training, Interactive, Toolkit, Remote execution, Web service, SLURM.

1. INTRODUCTION

The Cornell Virtual Workshop is a set of web-based, asynchronous learning modules on advanced computing topics ranging from high-performance parallel computing to data analysis and visualization. Over 100,000 learners have accessed Cornell Virtual Workshop modules since the online learning platform was launched by the Cornell University Center for Advanced Computing (CAC). CAC has received numerous grants from the National Science Foundation (NSF), the Department of Defense (DOD), and private industry to develop and deploy Cornell Virtual Workshops. For example, under an NSF grant, CAC developed the *Ranger Virtual Workshop* to train educators and students on how to effectively use the Ranger supercomputer [1]. Subsequently, CAC developed the *Stampede Virtual Workshop* [2] which included modules on new technologies such as *Many Integrated Core (MIC)* and *Vectorization in Modern CPUs and the Intel® Xeon® Phi*. Today, thirty-three Cornell Virtual Workshop modules are available to the research and education community through the XSEDE User Portal [3]. These include specific training modules and complete Computer Science courses such as Jim Demmel's *Applications of Parallel Computers* [4].

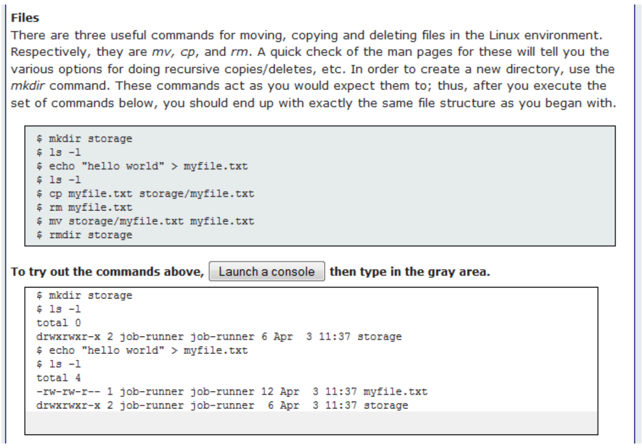
Since the launch of the first Cornell Virtual Workshop, CAC's goal has been to continually improve the learning environment and experience. With this goal in mind, our training and consulting team has developed and integrated a new mechanism into the Cornell Virtual Workshop platform called the Cornell Job Runner Service (CJRS). CJRS provides the learner immediate access to a compute environment that looks and feels like a typical HPC login node or batch job. It lowers barriers to learning by eliminating the need to obtain an account on an appropriate resource, sign into a remote machine, and understand the local compute environment. The CJRS developer toolkit enables courseware developers to quickly incorporate platform-appropriate HPC exercises into web-based training modules.

2. APPLICATIONS

The Cornell Job Runner Service toolkit provides a general-purpose ability to launch jobs on a computing resource, and interact with their runtime environment (for example, writing to the STDIN of a process, reading the contents of a file, etc.). In the context of the Cornell Virtual Workshop, we leverage this ability to enable three learning application environments: a Linux console configured as a general login node, a form element that launches a pre-defined SLURM job, and a guided session which allows the user to walk through pre-planned steps such as run, edit, and run again. All three environments use SLURM to launch jobs on a Virtual Machine (VM) as needed. All jobs are terminated when the web page is no longer active or the job completes on its own terms.

2.1 Console Application

The console application of the CJRS toolkit enables a rudimentary console to be embedded in a web page. The console displays the STDIN and STDOUT of any command that is typed in, providing the learner with a simple, easy-to-use interactive session. This example uses a simple, scrollable preformatted text box to show console output, and has a text input for the learner to issue commands in real time. When the learner selects "Launch a console," the SLURM `srun` command runs and launches a VM with a bash shell, as shown in Figure 1. Each time a learner types content into the text input box and selects enter, the command that the learner entered runs on the VM, with the resulting content posted back to the text box. This does not provide a window to a real console, and it is not a terminal emulator. It can be used to execute individual commands, but at present cannot be used for activities that assume that the learner is at a terminal, such as editing with vi.



Files
There are three useful commands for moving, copying and deleting files in the Linux environment. Respectively, they are *mv, cp*, and *rm*. A quick check of the man pages for these will tell you the various options for doing recursive copies/deletes, etc. In order to create a new directory, use the *mkdir* command. These commands act as you would expect them to; thus, after you execute the set of commands below, you should end up with exactly the same file structure as you began with.

```
$ mkdir storage
$ ls -l
$ echo "hello world" > myfile.txt
$ ls -l
$ cp myfile.txt storage/myfile.txt
$ rm myfile.txt
$ mv storage/myfile.txt myfile.txt
$ rmdir storage
```

To try out the commands above, Launch a console **then type in the gray area.**

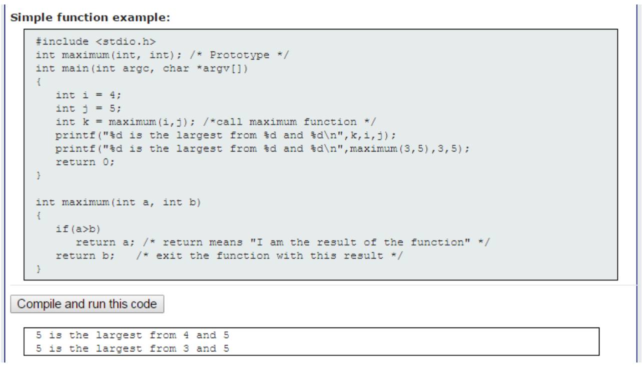
```
$ mkdir storage
$ ls -l
total 0
drwxrwxr-x 2 job-runner job-runner 6 Apr  3 11:37 storage
$ echo "hello world" > myfile.txt
$ ls -l
total 4
-rw-rw-r-- 1 job-runner job-runner 12 Apr  3 11:37 myfile.txt
drwxrwxr-x 2 job-runner job-runner  6 Apr  3 11:37 storage
```

Figure 1: Console Application Example

2.2 Job Launch Application

The job launch application consists of a web page form element that can be used to issue either a SLURM `srun` or `sbatch` command to submit a predefined job or command to a previously started VM. This functionality can be used to run any command or program on-the-fly to show live output or to demonstrate a run that is dependent on changing input. Figure 2 demonstrates compiling and running a C program. It can also be used as a building block to demonstrate a more complicated set of tasks.

A single web page can contain two or more independent forms that run unrelated commands. The forms can be submitted any number of times, in any order, but any two forms cannot be executing at the same time. This only becomes an issue if execution time for a given job is long, or a job waits in a queue. If a form is submitted while another form is still running, the submission will disable the job that was already running.



Simple function example:

```
#include <stdio.h>
int maximum(int, int); /* Prototype */
int main(int argc, char *argv[])
{
   int i = 4;
   int j = 5;
   int k = maximum(i,j); /*call maximum function */
   printf("%d is the largest from %d and %d\n",k,i,j);
   printf("%d is the largest from %d and %d\n",maximum(3,5),3,5);
   return 0;
}

int maximum(int a, int b)
{
   if(a>b)
      return a; /* return means "I am the result of the function" */
   return b;   /* exit the function with this result */
}
```

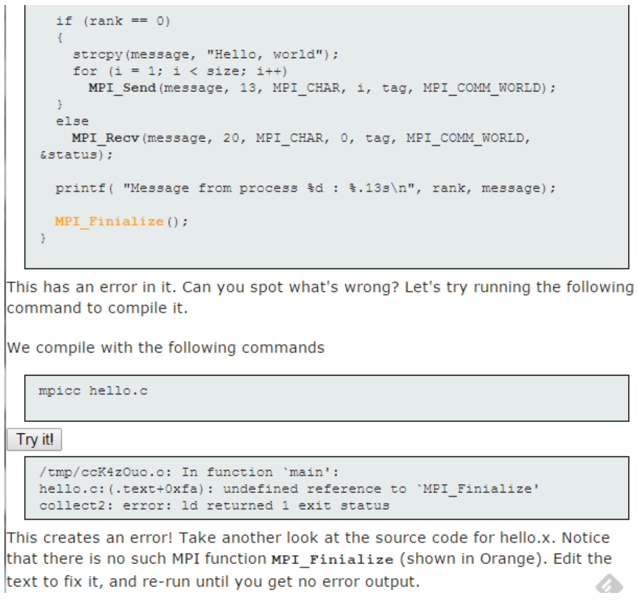
Compile and run this code

```
5 is the largest from 4 and 5
5 is the largest from 3 and 5
```

Figure 2: Job Launch Application Example

2.3 Guided Session Application

In a Guided Session, we present the learner with several different tasks to complete. The learner interacts with multiple "widgets" that dynamically react to the leaner's input. We call this a "guided session" because the tasks are not performed in isolation like the previous simple multi-form example, but are all related and share state. In guided sessions, we use `srun` to run a shell just like the simple console example, but instead of presenting the learner with a shell-like interface, we present a series of widgets that interact with the bash session. We also run `sbatch` and `srun` from within jobs. This allows us to use srun to run a shell, then from within that shell issue subsequent srun and sbatch commands. We can then use a background bash shell managed by `srun` to demonstrate the launching of batch jobs. There is no explicit launch button in this guided session example. The header section contains code that automatically starts the srun session when the page is loaded. We can use the class `xjr-display-hide-until-run` to hide elements until the job is running, or we can do our own special processing by using the `xjc_client.onRun()` hook. For example, if MPI is needed, we demonstrate this effect by submitting a '`module load`' command right away to make sure MPI is available for the whole session. Alternatively, the learner may perform this step. In the example shown in Figure 3, the learner is asked to run an MPI code that fails, edit the code, then run again.



```
  if (rank == 0)
  {
    strcpy(message, "Hello, world");
    for (i = 1; i < size; i++)
      MPI_Send(message, 13, MPI_CHAR, i, tag, MPI_COMM_WORLD);
  }
  else
    MPI_Recv(message, 20, MPI_CHAR, 0, tag, MPI_COMM_WORLD,
&status);

  printf( "Message from process %d : %.13s\n", rank, message);

  MPI_Finialize();
}
```

This has an error in it. Can you spot what's wrong? Let's try running the following command to compile it.

We compile with the following commands

```
mpicc hello.c
```

```
/tmp/ccK4zOuo.o: In function `main':
hello.c:(.text+0xfa): undefined reference to `MPI_Finialize'
collect2: error: ld returned 1 exit status
```

This creates an error! Take another look at the source code for hello.x. Notice that there is no such MPI function `MPI_Finialize` (shown in Orange). Edit the text to fix it, and re-run until you get no error output.

Figure 3: Guided Session Application Example

3. THE CJRS CLIENT: HOW IT WORKS

The Cornell Job Runner Service toolkit consists of a web service, a JavaScript client, and html constructs that allow the client to interact with the page content.

3.1 Web Service

The web service exposes a REST API for clients to run and manage jobs on an execution resource. It is composed of two main components: the JobExecutionService and JobContextService. The JobExecutionService executes a single command as a submitted "job" and provides access to the job's state while it is running. The JobContextService provides a location on the file system for the input/output files relevant to the job, and provides client access to those files (i.e., allows an external client to list the files, read their contents, write to them, etc.), and performs management tasks such as cleanup.

3.1.1 JobExecutionService

The JobExecutionService manages the execution of a job in a resource environment. Its parameters include a single command with associated arguments, and a list of files that shall be present when the job executes (e.g., data files, source code, etc.). The JobExecutionService can be used to enforce job limits, such as a run time limit. It monitors all jobs and terminates jobs that are no longer active. It allows the client to look up the job state, including whether the job is pending, running, or completed, and obtain a list of files that are present at any point during a job's execution. An implementation of the execution service may restrict the set of commands that may run.

3.1.2 JobContextService

The JobContextService sets up and tears down an *empty* directory at job start and cleanup. This is the job context. It serves as the home directory for any jobs running in the context. The JobContextService creates and populates any files specified upon job submission, and provides a list of file information for all files in the context of a given job during or after its execution including name, modification time, size, and location. This includes any files created or modified during the execution of a job. In addition, the JobContextService creates special files .STDIN and .CONSOLE to be used for the job's I/O. Any content appended to the .STDIN file will be piped to the running command's STDIN, and the process's STDOUT and STDERR are appended to the .CONSOLE file.

3.2 JavaScript Client

The JavaScript client is intended to be a convenient tool for constructing dynamic pages that interact with the CJRS web service via its REST API. It provides methods to inspect and interact with job state (e.g., read the content of files in the job's environment, write to file in the job's environment), as well as trigger actions in response to changes in job state (e.g., a new file is created, a file is updated, the job finishes, etc.). For example, the JavaScript client may be used to upload user-supplied code from a text box, run a command to compile the code, run the resulting executable, and display the contents of an expected output file once it has been written by the executable as a triggered action.

3.3 HTML Constructs

The JavaScript client defines a list of html elements and classes that, if present in the document, result in certain behaviors by the client. For example, any HTML element with the attribute `class = "xjr-input-file"` and a `title="<filename>"` attribute will automatically be uploaded by the client into the environment of a job before it executes, with the filename matching the value of the `title` attribute. Given the fairly extensive list of html constructs interpreted by the client, in some cases it is possible to create pages that use the CJRS without having to write additional JavaScript at all.

The Cornell Job Runner Service comes packaged with an implementation of JobRunnerService that uses the SLURM scheduler to run all submitted jobs. The client supplies a `srun` command with arguments to run a single command, or a `sbatch` command along with a batch script to execute everything in the batch script. The SLURM JobRunnerService forwards the supplied SLURM commands to a running SLURM scheduler. The SLURM scheduler is infrastructure independently deployed by a site to manage allocation of compute resources and job execution. The resources managed by the SLURM scheduler known to the JobRunnerService are available as an execution environment for running jobs submitted by the client, be it an institutional compute cluster, an XSEDE resource, or dedicated training nodes. In this sense, SLURM is an abstraction layer that creates flexibility in the type of compute resources that are exposed by the CJRS.

4. DEPLOYMENT OPTIONS AND SYSTEM REQUIREMENTS

The Cornell Job Runner Service was intentionally designed to be a light weight mechanism, leaving much of the exposed capabilities and performance characteristics to the environment in which it is deployed. For example, the SLURM JobExecutionService interacts with a SLURM scheduler, but it is completely agnostic as to the nature of the underlying resources managed by SLURM. Issues such as how large the SLURM cluster is, what software is installed on it and inherent resource limits such as compute time or number of nodes are concerns outside of the scope of the CJRS, but they profoundly affect the user experience.

There are a few system requirements for running the CJRS. It is distributed as an executable JAR file (Java ARchive) that can be run on the command line, or a WAR (Web application ARchive) file that can be dropped into a servlet container such as Tomcat or Jetty. To run as an executable JAR file, the service requires, at minimum, an installed Java Runtime Environment (JRE), and access to a high unprivileged port for exposing the web service API to a browser. The browser needs to be capable of executing JavaScript.

In its default configuration, the CJRS web service (either deployed as an executable jar, or a war file in a servlet container) is configured to use the SLURM JobExecutionService, and directly invokes ‘srun’, ‘sbatch’, and ‘salloc’ commands that are available on the host it is running on. A natural consequence of this is that SLURM jobs are submitted using the same user ID as owner of the CJRS web service process. For the purposes of training and demonstration, it is recommended to deploy the application so that it runs as a single, unprivileged user created specifically for the purpose of training. In theory, however, anybody who obtains the executable jar file may run it on a machine they have access to, bound to some random high port exclusive to that user, allowing it to launch SLURM jobs on their behalf via the REST API.

As an example of a production-level deployment used for training, the Cornell University Center for Advanced Computing operates an instance of the CJRS that supports its training modules on a single virtual machine instance in Red Cloud. That single machine hosts the CJRS, the SLURM scheduler, and runs all the jobs submitted to SLURM. SLURM is configured with a queue containing only one node, capable of running 32 scheduled tasks. Such a configuration is not suited to performing large HPC computations, but works well for training exercises that involve teaching, for example, MPI or OpenMP fundamentals, and is inexpensive to run 24/7. Responsiveness is high, with most jobs experiencing little or no perceptible delay before running.

The machine is configured with Open MPI and offers several software packages operating on XSEDE resources such as Stampede. While users need to be logged in to our training modules in order to see the interactive exercises that submit to the SLURM machine, all jobs on the machine execute as one single, unprivileged user placed into a temporary home directory that lasts for the duration of a single job. Since the machine is a virtual cloud instance created from a master image, it can be destroyed and re-created at any time, resetting it to its initial state.

5. SECURITY CONSIDERATIONS

Most security concerns are addressed by technologies and techniques external to the web application. The CJRS API has only one feature directly related to security: it can be configured to require the client to provide an opaque ‘token’ string in a request, and can be configured to pass this token to some external service that must validate and accept it before the job can be executed.

Authentication, authorization, and network security are provided externally to the application. Network traffic between the client and service can be secured by deploying the web service behind an SSL termination proxy such as an appropriately configured Apache or Nginx web server. Network traffic out of and between compute nodes can be controlled by firewalls or network traffic routing rules.

Authentication and authorization are provided by the web server that serves the pages that contain exercises that use the CJRS toolkit. If the CJRS web service is configured to require an authentication token as a prerequisite for executing jobs, then a backend service for validating this token would need to be provided.

When using the SLURM-based JobExecutionService, SLURM itself provides a degree of security for executing jobs. For example, SLURM can enforce time or resource limits, and can terminate all processes created in the context of a job when it terminates. If SLURM is configured to launch virtual (Cloud) nodes on which to execute jobs, then the compute nodes can be considered ephemeral and disposable.

At the Cornell Center for Advanced Computing, all jobs execute as a single, unprivileged user on a single virtual machine running in Red Cloud. The node may be periodically terminated and re-launched from its base image, assuring that the results of malicious activity (such as through a local exploit allowing escalated privileges) are limited in duration. Because all jobs execute on the same node and the node is overcommitted in the scheduler (i.e., more jobs can run on the node than there are cores), it is possible for a user to negatively affect the performance of the machine for others. This can be viewed as a denial of service vulnerability. We decided that since the consequences are rather low due to the low-volume training nature of the machine, this risk was acceptable. While a more sophisticated configuration of SLURM or the use of a container technology such as Docker would add an additional layer of flexibility for security and resource limitations, our needs do not require that at this point.

6. EXTENSIBILITY

The Cornell Job Runner Service itself can be configured to use different JobExecutionService implementations. For example, there is a local execution service which simply executes one of a list of allowed commands locally on a machine, rather than submitting a job to SLURM. We have found that a JobExecutionService that submits to a scheduler (such as SLURM) offers the most capability, flexibility, and security for the smallest investment in developer effort when using the CJRS toolkit for HPC training. Alternate JobExecutionService implementations that can submit to other schedulers (such as SGE) would be fairly simple to implement, but were not necessary for our training objectives.

In our view, much of the extensibility in using the CJRS comes from the abstraction offered by the scheduler (e.g., SLURM). For example, Cornell has leveraged this capability to configure SLURM to execute jobs on the same node that hosts the SLURM scheduler and the CJRS web service, providing a highly responsive and cost-effective environment that suits the modest demands of training exercises.

If our training exercises required the performance characteristics of a traditional multi-node cluster, we simply would have configured SLURM differently so as to schedule jobs to run on a cluster. Likewise, the Cornell CAC has created a toolkit and plugin for SLURM that allows SLURM to dynamically launch and tear down cloud nodes for the sole purpose of running a single job. Had we chosen to configure the SLURM instance to use this toolkit, the Cornell Virtual Workshop training exercises would run entirely on virtual compute nodes that appear and disappear on demand.

7. COMPARISON TO OTHER AVAILABLE TOOLS

Finding existing software that could fulfill the educational and ease of use requirements set forth in this project proved to be challenging. There were four key requirements that needed to be met in order to satisfy all our use cases:

1. Integrate into existing, mature training documentation in the form of .html or .aspx files,
2. Emulate the software stack and fundamental HPC technologies (MPI, OpenMP, job submission) of XSEDE resources such as Stampede,
3. Allow users to do potentially dangerous things in a secure fashion, such as compile and run C code,
4. Support interaction paradigms ranging from a full command line interface (shell), to clicking a button to run a code snippet embedded on a page.

IPython Notebook [5] is a Python-based platform for creating interactive web pages containing mixed code, text, and media content. Aimed at the scientific computing community, it has successfully been used as an educational tool in the classroom [6,7,8]. While it is primarily oriented around Python code, IPython's extensibility architecture allows 'kernels' to be developed that allow the incorporation of code in other languages [9].

On its face, IPython offers an excellent platform for creating interactive learning modules. In our case, however, it did not offer a clear way to incorporate exercises into existing static training material in a piecemeal fashion (requirement 1), nor was it amenable to interactively performing tasks commonly seen in HPC workflows. Tasks such as compiling C code and launching parallel MPI jobs to run the compiled code using a batch submission scheduler (requirement 2) were not something that IPython was designed to do.

The LinkSCEEM Supercomputing Training portal [10] provides scientific computing training materials ranging from programming concepts to using HPC systems. While it does not offer a public toolkit or software platform for creating training modules, the interaction paradigm used in the LinkSCEEM training modules is notable. Each training module page contains a small icon which will hide or unhide a transparent SSH console connected to a node on the Euclid training cluster. Users of the training portal must be given accounts on the training cluster in order to log in to their account via the console. The leaner may go back and forth between reading content in the module pages and performing actions in the shell.

The SSH console paradigm employed by the LinkSCEEM training portal is easily added as a layer to any existing html page (satisfying requirement 1), and inherently meets requirements 2 and 3 by providing the user with an unrestricted command prompt on a node in a fully-functional compute cluster. Unfortunately, it does not meet requirement 4 as there is no way to embed interactive content into a web page since *all* interaction is through the console.

ISLET [11] (Isolated, Scalable, & Lightweight Environment for Training) is a framework for creating disposable Linux instances as Docker [12] containers, and providing access to them via SSH. The approach of creating a throwaway virtual Linux instance satisfies requirement 3 handily, as any malicious acts by the user are confined to a disposable sandbox instance. However, ISLET does not provide any sort of user interface, toolkit, or API for integrating with web pages (as in the popup console in LinkSCEEM) as implied by requirements 1 and 4.

Geordi [13] is an IRC (Internet Relay Chat) server that can compile and run C++ code. It is intended for teaching and discussing C++. Its backend architecture is designed to allow arbitrary C++ code to run safely; avoiding infinite loops, memory leaks, and other potentially dangerous problems (fulfilling requirement 3). Because its functionality is specialized towards executing C++ as its sole function, Geordi lacks the features required for a full HPC toolset (requirement 2).

8. FUTURE DEVELOPMENT

The effectiveness of the Cornell Job Runner Service as a teaching tool is highly dependent on the computing environment available to the user. To learn MPI, OpenMP, or scientific computing in Python, the computing environment must have the correct software and frameworks installed and available. To learn how to use a specific XSEDE resource such as Stampede, the computing environment must look and feel like Stampede. Outside of executing jobs on Stampede itself, re-creating an environment similar enough to serve a useful training purpose is a challenge.

As part of the XSEDE Campus Bridging [14] effort, Cornell is currently researching the use of containerization [15] technology such as Docker to package and distribute container images containing software stacks that mimic various XSEDE resources. By running jobs within a container, it is possible to emulate the look and feel of an XSEDE resource without requiring the host system to have a particular software stack installed for that purpose. Leveraged for the purpose of training, one can imagine having a modest compute resource available for training (such as a VM or small cluster) with a variety of XSEDE Campus Bridging container images to choose from. In this scenario, the Cornell Job Runner Service and a Campus Bridging container image could be used in a lightweight manner to develop training materials geared toward a specific XSEDE resource or a software package located on an XSEDE resource. We plan to explore container technologies in the upcoming year.

9. ACKNOWLEDGEMENTS

The Cornell Job Runner Service was developed in part with funding from the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1053575.

10. REFERENCES

[1] Mehringer, S., Woody, N., Dolgert, A., Lantz, S. & Stanzione, D. (2011). Maximizing Computational Learning for Faculty and Student Scientists: The Ranger Virtual Workshop. *TeraGrid Conference Proceedings*. Retrieved from: http://www.cac.cornell.edu/about/pubs/RangerVirtualWorkshop.pdf

[2] Stampede Virtual Workshop: TACC User Portal (n.d.). Retrieved from https://portal.tacc.utexas.edu/stampede-virtual-workshop

[3] XSEDE User Portal: On Demand Training (n.d.). Retrieved from https://portal.xsede.org/web/xup/online-training

[4] Cornell Virtual Workshop: Applications of Parallel Computers (2013). Retrieved from http://www.cac.cornell.edu/VW/apc/

[5] Shen, H., 2014. Interactive notebooks: sharing the code: the free IPython notebook makes data analysis easier to record, understand and reproduce" in *Nature, 515* (7525), 151

[6] Raju, A.B. and Annigeri, S. 2014. Computing in engineering education: The current scenario, in *IC3I 2014: International Conference on Contemporary Computing and Informatics*, (Mysore, India 2014), IEEE, 130-134

[7] Wilson, G., Perez, F., Norvig, P., 2014. Teaching Computing with the iPython Notebook, in *SIGCSE '14 Proceedings of the 45th ACM technical symposium on Computer science education,* (Atlanta, GA 2014), ACM, 740

[8] Ketcheson, D., 2014. Teaching Numerical Methods with IPython Notebooks and Inquiry-based Learning, in *SciPy 2014: Proceedings of the 13th Python in Science Conference*, (Austin, TX 2014), 19-25

[9] Rossant, C., 2014, Creating a simple Kernel for iPython in *IPython interactive Computing and Visualization Cookbook*, Packt Publishing, Birmingham UK.

[10] Supercomputing Training Portal. Retrieved from http://supercomputing.cyi.ac.cy

[11] ISLET. Retrieved from https://github.com/jonschipp/islet

[12] Merkel, D., 2014. Docker: Lightweight Linux Containers for Consistent Development and Deployment, *Linux Journal*, *2014*(239)

[13] Geordi – C++ eval bot. Retrieved from http://www.eelis.net/geordi/

[14] Stewart, C. et al., 2012. What is Campus Bridging and What is XSEDE Doing About It? *in XSEDE '12: the 1st Conference of the Extreme Science and Engineering Discovery Environment: Bridging from the eXtreme to the Campus and Beyond*, (Chicago, IL 2012), ACM, 47:1-47:8

[15] Dua, R., Raja, A. R., & Kakadia, D. Virtualization vs Containerization to Support PaaS, in *IC2E '14: IEEE International Conference on Cloud Engineering*, (Boston, MA 2014), IEEE, 610-614.