2018 IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC)

Jupyter Notebooks and User-Friendly HPC Access

Ben Glick* and Jens Mache†,
Department of Mathematical Sciences, Lewis & Clark College
Portland, Oregon
Email: *glick@lclark.edu, †jmache@lclark.edu,

***Abstract*—In the modern world, having at least a basic understanding of High Performance Computing is critical even for undergraduate students. In this project, we describe our extensible ecosystem of tools which has lowered the access barrier to HPC systems for less experienced users. Our ecosystem of tools serves as the "one-stop" interface which allows users to focus on their research and teaching without remembering a lot of commands and syntax, or struggling with details like job submission or file systems. This environment is specially tailored for education. We provide a means for teachers to easily provide students access to information and assignments, and give students their own personal HPC sandbox, including access to a fully-functional, personalized Jupyter Notebook server.**

I. INTRODUCTION

High Performance Computing (HPC) is becoming more and more important in many domains. However, HPC can be complex and hard to use, especially for people from domains other than Computer Science. One way colleges and universities provide access to this vital resource to faculty, staff, and students is through operation of campus cluster systems. Many campuses have dedicated scientific computational resources, and ensuring that those resources are secure and accessible is important. One challenge with high performance clusters is ensuring that researchers without a systems administration background are able to easily and effectively use the high-performance clusters. Often, tasks submitted to these systems are controlled by resource managers. While resource managers offer a high degree of control over job submission, they are often hard to use. This paper introduces our ecosystem of tools which serves as the "one-stop" interface which allows users to focus on their research without remembering a lot of commands and syntax, or struggling with details like job submission or file systems.

The organization of the rest of this paper is as follows. Section II has related work, and Section III describes the technologies used. Section IV discusses security, usability and extensibility, performance, scalability, and administrative overhead. Section V describes educational impact. Future work and conclusions are Section VI and VII, respectively.

II. RELATED WORK

Wheras related work about web-based access to HPC exists [1] [2] [3] [4] [5] [6] [7] [8] [9] [10], not many support Jupyter Notebooks [11] [12], and other works have not presented a single unified, integrated environment in which users can design, implement, execute, analyze, and share their HPC tasks without changes in context or environment.

When designing and building our ecosystem, we considered a number of factors which had design and usability implications. Among those factors are security, usability, performance, and scalability. We feel that our system improves upon other systems in each of those categories. We believe that these factors all contribute to making our system easier to use and manage, as well as making potential users more likely to choose our system over others.

Our system is also different from other systems in its overall vision. Many of the other systems attempt to provide access to a single function or tool. The system designed by Kepner, et al. provided access to the Jupyter Notebook environment for its users, and was focused almost exclusively on allowing users to deploy web applications [11]. Our system allows users to deploy any arbitrary task or application. The system implemented by Misra et al. was designed almost exclusively for submission and to a lesser extent, monitoring of HPC jobs over the web [10]. UNICORE, the system presented in Benedyczak1 et al., supports execution of workflows, but not design [9]. Our system is designed to provide a completely integrated experience, where users do not need to interact with any other system for the duration of the time they are designing, implementing, executing, analyzing, and sharing their workflows.

III. TECHNOLOGIES

We found that our users' largest challenges with HPC systems were that many preferred web-based access over the command line. They, on the whole, did not feel spending time on correctly specifying all of the required syntax and details was valuable, instead preferring to focus on the science. They also found that spending time learning to use a resource manager was not useful. Our response was to build a system of tools our users could interact with instead of directly addressing our HPC system through the command line. We have decreased the amount of required systems administration skill necessary to run tasks on our cluster, and at the same time, we have decreased the amount of time and effort necessary to manage and operate the system, because users of our system require less training and oversight than with traditional cluster interaction layers.

Our cluster has a relatively straightforward architecture. Users interact with a login node over SSH, and have terminal-based access to the resource manager, the worker-accessible filesystem, and their private home directories. The workers are connected to the login node through a 10-gigabit per second Ethernet switch. The login node's filesystem is mounted to the

978-1-7281-0190-3/18/$31.00 ©2018 IEEE
DOI 10.1109/EduHPC.2018.00005

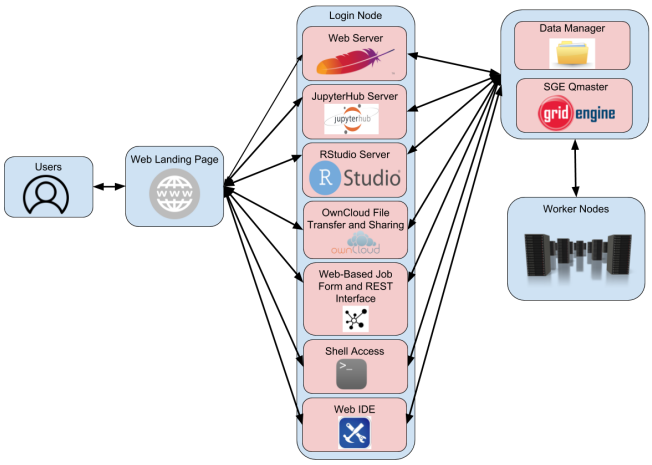

Fig. 1. **Architecture diagram of our ecosystem exhibiting the "One-Stop-Shopping" model of interaction where users only work with with one module**

worker nodes by the network filesystem (NFS) protocol. Work is scheduled to workers by Univa GridEngine[13], which has a qmaster running on the login nodes and execution daemons running on each worker. We have built tools on top of this architecture with which users can avoid terminal-only access.

We call the interaction model we have implemented "one-stop shopping." This means the user should never need to go to more than one place in order to get started with our system. Everything from launching large jobs to requesting personal support from an administrator happens in the same app. The user should never need to access our cluster in a way they find uncomfortable or overwhelming. A diagram of the components that make up this model can be seen in Fig. 1. We have configured and run a specific set of tools, including Jupyter notebooks [14], web-based integrated development environments (IDEs), and file management systems that we believe will benefit our users, and because our system is modular, we will be easily able to extend the system by adding more tools in the future.

A. User-Facing Infrastructure

For a user, the one-stop shopping model drastically simplifies the process of interacting with our cluster. Instead of attempting to either run all of their needed programs over the command line or developing programs on their computers before uploading to the HPC system, users are able to design, develop, and run complete workflows without any context switching. Switching from working on a local machine such as a laptop to working on a high-performance system is one of the most time-consuming parts of computational science. Users need to understand how programs run on their personal computers differ when run on high performance systems, how to upload and download data files and executables to and from the system, and how to make changes to their code on the remote systems. For many computational scientists, these tasks are unintuitive and uninteresting. In our system, users only need to know how to use a web browser in order to design, implement, and run full-scale workflows on a high-performance system.

When a user visits our system's landing page, they are prompted with a familiar web interface which contains links to many useful services, all of which are accessible over a web browser. The web interface has an awareness of what it is able to do, and advertises this to other tools in the ecosystem through the interface we have designed for it. This allows different parts of the ecosystem to seamlessly interact. For example, a script a user has written in the web IDE component can be run on worker nodes through the web-based job submission form very easily. The goal of this model is that users never need to go anywhere other than our web-facing infrastructure for any interaction with the system of any kind. After tasks are completed, users are optionally notified via email that their output is available and where it is available. Additionally, in some cases, their output can be directly attached to the email notifying them of job completion. This allows for users to "fire and forget" their HPC jobs. The email notification is optional so that people who are programmatically launching large numbers of tasks aren't overwhelmed by hundreds or thousands of emails. The ecosystem simplifies the act of accessing an HPC system for users who may not feel comfortable working with such machines.

Jupyter[14] [15] notebooks are a commonly used platform for designing, developing, executing, and communicating code. Notebooks documents, internally represented in JSON, produced by the Jupyter Notebook App, which contain both computer code and display elements, including equations, descriptive text, and images. Notebooks can be both human-readable documents containing the description and the results of code, as well as executable documents which can be run to execute arbitrary computer code. The Jupyter Notebook App is a server-client application that allows editing and running notebook documents via a web browser. The Jupyter Notebook App can be installed on a remote server and accessed through the Internet. We have set up an instance of the Jupyter Notebook App as a server allowing users to run Jupyter notebooks directly on our HPC system[16].

We have tried to provide as many options as possible in order to fill the needs of many different types of users, from people most comfortable with the traditional UNIX command line interface to users who prefer Jupyter Notebooks as an execution model, to people who are most comfortable submitting tasks over a REST[17] interface or web form. We have provided an interface which allows for many different interactive, user-facing components, but which also sits on top of a unified execution interface to ensure secure, performant, and scalable execution of HPC tasks. From our landing page, we want our users to feel comfortable not only using the tools we have provided in our system, but also in requesting assistance and other personal communication from our administration team. To this end, we have a section of the website

from which users are encouraged to reach out to our support team for assistance. Though this section is not implemented as a module or set of modules in our system, it does fit with the overall design philosophy of our ecosystem, as users are encouraged to ask for help without going to a place than the one they would use for other interaction with the system.

The tools we decided would go the farthest towards allowing all of our users to interact with only one outward-facing section of our HPC system were chosen because we identified the challenges they solve as being common problems for our users. One of them is the JupyterHub Notebook server[18]. Jupyter notebooks[14] attempt to capture the whole computation process, including developing, documenting, and executing code, as well as communicating the results. Jupyter notebooks are becoming extremely common in computational science, with more and more scientists preferring the Jupyter environment as a means of carrying out simulations and other computational science work. Our system provides access to Jupyter notebooks which execute computationally intense work on the worker nodes, while still running the JupyterHub process on the login node.

Another common challenge we found among our users was the issue of transferring data on and off of HPC environments. Many of our users did not feel comfortable using command line clients (scp, sftp, rsync, etc.) to transfer files on and off of the system, so we use an open-source, web-based frontend which would provide our users with an interface similar to commercial cloud storage options like Google Drive. This tool is called OwnCloud[19], and we were able to write a wrapper for it to allow seamless drag-and-drop file uploading and downloading. Any files that are created by an HPC job that a user runs are automatically represented in the OwnCloud environment and can easily be downloaded to their personal computers.

We also found that in many cases, users had scripts or executables which they had either already written or had been given to them in a form that would be difficult to move to a Jupyter Notebook. Other users may simply not enjoy using the Jupyter interface. These users also often did not feel comfortable launching jobs directly from the command line, and our response to this common user challenge was to design and build an easily accessible web-based frontend for job submission and management on our HPC system. We implemented a secure, easy to use, web-based access manager designed to make interacting with our computational system easy. Currently, the web-based component uses Python's Flask library[20] and the high-performance interactions are handled by the job submission library libsubmit[21]. The system has a web-based GUI form as well as a RESTful[17] interface for automated job submission. The web form is simple and familiar to users, while the REST interface is easily extensible and can be wrapped to allow arbitrary components of our ecosystem to submit HPC tasks, request status of HPC tasks, and cancel unwanted HPC tasks.

B. Implementation Details

There are three main layers that make up the ecosystem we have designed. Those three layers are (1) the most user-facing, interactive layer, (2) the layer composed of various tools for executing HPC jobs, and (3) the layer that ensures that jobs will always execute in an expected, understandable, and, performant way and also with a simple, unified interface which ensures that it is easy to write expansion modules in the future. We have designed and implemented the first and third layers described here in order to allow people to interact with the tools that make up the second layer easily and securely. A diagram of the interaction and the makeup of these layers is provided in Fig. 2.

Within our system, the second layer, which is composed of the tools for executing HPC jobs, is represented as a modular list of tools which run from the login node but are able to interact with the worker nodes without any user interaction. In order to accomplish this, we have designed an interface comprised of verbs that the underlying third layer must be able to carry out upon request from the second layer. Because of this interface, it is very easy to add new tools to our list of modules, which makes it easy to incorporate user feedback into the system quickly and adopt new features and tools into the ecosystem. The third layer, which the tools sit on top of, is the layer responsible for delegating the tasks launched by users to the worker nodes. This layer, which is comprised of three main components, is responsible for balancing load on the login node, delegating work to worker nodes when possible, and ensuring that user-requested tasks are performed as efficiently as possible. After the tool's wrapper is added, the ecosystem is able to communicate what the new tool allows it to do with the user and the other tools. Users then are able to use the tool and the tool is able to push data and commands to other constituent tools.

The three aforementioned component modules are the execution manager, the filesystem manager, and the external data manager. The interface, which the tools in layer two must conform to in order to be modules in the ecosystem, ensures that all tools which are to become modules in our system are able to interact with each one of these components. However, some modules may not make use of all of the components. Before a tool is added to the system, a "wrapper" layer needs to be written for it. This wrapper layer allows the tool to interact with the rest of the ecosystem and ensures that it follows the interfaces provided by the ecosystem.

An example of this is the RStudioServer [22] module, which needs access to the execution and filesystem managers, but does not support file staging from external sources. Cases like this are handled by the interface returning that the requested functionality has not been implemented. We have implemented a "wrapper" interface which interacts with RStudioServer, waiting for a user to request that an R script is run before wrapping that R script in a GridEngine submission script. It then will push all output from the R script back into RStudioServer to display to the user. In the future, we plan

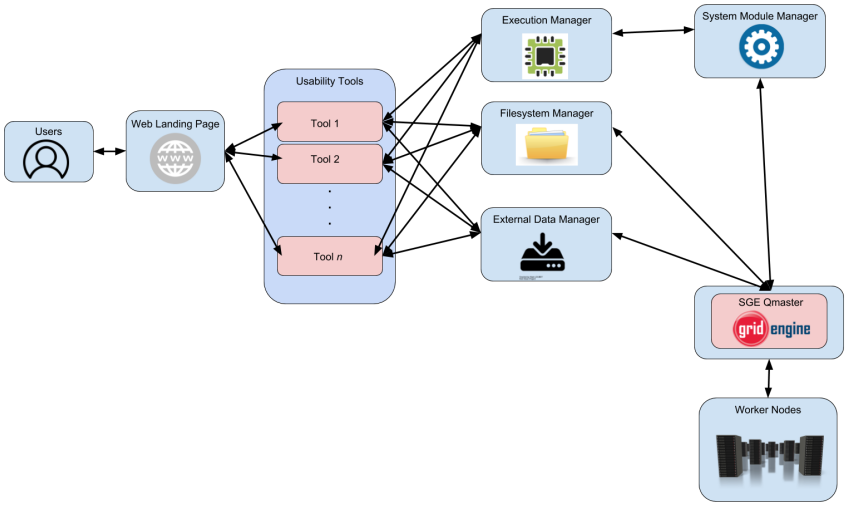

Fig. 2. **Diagram of the makeup of different interaction layers in our ecosystem**

to reconfigure this so that RStudioServer tasks will request interactive GridEngine[13] jobs so that users have no waiting time when their job has run but has not reported output yet.

An example of a tool that needs to use all three components of the interface is our Jupyter Notebook server. Jupyter Notebooks[16] are a powerful and popular tool used by many different types of scientists for a wide variety of tasks from presenting work for publication and teaching to heavy-duty high-performance computing tasks. Many scientists feel more comfortable working with computation through the Jupyter Notebook than through traditional file-based scripts and programs.

Jupyter Notebooks, however, are natively limited to running computational tasks, represented as runnable ”cells” of code, on the computer which it is running on. Through its IPyParallel[23] API, Jupyter Notebooks provide a way to create a virtual cluster on which Jupyter cells can be run, but natively, these cells must be on the same physical machine as the Jupyter kernel is running on, which is usually a login node or web-facing server. Jupyter needs to interact with all three of the pieces of our interface, because it needs to be able to write files to disks which are accessible from the worker nodes, it needs to be able to pull data in from the external Internet, and it needs to be able to execute all tasks on worker nodes so as to not place excessive stress on the login node.

As mentioned in the User-Facing Infrastructure section, we decided to run the JupyterHub[18] server directly on the login node rather than on the worker nodes. Previously, similar systems, such as the one described by [Kepner, et. al 2017], have run individual Jupyter notebook servers as required on a per-user basis, where single-user Jupyter servers are started as an interactive HPC job and once they are started up, the user is given a link to access their specific notebook environment [11]. This forces users to wait for the system to start up their notebook, and it also presents a confusing user flow which they must follow. In our system, a user can login to our JupyterHub server whenever they want with no wait time and confidence that the Jupyter Notebook[15] files they have previously used or created are still there, in a place where they can be accessed easily. Our system also removes the complexity of running scripts which impersonate the user and log in to the worker nodes, where they start up a single-user Jupyter notebook server[16]. The lack of this step makes it both easier for users to interact with and also easier to ensure that files users may need access to are easily accessible from the Jupyter environment. Our design prioritizes user-facing simplicity with only a slight penalty of increased load on the login node.

For the specific case of Jupyter with Python as a backend, we have employed the Parsl Parallel Scripting Library [24] in order to submit tasks directly to the Univa GridEngine[13] resource manager. We have designed and implemented a Jupyter Notebook Extension which we have enabled by default on our system. This notebook extension takes python code and automatically annotates it and prepares it to be run on worker nodes through parsl. As part of our interface, there is an instance of the python library libsubmit[21], which assists in the abstraction of scheduler commands and provides a simple, easy interface for submitting, canceling, and requesting status of resource manager defined tasks [24]. In order to be Parsl-compliant, there are some slight restrictions on the type of code that can be run through our ecosystem, but most python code which is written is compatible with no changes.

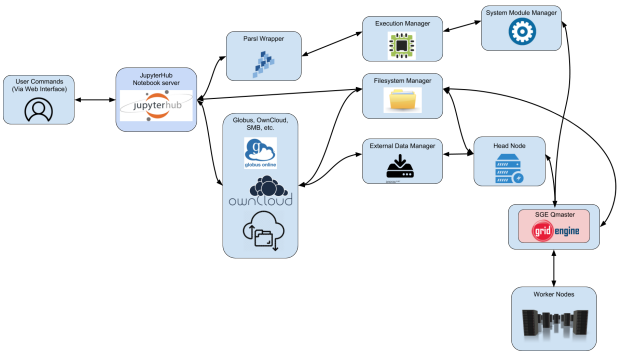

Fig. 3. **Architecture diagram of the JupyterHub notebook server's interactions in our ecosystem.**

Libsubmit uses the concept of an "execution provider" in order to provide a uniform interface for submitting arbitrary jobs to various different types of execution environments, includng clouds, resource managers, and local, thread-based task execution[25]. As part of this project, we wrote a Libsubmit[21] execution provider for Univa GridEngine. A diagram of the way our Jupyter server interacts with the rest of our ecosystem is in Fig. 3.

For other tools, there are different considerations which need to be made. As mentioned earlier, there are some tools which only need to interact with one or two pieces of our cluster abstraction interface. One of these such tools is our file transfer manager. As a front-end, we use the open-source project called OwnCloud[19]. OwnCloud is a cloud file storage system that provides users with an interface similar to that of proprietary tools like Google Drive. Because the interface and experience is familiar to users, they are comfortable using it to transfer files on and off our system.

OwnCloud on its own is a powerful tool for allowing users to share, sync, and transfer files in the cloud, but it does not provide all of the functionality we need in order for a completely integrated job execution environment. It lacks proper authentication, file ownership, and system interaction without our interface underneath it.

Essentially, the only functionality provided "out of the box" with OwnCloud[26] is its web interface. The web interface is designed to store all data in a specified directory, which is an understandable design choice if most users are not expected to touch the data other than from the OwnCloud front-end, but for our use case, this is not a safe assumption to make. Users of our system will almost always want to use the files they have just uploaded in a HPC job, so they will need to know where it is stored and have proper permissions on the files.

Because OwnCloud cannot fill this requirement natively, we have attached it to the Filesystem Manager of our interface, which is able to translate the file structure that OwnCloud creates into something more useful for HPC file transfers. The filesystem manager is able to track what files are uploaded by which OwnCloud users and is able to use its index to translate OwnCloud users to system users and automatically move files to an easily accessible location within the user in question's home directory and change the file's ownership so that the user will have access to their own files, even though OwnCloud automatically creates the files as the user that runs the web server.

IV. DISCUSSION

A. Security

When designing systems which run on high performance, multi-user, production-level systems, security is an extremely important consideration. High-performance systems are often both public-facing and highly publicized. This, as well as the extremely high potential usability and power to a hacker, makes them common targets for malicious cyberattacks. A hacker using a high-performance system as a botnet could cause a large amount of damage. Additionally, without security, people are not likely to use the system, and if they do use the system, it is likely that their data will be compromised.

We have taken a multi-tiered approach to security in our ecosystem. The first tier is to attempt to keep unwanted users out of the ecosystem entirely, and the second tier is to make sure that even if an attacker can access the ecosystem, they will still be locked out of any important functions and tools. The third tier ensures that users only have access to the specific data they own or have been given rights to by its owner. This is important because some research projects require access to sensitive data, and that data must be carefully secured from users not on the project. We support both the traditional command line interface for changing file permissions, as well as a web-based GUI for sharing files to users and groups, and changing read, write, and execute permissions on the files.

In order to completely lock unwanted malicious users out of the system, we have isolated the system from the public Internet behind a virtual private network (VPN). The VPN ensures that only users registered to the system can even find the system on the Internet, as well as encrypting all web traffic to and from the system. The system is still able to make outgoing requests, but will not receive any incoming requests unless they come from within the VPN. The effect of this is that our cluster is not even visible to potentially malicious users. We also enforce that all HTTP based traffic to and from our system is encrypted with TLS. This ensures that even if packets are intercepted and saved by a hacker, they are essentially unreadable. This is especially important because many of our tools require transmission of passwords over a network connection, and this ensures that the passwords sent over the network will remain unreadable except by our system on the other end. These two design choices ensure that the requirements laid out in the first tier of our security are met.

We have used the pluggable authentication module (PAM) for all of our tools. In some cases, where the tool's built-in security was not sufficient, we wrote a PAM authenticator as a part of its wrapper. Our user-facing interface requires that the user be able to authenticate to the tool they wish to use with

their system username and password. PAM allows for this to happen securely and easily, and it is usually easy to modify a tool so that it uses PAM authentication in the event that it does not natively support PAM. This satisfies the requirements in the second tier of our security design.

As explained more thoroughly in the implementation details section, whenever files are written or read by user-facing components of our ecosystem, they must go through the filesystem manager. This ensures that file permissions are always set such that only authorized users have access. This allows the system to automatically ensure that files have their permissions set as users wish. The filesystem manager's capabilities ensure that the third tier of our security system is properly implemented.

B. Usability and Extensibility

Usability is by far the most important consideration we made when designing our ecosystem. We are based at a small liberal arts college and many of our users have no experience with UNIX or HPC systems. We want to make our system as easy to use for them as possible, while still maintaining the ability to carry out high-performance tasks efficiently and quickly. We want our system to provide enough fine-grained control that the most advanced users of our system may customize their workflows in any way they wish, while also providing a simple enough interface that our first time users who may have never used a command line interface before are able to submit jobs to our system easily and without much outside assistance.

Because we are based at a small liberal arts college and this is the college's first dedicated HPC system, one of the main challenges we anticipate is simply getting people to use our system. We believe that people—both students and professionals—from many disciplines, including humanities, physical sciences, life sciences, and social sciences, would benefit greatly from using our HPC system with their education and research, but we also understand that many people from disciplines outside of computer science will feel uncomfortable using a system like ours because it may seem foreign, confusing, and difficult to understand. The ecosystem we have built helps to solve this problem by presenting a friendly, easy to use, familiar interface to a system which many of our users are initially uncomfortable with. We hope our users' experiences with our ecosystem will make them want to continue using more HPC in their future work, be it through teaching, research, or just for fun.

Our ecosystem has already been used for multiple computational science projects, both research and teaching based, and our users, both professional researchers and students, have found our ecosystem to be easier to use than they expected. Additionally, more components of the system have been developed in response to requests from and in collaboration with our users. Through the method of allowing users to help design the environment which they will be using, we have been able to provide an environment in which users feel comfortable. Users have a simple, intuitive workflow which allows them to design, implement, execute, analyze, and share their HPC tasks easily. This user flow is shown in Fig. 4. The end goal of this project is to provide a single unified environment from which all of our users' HPC needs are filled, regardless of the scale of the task they are trying to accomplish, their level of experience with HPC systems, or their familiarity with UNIX and command line interfaces. We believe we have taken a step towards this goal by providing users with an ecosystem designed to offer them enough options that they can compose, execute, analyze, and publish their HPC workflows, as well as requesting personalized assistance when necessary, all without ever leaving the web-based frontend of our ecosystem. One of the most important and popular tools we have made available to users are Jupyter notebooks. Our Jupyter notebook installation is simple and easy to use. As an example, a code sample which calculates pi via monte carlo simulation through our Jupyter notebook interface is in Fig. 5.

Along with usability goes extensibility, because many of the things that make our ecosystem more usable than traditional HPC environments are the third-party tools in the user-facing layer of our ecosystem. The modular interface we have designed, which any tool added to our ecosystem must conform to, is written so that simple wrappers can be easily written around tools so that any tool a user requests can be quickly and easily installed on our system. This interface consists of a set of commands that the ecosystem can give to any subset of the tools on the system simultaneously. Some of those commands are starting and stopping the tools, turning on and off user-facing parts of the tools, executing HPC jobs through a resource manager, writing to and reading from files, and connecting and disconnecting from worker nodes. The result of having a clearly defined, simple interface is that adding new usability tools to the system is easy, and the system can grow as its users need.

C. Performance

HPC jobs, by definition, require a highly performant system on which to run. If our ecosystem increased usability but reduced performance to the point that many of our users' jobs would not properly run through our ecosystem, it would be almost completely useless. The system needs to maintain its performance enough that HPC jobs can run without a large performance penalty.

Our ecosystem slightly impacts performance in terms of memory usage as well as CPU usage. At idle, our ecosystem in total uses about 1.5 gigabytes of memory. As users log in, slightly more memory is used up. With roughly 25 users logged in and using the ecosystem, this went up to about 2 gigabytes. At one time, 2 gigabytes of memory was a large amount, but now, programs as common as web browsers can take up well over 2 GB of memory on people's personal computers. Our system's login node has over 100 GB of memory, so even with hundreds of concurrent users, our ecosystem would be able to keep up with demand.

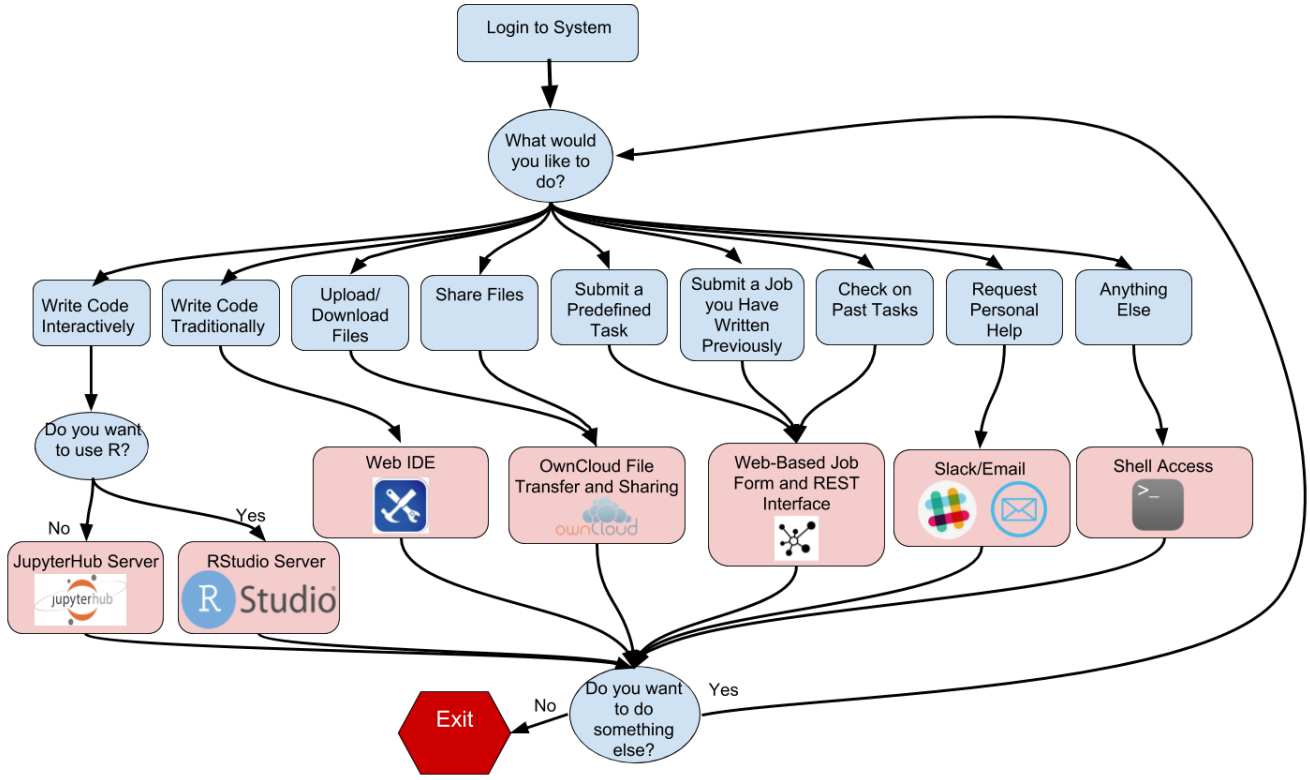

Fig. 4. **Flowchart summarizing the user workflow process**

```
In [ ]:
# Workflow defined here:

@App('python', dfk)
def pi(total):
    # App functions have to import modules they will use.
    import random
    # Set the size of the box (edge length) in which we drop random points
    edge_length = 10000
    center = edge_length / 2
    c2 = center ** 2
    count = 0

    for i in range(total):
        # Drop a random point in the box.
        x, y = random.randint(1, edge_length), random.randint(1, edge_length)
        # Count points within the circle
        if (x - center)**2 + (y - center)**2 < c2:
            count += 1

    return (count * 4 / total)


@App('python', dfk)
def avg_n(inputs=[]):
    return sum(inputs) / len(inputs)
```

```
In [ ]:
# Call the workflow:
sims = [pi(10**6) for i in range(10)]
avg_pi = avg_n([task.result() for task in sims])
```

```
In [ ]:
# Print the results
print("Average: {0:.63f}".format(avg_pi.result()))
```

Fig. 5. **Jupyter cell which calculates the decimal expansion of π by Mote Carlo simulation**

D. Scalability

Our ecosystem is also highly scalable. Though we do not have a larger cluster on which to install our ecosystem, we are confident that it would be easy to replicate our setup on a larger system with similar performance. If the cluster we were to install our ecosystem on has only one login node, as ours does, it would be essentially the same installation as we have on our cluster, and would be extremely simple to replicate. If the larger system in question has more than one login node, the only major change we would need to make is ensuring that an instance of each user-facing tool is installed and running on each login node, and ensuring that those instances of tools are able to communicate with each other through our interface. Though we have not done this, there is no reason that it couldn't be done, as our interface already supports network traffic.

E. Administrative Overhead

Most HPC systems have full-time systems administrators and developers whose job it is to keep the system running and to keep the users happy, as well as full-time researchers who spend most of their days using the system. Because our school is so small and does not have a large research staff, we also do not have a large HPC administration staff. Our cluster is managed by our school's library, and there are no full time employees dedicated completely to HPC operations. Instead, there is one library employee whose responsibilities include library collection management, research computation support, support for technology as a teaching tool, and now, HPC administration and management as well. There are also two students who work part time on those same topics.

Because of the limited amount of time we can spend on HPC administration, having a system which is easy to administer is very important. We have found that after the initial design of our ecosystem, the amount of time we have needed to spend managing our HPC system has been minimal. Users have found interacting with the system to be easy and intuitive, and as such, have not needed very much hands-on assistance with HPC operations. We have also found that our ecosystem is not hard to maintain, upgrade, and operate. Starting and stopping the whole system or various parts of the system can be controlled with a single command, and adding and maintaining various modules is fairly straightforward.

V. EDUCATIONAL IMPACT

A. Use in Courses

Our system has been operational since winter semester 2018, and has been used as a part of coursework for two courses, one in biology and one in computer science (Machine Learning), as well as several independent studies. In academic year 2018/ 2019, it will be used for at least three courses. The 200-level course "Computer and Network Security" plans to use it for password cracking.

We also plan to use our system at least once in our CS1 course, to experience the power of HPC. The CS1 course typically covers an exercise of computing π by Monte Carlo method. Consider a circle inscribed in a square. Uniformly scatter a given number of points over the square. Count the number of points inside the circle, i.e. having a distance from the center of less than the radius. The ratio of the inside-count and the total-sample-count is an estimate of the ratio of the two areas, $\pi * r^2/(2*r)^2 = \pi/4$. Multiply the result by 4 to estimate π.

The approximation accuracy improves as more points are placed. With a good random number generator and n points, accuracy improves by a factor of $\sqrt{n}$. Thus muliplying numbers of points by 100, improves accuracy by a factor of 10. The computation is embarassingly parallel. Fig. 5 shows a Jupyter notebook.

B. Education-Specific Features

There are a some education-specific features of our system which are designed to help teachers do things they would otherwise need to do manually. One of these features is a Jupyter Notebook Grading system. Jupyter notebooks lend themselves to be used in course quite nicely. Because of the cell-based nature of a notebook, it is easy to create an assignment We have designed a Jupyter notebook extension which allows teachers to assign a Jupyter notebook as either a test or project assignment to a number of students with the click of a button. Jupyter will then automatically make a copy of that assignment in the students' home directories, ensuring that only that student and the instructor have read access, and notes when the assignment is due. Then, when the assignment is due, it will automatically timestamp and copy the updated student assignment into a folder which is only readable by the teacher. The teacher can then grade the student assignments, before returning them to students' home directories, also with the click of a button.

Another one of these education-specific features also expands on Jupyter. We have created a "textbook" based entirely on Jupyter notebooks, which is intended to be used either as an interactive, self guided course or as supplemental material for a traditionally taught course [27]. The course interactively guides students through the basics of high-performance computing. We have prepared eight notebooks, each one corresponding to a topic we decided was important for a foundational understanding of high performance computing. We call each of these notebooks a chapter. The structure of each chapter is as follows: each chapter has a number of sections, and each section has at least one, but in many cases more, example of a real-world problem which is pertinent to the section, mostly written in python. These examples are all interactive and can be run and tweaked as needed by students. We surveyed students about how they felt this notebook-based textbook compared to traditional textbooks, and they found the interactive textbook more useful than a traditional textbook. One student said "I enjoyed the images and appreciated that the examples could be run, so users can change values and see how that effects the output", while another student said "I really liked having the cells throughout the chapter to explain and showcase the material from that section". These notebooks can easily be used as projects and assignments along with the features described in Sec. 5A.

VI. FUTURE WORK

In the future, we would like to continue to expand the ecosystem both by adding new tools to the system as well as adding entirely new components to the ecosystem. One of the entirely new components we would like to design is an automated publication pipeline, which would allow users to specify that they would like output data from certain types of jobs to be automatically published to external destinations including Globus[28] endpoints, Amazon S3 [29] buckets, personal web servers, Google Drive folders, or any other location users may want output data to go. One example of how this would be used is if a user had a web server which was showcasing their research with sample data, they could set up this system to automatically show their newest data on that website, updating every time they changed or re-ran the workflow producing that data. Another use for this output publishing system could be for a pipeline for an interactive demo. In this case, the user would have a public-facing web site which would automatically call a workflow on our cluster and then results would use the publication module to send data back to the person who had called the workflow, thus showing how HPC works interactively and in real-time.

Some of the smaller changes we would like to make in the future concern the web-based job submission form. First, we would like to change the UI and make the website somewhat better looking. We believe that though the website is functional and simple to understand now, it would be even easier to use if the UI was a bit cleaner. To that system, we would also like

to continue to expand the capabilities of the access manager to include automated external data staging and workflow optimization, as well as enhanced monitoring and interactivity of jobs. Additionally, we would like to expand our external data manager to support more external data sources such as Apache HDFS[30], as well as enabling users to mount remote filesystems directly onto our system in a safe, fast, and secure way.

Finally, we would also like to design a software layer which can differentiate jobs by what kinds of hardware they would benefit from. This would be useful for a number of different things. First, it would be useful because it would allow jobs that benefit from different types of accelerators to run on hardware with those accelerators available automatically. For example, if we know that a user is running a job which involves TensorFlow[31], we could suggest to them that their job should be run on a compute node with a GPU, while other software frameworks may be conducive to running on nodes with Intel Xeon Phi coprocessors, or may benefit from running on a node with just traditional CPUs.

Another thing this technology would be useful for is determining what kinds of queues to submit jobs to. For example, if we know that a job is mostly IO bound, we can then attempt to make sure that, if it is small enough, it will only run on one node, and if not, then we can ensure that the nodes it runs on are closely networked together. Additionally, we are interested in setting up a High-Throughput (HTC)[32] queue in our system, where we gain additional compute power by scavenging spare compute cycles from computers around campus which are often idle, similar to how HTCondor operates[33]. In order to effectively do this, we must have a way of differentiating what types of jobs should be run directly on the HPC system [34] and what jobs are lower priority or otherwise suitable to run on the HTC queue[35], while still coming from the same user-facing ecosystem we have described here.

VII. CONCLUSION

High Performance Computing (HPC) is becoming more and more important in many domains. However, HPC can be complex and hard to use, especially for people from domains other than Computer Science. We identified what our users' largest challenges with HPC systems were and found that they preferred web-based access over the command line. They, on the whole, did not feel spending time on correctly specifying all of the required syntax and details was valuable, instead preferring to focus on the science. We call the interaction model we have implemented "one-stop shopping." This means the user should never need to go to more than one place in order to get started with our system. While related work about web-based access to HPC exists, not many support Jupyter Notebooks[15], and other works have not presented a single unified, integrated environment in which users can design, implement, execute, analyze, and share their HPC tasks without changes in context [11][9][10]. We have provided this "one-stop shop" of useful tools and modules to our users with minimal performance overhead, and we have found that this ecosystem has reduced the amount of administration necessary on our system.

Our ecosystem has been tailored towards educational use cases. We have designed specific features for teachers to use with their students, in order to help optimize tasks which would otherwise be time-consuming. One example of such a feature is automated assignment and collection of project-style and test-style assignments to a group of students which ensures students cannot view each others' work and ensures that students lose edit-level access to their work after it is due. We have also demonstrated use of our ecosystem as a means to provide students with an interactive textbook, which they found to be more interesting and useful than a traditional textbook.

ACKNOWLEDGMENT

The authors would like to thank Chris Sullivan, Asistant Director of Biocomputing at Oregorn State University for his assistance with design and setup of the hardware of our system. The authors would also like to thank Jeremy McWilliams, Digital Services Coordinator at Lewis & Clark College for his support and assistance throughout this project.

REFERENCES

[1] M. Thomas, S. Mock, and J. Boisseau, "Development of web toolkits for computational science portals: The npaci hotpage," in *High-Performance Distributed Computing, 2000. Proceedings. The Ninth International Symposium on*. IEEE, 2000, pp. 308–309.

[2] G. Aloisio and M. Cafaro, "Web-based access to the grid using the grid resource broker portal," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1145–1160, 2002.

[3] G. Alosio, M. Cafaro, C. Kesselman, and R. Williams, "Web access to supercomputing," *Computing in Science & Engineering*, vol. 3, no. 6, pp. 66–72, 2001.

[4] G. Aloisio, M. Cafaro, R. Williams, and P. Messina, "A distributed web-based metacomputing environment," in *International Conference on High-Performance Computing and Networking*. Springer, 1997, pp. 480–486.

[5] I. Milne, D. Lindner, M. Bayer, D. Husmeier, G. McGuire, D. F. Marshall, and F. Wright, "Topali v2: a rich graphical interface for evolutionary analyses of multiple alignments on hpc clusters and multi-core desktops," *Bioinformatics*, vol. 25, no. 1, pp. 126–127, 2008.

[6] M. A. Miller, W. Pfeiffer, and T. Schwartz, "Creating the cipres science gateway for inference of large phylogenetic trees," in *Gateway Computing Environments Workshop (GCE), 2010*. Ieee, 2010, pp. 1–8.

[7] S. Cholia, D. Skinner, and J. Boverhof, "Newt: A restful service for building high performance computing web applications," in *Gateway Computing Environments Workshop (GCE), 2010*. IEEE, 2010, pp. 1–11.

[8] C. A. Atwood, R. C. Goebbert, J. A. Calahan, T. V. Hromadka, T. M. Proue, W. Monceaux, and J. Hirata, "Secure web-based access for productive supercomputing," *Computing in Science & Engineering*, vol. 18, no. 1, pp. 63–72, 2016.

[9] K. Benedyczak, B. Schuller, M. P.-E. Sayed, J. Rybicki, and R. Grunzke, "Unicore 7 — middleware services for distributed and federated computing," in *2016 International Conference on High Performance Computing Simulation (HPCS)*, July 2016, pp. 613–620.

[10] G. Misra, S. Agrawal, N. Kurkure, S. Pawar, and K. Mathur, "Chreme: A web based application execution tool for using hpc resources," in *International Conference on High Performance Computing, 2011 (HPC-UA2011)*, October 2011, pp. 19–25. [Online]. Available: http://hpc-ua.org/hpc-ua-11/files/proceedings/1.2(19).pdf

[11] A. Prout, W. Arcand, D. Bestor, B. Bergeron, C. Byun, V. Gadepally, M. Hubbell, M. Houle, M. Jones, P. Michaleas, L. Milechin, J. Mullen, A. Rosa, S. Samsi, A. Reuther, and J. Kepner, "MIT supercloud portal workspace: Enabling HPC web application deployment," *CoRR*, vol. abs/1707.05900, 2017. [Online]. Available: http://arxiv.org/abs/1707.05900

[12] M. Milligan, "Interactive hpc gateways with jupyter and jupyterhub," in *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*, ser. PEARC17. New York, NY, USA: ACM, 2017, pp. 63:1–63:4. [Online]. Available: http://doi.acm.org/10.1145/3093338.3104159

[13] W. G. S. Microsystems), "Sun grid engine: Towards creating a compute power grid," in *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, ser. CCGRID '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 35–. [Online]. Available: http://dl.acm.org/citation.cfm?id=560889.792378

[14] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, "Jupyter notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt, Eds. IOS Press, 2016, pp. 87 – 90.

[15] F. Pérez and B. E. Granger, "IPython: a system for interactive scientific computing," *Computing in Science and Engineering*, vol. 9, no. 3, pp. 21–29, May 2007. [Online]. Available: http://ipython.org

[16] J. Team, *What is the Jupyter Notebook?*, IPython, Inc., USA, 2017. [Online]. Available: http://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html

[17] R. Battle and E. Benson, "Bridging the semantic web and web 2.0 with representational state transfer (rest)," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 6, no. 1, pp. 61 – 69, 2008, semantic Web and Web 2.0. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570826807000510

[18] J. Team, *Jupyterhub: Multi-user server for Jupyter notebooks*, Ipython, 2017. [Online]. Available: https://github.com/jupyterhub/jupyterhub

[19] O. Team, *OwnCloud - The last cloud collaboration platform you'll ever need*, OwnCloud, Inc., 2017. [Online]. Available: https://owncloud.org

[20] A. Ronacher, *Flask Documentation*, Pocoo, 2017. [Online]. Available: http://flask.pocoo.org/docs/

[21] Y. N. Babuji, *Libsubmit Documentation*, University of Chicago, Chicago, USA, 2018. [Online]. Available: http://libsubmit.readthedocs.io

[22] RStudio Team, *RStudio: Integrated Development Environment for R*, RStudio, Inc., Boston, MA, 2015. [Online]. Available: http://www.rstudio.com/

[23] The IPython Development Team., "Using ipython for parallel computing," https://ipyparallel.readthedocs.io/en/latest/, 2015.

[24] Y. Babuji, A. Brizius, K. Chard, I. Foster, D. S. Katz, M. Wilde, and J. Wozniak, "Introducing Parsl: A Python Parallel Scripting Library," Aug. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.891533

[25] B. Glick, Y. Babuji, and K. Chard, "Scalable parallel scripting in the cloud," in *Proceedings of The International Conference for High Performance Computing, Networking, Storage, and Anaysis*, ser. SC '17. Denver, USA: IEEE Computer Society and ACM SIGHPC, 2017. [Online]. Available: https://doi.org/10.13140/RG.2.2.20048.81922

[26] A. Patawari, *Getting Started with ownCloud*. Packt Publishing, 2013.

[27] B. Glick and J. Mache, "Using jupyter notebooks to teach high-performance computing," in *Northwest Regional Conference of the Consortium of Computing Sciences in Colleges (CCSC-NW) 2018*. CCSC, 2018, pp. 180–188.

[28] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster, "The globus striped gridftp framework and server," in *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, ser. SC '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 54–. [Online]. Available: https://doi.org/10.1109/SC.2005.72

[29] A. S. Team, *Amazon Simple storage Service Documentation*, Amazon, Inc., Seattle, USA, 2018. [Online]. Available: https://aws.amazon.com/documentation/s3/

[30] T. White, *Hadoop: The Definitive Guide*, 1st ed. O'Reilly Media, Inc., 2009.

[31] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[32] I. Foster and C. Kesselman, *The Grid 2: Blueprint for a New Computing Infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[33] M. Litzkow, M. Livny, and M. Mutka, "Condor - a hunter of idle workstations," in *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[34] K. Dowd and C. Severance, *High Performance Computing*, 2nd ed., M. Loukides, Ed. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 1998.

[35] I. Foster and C. Kesselman, Eds., *The Grid: Blueprint for a New Computing Infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.