Sandstone HPC - A Domain-General Gateway for New HPC Users

Zebula Sampedro
University of Colorado Boulder
Research Computing
597 UCB Boulder, CO, United States
zebula.sampedro@colorado.edu

Thomas Hauser
University of Colorado Boulder
Research Computing
597 UCB Boulder, CO, United States
thomas.hauser@colorado.edu

Saurabh Sood
University of Colorado Boulder
Research Computing
597 UCB Boulder, CO, United States
saurabh.sood@colorado.edu

ABSTRACT

The complexity of high-performance computing (HPC) resources poses many challenges to new users. A number of science gateways have been developed to increase the productivity of novice users by hiding the underlying infrastructure, however these solutions tend not to teach HPC skills that transfer easily outside of the gateway. In this paper we introduce a domain-general gateway, Sandstone HPC, that represents the HPC environment more naturally to novice users by abstracting the command-line interface and providing contextual help. We assess the degree to which Sandstone HPC improves upon the usability of the command-line interface by analyzing the results of a usability study conducted on both environments. We will also detail how the architecture promotes long-term sustainability and a community-development model.

CCS Concepts

•**Human-centered computing** → **User studies**; *Empirical studies in HCI*; *Web-based interaction*; •**Software and its engineering** → **Integrated and visual development environments**;

Keywords

Web-Based IDE; High-Performance Computing; User Studies

1. INTRODUCTION

Users of high-performance computing resources tend not to be computational experts, but instead scientists with a primary focus on scientific results [20]. Although in order to

produce scientific results, HPC users must become knowledgeable practitioners in the wide variety of skills necessary to navigate these increasingly complex resources. Before they can even login and submit their first job, users must learn and accomplish over a dozen tasks, many of which are strictly outside of their domain [21].

This skill gap poses a particular challenge, as many users approach this discipline unfamiliar with not only core HPC skills like scheduling and parallel programming, but also Linux and command-line interfaces (CLI). One survey of PRACE users observed that over 70% of respondents had no formal education in BASH/Linux or fundamental HPC skills [19]. A survey of our own suggests that this autodidactic approach creates a number of pitfalls. Over a six-month period, we reached out to new users whom either failed to login to our system, or successfully run a job. 77% of these users indicated that they would require training in basic Linux skills and job scheduling in order to successfully accomplish their work.

Minimizing the impact that this skill gap has on user-productivity has been an active area of focus and development for a number of years now. The most common approach is the *Science Gateway*. XSEDE defines a Science Gateway as such:

A Science Gateway is a community-developed set of tools, applications, and data that are integrated via a portal or a suite of applications, usually in a graphical user interface, that is further customized to meet the needs of a specific community. Gateways enable entire communities of users associated with a common discipline to use national resources through a common interface that is configured for optimal use [6].

An intrinsic quality of most gateways is that they hide the underlying infrastructure. The goal of these gateways is to improve user productivity as quickly as possible by eliminating the need for the user to learn system specific policies and workflows to accomplish certain domain tasks. Creating an interface that is highly optimized for a particular discipline by abstracting system details necessarily creates a workflow that is distinct from its traditional, command-line interface equivalent. The result of this is that skills learned on either

This work is licensed under a Creative Commons Attribution International 4.0 License.

PEARC17 July 09-13, 2017, New Orleans, LA, USA

© 2017 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-5272-7/17/07.

DOI: <http://dx.doi.org/10.1145/3093338.3093360>

interface cannot be transferred in between.

A domain-specific gateway designed to hide infrastructure can help a user —*particularly a new user*— get started on an HPC resource quickly. However this kind of gateway necessitates that a user be able to realize her own work within the confines of the gateway; the gateway is explicitly designed to circumvent the traditional HPC workflow. When one considers that scientists tend to use what their peers use (and often need to pick up where their peers left off), and that most HPC development still happens on the CLI [15], it becomes clear that there is a need to accommodate users whom cannot accomplish their workflows within a science gateway.

In this paper, we propose an alternative perspective on HPC user gateways. This perspective is to develop a gateway that focuses on those aspects of the HPC workflow that are common to all users, regardless of discipline. Instead of hiding system details, this gateway would instead make these details apparent while facilitating use of the system. In §2 we introduce Sandstone HPC, a gateway built upon this alternative perspective. We will also discuss how the architecture of the system favors long-term sustainability, and heavily emphasizes community-development. Lastly, we assess the degree to which Sandstone is successful at facilitating system use, and detail the methods and findings of a usability study conducted over part of the gateway in §4.

2. SANDSTONE HPC

To provide a solution for those users that cannot achieve their goals using a science gateway we have developed Sandstone HPC, a web application that decomposes the CLI workflow into meaningful, interactive, visual components. Building a gateway that represents the CLI targets an aspect of the HPC workflow that is common to all disciplines and users. This domain-general approach allows the gateway to not only minimize time-to-science, but also teach skills that will transfer to the underlying system. Sandstone is conceptually similar to a Web-based IDE, however its focus has been placed on facilitating use of the underlying system as opposed to facilitating the software development process. Adapting the Web-based IDE model provides a platform that is operating-system-agnostic, requires minimal setup from the user’s perspective, and is able to provide a superior interface to the the CLI [15].

2.1 Design Considerations

Sandstone HPC is designed to represent the command-line environment with components and interaction paradigms that are more familiar and accessible to new users. This makes it possible for the gateway to supplement the usability shortcomings of the CLI, without replacing it entirely. Sandstone can assist the new user in being productive sooner, while still being transparent enough to teach skills that will transfer to future workflows. This is accomplished by adhering to three primary design principles, described below:

Minimize up-front knowledge. To accelerate user productivity, the amount of up-front knowledge any task requires is minimized. Documentation and meaning are folded into task structure, allowing learning to happen within the context of performing an action. Components of the CLI workflow are mapped to visual paradigms with which the user is already familiar to allow the interface to more clearly articulate the purpose, function, and space of possible actions

for each component. This allows the gateway to guide the learning process, leveraging the user’s existing knowledge and minimizing the time the user will spend searching for and interpreting documentation.

Provide clear feedback. To provide an interface that is easy to interpret and use efficiently, Sandstone is designed to maximize the amount and quality of feedback given to the user. Each action available to the user is designed to indicate its expected end-state, and the results of every user-interaction are clearly presented. This allows the user to deliberately navigate the interface in accordance with her goals, and encourages exploration. The interface also clearly articulates errors, so that the user can diagnose and resolve problems independently.

Reinforce the CLI workflow. Each interface component is designed to reinforce the structure and use of its CLI-equivalent. Terminology and component descriptions in Sandstone are adopted directly from the CLI, so that the user may move freely between platforms using the same vocabulary. Each component in Sandstone maps user interactions to effects on system state in a way that is consistent with the CLI, so that the user may learn the use of each component in a context-rich environment. Using interaction to reinforce the CLI workflow allows both environments to be used synergistically, without forcing the user to shift paradigms.

2.2 Modules

Sandstone HPC is composed of an extendable set of *modules*, each one representing a distinct element of the CLI workflow. Modules are designed to be loosely coupled to make it possible for adopting sites and research groups to easily extend, modify, delete, and create individual modules to meet their needs without affecting other modules within the Sandstone runtime. What follows is a description of modules currently available in Sandstone HPC.

Code Editor. This module provides the user with a web editor similar to those seen in other Web-based IDEs like Cloud9 [3]. The Sandstone editor provides syntax highlighting, find and replace, and tabbed file editing. Providing the user with a convenient interface for making in situ edits over project files obviates the need for either excessive file transfer or immediately learning cumbersome terminal editors.

File Browser. This module provides the user with the very familiar file browser paradigm, which plays a critical role in giving the user context for the underlying system. This module shows accessible volumes and volume information, file and directory details, and provides HTTP upload functionality to use in place of `scp`.

Web Terminal. Sandstone includes a Terminado web terminal [8] that allows the user to use the CLI without having to perform any local setup, such as SSH clients. This easy-to-access web terminal also allows for the gradual integration of more advanced skills over time, without having to leave the Sandstone interface.

Slurm Assist. An interactive, visual utility for scheduling jobs with Slurm [7]. This module features a highly-configurable web form that aids the user in specifying directives for her Sbatch script. The module also provides utilities for estimating job cost, viewing and controlling job status, and loading previously-created scripts. This module will be covered more extensively in §4.1.

JupyterHub Spawner and Login. A set of modules that enable dynamic Sandstone deployment via JupyterHub, a

multi-user hub for spawning and managing Jupyter notebooks [5]. This deployment strategy makes it possible to use and maintain per-user Sandstone servers alongside Jupyter notebooks.

The above modules are presently available, and expanding this list is an ongoing effort. There are plans to add an extension to Sandstone to provide an interactive, searchable listing of environment modules generated automatically from system configuration. There are also plans to add an HTML-based VNC client to Sandstone to support a wider variety of applications and use cases.

2.3 Architecture

Sandstone HPC system architecture is designed with careful consideration paid to interface quality and community adoption. An interface with the responsiveness and quality that users now expect from a web application is achieved by employing modern client frameworks and programming techniques. Architecting an application that can be widely adopted by the HPC community however, poses a great many challenges. Chief among these challenges is the need to develop a site-general solution such that installing the application on a variety of systems only requires configuration, not additional development. Such a solution needs to accommodate a wide variety of policies, system environments, and deployment strategies. The solution also needs to be easy to extend and modify, if it is necessitated by the needs of an adopting site.

At a high level, Sandstone is a self-contained Python web application which hosts a JavaScript single-page application (SPA). An SPA is a web app that loads a single HTML page from the server and uses client-side code to update that page, eliminating the need to reload the page during use. Using an SPA for the front-end allows for the development of a modern interface that is more responsive than a traditional website, as the application no longer relies on the server for page updates [14]. The structure of an SPA also allows for better error recovery and reporting, as failed requests to the server no longer prevent the client from receiving content necessary to render the page.

The SPA that constitutes the front-end is hosted by a *self-contained* Python web application. This application is described as being self-contained because it manages all elements of its own execution such as hosting of static content, SSL encryption, and server code. This lies in contrast to a traditionally-structured web application which relies on external services such as Apache to manage the above mentioned aspects of deployment. The result of self-containment is a highly portable system that can be run as a user command.

Running Sandstone as a user command is particularly important to system security, as the application inherits all the permissions and capabilities of the executing user. This allows the operating system to enforce security and access constraints, and ensures that no user can elevate her access level if the application is compromised. To accommodate deployment strategies where a Sandstone instance is publicly-accessible, the application requires the user to log in with her system account. Credentials supplied during a login request are forwarded to PAM (Figure 1), therefore any login strategy that sits behind PAM will automatically be supported. If the authentication attempt against PAM is successful, a secure session cookie is handed back to the

Figure 1: This diagram illustrates how the Sandstone HPC core framework and an installed module communicate with one another, and the operating system.

client to verify future requests.

Implementing the backend as a Python web application also contributes greatly to the modularity of the Sandstone architecture. Sandstone provides an extremely lightweight framework that implements authentication, web application lifecycle management, subsystem interfaces, core front-end components, and configuration utilities. The bulk of the Sandstone interface and functionality however is implemented by a set of independent modules developed on this framework (Figure 1).

Each module in Sandstone is a Python module that includes a manifest of static web content and runtime configuration. Modules are easy to install and manage, as the administrator need simply to place the module in the same Python environment as the Sandstone application, and then declare the module path in the central configuration [18]. When the application is starting up it loads all declared modules into its runtime.

Two separate technologies compose the SPA front-end for Sandstone. AngularJS, a JavaScript Model-View-Controller framework [1] is used to implement the behavior of the SPA, and Twitter Bootstrap [2] is used for the style and content. The backend is implemented in Tornado, a performant Python web framework with native support for web-sockets [9]. All three components of the Sandstone HPC stack possess attributes that favor long-term sustainability. Among these attributes are project maturity, use of permissive open-source licenses, and a sizable community of users and developers.

Sandstone IDE EDITOR FILEBROWSER NBTERRM **SLURM ASSIST** LOGOUT

SCHEDULE STATUS

Directives

```
#!/bin/bash
#SBATCH --qos=janus-debug
#SBATCH --nodes=1
#SBATCH --time=00:30:00
```

Script

```
1 # Add your script below
2
3 hostname
4
```

Select Job Template

Debug Job

Add Directives

qos

janus-debug

nodes

1

time

00:30:00

SAVE SCRIPT FOR LATER LOAD SAVED SCRIPT COST ESTIMATE **SCHEDULE JOB**

Figure 2: An example job being constructed in the Slurm Assist module. The user specifies Sbatch directives in the form on the right, and the values are propagated to the script viewer/editor on the left.

3. RELATED WORK

Cloud IDEs are becoming a dominant mode of access for remote compute resources as they are operating-system-agnostic, based on familiar interaction patterns, and require no installation for the end user. Two popular examples of cloud IDEs are Cloud9 [3] and Koding [4]. Both projects are commercial however open-source forks are available for download. In the early stages of the project we evaluated the possibility of extending one of the above cloud IDEs, but there was concern about the ease of extension and flexibility of these platforms for our use case. There was also some concern about the full stack use of web languages in these projects, as we wanted to focus on an implementation that would be easily accessible to users and administrators at deploying sites. Sandstone adapts the cloud IDE model for HPC, placing the emphasis less on code development and more on facilitating access and learning.

The Jupyter project has gained a large amount of popularity and support over the past few years, particularly in the HPC space. Reproducible computational narratives have proven to be a powerful tool for researchers and instructors [16], and the platform has also gained popularity due to its flexibility and ease of deployment. For this reason Sandstone borrows heavily from the Jupyter Notebook architecture, employing a self-contained Python web application that runs as a user command and can be installed anywhere.

Within the realm of HPC gateways there are also several efforts that share a focus on providing web interfaces for domain-general tasks. Among these related HPC gate-

ways are the Viewpoint Portal [10] and Open OnDemand [12]. Viewpoint is a commercial product whereas Open OnDemand is an open-source platform built for end-user extension. These projects offer overlapping functionality with Sandstone such as interfaces to browse and edit files and creating compute jobs from templates, but differ in their approach to architecture and dependencies. Sandstone is designed to support a wider range of deployment strategies, including installation by an unprivileged user.

4. USABILITY STUDY

To determine how successful Sandstone HPC is in solving the usability challenges posed by the CLI, a usability study was conducted on the *Slurm Assist* module. This module was selected to illustrate particular strengths and weaknesses in aiding novice users with HPC-specific tasks that require knowledge outside of the users' domains. This study focused primarily on measuring how the use of Sandstone, when compared to the command-line environment, affected the following aspects of user performance: task success and accuracy, error recovery, and discovery and interpretation of documentation.

4.1 Slurm Assist Module: A Closer Look

The scope of this module is to aid the user in submitting compute jobs with the Slurm job scheduler. In keeping with the Sandstone design principles, the job submission workflow is reinforced using interaction paradigms more familiar to the user, with focus placed on reducing the amount of knowledge required to get started and clear articulation of

errors. If scheduling a job with the CLI, a general workflow would be as follows:

1. Assess the characteristics of the job to be run (nodes, wall time, output file, etc.).
2. Translate job characteristics to corresponding Sbatch directives, and then prepend those directives to a shell script.
3. Submit the shell script to Slurm via the `sbatch` command.

The overall structure of the Slurm Assist module reinforces this workflow, but each step is contextualized more effectively for the user. Step 1 is represented by a visually-prominent dropdown labeled *Job Templates*, containing plain-language descriptions of different job types that may be run on the system. Upon selection of a template the directive form (discussed below) appears, pre-populated with suggested directive values. Beyond offering context regarding what types of jobs may be run on the system, these templates also provide a starting point for directive structure and likely values. This eliminates the need for the user to search for meaningful equivalents to the job she wishes to run in documentation, which proved to be a primary struggle for users.¹

Step 2, the translation of job characteristics into appropriate Sbatch directives, is by many measures the most arcane and difficult aspect of scheduling a job. Sandstone represents directives in two ways: directly in a textbox that is placed atop the script editor, and in a dynamic web form which allows the user to edit directives by filling out associated form fields (Figure 2). This dual representation allows the directives to be added and edited using the more descriptive web form, while still showing the users how those values translate to a job script built outside of Sandstone. Placement of the directives viewer above the script editor visually implies where in a job script directives are located.

The web form (which can be configured to enforce system constraints) validates input as the user types, highlighting errors and disabling the *Schedule Job* button if the user inputs directive values that are incorrect for the selected Job Template (Figure 3). This not only prevents the user from submitting an invalid job, but it also provides the feedback and information needed to diagnose and resolve the error.

The directives form is also designed to facilitate exploration, which helps the user discover and visualize the directives that are most appropriate to describe her job. This is first accomplished by a typeahead input which allows the user to browse all available directives, or narrow down the selection by typing in key terms. Each form field also contains a help button that provides documentation for the Sbatch directive, obviating the need for the user to find and interpret Slurm documentation or manual pages.

Submission is the final step and is represented by the *Schedule Job* button. Once the user generates the contents of a valid job script, the button changes states and becomes clickable. Upon clicking the user is presented with a *Save As* modal which structurally implies that the contents generated in Slurm Assist are components of a BASH script,

¹In our usability study, no participant successfully utilized the listing of qos purposes and limits.

Figure 3: The user has specified values for both *nodes* and *time* that exceed the configured limits for the *janus-debug* qos. These errors are highlighted to help the user diagnose and resolve the problem.

and saving that script is a necessary precursor to submission. After saving a confirmation dialogue is shown, providing immediate feedback on the success of the job submission.

4.2 Methods

This study involved 8 participants that self-identified as having low-to-moderate skill on the CLI, and limited-to-no skill with the Slurm job scheduler. The number of participants was determined by the product of the base participant count per demographic (four for this type of study), the number of distinct demographics involved in this study (one demographic representing novice users), and the number of conditions in the study (two conditions: tasks were performed using either the CLI or Sandstone) [11].

Each participant was asked to complete four scheduling tasks, two using the Sandstone Slurm Assist module, and two using the command-line interface (CLI). A *Between-Subjects* design with counterbalancing (as described by Rubin [17]) was used, so no participant performed the same task on both Sandstone and the CLI. This prevented learning affects between platforms from invalidating test results. While performing the tasks, participants used the *Think-Aloud Protocol* to better articulate their rationale for the actions they were taking [13]. Below are the four tasks:

1. Schedule a Slurm job that runs in `janus-debug` for no more than 50 minutes, and outputs to a file called `output.log` in your home directory. The job should execute the `hostname` command.
2. Schedule a Slurm job that runs in `janus-long` on 2 nodes for no more than 18 hours, and outputs to a file called `output.log` in your home directory. The job should execute the `hostname` command.
3. Schedule a Slurm job that runs in `janus-debug` on 1 node for no more than 10 minutes, outputs to a file called `output.log`, and prints error out to a file called `error.log`. Both log files should be in your home directory. The job should execute the `hostname` command.

4. Try to schedule a Slurm job that runs in `janus-debug` on 12 nodes. The job should execute the `hostname` command. Explain why your job is not starting.²

4.3 Measures

The two kinds of data collected from this study were quantitative performance data, and qualitative preference data. The performance data was a set of metrics determined by post-task analysis of the recorded sessions. The metrics included time-on-task, time spent searching for and reading external documentation, errors of omission and commission, error recoveries, and tasks completed successfully. Task success was recorded as one of three values:

- The participant completed the task correctly, and required no prompting from the test moderator.
- The participant completed the task correctly, but required prompting from the test moderator.
- The participant did not complete task within the time limit, or the final solution was incorrect.

Preference data was collated from the Think-Aloud Protocol and a post-task survey given to each participant. Beyond identifying general impressions of each platform, this data also provided insight on why the participants encountered certain obstacles, and how they attempted to overcome them throughout the course of each task.

4.4 Findings

Compared to the CLI, Sandstone HPC improved task success, accuracy, and use of documentation. When analyzing each platform as a whole, participants had a 31% successful task completion rate when using the CLI, compared to 75% when using Sandstone. Accuracy followed a similar pattern, with an average of 2.6 errors committed per task and an average time-on-task of 8 minutes on the CLI, while an error rate of .5 errors per task and an average of 4 minutes time-on-task was observed on Sandstone. Contextual documentation in Slurm Assist (tooltips, form field directive documentation, and general help text) reduced the average time spent searching for and reading documentation to under one minute per task, whereas an average of 4 minutes per task was observed on the CLI.

An analysis of the problems participants encountered in correlation with the preference data demonstrates a core problem with the CLI that negatively impacted task performance: the CLI provides few, and sometimes no contextual information. The lack of contextual information in the CLI most negatively impacted the participants' abilities to recognize, identify, and resolve errors. Task D clearly demonstrated this impact, as it required participants to explain that the job they submitted exceeded the configured limits for the `janus-debug` qos.

Only 25% of participants who performed this task on the CLI completed it successfully. Even when prompted to view the output of `squeue` many participants did not recognize the error. Those whom identified that the job state had changed from *Pending* to *Failed* could not correctly explain the cause. The immediate form field validation and contextual documentation in Slurm Assist led to a much improved

100% successful completion rate for Task D. Each participant correctly identified that an error occurred as soon as they input the invalid value. By either reading the adjacent help text or experimenting with the field values, they also correctly articulated the source of error and the correct solution (decrease the node count).

User impressions of the two platforms were collated from both the Think-Aloud protocol and the post-task survey. While all participants remarked that the terminal seemed like it would eventually become a more powerful and flexible solution, all participants also indicated that they would prefer to use Sandstone, particularly in the earlier phases of the learning process. They felt that it was the superior option in regards to describing workflow, expressing component meaning and function, and making errors much more explainable and evident.

5. CONCLUSIONS

Science gateways are a proven and effective means for improving the productivity of a new user. Sandstone HPC applies the science gateway model directly to the HPC resource itself, facilitating its use while still teaching the skills necessary to expand one's workflow outside of the boundaries of the gateway.

Facilitating access and use to an HPC resource while still representing it authentically allows for new applications of HPC gateways. One such application is to use Sandstone as a platform for tutorials. This allows the tutorial content to focus more narrowly on topics that have direct utility to the user's own research. The Web-based IDE model obviates the need for the user to successfully configure SSH clients, use terminal editors, and interact with a job scheduler via terminal in order to follow along with instruction.

Sandstone makes it easy for an HPC site or research group to realize new applications of their own by encouraging community development and adaptation. Modular architecture makes it easy to modify and extend modules independently from the rest of the software, as well as develop new components that will then become part of the Sandstone HPC ecosystem. The freedom to easily reshape Sandstone widens the pool of potential developers and adopters of this project, which contributes greatly to its long-term sustainability.

Sandstone functions as a complementary component in the greater ecosystem of HPC gateways. It builds upon the successes of gateways that have come before it, and further expands the scope of research that these valuable community resources can support.

6. REFERENCES

- [1] Angularjs - superheroic javascript mvw framework. <https://angularjs.org/>. Accessed: 2017-02-14.
- [2] Bootstrap - the world's most popular mobile-first and responsive front-end framework. <http://getbootstrap.com/>. Accessed: 2017-02-14.
- [3] Cloud9 - your development environment, in the cloud. <https://c9.io/>. Accessed: 2017-02-14.
- [4] Modern dev environment delivered - koding. <https://www.koding.com/>. Accessed: 2017-06-01.
- [5] Project jupyter home. <http://jupyter.org/>. Accessed: 2017-02-14.
- [6] Science gateways. <https://www.xsede.org/gateways-overview>. Accessed:

²the `janus-debug` qos has a maximum node limit of 2

2017-02-13.

- [7] Slurm workload manager. <https://slurm.schedmd.com/>. Accessed: 2017-02-14.
- [8] Terminado web terminal. <http://terminado.readthedocs.io/en/latest/>. Accessed: 2017-02-14.
- [9] Tornado web server. <http://www.tornadoweb.org/en/stable/>. Accessed: 2017-02-14.
- [10] Viewpoint. <http://www.adaptivecomputing.com/products/hpc-products/viewpoint/>. Accessed: 2017-06-01.
- [11] K. Goodwin. *Designing for the digital age: How to create human-centered products and services*. John Wiley & Sons, 2011.
- [12] D. E. Hudak, D. Johnson, J. Nicklas, E. Franz, B. McMichael, and B. Gohar. Open ondemand: Transforming computational science through omnidisciplinary software cyberinfrastructure. In *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale*, page 43. ACM, 2016.
- [13] R. Jääskeläinen. Think-aloud Protocol. In Y. Gambier and L. van Doorslaer, editors, *Handbook of Translation Studies*. Handbook of translation studies, 2010.
- [14] M. S. Mikowski and J. C. Powell. Single page web applications. *B and W*, 2013.
- [15] J. Overbey and M. Price. Toward a Cloud IDE for HPC. 2015.
- [16] F. Perez and B. E. Granger. Project jupyter: Computational narratives as the engine of collaborative data science. Technical report, Technical report, Project Jupyter, 2015.
- [17] J. Rubin and D. Chisnell. *Handbook of usability testing: how to plan, design and conduct effective tests*. John Wiley & Sons, 2008.
- [18] Z. Sampedro and S. Sood. SandstoneHPC/sandstone-ide: v1.0.1 (Beta) 2017-05-03, May 2017.
- [19] T. Stitt and T. Robinson. A survey on training and education needs for petascale computing, 2008.
- [20] N. Wolter, M. O. McCracken, A. Snavelly, L. Hochstein, T. Nakamura, and V. Basili. What’s working in hpc: Investigating hpc user behavior and productivity. *CTWatch Quarterly*, 2(4A):9–17, 2006.
- [21] A. Zimmerman and T. A. Finholt. Growing an infrastructure: The role of gateway organizations in cultivating new communities of users. In *Proceedings of the 2007 international ACM conference on Supporting group work*, pages 239–248. ACM, 2007.