Obelisk: Summoning Minions on a HPC Cluster

Abstract

In scientific research, having the ability to perform rigorous calculations in a bearable amount of time is an invaluable asset. Fortunately, the growing popularity of distributed systems at universities makes this a widely accessible resource. However, in order to use such computing resources, one must understand Linux, parallel computing, and distributed systems. Unfortunately, most people do not have the time or patience to learn these skills, indicating that the high performance computing (HPC) cluster has not been abstracted far enough. The purpose of Obelisk is to facilitate the submission of jobs via an intuitive web interface, making HPC clusters accessible to all. With the Obelisk web portal, users select a scientific application, configure parameters, and submit their task to the Obelisk manager. This will in turn translate the request to a batch job on the HPC cluster and provide the user with progress information via a status page. Using Obelisk, novice users no longer need to know the arcane incantations necessary to harness the power of HPC minions and instead can utilize a straightforward wizard interface.

1 Introduction

To someone that is familiar with Linux, parallel computing, and distributed systems, making a high performance computing (HPC) cluster do your bidding is relatively straightforward: you simply need to know the right commands! However, most people do not have this type of specialized knowledge, nor should they be expected to learn the seemingly magical incantations necessary to harness the power of multiple machines. For this reason, it is almost necessary that a bridge be built between users and the cluster if the awesome power of HPC systems is to be unleashed. The purpose of our project, **Obelisk**, is to build this bridge and to connect users and HPC systems through an intuitive web interface that automates the creation, submission, and retrieval of scientific computing tasks.

2 Design

An overview of Obelisk is shown in Figure 1. Rather than dealing with arcane Unix command shells and fighting daemons, Obelisk allows users to simply `transcribe` their computational goals and `summon` minions to perform their bidding via a responsive and easy-to-use web application.

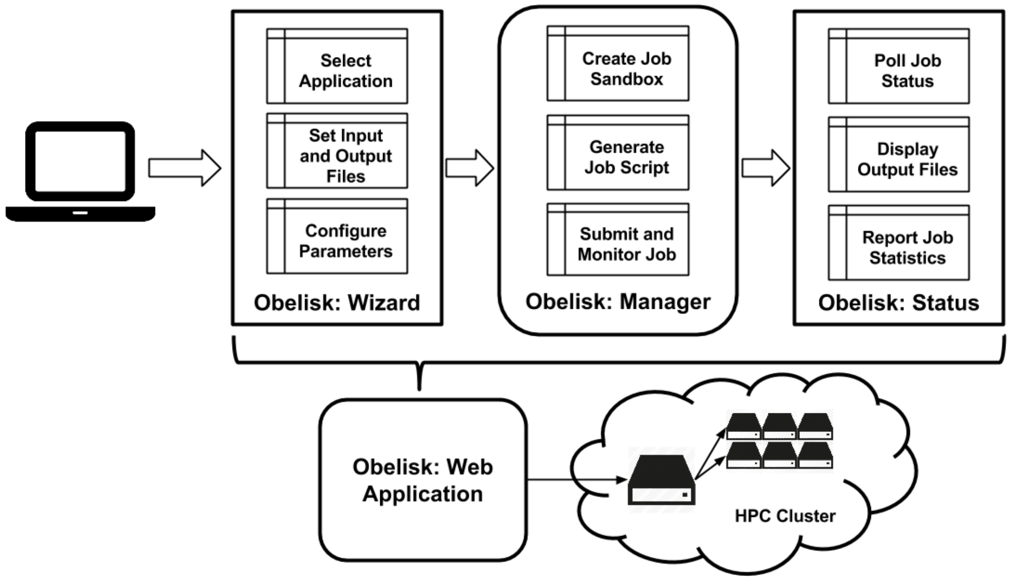
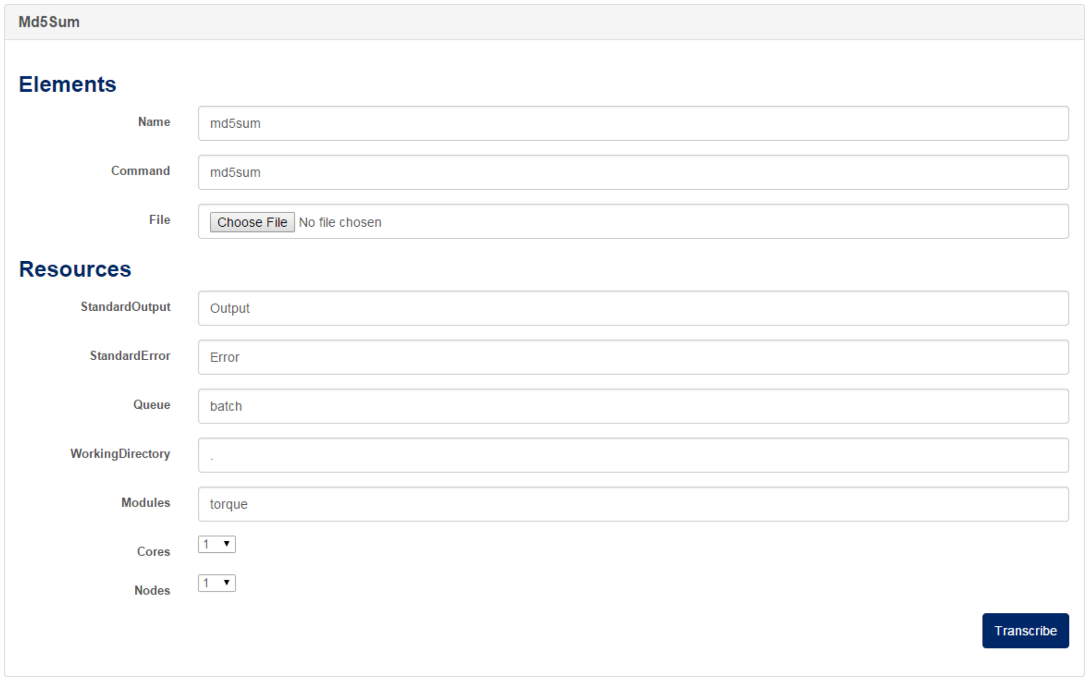


Figure 1: Obelisk Overview

2.1 Wizard

Often times, when a person wants to perform a task (ie. run a program) on an HPC Cluster, they know what they want to do, but don't know how to do it. This is where **Wizards** come into play. When a user opens the Obelisk web application, they will confront a list of predefined applications available on the HPC system. Upon selection of one of these applications, the user will be redirected to a web page containing a dynamically generated form containing various input fields, such as numbers, strings, files, and check-boxes. Each of these fields represents parameters for the command the user wishes to execute, and includes descriptions of their purpose.


Md5Sum

Elements

Name	md5sum
Command	md5sum
File	Choose File No file chosen

Resources

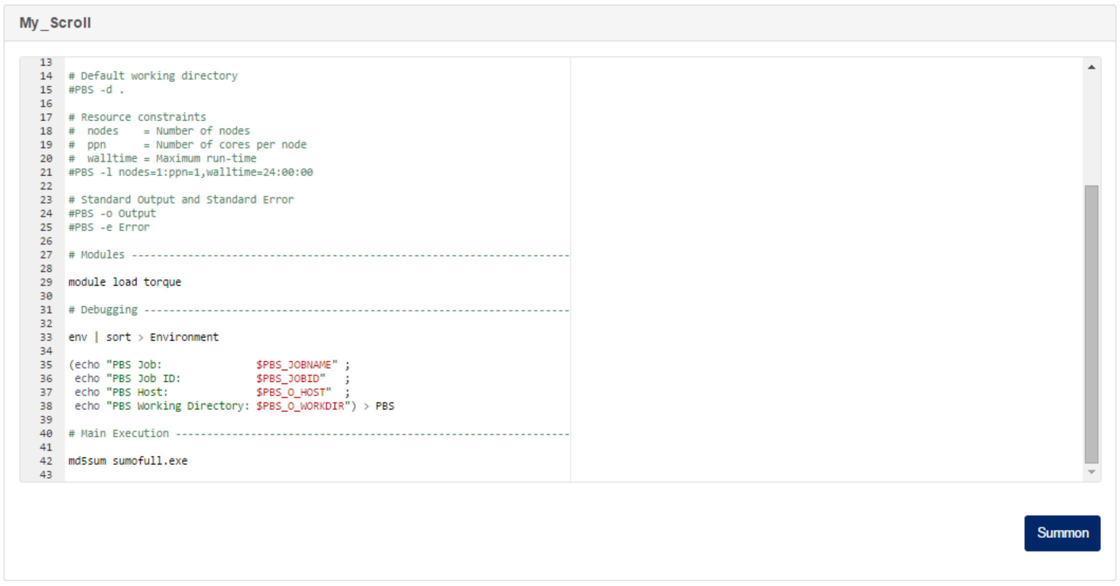
StandardOutput	Output
StandardError	Error
Queue	batch
WorkingDirectory	.
Modules	torque
Cores	1
Nodes	1

Figure 2: Wizard Page

An example of a simple Wizard is shown in Figure 2. As can be seen, each Wizard consists of a set of `Elements` specific to the task being specified along with a set of `Resources` which define the computational resources required from the HPC system. After filling out the form, the user may complete the Wizard and continue onto the next step by clicking the `Transcribe` button.

2.2 Scroll

When the user clicks on `Transcribe`, the values in the form are set to the Obelisk web application, where the data is parsed and analyzed to generate a script that can be understood by the HPC Cluster's batch system. In Obelisk, this materialized script is referred to as a `Scroll` and contains all the necessary magical incantations for executing the user's specified job. After this script is generated, the application redirects the user's web browser to the corresponding `Scroll` page as shown in Figure 3. From here, the user can verify the results of the transcription process and ensure Obelisk has interpreted their intentions correctly. When the user has approved of the script, they may click `Summon` to bring the `Scroll` to life in the form of a `Minion`.

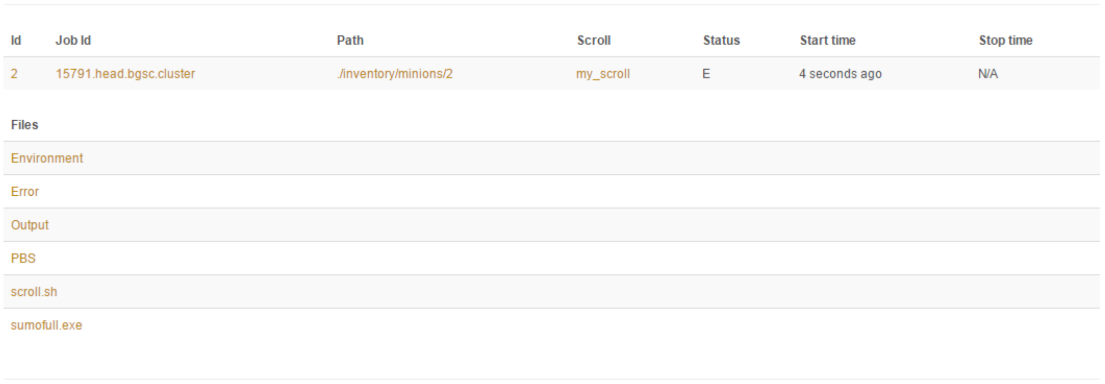

My_Scroll

```
# Default working directory
#PBS -d .

# Resource constraints
#  nodes    = Number of nodes
#  ppn      = Number of cores per node
#  walltime = Maximum run-time
#PBS -l nodes=1:ppn=1,walltime=24:00:00

# Standard Output and Standard Error
#PBS -o Output
#PBS -e Error

# Modules ----------------------------------------------------------------------

module load torque

# Debugging --------------------------------------------------------------------

env | sort > Environment

(echo "PBS Job:               $PBS_JOBNAME" ;
 echo "PBS Job ID:            $PBS_JOBID"   ;
 echo "PBS Host:              $PBS_O_HOST"  ;
 echo "PBS Working Directory: $PBS_O_WORKDIR") > PBS

# Main Execution ---------------------------------------------------------------

md5sum sumofull.exe

```

Figure 3: Scroll Page

2.3 Minion

After summoning a `Scroll`, the user will be redirected to a `Minion` page. A Minion is essentially a personification of a HPC cluster batch job. That is, a `Minion` corresponds to a running task in the HPC job scheduler. Each `Minion` has a workspace, status, and output. The user can easily check on a job status by consulting with their `Minion`. When the `Minion` has completed the `Scroll`, its status will be changed to complete, and the output files will be available for download or viewing as shown in Figure 4.



Id	Job Id	Path	Scroll	Status	Start time	Stop time
2	15791.head.bgsc.cluster	./inventory/minions/2	my_scroll	E	4 seconds ago	N/A

Files
Environment
Error
Output
PBS
scroll.sh
sumofull.exe

Figure 4: Minion Page

In summary, in order to mask the complexity of traditional HPC systems, Obelisk provides an intuitive web interface that allows users to employ `Wizards` to transcribe `Scrolls`, which in turn are used to summon `Minions` to perform their desired computational tasks.

3 Implementation

To implement Obelisk, we used a number of different web development and HPC technologies. The Obelisk web application itself is written in Python and the Tornado web framework [4]. It is structured as a RESTful [5] web service that receives HTTP requests (i.e. `GET` and `POST` methods) and processes them via *handlers* to perform the appropriate actions. For the front-end user interface, Obelisk utilizes Bootstrap and JQuery to mediate the interactions between users and the RESTful backend. All of the this activity is tracked and recorded in a database using the SQLite library, making information pertaining to jobs readily available to users and the program itself. Finally, batch jobs are submitted and managed by Torque, an open source project that is often used in HPC clusters for resource management and allocation. The following is a detailed explanation of how all these components are integrated within the Obelisk to achieve the actions described previously in the Design section.

3.1 WizardHandler

The user begins the process of submitting a job with Obelisk by making a HTTP `GET` request to the `/wizard/` endpoint. Internally, this is processed by the `WizardHandler`, which in this case produces a list of `Wizards` that the user may run. These `Wizards` are generated dynamically from information inside of a collection of `YAML` files stored remotely on the web server. When the user selects a `Wizard`, they are redirected to `/wizard/wizard-name`.

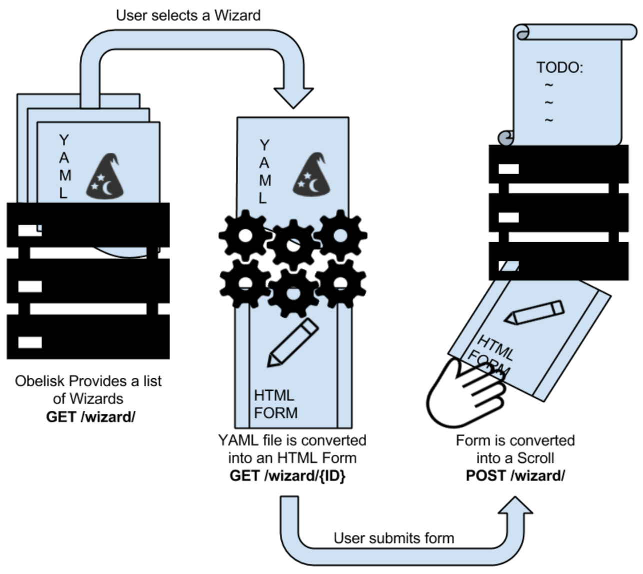


Figure 5: Wizard Handler

Once again, this page is handled by the `WizardHandler`, which parses the corresponding `YAML` file and produces an `HTML` form with inputs that determine parameters like resources allocated, queue name, error and output file names, and parameters for the command to be run. When the user clicks `Transcribe`, the form is sent back to `/wizard/`

via a POST and its values are converted into a batch script, also called a Scroll, that can be understood by Torque. This Scroll is stored in a unique directory dedicated to that Scroll that contains the script and any files uploaded by the user. Likewise, an entry for the Scroll is recorded in the database which tracks the Scrolls name, path, creation time and modification time. Once all of this information is stored, the user is then redirected to the Scroll's page and its contents are displayed. This whole process is outlined in Figure 5.

3.2 ScrollHandler

As with the Wizards, a list of Scrolls can be viewed at /scroll/, which is processed by the ScrollHandler. When the user selects a Scroll from the listings, they are redirected to /scroll/scroll-id, where they can view the content of the Scroll and choose to execute the script by clicking Summon as shown in Figure 6. When this button is clicked, a POST request is made to the current page, and the job is submitted to a queue with the Torque command, qsub.

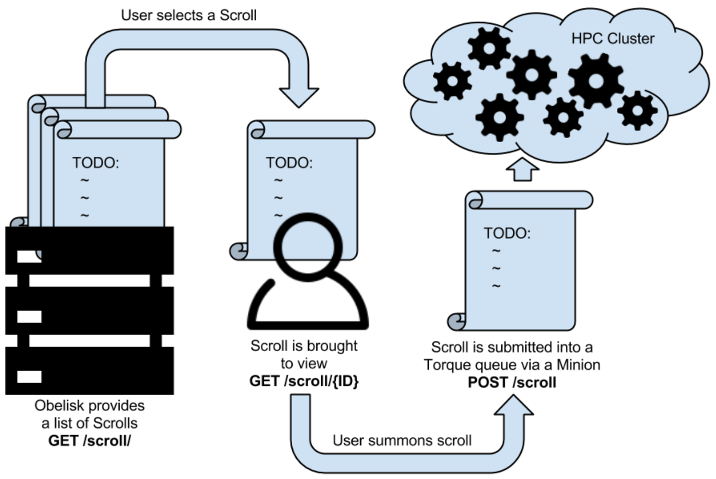

User selects a Scroll
HPC Cluster
TODO:
TODO:
TODO:
Obelisk provides a list of Scrolls GET /scroll/
Scroll is brought to view GET /scroll/{ID}
Scroll is submitted into a Torque queue via a Minion POST /scroll
User summons scroll

Figure 6: Scroll Handler

Before submitting the job to Torque, however, the ScrollHandler will first create a unique Minion directory and copy the contents of the Scroll directory to this new Minion workspace. Once this is accomplished, Torque's qsub command is invoked with the Scroll generated previously and the current working directory is set to the Minion's directory. This approach to materializing new directories for the Scroll and the Minion allows users to summon multiple Minions from the same Scroll without worrying about overwriting previous results. Moreover, having separate Minion workspaces provides sandboxing and isolation for concurrently running batch jobs.

3.3 MinionHandler

Like the other handlers, the `MinionHandler` provides a list of `Minions` that can be viewed by going to `/minion/`. The status and output of any `Minion` can be viewed by going to `/minion/minion-id` as shown in Figure 7. The progress of each `Minion` is updated by a timer that periodically executes `qstat` on the user's jobs, and the output files are contained inside of the `Minion` workspace. When a file is selected, the user is redirected to an additional `MinionFileHandler` that serves the file based on its `MIME` type.

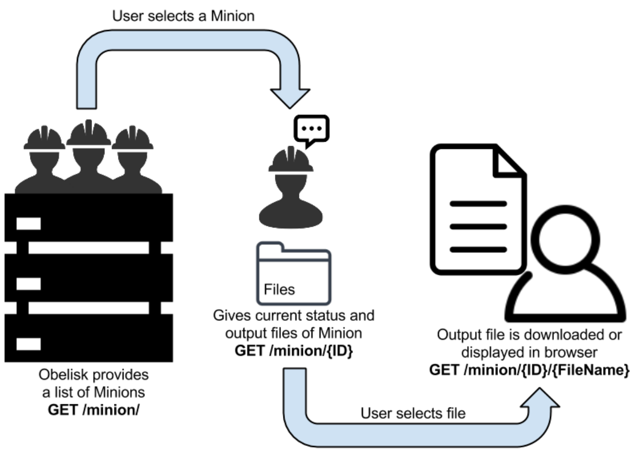

User selects a Minion
Files
Gives current status and output files of Minion
GET /minion/{ID}
Output file is downloaded or displayed in browser
GET /minion/{ID}/{FileName}
Obelisk provides a list of Minions
GET /minion/
User selects file

Figure 7: Minion Handler

3.4 Database

As mentioned above, Obelisk keeps track of `Scrolls` and `Minions` with database tables managed by SQLite. The `Scrolls` table includes the name, id, path, creation time, and last modification time of every `Scroll`. A new row is added every time the user creates a scroll and the number of times it is used is determined by the number of `Minions` with said `Scroll` as a foreign key. Likewise, a new `Minion` is added every time the user summons a `Scroll`. The `Minions` status and finish time are updated by the timer described in the Minion handler section.

This timer process is shown in Figure 8. Once every 5 seconds, the updater is activated and calls Torque's `qstat` command with the user's `UID` to only get a listing of the user's jobs. This information is then parsed for the Torque job id, job status, and completion time. If these fields are found, then the job id is used by the updater to lookup the corresponding `Minion` and update the appropriate job status and completion time. Because of this approach, Obelisk can accurately report the status of each `Minion`, even in the cases of application failure.

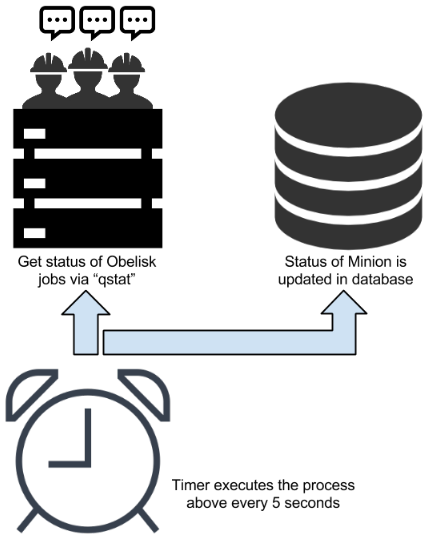

Get status of Obelisk jobs via "qstat"
Status of Minion is updated in database
Timer executes the process above every 5 seconds

Figure 8: Database Updater

3.5 DashboardHandler

To tie all of this information together is the `DashboardHandler`, which provides an overview of the `Wizards`, `Scrolls`, and `Minions` available on the user's instance of Obelisk. From this landing page, the user can jump immediately to a `Wizard` page to `Transcribe` a new `Scroll`, a `Scroll page` to `Summon` a new `Minion`, or a `Minion` page to view the progress of their work.

3.6 Deployment

Currently, to deploy and start Obelisk, a user needs to checkout the source code from our Bitbucket repository: `https://bitbucket.org/pbui/obelisk`. Once they have the source code and have installed the necessary dependencies (ie. Python and Tornado), they can start it by executing the command `python obelisk` on the HPC submit node. When this command is invoked, the web application will be started and the IP address and port number will be displayed to the user. At the moment, there are no security or authentication measures implemented, although that is on our list of future improvements.

The benefit of this approach is that it was simple and straightforward, while allow us to avoid the complications of user authentication and system security. Since the Obelisk application is executed by the owner, any jobs submitted to the HPC batch system are also tied to the user. Unfortunately, the current method of starting Obelisk breaks the facade of a magical web portal for summoning minions, so we hope to create a system for managing user instances of Obelisk in the future. This new system will allow users to avoid manually deploying and invoking Obelisk; instead they would use the a portal setup by the system administrator for accessing their Obelisk session.

4 Evaluation

We have tested the functionality of Obelisk and verified that it in fact works correctly. We will continue to add more features and fix bugs, but for now we will focus on getting more `Wizards` available for demonstration and use. In the future, we will work with domain scientists and have science faculty and students utilize our project to summon minions for class assignments and research experiments. In doing so, we will collect data that allows us to determine the effectiveness of Obelisk and find ways to improve the user experience.

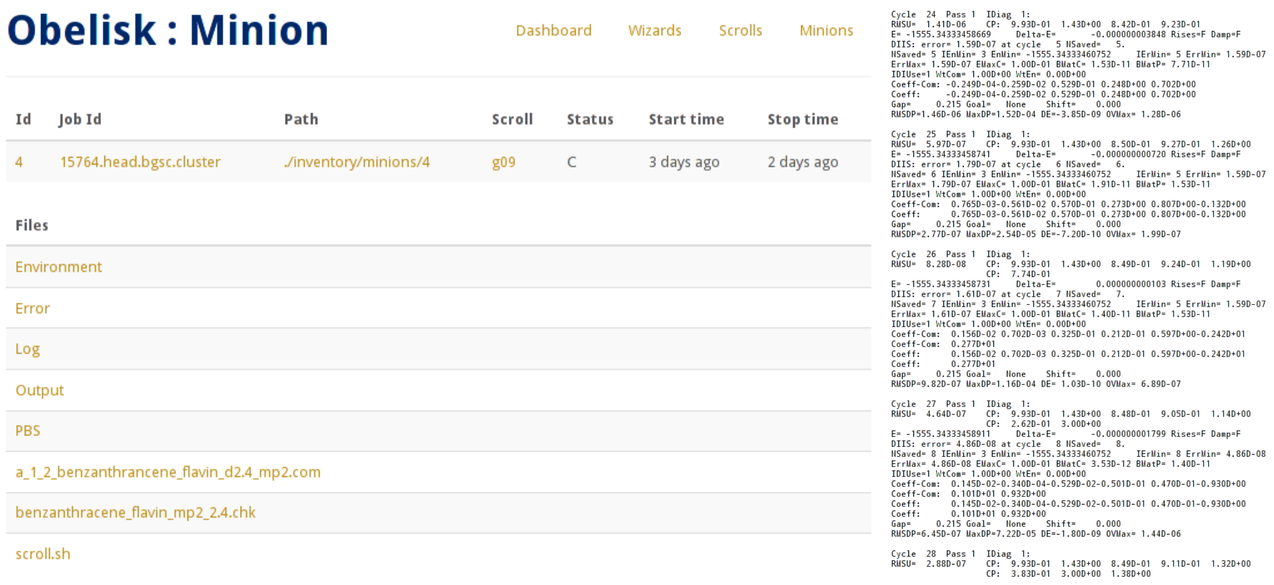
To test Obelisk, we took applications that are typically ran by faculty and students on our local HPC cluster, and converted them into Obelisk `Wizards`. By doing so, we were able to verify that Obelisk is applicable to their computational needs and works properly. One of these programs is Gaussian, a computational chemistry software suite for electronic structure modeling [3] that is used both for research and for teaching at our university. The only step required in creating a `Wizard` for Gaussian is to create a `YAML` file outlining its parameters and dependencies.

```
icon:   'flask'
elements:
    -   name:          Name
        type:          string
        value:         g09
        description:   Name of scroll

    -   name:          Command
        type:          string
        value:         g09
        description:   Command to execute

    -   name:          File
        type:          file
        value:
        description:   Com File

formula: '{Command} < {File} > Log'
```

Listing 1: Gaussian YAML

The `gaussian.yaml` can be seen in Listing 1. As can be seen, the Gaussian `Wizard` allows for three `Elements` to be defined by the user: `Name`, `Command`, and `File`. The first two are string types which correspond to the name of the `Scroll` to transcribe and the command to execute when a `Minion` is summoned. In the `HTML` form that is generated by the `WizardHandler` for this application, these fields will take the form of standard input text fields. The last `Element` is the input `COM` file required by the Gaussian application. Because it is marked with the `file` type, the `WizardHandler` will present the user will a button they can use to select a local file to upload to Obelisk. The last portion of the `YAML` file is the `formula`, which is the recipe used by the `WizardHandler` in Obelisk to transcribe the `Elements` into a `Scroll`.

After saving the `gaussian.yaml` file, we simply need to restart Obelisk to load the new `Wizard` template. Afterwards, we can open the `Wizard` web page and fill in the fields. For this `Wizard`, we only need to upload a COM file for input into Gaussian. There are of course other parameters for the Gaussian program, it is by no means a limited piece of engineering. However, for the sake of simplicity, we will use a limited set of options. After completing the form, we can summon a `Minion` to work on the `Scroll`. The `Minion` will eventually finish the `Scroll`, leaving files containing the results of the job in its workspace. When this has occured we can sift through the results and discover the next breakthrough in chemistry.



Id	Job Id	Path	Scroll	Status	Start time	Stop time
4	15764.head.bgsc.cluster	./inventory/minions/4	g09	C	3 days ago	2 days ago

Files
Environment
Error
Log
Output
PBS
a_1_2_benzanthrancene_flavin_d2.4_mp2.com
benzanthracene_flavin_mp2_2.4.chk
scroll.sh

```
Cycle  24  Pass 1  IDiag  1:
RMSU=  1.41D-06    CP:  9.93D-01  1.43D+00  8.42D-01  9.23D-01
E= -1555.34333458669     Delta-E=       -0.000000003848 Rises=F Damp=F
DIIS: error= 1.59D-07 at cycle   5 NSaved=   5.
NSaved= 5 IEnMin= 3 EnMin= -1555.34333460752     IErMin= 5 ErrMin= 1.59D-07
ErrMax= 1.59D-07 EMaxC= 1.00D-01 BMatC= 1.53D-11 BMatP= 7.71D-11
IDIUse=1 WtCom= 1.00D+00 WtEn= 0.00D+00
Coeff-Com: -0.249D-04-0.259D-02 0.529D-01 0.248D+00 0.702D+00
Coeff:     -0.249D-04-0.259D-02 0.529D-01 0.248D+00 0.702D+00
Gap=     0.215 Goal=   None    Shift=    0.000
RMSDP=1.46D-06 MaxDP=1.52D-04 DE=-3.85D-09 OVMax= 1.28D-06

Cycle  25  Pass 1  IDiag  1:
RMSU=  5.97D-07    CP:  9.93D-01  1.43D+00  8.50D-01  9.27D-01  1.26D+00
E= -1555.34333458741     Delta-E=       -0.000000000720 Rises=F Damp=F
DIIS: error= 1.79D-07 at cycle   6 NSaved=   6.
NSaved= 6 IEnMin= 3 EnMin= -1555.34333460752     IErMin= 5 ErrMin= 1.59D-07
ErrMax= 1.79D-07 EMaxC= 1.00D-01 BMatC= 1.91D-11 BMatP= 1.53D-11
IDIUse=1 WtCom= 1.00D+00 WtEn= 0.00D+00
Coeff-Com:  0.765D-03-0.561D-02 0.570D-01 0.273D+00 0.807D+00-0.132D+00
Coeff:      0.765D-03-0.561D-02 0.570D-01 0.273D+00 0.807D+00-0.132D+00
Gap=     0.215 Goal=   None    Shift=    0.000
RMSDP=2.77D-07 MaxDP=2.54D-05 DE=-7.20D-10 OVMax= 1.99D-07

Cycle  26  Pass 1  IDiag  1:
RMSU=  8.28D-08    CP:  9.93D-01  1.43D+00  8.49D-01  9.24D-01  1.19D+00
                   CP:  7.74D-01
E= -1555.34333458731     Delta-E=        0.000000000103 Rises=F Damp=F
DIIS: error= 1.61D-07 at cycle   7 NSaved=   7.
NSaved= 7 IEnMin= 3 EnMin= -1555.34333460752     IErMin= 5 ErrMin= 1.59D-07
ErrMax= 1.61D-07 EMaxC= 1.00D-01 BMatC= 1.40D-11 BMatP= 1.53D-11
IDIUse=1 WtCom= 1.00D+00 WtEn= 0.00D+00
Coeff-Com:  0.156D-02 0.702D-03 0.325D-01 0.212D-01 0.597D+00-0.242D+01
Coeff-Com:  0.277D+01
Coeff:      0.156D-02 0.702D-03 0.325D-01 0.212D-01 0.597D+00-0.242D+01
Coeff:      0.277D+01
Gap=     0.215 Goal=   None    Shift=    0.000
RMSDP=9.82D-07 MaxDP=1.16D-04 DE= 1.03D-10 OVMax= 6.89D-07

Cycle  27  Pass 1  IDiag  1:
RMSU=  4.64D-07    CP:  9.93D-01  1.43D+00  8.48D-01  9.05D-01  1.14D+00
                   CP:  2.62D-01  3.00D+00
E= -1555.34333458911     Delta-E=       -0.000000001799 Rises=F Damp=F
DIIS: error= 4.86D-08 at cycle   8 NSaved=   8.
NSaved= 8 IEnMin= 3 EnMin= -1555.34333460752     IErMin= 8 ErrMin= 4.86D-08
ErrMax= 4.86D-08 EMaxC= 1.00D-01 BMatC= 3.53D-12 BMatP= 1.40D-11
IDIUse=1 WtCom= 1.00D+00 WtEn= 0.00D+00
Coeff-Com:  0.145D-02-0.340D-04-0.529D-02-0.501D-01 0.470D-01-0.930D+00
Coeff-Com:  0.101D+01 0.932D+00
Coeff:      0.145D-02-0.340D-04-0.529D-02-0.501D-01 0.470D-01-0.930D+00
Coeff:      0.101D+01 0.932D+00
Gap=     0.215 Goal=   None    Shift=    0.000
RMSDP=6.45D-07 MaxDP=7.22D-05 DE=-1.80D-09 OVMax= 1.44D-06

Cycle  28  Pass 1  IDiag  1:
RMSU=  2.88D-07    CP:  9.93D-01  1.43D+00  8.49D-01  9.11D-01  1.32D+00
                   CP:  3.83D-01  3.00D+00  1.38D+00
```

Figure 9: Gaussian Minion and Output Log

Some of this output can be seen in Figure 9. On the left, we have the `Minion` web page which lists the `Minion` id, Torque job id, project workspace, the executed `Scroll`, the status of the `Minion`, and the start and stop times for the job. Below this, we have a listing of the files associated with the `Minion`, including the resulting `Log` file of which a portion is shown on the right. Altogether this example serves to show how simple and straightforward it is for users to create `Wizards`, which in turn can be used to transcribe `Scrolls`, which are later executed by summoned `Minions`, who carry out the commands of the user on high performance computing systems.

As we have demonstrated, using Obelisk is relatively straightforward and easy. There are very few barriers inhibiting researchers and students from obtaining the results they need. By giving the user complete control over their `Wizards`, Obelisk provides versatility, while the web based interface of Obelisk removes the painstaking tedium involved with manually operating `Torque` from the Unix command line. Obelisk abstracts this complexity from the user via an intuitive and easy-to-user web application and empowers them to harness armies of `Minions` to do their bidding.

5 Related Work

Our project is similar to two projects from MICS 2014: “A Web Portal For An Animation Render Farm” [1] and “On Ramp to Parallel Computing” [2]. Obelisk is different from the former project in that it does not target a specific application, but rather is a general purpose framework for different types of scientific software. Unlike the latter project, Obelisk provides wizards for specific scientific applications rather than giving the user a web-based programming environment. In some sense, Obelisk is a middle ground between the two projects, providing an flexible and extensible but easy-to-use abstraction for HPC systems.

As evident by the previous work mentioned, this overall approach to constructing web interfaces to scientific applications or high performance computing systems is an ongoing trend in the scientific computing community. Ian Foster notes that scientific tools tend to be difficult to deploy, configure, and utilize effectively and that an alternative to forcing users to become distributed system experts is to provide different service abstractions and in particular web service applications [6]. For instance, XSEDE has a large listing of such ”Science Gateways”, which provide simplified online access to a variety of HPC and scientific resources around the United States [7]. Our project follows along in this trend by providing a simple but effective general purpose web service for summoning minions on HPC systems.

6 Conclusion

It is our belief that Obelisk helps remove some of the technical barriers associated with scientific computing on HPC clusters. To abstract the complexity of harnessing these vast amounts of computing resources into a simple but effective interface, we developed Obelisk as an intuitive web interface to traditional HPC systems. As demonstrated in our paper, the design of Obelisk is relatively straightforward, while the implementation provides all the functionality required to describe, submit, execute, and monitor scientific computational tasks on a high performance computing cluster. By utilizing Obelisk, countless hours can be saved in the classroom by reducing the time spent teaching groups of students how to use Linux and its utilities. In the future, we hope to create a library of programs that can easily be installed on Obelisk and get wider user testing to fine-tune the user experience.

References

[1] John Rankin, Travis Boettcher, and Peter Bui. “A Web Portal For An Animation Render Farm”. Midwest Instruction and Computing Symposium (MICS2014). Verona, WI. April, 2014.

[2] Zackory Erickson and Samantha Foley. “On Ramp to Parallel Computing”. Midwest Instruction and Computing Symposium (MICS2014). Verona, WI. April, 2014.

[3] Gaussian. ”Official Gaussian Website”. http://www.gaussian.com/. 2015.

[4] Tornado. ”Tornado Web Server”. http://www.tornadoweb.org/en/stable/. 2015.

[5] C. Pautasso, O. Zimmermann, and F. Leymann. ”Restful web services vs. ”big” web services: Making the right architectural decision”. In Proceedings of the 17th International Conference on World Wide Web, WWW 08, pages 805814, New York, NY, USA, 2008. ACM.

[6] I. Foster. Service-oriented science. Science, 308(5723):814817, May 2005.

[7] Extreme Science and Engineering Discovery Environment. ”XSEDE Science Gateways”. https://www.xsede.org/gateways-overview. 2015.