Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Unifying computing resources and access interface to support parallel and distributed computing education

Linh B. Ngo [a,b,*], Ashwin Trikuta Srinath [a], Jeffrey Denton [a], Marcin Ziolkowski [a]

[a] *Clemson Computing and Information Technology, Clemson University, Clemson, SC, United States*
[b] *School of Computing, Clemson University, Clemson, SC, United States*

HIGHLIGHTS

- Design of a framework combining various computing resources and browser-based access interfaces.
- Description of learning modules for a CS course leveraging this framework.
- Reports on students' skill improvement and feedback on lecture contents and course project.

ARTICLE INFO

Article history:
Received 31 July 2017
Received in revised form 9 February 2018
Accepted 26 February 2018
Available online 7 March 2018

Keywords:
Computing resources
MPI
MapReduce
Spark
Python
Distributed systems
Parallel computing education

ABSTRACT

This article presents how various on-site and remote computing resources are combined into a framework to support teaching parallel and distributed computing (PDC) at the undergraduate level. The combination of these resources enables the delivery of PDC programming, system, and architectural concepts via a browser-based common interface (JupyterHub) and a single programming environment (Python and its supported libraries). This also allows lecturers and students to focus more on the principles of PDC and less on the technicalities of native languages for different platforms. We describe how this framework can support a comprehensive set of PDC course modules, including lectures, assignments, and projects, for a full semester junior-level class. Adoption of this framework in various teaching environments at Clemson University has received positive feedback from both instructors and participants.

Published by Elsevier Inc.

1. Introduction

Parallel and distributed computing (PDC) has become integral to various aspects of IT across all academic disciplines and industrial areas. It is critical that college graduates are properly introduced to PDC core concepts and technologies for their future careers. The existing body of PDC knowledge spans across four different areas: Data Structures and Algorithms, Software Design, Software Environments, and Hardware [8]. A traditional Beowulf-based computing cluster [40], available through either on-site or public resources, can adequately provide a classroom computing environment for this knowledge, as shown in Fig. 1.

However, there remains a number of issues in working within this environment. First, interactions with computing clusters are typically done through a Linux-based command line interface (CLI). Terminal tools to support these interactions are available by default on Linux and Mac, but not on Windows. A portion of the class time must be spent on ensuring that all students can access the computing infrastructures. This necessitates that instructions and technical support for Linux and Mac terminals, and terminal emulators for Windows (e.g., PuttY or SSH Secure Shell) need to be prepared. The second issue arises from the need to include various new technologies in the curriculum as PDC moves beyond the traditional high performance computing concepts and into areas of data-intensive computing, big data analytics, and large-scale streaming systems. This leads to an increase in the number of computing tools and platforms that need to be taught together with the corresponding PDC concepts. To deliver this expanding set of new PDC concepts and enable students to have adequate hands-on practice within the limited class time, instructors will need to face in-class technological issues that will most certainly arise from working with these various tools and platforms. Furthermore, while most of these new platforms support Java, a commonly taught language within the core curriculum, the code complexity

* Corresponding author at: Clemson Computing and Information Technology, Clemson University, Clemson, SC, United States.
E-mail addresses: lngo@clemson.edu (L.B. Ngo), atrikut@clemson.edu (A.T. Srinath), denton@clemson.edu (J. Denton), zziolko@clemson.edu (M. Ziolkowski).

https://doi.org/10.1016/j.jpdc.2018.02.020
0743-7315/Published by Elsevier Inc.

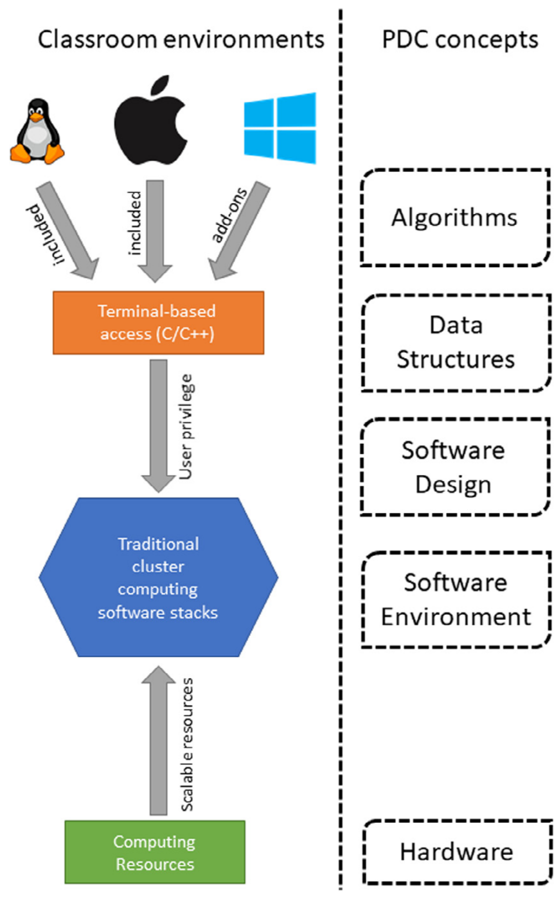


Fig. 1. A traditional classroom computing environment for teaching PDC concepts.

is significant. For example, the well-known Hadoop MapReduce WordCount program [1] requires fifty five lines of Java code. Out of these lines, only ten are directly related to the MapReduce programming portions, while the remaining forty five include library imports, class and function declarations, and standard job configurations. In both formal and informal educational settings, explanation of these lines present significant timing overheads during the hands-on practice segments of the lectures. They distract learners from programming paradigms and workflow designs, which are more important aspects of the learning process. In many cases, they require instructors to spend valuable class time for trouble shooting and technical support.

In this paper, we describe our approach at Clemson University in addressing these issues by unifying on-site and remote computing resources, access platforms, and programming tools into a common PDC educational framework. This framework is shown in Fig. 2. To support the traditional PDC concepts, we utilize Clemson University's research cluster, the Palmetto Supercomputer (Palmetto). For advanced PDC concepts, educational modules are developed using CloudLab, a public research computing testbed [37]. To combine these two resources, we deploy and integrate JupyterHub as the front-end for Palmetto. JupyterHub allows users to spawn Jupyter server, a platform that provides convenient access to computing environments for high performance computing, big data, and data intensive computing infrastructures. With Jupyter's ability to support a diverse set of programming languages, a browser-based terminal, and a text editor, students can be introduced to PDC through a standardized browser-based interface across different operating systems. The browser-based terminal also allows students to seamlessly interact with CloudLab from Palmetto.

The default language for JupyterHub Notebooks is Python. The availability of various community-supported Python libraries for PDC allows instructors to teach PDC concepts from different platforms without having students learn the various native languages of these platforms. As a result, students can spend more time understanding PDC concepts and less time on syntax correctness of specific tools and languages. This facilitates the instructional delivery of the most common PDC areas such as high performance computing, data-intensive computing, and in-memory distributed computing, which were originally designed for different computing platforms using different languages. With the popularity of Python and the variety of Python libraries/APIs that support parallel and distributed programming, there exists a number of tutorials and teaching materials for PDC using Python. Examples include the exhaustive tutorial for Python and MPI by the creators of mpi4py [12] or the framework to teach MapReduce programming via a web browser [17]. Our framework will further extend the ability to utilize Python to teach PDC by developing Python-based learning modules on a common and consistent interface, the Jupyter notebooks.

The unified educational framework for PDC presented in this article will enable important aspects of teaching PDC [21] such as showing speedup, real time results, visual results, interactivity, active learning, reproducibility, and accessibility. All resulting educational modules are available online. The remainder of this paper is organized as follows. Section 2 describes Palmetto and CloudLab, the on-site and remote computing resources, as well as the design and deployment of JupyterHub on Palmetto. Section 3 contains three topics. First, We discuss previous work that used Python in PDC education. Second, we describe course modules that use Jupyter and Python to teach high performance computing, data-intensive computing, and in-memory distributed computing concepts. Third, we show how the course modules can combine CloudLab, Jupyter, and Python in teaching advanced topics such as distributed system architectures, schedulers, and distributed file systems. Section 4 describes user evaluation of this unified educational computing framework from participants in various workshops and students from a class that teaches high performance and data-intensive computing. The evaluation includes performance observations and feedback from students and instructors who use the platform. Section 5 concludes the paper and discusses future work.

2. Unifying computing resources and access platforms

It is typical for PDC to be primarily taught as a single course within the entire undergraduate CS curriculum while having some PDC concepts embedded in other courses [9,42]. At Clemson University, PDC education is delivered through two settings, formal academic courses and informal half/full-day training workshops.

The PDC academic course is CPSC 3620, a computer science course on the topic of distributed and cluster computing. This course is taught in fifty-minute classes three times a week. While this is a required junior-level course, most students in the class wait until the first or second semester of their senior year before taking the course. The enrollment of the class ranges between 40 and 45 students. The course is intended to present students with a broad overview of PDC. The topics covered include distributed file systems, the message-passing programming paradigm, scheduling on a cluster of computers, big data and data-intensive computing, the map-reduce programming paradigm, in-memory distributed computing, cloud computing, message-oriented middleware, and distributed stream processing [28].

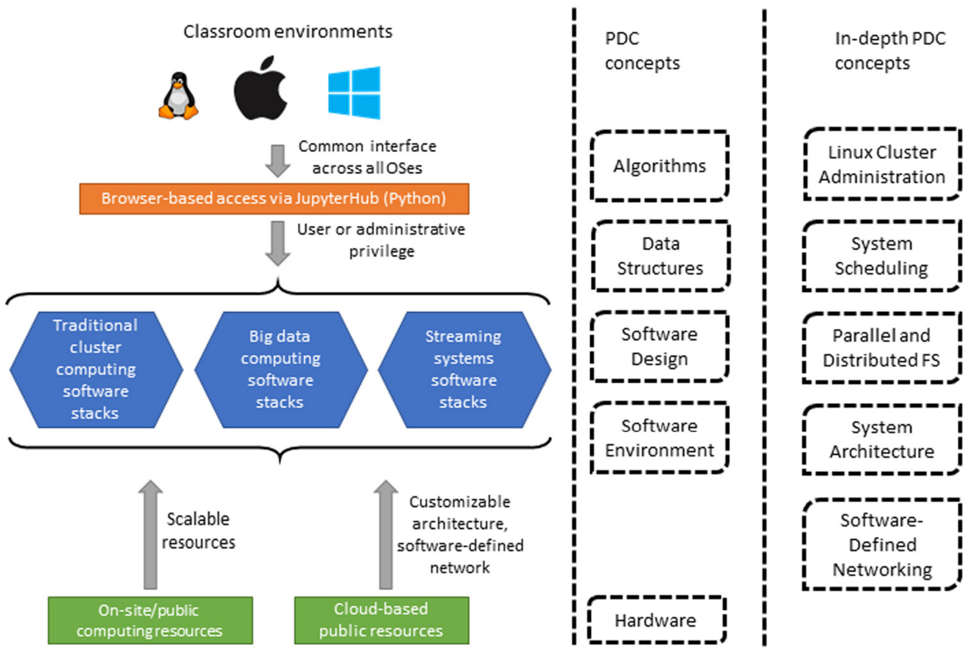

Classroom environments
Common interface across all OSes
Browser-based access via JupyterHub (Python)
User or administrative privilege
Traditional cluster computing software stacks
Big data computing software stacks
Streaming systems software stacks
Scalable resources
Customizable architecture, software-defined network
On-site/public computing resources
Cloud-based public resources
PDC concepts
Algorithms
Data Structures
Software Design
Software Environment
Hardware
In-depth PDC concepts
Linux Cluster Administration
System Scheduling
Parallel and Distributed FS
System Architecture
Software-Defined Networking

Fig. 2. A unifying classroom computing environment that supports a variety of infrastructures for teaching PDC concepts.

At Clemson University, the Cyberinfrastructure and Technology Integration group (CITI) also offers regular concentrated half/full-day training workshops teaching PDC to researchers and graduate students engaging in research activities that require advanced research computing infrastructures. The workshops focus on hands-on skills that quickly familiarize participants with using PDC, while fundamental concepts are only covered when necessary. Examples include working with Linux environments, interacting with research computing resources, developing parallel programs, and working with Hadoop/Spark big data infrastructures. Workshop materials are designed to be self-paced [11].

In both cases, the amount of content that needs to be delivered in each session is significant. It is critical to minimize the technical hurdles for students and workshop participants in their interactions with various computing resources when learning about PDC. To accomplish this goal, we develop our course modules around the unifying computing framework which includes one on-site computing resource, one remote public research cloud environment, and a browser-based access platform.

2.1. On-site computing resource: Palmetto cluster and Cypress cluster

The Palmetto Cluster (Palmetto) is a shared local high performance computing environment available to all Clemson students, faculty and staff as a resource for research and education. Palmetto has been deployed and managed using a condominium model since 2006, with the university subsidizing most of the costs, and faculty contributing to the baseline cost of the computers in exchange for priority access to Palmetto resources. Every year, Palmetto's computing capacity is expanded by the addition of new hardware to the existing infrastructure. Currently, Palmetto consists of more than 2000 compute nodes with various hardware profiles, including computers with specialized components such as large memory and GPU cards. Palmetto also has a 40-node Hadoop cluster named Cypress attached to the same network. The Hadoop ecosystem is a prime example of a significant area of interest which is "continually evolving as topics mature and even newer topics appear on the scene" [33]. This is made evident by recent additional software libraries and frameworks associated with the Hadoop ecosystem, including cluster resource manager [2], in-memory distributed computing [44], interactive programming [5], and distributed data store [3]. At Clemson University, Palmetto and Cypress are set up such that workflows can be designed to involve both Palmetto for high performance computing tasks and Cypress for big data analytics tasks.

Palmetto usage policy allows educational access for students. As a result, they are able to observe examples and work on tasks that demonstrate concepts such as speedup, efficiency, and parallel I/O. However, since students are limited to having only user privileges on Palmetto, these tasks are limited to primarily programming-based assignments. It should be noted that in the absence of an on-site centralized computing resource such as Palmetto, there exist other public resources that can be utilized in this educational framework. One prominent example is the Extreme Engineering and Science Discovery Environment (XSEDE) [41]. XSEDE consists of more than a dozen research computing sites from different institutions across the United States in order to provide computing resources for both research and educational purposes. Instructors can apply for an educational allocation and distribute this allocation among their students.

2.2. Remote cloud resource: CloudLab

Built on the successes of the Global Environment for Network Innovations (GENI) [7], CloudLab provides a robust cloud-based environment where researchers can design and deploy experiments for next generation computing research [37]. An experiment using various computers and network topologies in CloudLab is represented by a resource description file. One method to generate this file is to develop a Python program to describe how the components of the experiment are to be reserved, configured, and deployed on CloudLab hardware. As the computing components are virtual images booted on top of bare metal infrastructures, CloudLab users are granted complete administrative privilege over their experiments. Similar to XSEDE, CloudLab allows instructors to apply for educational projects and to add students to these projects.

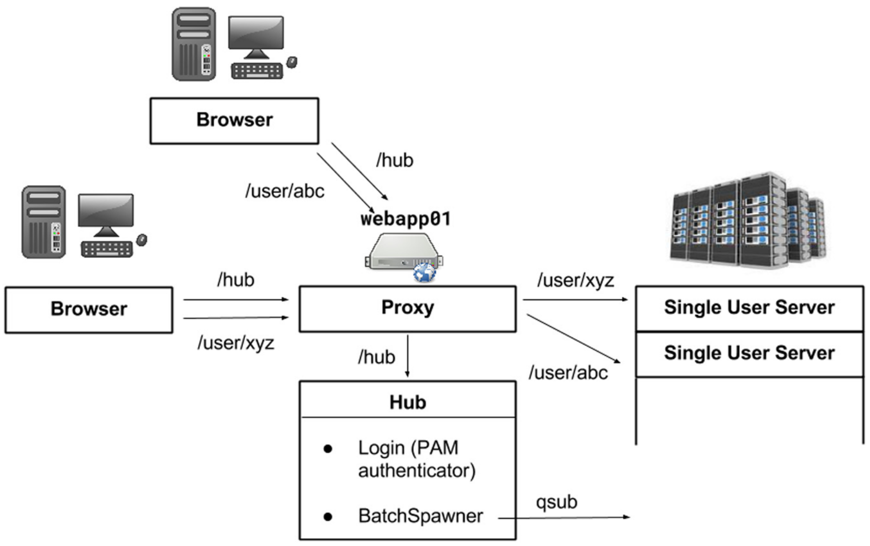


Fig. 3. Design of JupyterHub's deployment on Palmetto.

2.3. *Access platform: Jupyter server*

The above computing resources run various distributions of Linux, whose main mode of access is via a SSH connection established from a command line terminal. This presents an additional overhead of providing impromptu instructions for students and workshop participants who have not worked in a non-GUI Linux environment before. While it is possible to impose a prerequisite on having learned how to use Linux, there remains a significant technical gap between those who just pick up the skill to satisfy the prerequisite and those who have used Linux throughout their academic career. We have observed students that are either unable to keep up with class activities or frequently request assistance from instructors.

To create a consistent environment to access these computing resources, we turn to Jupyter server, a Python-based infrastructure. A Jupyter server allows users to manage their files and directories, edit text files, and launch notebooks that are essentially live documents containing executable code, visualizations, and text. All Jupyter functions are available through a browser interface. This provides a standard platform for all students, effectively lowering the barrier to entry of working with large scale computing resources.

At Clemson University, we deployed JupyterHub [35], a centralized service for users to manage Jupyter servers running on Palmetto's computing resources. Utilizing JupyterHub, we are able to develop a series of Python-based teaching materials that allow the delivery of different aspects of PDC within the context of a single interactive programming environment while minimizing potential technology-related timing overheads. This enables teachers and learners to focus more on PDC concepts and less on the various syntax and coding details. In the remainder of this section, we will briefly describe the technical configuration for this JupyterHub deployment.

2.3.1. *Deployment design*

JupyterHub is a multi-user hub for spawning, managing, and proxying multiple instances of the single-user Jupyter notebook (formerly IPython notebook) server. It has three main components: (1) a configurable http proxy, (2) a multi-user hub, and (3) multiple single-user Jupyter notebook servers (NBS(s)). The proxy is a single point of contact for using JupyterHub from a user perspective and handles all remote connections coming from users' browsers. The proxy communicates with the hub and each active user's NBS on behalf of the user. Communication between users' browsers and the proxy is secured. The hub is responsible for authenticating users and spawning their NBSs. The hub also configures the proxy to forward requests for specific URL prefixes to the correct NBS. The design of the JupyterHub deployment on Palmetto is shown in Fig. 3.

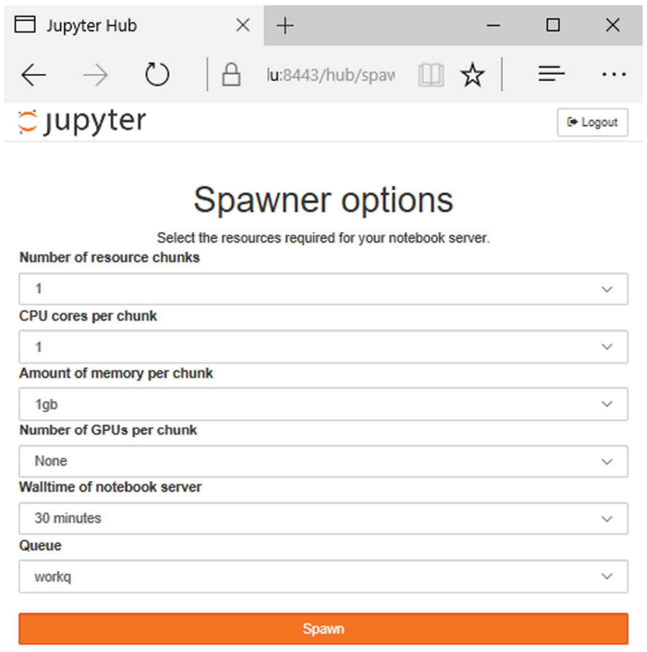
The proxy process is accessible from the Internet by users' browsers by running on a node dedicated for web applications. The hub process is run on the same node as the proxy and is configured to interface with the batch scheduler's client software on behalf of users. The hub is run as a system service and the proxy process is started when the hub service is started. JupyterHub has a modular design that allows administrators to select the plug-ins best suited for both the environment in which JupyterHub will be deployed as well as the needs of the Jupyter notebook user.

By default, the hub spawns an NBS on the same physical resources as the node running the hub. To enable the hub to interface with Palmetto and its scheduler, we utilize the BatchSpawner plug-in [35]. Using BatchSpawner, we configure the hub to spawn NBSs as batch jobs that are submitted to the Palmetto's scheduler.

2.3.2. *User interaction*

After users authenticate with the Clemson JupyterHub interface, they are presented with a web form which they may customize each time prior to spawning their NBS, as shown in Fig. 4. This enables the users to specify their resource requirements through nodes, cores, RAM, GPUs, etc. Job wall-time is one of the configurable fields, and a user's job, running an NBS, is automatically deleted once the wall-time of the job has been reached. Once the user's job begins running, the user may access their NBS, which is running on a Palmetto compute node, as shown in Fig. 5.

The Jupyter interface provides many useful features: a text editor, a terminal, the ability to traverse and manipulate a file system, and most importantly, the notebook interface. The notebook interface lets users create notebooks, which may contain: notes, program code, equations, visualizations, as well as references to images and videos that may be viewed in the notebook. The notebook is a great environment for iterative development of programs because it encapsulates both the code and its output in a single notebook file. This file can be downloaded, shared, and re-played by other users. Different programming languages are supported by Jupyter Notebooks in the form of different kernels [34]. We have



Spawner options

Select the resources required for your notebook server.

Number of resource chunks	1
CPU cores per chunk	1
Amount of memory per chunk	1gb
Number of GPUs per chunk	None
Walltime of notebook server	30 minutes
Queue	workq

Fig. 4. A form enables user customizable resource requests for NBS.

successfully configured Python version 2 and 3, R, Apache Spark, and Matlab kernels on Palmetto.

The integration of JupyterHub with Palmetto enables support for many users to have concurrent access to distributed computing resources in an interactive and streamlined manner. The Jupyter Notebook interface represents a highly enriched form of an interactive computing environment that, when coupled with Palmetto, lowers the barrier of entry for accessing computing infrastructures and learning parallel and distributed computing topics.

3. PDC course modules using unifying framework's resources

Section 2 describes the complex set of computing resources that power the unifying framework. Various technical decisions have been made in the selection and deployment of these resources. It is difficult to map each technical decision and resource to individual pedagogical concerns, as these decisions create impacts across the entire educational process. In general, they follow these two principles:

- *Reduction of irrelevant technical overhead:* For example, instead of trying to edit files through command line interfaces, Jupyter notebook allows students to make these modifications through a browser-based GUI, thus preventing unnecessary technical complications.
- *Exposure of fundamental concepts:* A traditional cluster of computers allows students to observe characteristics of parallel computations. A big data cluster allows students to understand the benefits of data locality and bringing computation to the data. Software-defined infrastructures such as CloudLab allow students to be hands-on with how different components of a distributed computing system are deployed in production.

In the remainder of this section, we discuss how PDC modules covering a wide range of topics are covered within this course by leveraging computing resources of the unifying framework. These topics are not grouped into specific areas of PDC (e.g., high performance computing, big data analytics, and distributed in-memory computing). Instead, the design and presentation of these topics highlight the order in which advances in PDC knowledge and technologies have arisen from relevant computing demands.

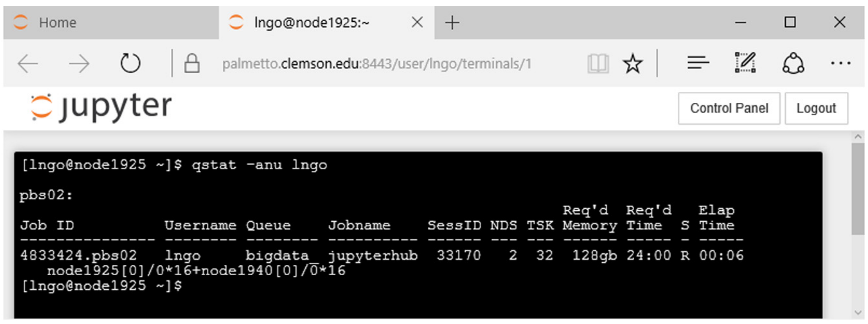
3.1. Module 1: introduction to parallel and distributed computing

This module first discusses the necessity of investing in distributed and cluster computing (*scaling out*) rather than upgrading a single computer (*scaling up*). While there are many examples demonstrating the needs for PDC, we choose to discuss a real life event, Hurricane Sandy. In this discussion, we look at how the significant weaker performance of NOAA supercomputer at that time, compared to those of other international agencies, seems to correlate with the inaccurate early prediction of Sandy's path. Next, we begin to introduce basic scaling concepts for PDC such as speedup, efficiency, and Amdahl's Law. The first assignment is a writing assignment where students are required to read and discuss the Beowulf paper [40], which provides them with definitions and motivations regarding the traditional cluster-of-computers model, where a cluster is built using off-the-shelf commodity components, and large-scale computing tasks are parallelized and separated from large-scale storage. The students are also tasked with finding and discussing other examples of the impacts of PDC in research and industry. The other two units of this module introduce students to Palmetto and CloudLab, the two computing resources that they will utilize for the remainder of the course. After a brief overview of these resources, students will use the Jupyter interface to practice accessing Palmetto and CloudLab. The second assignment in this module is a minor task involving students interacting with Palmetto and registering for accounts on CloudLab. The third assignment introduces students to the concept of parallel communication by completing the following tasks:

- Requesting a multi-node, multi-core allocation on Palmetto.
- Building a simple computer cluster on CloudLab such that this cluster has as many nodes as the total number of cores on the previous Palmetto allocation.
- Extending the Palmetto allocation script such that each individual core of the allocation contacts one node on the CloudLab cluster and requests the full domain name.

3.2. Module 2: introduction to parallel and distributed file systems

One question from Assignment 1 specifically asks students to identify potential bottlenecks in cluster computing as discussed in the Beowulf paper. The correct answer includes the Beowulf paper's discussion on the need for a parallel file system. This allows us to transition to the next module, which introduces students to Parallel and Distributed File Systems. We examine concepts such as block-based versus object-based striping, distributed locks, and meta-data servers. Unlike parallel programming, which can be assessed via a programming assignment, this type of knowledge is difficult to assess in a normal task-based assignment without resorting to using quizzes and exams to evaluate memorization. At this point in the semester, we instruct students to form teams for a semester-long project, broken into multiple stages. One task of the first stage requires each team to select one research computing site to investigate, which can be part of XSEDE or Palmetto itself. Each team will pick three sites, in order of preference, and write brief documentation describing characteristics such as operating systems, parallel file systems, and the size of the selected sites. We then assign each team to one site from their list such that each team has a unique site. This becomes the basis for subsequent project stages.



```
[lngo@node1925 ~]$ qstat -anu lngo

pbs02:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
4833424.pbs02   lngo     bigdata  jupyterhub  33170   2  32  128gb 24:00 R 00:06
   node1925[0]/0*16+node1940[0]/0*16
[lngo@node1925 ~]$
```

Fig. 5. The notebook is spawned on the user-allocated node.

3.3. *Module 3: introduction to MPI using python*

Modules 1 and 2 present students with the fundamental infrastructure for a traditional large-scale computing cluster. In Module 3, they are introduced to MPI, the parallel programming library designed to take advantage of this infrastructure. Support for MPI for Python is available through the `mpi4py` library, which provides an interface to the C-based MPI-2 specification. The syntax and semantics of this interface strongly resembles those of the original MPI library in C/C++. The Python bindings of `mpi4py` allow MPI-related PDC concepts such as pleasantly parallel, workload assignments, divide and conquer, and sorting to be implemented with less boilerplate code and fewer compilation/execution commands. This package has been proven to be a suitable teaching tool for parallel computing education [31,32].

One approach to enable parallel computing for Jupyter Notebook is to use the `ipyparallel` extension [20]. This extension allows users to create a cluster of processing engines, to which the Notebook can be connected. Engines support message passing via `mpi4py`, thus enabling users to write parallel programs entirely within the Notebook. However, in the case of deadlock errors, the process to restart `ipyparallel` is complex and error-prone. Since Fall 2016, we switched to using Jupyter's cell magic, which allows the code written within a cell to be saved to a file, and the `'!'` symbol, which allows users to define and execute Linux shell commands from within a notebook cell. In this case, the Linux shell command is the command to run the Python MPI program. This process is illustrated in Fig. 6.

Besides interactive lectures, having MPI Python code and lectures inside a Jupyter notebook allows instructors to quickly elaborate on students' questions. For example, in one class, we had a student that raised a question about what can happen when the array indices for `Scatterv` are not calculated properly. We were able to demonstrate the case of memory buffers being overwritten due to overlapping array segments, which belonged to different MPI processes. This impromptu demonstration happened instantaneously by modifying one variable and re-executing the notebook's cell without leaving the presentation mode.

In this specific module, students are first introduced to two fundamental communication patterns: point-to-point communication using Send and Recv and collective communication using Broadcast, Reduce, Gather, and Scatter. Next, students learn about common parallel workflows including pleasantly parallel and divide-and-conquer. This module has one programming assignment. The students are required to process the Google Trace Data [36] to extract and calculate counts such as number of users, number of jobs, and number of CPU hours for each job. As the trace data contains timestamped snapshots of the system, students must establish communication patterns to let processes exchange and gather all events belonging to the same job for calculation purposes. The size of the trace data is 170 GB, thus using a multi-core, multi-node allocation on Palmetto is required. As part of the assignment, students must increase their allocation requests to measure and graph speedup and efficiency. This is a large assignment that takes several weeks to complete. This module is also the basis upon which a half-day MPI introductory workshop is created for non-computer-science participants.

3.4. *Module 4: big data and data intensive computing*

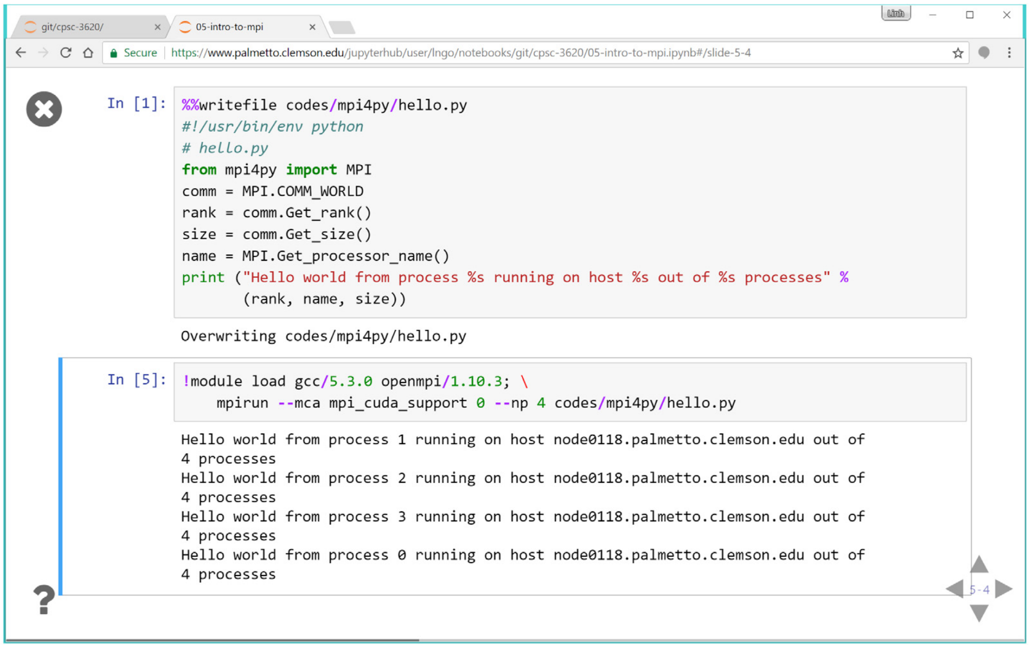
The large size of the Google Trace Data in Module 2's MPI assignment provides a pretext to illustrate the growth of interest in big data and data-intensive computing technologies since the early 2000s. This module presents students with the perceived characteristics of Big Data (Volume, Velocity, Veracity, Variety, and Value), and how network bandwidth becomes a bottleneck in processing a massive amount of data. The second lecture of this module discusses the Google File System (GFS) paper [18] and its open source counterpart, the Hadoop Distributed File System (HDFS) [39]. Attention is paid to the concept of data locality to reduce network traffic as well as the emphasis on failure recovery and resiliency in big data infrastructures. We have multiple-choice quizzes that assess students on these concepts but no assignment for this module.

3.5. *Module 5: MapReduce programming paradigm*

The design of GFS/HDFS does not support standard point-to-point communication. Collective communication is facilitated without manual management from programmers. As a result, the standard MPI programming paradigm does not work, and students are introduced to a new programming paradigm, MapReduce [13].

Previously, there were two infrastructures that could be leveraged to teach MapReduce at Clemson University. The first was a 16-node persistent Hadoop infrastructure accessible only through a dedicated user node and networked with the existing Palmetto cluster. The infrastructure supported several regular users, which made it only suitable for a half-day MapReduce workshop with a small number of attendees. Academic courses relied on an extended implementation of myHadoop [23] to facilitate the deployment of dynamic Hadoop clusters on Clemson University's Palmetto Supercomputer for students in the class. For the half-day workshops, Python and Jupyter were already used to teach MapReduce concepts, but attendees were required to launch a local Jupyter server from their allocated resource on Palmetto and then access that server via X11 tunneling. For the modules taught in academic courses, students are used to Java as the default language; however, much effort is spent writing, compiling, and debugging Java programs. The important MapReduce workflows are often obscured within a larger amount of boiler-plate code overhead required when working with Java.

The unifying framework has enabled the consolidation of MapReduce-related contents for both the half-day workshop and


git/cpsc-3620/
05-intro-to-mpi
Secure
https://www.palmetto.clemson.edu/jupyterhub/user/lngo/notebooks/git/cpsc-3620/05-intro-to-mpi.ipynb#/slide-5-4
In [1]:
In [5]:
5-4

```
%%writefile codes/mpi4py/hello.py
#!/usr/bin/env python
# hello.py
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
name = MPI.Get_processor_name()
print ("Hello world from process %s running on host %s out of %s processes" %
       (rank, name, size))
```

```
Overwriting codes/mpi4py/hello.py
```

```
!module load gcc/5.3.0 openmpi/1.10.3; \
    mpirun --mca mpi_cuda_support 0 --np 4 codes/mpi4py/hello.py
```

```
Hello world from process 1 running on host node0118.palmetto.clemson.edu out of
4 processes
Hello world from process 2 running on host node0118.palmetto.clemson.edu out of
4 processes
Hello world from process 3 running on host node0118.palmetto.clemson.edu out of
4 processes
Hello world from process 0 running on host node0118.palmetto.clemson.edu out of
4 processes
```

Fig. 6. Running a simple parallel program on Jupyter Notebook.

academic modules. Through the new Jupyter interface, the content of this MapReduce module has been developed [10] based on previous work [29] and converted to Python. Python's map and reduce functions are used to explain the concepts in MapReduce. The development of MapReduce workflows is demonstrated by leveraging Hadoop Streaming, where mapper and reducer tasks can be implemented as Python scripts. To eliminate redundant creation of sampling data, which is typically the case when Python-based implementations of mappers and reducers are to be tested in isolation on non-Hadoop platforms, Hadoop's `hdfs dfs -cat` command and Linux's pipe (|), `sort`, and `head` calls are utilized. Through this combination, we can demonstrate the creation of Key/Value pairs, sorting of the pairs, and final reduction of values belonging to the same keys. The final validation on the Hadoop infrastructure is straightforward because the Python scripts can be reused as is by Hadoop's *yarn jar hadoop-streaming.jar* call as mapper and reducer functions. Fig. 7 shows an example Jupyter notebook where students can perform a correctness test by using *grep* on the mapping phase and then validate the results with the outcomes after the reducing phase.

The contents for this module are developed inside Jupyter notebooks, enabling seamless transitions between lecture notes and code examples. The notebooks allow students to follow instructions, observe live output recorded in the notebooks, and repeat the steps on their own. One important challenge in working with the Hadoop-based platform is learning how to debug error messages which are often hidden within a massive amount of log information. Analyzing MR logs in a traditional command-line setting is nearly impossible due to the typical length of the logs, which are often automatically scrolled off screen. A notebook's cell can enable scrolling for its output windows, allowing instructors to slowly and methodically go through contents of the log and show students where to find the relevant error messages and how to interpret them. This is demonstrated in Fig. 8.

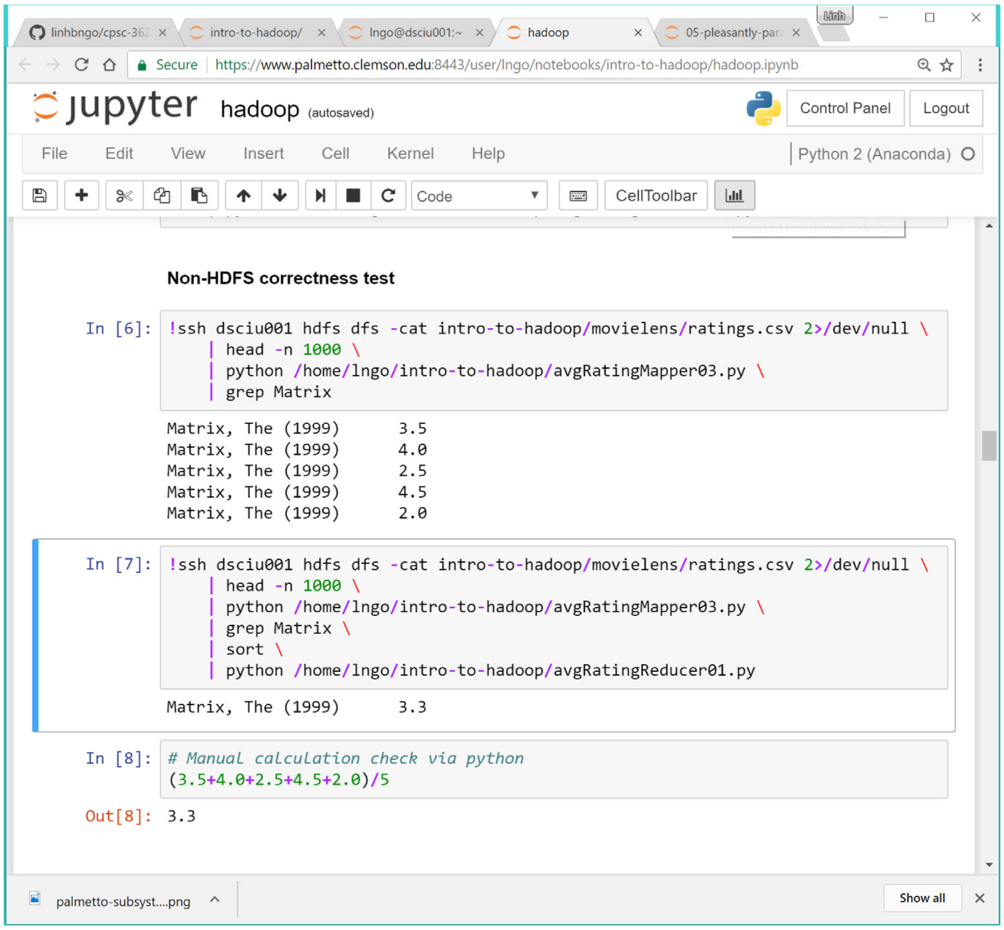
The assignment for this module reuses the Google Trace Data and the questions of the MPI module; however, students are required to implement the solutions using the MapReduce programming paradigm. This serves two purposes. First, students will be able to apply their understanding about MapReduce in actual coding practice. Second, this highlights the trade-offs between message passing and MapReduce: more convenient programming capability at the loss of communication and workflow control. The second stage of the semester project is also assigned at the end of this module. Each team is tasked with creating a mock-up design of their assigned cluster computing site and deploy this design on CloudLab. Typical components of a cluster of computers are required. These include a login node, a set of computing nodes, a NFS server to support a shared `/home` file system, and a parallel file system. For the parallel file system, each team is required to install and configure the corresponding file system used at their assigned site. The teams are expected to automate the deployment process via Python scripts that use CloudLab's `geni-lib`, an open source Python library. Through this stage, students become familiar with the administrative side of PDC, including design, deployment, and configuration of PDC systems.

3.6. Module 6: parallel I/O and cluster scheduling

This module contains two lectures focusing on parallel I/O for MPI and scheduling of computing clusters. We consider the topic of cluster scheduling to be more appropriate, and parallel I/O can be an optional part of this specific course because parallel I/O for MPI is considered an advanced topic that is covered in another technical elective course focusing entirely on MPI programming. This technical elective course is cross-listed for both the departments of Electrical and Computer Engineering (ECE) and School of Computing (SoC) at Clemson University. As a result, we pay more attention to aspects of job scheduling, a critical component of any large-scale computing cluster. These lectures also provide students with the background knowledge to implement an optional stage of the semester project, which is to install and configure the scheduler used by the assigned site. Examples of schedulers set up by teams include PBS [19], SLURM [43], and Moab [14].

3.7. Module 7: in-memory distributed computing

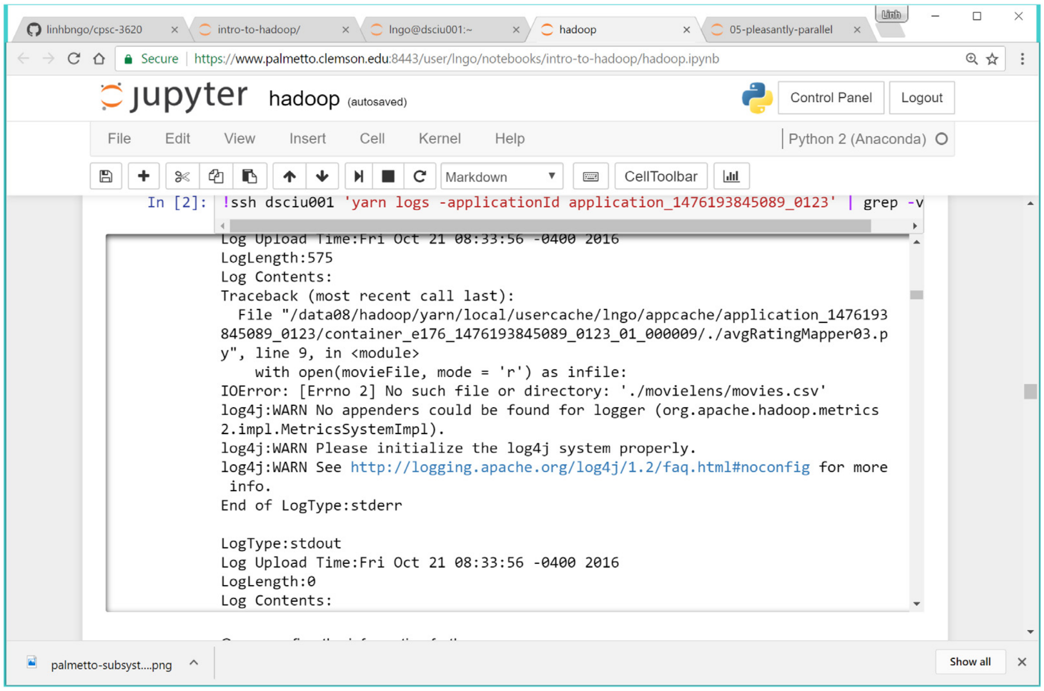
This module consists of one lecture and two hands-on labs. In the previous transition between the MPI and MapReduce lectures, we demonstrated how real world demand for big data processing has led to new PDC infrastructures and programming


linhbngo/cpsc-36
intro-to-hadoop/
lngo@dsciu001:~
hadoop
05-pleasantly-par
Secure | https://www.palmetto.clemson.edu:8443/user/lngo/notebooks/intro-to-hadoop/hadoop.ipynb
jupyter hadoop (autosaved)
Control Panel
Logout
File Edit View Insert Cell Kernel Help
Python 2 (Anaconda)
Code
CellToolbar
palmetto-subsyst....png
Show all

Non-HDFS correctness test

```
In [6]: !ssh dsciu001 hdfs dfs -cat intro-to-hadoop/movielens/ratings.csv 2>/dev/null \
            | head -n 1000 \
            | python /home/lngo/intro-to-hadoop/avgRatingMapper03.py \
            | grep Matrix

        Matrix, The (1999)      3.5
        Matrix, The (1999)      4.0
        Matrix, The (1999)      2.5
        Matrix, The (1999)      4.5
        Matrix, The (1999)      2.0

In [7]: !ssh dsciu001 hdfs dfs -cat intro-to-hadoop/movielens/ratings.csv 2>/dev/null \
            | head -n 1000 \
            | python /home/lngo/intro-to-hadoop/avgRatingMapper03.py \
            | grep Matrix \
            | sort \
            | python /home/lngo/intro-to-hadoop/avgRatingReducer01.py

        Matrix, The (1999)      3.3

In [8]: # Manual calculation check via python
        (3.5+4.0+2.5+4.5+2.0)/5

Out[8]: 3.3
```

Fig. 7. Execution of Linux commands inside Jupyter notebook to validate implementations of mapper and reducer via HDFS data pipelining.



```
In [2]: !ssh dsciu001 'yarn logs -applicationId application_1476193845089_0123' | grep -v

        Log Upload Time:Fri Oct 21 08:33:56 -0400 2016
        LogLength:575
        Log Contents:
        Traceback (most recent call last):
          File "/data08/hadoop/yarn/local/usercache/lngo/appcache/application_1476193
        845089_0123/container_e176_1476193845089_0123_01_000009/./avgRatingMapper03.p
        y", line 9, in <module>
            with open(movieFile, mode = 'r') as infile:
        IOError: [Errno 2] No such file or directory: './movielens/movies.csv'
        log4j:WARN No appenders could be found for logger (org.apache.hadoop.metrics
        2.impl.MetricsSystemImpl).
        log4j:WARN Please initialize the log4j system properly.
        log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more
         info.
        End of LogType:stderr

        LogType:stdout
        Log Upload Time:Fri Oct 21 08:33:56 -0400 2016
        LogLength:0
        Log Contents:
```

Fig. 8. A Hadoop MapReduce log is filtered to remove redundant INFO messages, and the remaining contents are scrollable within Jupyter notebooks' output cell, showing the specific error.

paradigms. Similarly, the lecture on in-memory distributed computing demonstrates how demands for faster big data processing and more complex workflows have led to various technological advances in PDC. The first *half-step* is Tez [38], a framework that supports chaining multiple MapReduce jobs together to improve complexity and avoid overhead in writing intermediate data to hard drives. Next, Spark [44] and its successor, Tachyon [25] completely re-implement the MapReduce programming paradigm to take advantage of in-memory distributed processing.

First presented at USENIX in 2010, Spark was created to address a subset of data flow programming applications that are not suitable to be implemented in Hadoop MapReduce: iterative jobs and interactive analytics. The strength of Spark comes from its design and implementation of resilient distributed datasets (RDD), immutable data structures that are distributed across a cluster. The core libraries of Spark allow users to invoke parallel operations such as map, reduce, join, and filter on the RDDs. RDDs can be cached in memory, resulting in significant improvement in performance for iterative and interactive operations. At the same time, operations on a RDD can be lazily evaluated and the lineage of operations is maintained. This helps with reducing memory usage and ensuring the ability to recreate RDDs in the event of system failures. The potential of Spark as a big data analytics programming framework has since been recognized by academia and industry, with the open source version quickly becoming a top-level Apache Software Foundation project [4].

We first introduced in-memory distributed computing into the materials covered in CPSC 3620 as a single lecture in Fall 2014. Since Spring 2015, a full module with one lecture and one lab has been included in the curriculum for the class. In Spring 2015 and Fall 2015, the Spark lab has been taught using Scala, the native language API for Spark. While both Java and Python APIs of Spark have been available during the aforementioned time periods, the lack of overhead boiler-plate code and the interactive capability and maturity of the Spark shell, written in Scala, reduces the steps needed for lab exercises and technical support. On the other hand, as Scala is a new programming language that is not taught in the standard computer science curriculum, we cannot justify assigning students any significant homework assignments. In Spring 2016, we made the transition to using Spark's Python API in combination with a single-user version of Jupyter. Leveraging Spark's integration into Hadoop YARN, the students could reuse the dynamic Hadoop deployment on Palmetto [29] and spawn a Jupyter notebook with a Spark kernel which connected to a Spark cluster on top of the dynamic Hadoop cluster. To interact with the notebook, the students needed to launch a browser from inside Palmetto via X11 tunneling. The combination of the Python API and Jupyter allowed for a more intuitive and understandable demonstration of programming flows. The technical aspects of setting up X11 tunneling and correct creation of the Spark kernels in Jupyter took up a significant portion of the lab, resulting in two lab sessions instead of one.

With the addition of the JupyterHub interface on Palmetto, we retain the ability to teach Spark in an interactive manner via Python while not having to deal with the issue of setting up an individual Jupyter server for each student. As the demonstration tools and labs become more streamlined, more content has been added to the module, assisting with the teaching of concepts such as data lineage, immutability, lazy-evaluation, and storage level. Similar to MPI and Hadoop MapReduce, Jupyter notebooks allow the integration of lecture contents with interactive code, as shown in Fig. 9.

The assignment of this module once again reused the Google Trace Data and the question set from the previous assignments. Students now can clearly see the advantage of having a more flexible MapReduce workflow and faster computing time due to in-memory processing. At the same time, they learn to struggle with requesting enough compute nodes to accommodate the required memory space. This set of assignments that use different programming paradigms to solve the same problem also serves to remind students to not become attached to a specific technological solution but to pick the right tools for the job at hand.

3.8. Module 8: other PDC infrastructures

At this time in the semester, we plan to leave students more time to work on the third stage of the team project, which is to correct misconfiguration from the second stage, perform evaluation and validation on the final cluster deployment, and write the final report. As a result, this module covers lectures on various PDC infrastructures and concepts but has no assignment.

The first lecture in this module discusses HPCCSystems, a proprietary-turned-open-source data processing framework [27]. HPCCSystems also supports big data processing using data locality I/O, but it focuses specifically on enterprise data, which are typically structured tabular formats. HPCCSystems has the same set of functional components as Hadoop. In this lecture, students observe how the concept of data locality can be implemented in a different approach, and how usability and functionality of HPCCSystems come at the cost of fault-tolerance and reliability. This lecture also comes with a hands-on lab. In previous semester, we had to rely on an educational allocation from HPCCSystems' parent company due to difficulty when trying to deploy HPCCSystems in Palmetto's user space [30]. With the framework, students are able to deploy their individual HPCCSystems clusters on CloudLab via Python scripting.

In the second lecture, we discuss a middleware infrastructure that is related to the velocity characteristics of big data: message-oriented middleware. Students understand the upcoming deluge of data due to streaming sensors and the need to have an infrastructure that can quickly deliver streams of data to processing centers. Attributes such as coupling, scalability, availability, and reliability are discussed regarding different approaches to message-oriented middleware implementations.

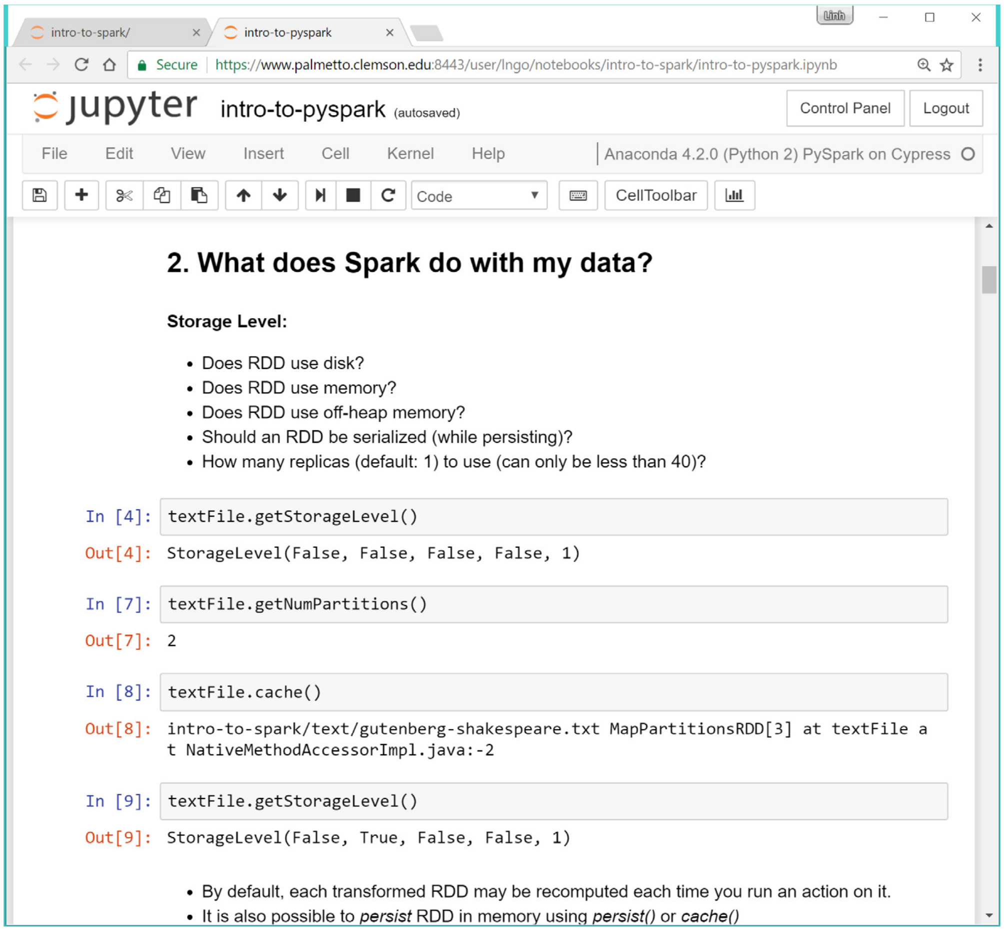
The third lecture discusses PDC approaches to processing streaming data in real time. Here, we demonstrate Spark's ability to support real-time/near-real-time data processing due to its in-memory design. Students are introduced to concepts such as mobility, availability, processing guarantee, partitioning, querying, deterministic versus non-deterministic processing, storage, and handling of imperfect data. Students also learn about the lambda architecture [26] where streaming processing and large-scale batch computation are combined into a distributed infrastructure.

Finally, students are introduced to the concepts of cloud computing. They will have a chance to understand the nuisance of setting up an infrastructure such as CloudLab which allows users to quickly deploy computing clusters on remote hardware components using virtual images.

4. Results and feedback

In this section, we discuss the performance and feedback from students and instructors about the usage of the unifying framework in several workshops and academic courses. The academic courses are offered during Fall 2016 (36 students) and Spring 2017 semesters (60 students). More than a dozen workshops have been offered during this time period; each workshop had an average of 20 participants.

As discussed in Section 2, the JupyterHub server acts as a proxy from which users can specify resource requirements and submit allocation requests to the Palmetto Supercomputer. The actual Jupyter NBS is only launched after the allocation is granted, and


intro-to-spark/
intro-to-pyspark
Secure
https://www.palmetto.clemson.edu:8443/user/lngo/notebooks/intro-to-spark/intro-to-pyspark.ipynb
jupyter
intro-to-pyspark (autosaved)
Control Panel
Logout
File
Edit
View
Insert
Cell
Kernel
Help
Anaconda 4.2.0 (Python 2) PySpark on Cypress
Code
CellToolbar

2. What does Spark do with my data?

Storage Level:

- Does RDD use disk?
- Does RDD use memory?
- Does RDD use off-heap memory?
- Should an RDD be serialized (while persisting)?
- How many replicas (default: 1) to use (can only be less than 40)?

```
In [4]: textFile.getStorageLevel()
Out[4]: StorageLevel(False, False, False, False, 1)

In [7]: textFile.getNumPartitions()
Out[7]: 2

In [8]: textFile.cache()
Out[8]: intro-to-spark/text/gutenberg-shakespeare.txt MapPartitionsRDD[3] at textFile a
        t NativeMethodAccessorImpl.java:-2

In [9]: textFile.getStorageLevel()
Out[9]: StorageLevel(False, True, False, False, 1)
```

- By default, each transformed RDD may be recomputed each time you run an action on it.
- It is also possible to *persist* RDD in memory using *persist()* or *cache()*

Fig. 9. Interleaving Spark's storage level concept with code demonstration in Jupyter notebook.

the NBS instance resides on the allocated compute node. As a result, the JupyterHub and proxy servers are unlikely to suffer from overloading if many users attempt to connect at the same time, which is the typical case in a classroom setting. The infrastructure currently supports 334 users (out of approximately 800 active accounts on the Supercomputer), and there are approximately thirty to forty active Jupyter servers running at any give time. During Fall 2016, when the class had the highest enrollments (65 students), we had in-class lab sessions where all students were able to deploy concurrent Jupyter servers without any performance degradation on the system. For institutions without local research computing access, the Jetstream resource of XSEDE provides virtual images with support for Jupyter servers and other educational tools at scale [16].

JupyterHub has been used as a regular teaching platform for CPSC 3620 since the beginning of the Fall 2016 semester, with more than forty students utilizing the platform to interact with Palmetto. Feedback from students and instructors has been positive. The most significant impact that comes from using JupyterHub is the disappearance of technical issues due to variation in students' computing devices. This allows instructors to spend more time lecturing and less time debugging. Students do not feel frustrated as they did previously when they were not able to get coding instructions to work properly on their own computers. In fact, additional content has been added to several workshops (debugging techniques for Hadoop MapReduce and SQLContext for Apache Spark) without increasing workshop duration because of the significant reduction in time overhead. While transitioning to teaching MPI and Hadoop in Python requires the redevelopment of materials, this effort is manageable as Python only acts as a wrapper and the Python syntax closely mirrors the native languages' counterparts. It has been found that all PDC concepts such as deadlocks, speedup, horizontal/vertical scaling, scheduling, broadcast trees, and peer-to-peer communication covered in previous semesters can still be taught and demonstrated in the new versions of the lectures. The integration of the RISE plugin [6] has been well received by instructors, as it lets instructors create highly interactive notebooks that are essentially slide decks that seamlessly weave between live code demonstrations and theoretical lecture notes.

All students have responded positively to having Palmetto and CloudLab for this course, citing the ability to work with realistic research computing environments. During Spring 2017, we have provided students with anonymous entry and exit surveys. The first set of questions ask students to rank their comfortability with various PDC tools and technologies in the class between 1 and 10, with 1 being not comfortable at all and 10 being most comfortable. We use this term to replace terms such as "level of proficiency" and "level of expertise" when asking students to self-assess their technical capabilities in entry and exit surveys. This decision is supported by findings from previous work that examines students' self-assessment skill. For example, it has been shown that the level of self-assessment carries a "weak to moderate accuracy" and that this accuracy correlates with students' academic performance [24]. As the course is required, all students must register. As a result, the wide range of students' academic competencies will negatively influence this accuracy level. While the class is attended mostly by seniors, it has been demonstrated that self-assessment accuracy has not improved over time [15]. Furthermore, it is unrealistic to

Table 1
Degree of comfortability at the beginning and at the end of Spring 2017.

PDC Tools/Concepts	Begin	End
Python	4.00	6.40
Linux	7.38	7.84
Parallel Programming	3.53	5.82
MPI	1.72	5.38
MapReduce	1.35	6.21
Spark	1.25	4.61

expect significant improvements in students' technical expertise, especially given the complexity of materials covered in this class. Therefore, instead of "expertise" or "proficiency", we use the term "comfortability" to ask the students to measure how comfortable they are in their interaction with a specific technology. How comfortable students are with certain technologies can be interpreted as the ease at which students approach these technologies for learning purposes. The averaged responses regarding degree of comfortability are shown in Table 1. The responses demonstrate a clear improvement in students' affinity toward MPI, MapReduce, Spark, and parallel programming in general. Having Python as the primary language throughout the course also leads to improvement. Most students seem to be comfortable with the Linux environment, and throughout the course, their comfort level only improves slightly.

The introductory survey also asks students to describe their experience working with clusters of computers. Out of all the responses, 14% have only used single workstations, 64% have used a single compute node that is part of a cluster via a Linux terminal, 19% have deployed software on multiple machines that are part of the same network, 2% have administered a network of computers, and 2% have deployed, configured, and administered a network of computers in a production environment. Given this lack of previous expertise, the usage of CloudLab in the semester project to design and deploy a cluster of computers has provided positive impacts on students. The exit survey shows that most students participate in the design and deployment of the cluster over the stages of the project. Only 5 students chose to work with research and documentation, and 3 students limited their participation to running test codes on the final cluster. Regarding the difficulty of the project, only 5 students thought the project was too difficult, while 37 thought the project required significant contribution from all members, and 15 thought it was just the right size for a team project. Most students also thought that using CloudLab and Palmetto can benefit other CS courses such as networking, operating systems, Linux administration, computer security, and software engineering. There were complaints, but they mainly concerned difficulties in applying PDC concepts and learning new technologies, and we have not received any complaints about being unable to utilize computing resources and the Jupyter interface of the framework.

Other feedback about the framework comes from guest lectures for advanced undergraduate/graduate classes that use new content and lecturing methods based on JupyterHub. In these cases, students that have taken CPSC 3620 in previous semesters expressed disappointment that the tools were not available to them before. The participants of the half-day workshops on MPI and Hadoop are mainly graduate students with non-computer science backgrounds, and they are able to quickly navigate the platform and perform exercises, even for those who have had little to no exposure to Linux and Python. Similarly, undergraduate students in CPSC 3620 are able to utilize the platform to connect to the Palmetto Supercomputer and to write and submit jobs to the supercomputer.

5. Conclusion and future work

We have deployed a unifying educational framework that combines on-site and remote computing resources with a common web-based access interface to provide a seamless approach to working with large-scale computing resources. Leveraging this framework, we have been able to utilize Python and its various collections of wrapper libraries to create a common platform upon which various PDC concepts and technologies can be taught. This allows students to focus more on the underlying principles rather than the complexity of various technologies. The feedback from instructors and students using the platform have been positive due to the new advantages such as reduction in required technical support, streamlined syntaxes, and the ability to create lecture slides with an embedded live coding interface for Jupyter notebooks. One important thing about our framework is that it can still work for institutions that do not have on-site computing resources. In this case, the JetStream site of XSEDE can provide a production cloud-based environment to host virtual machines with the Jupyter server software already installed and enabled. Instructors can leverage this cloud-based Jupyter interface to interact with other XSEDE resources (instead of having an on-site resource like Palmetto) and CloudLab.

Leveraging this framework, we demonstrate the ability to deliver a set of comprehensive educational modules describing most, if not all, critical aspects of parallel and distributed computing within the scope of a single academic course. The amount of knowledge covered is significant, and is only possible through the support of this framework. Through publicly available computing resources and open source software, this framework can be recreated at any academic institution, enabling complete or in-part adoption of the course's content.

A limitation of this work is the scarcity of quantitative measurements of students' skill improvements. Since the course is a required junior-level class, by the time students register for the course, most, if not all of them have built up their specialty that might not provide an adequate technical foundation for this course. Therefore, assessments through assignments, exams, and other summative methods can unfairly represent those that are not interested in a career path that involves distributed systems. In future work, we will explore the concept of "comfortability" from the perspective of the connectivism theory of learning [22]. All materials, including lectures and assignments, are made available online through a GitHub repository [28]. As the modules are updated frequently to keep up with changes in PDC technologies, the master branch of this repository contains the latest version, while the other branches store materials from previous semesters.

Acknowledgments

We are grateful for the support from NSF MRI Award # 1228312. We would like to acknowledge the assistance from Corey Ferrier and Xizhou Feng, Clemson Computing and Information Technology, in setting up and maintaining the hardware for JupyterHub. We also appreciate technical assistance from Carol Willing, Michael Milligan, and Min Ragan-Kelley of Project Jupyter.

References

[1] Apache Hadoop, Apache Hadoop Source Code Repository, 2016. https://github.com/apache/hadoop-common.
[2] Apache Hadoop 2.0: YARN (Yet Another Resource Negotiator), 2013. hadoop.apache.org/docs/current2/hadoop-yarn.
[3] Apache HBase, 2013, http://hbase.apache.org.
[4] Apache Spark, Lightning-Fst Cluster Computing, 2016. http://spark.incubator.apache.org.
[5] Apache Tez, A framework for near real-time big data processing, 2013. http://hortonworks.com/hadoop/tez.

[6] D. Avila, Reveal.js - Jupyter/IPython Slideshow Extension, 2016. https://github.com/damianavila/RISE.
[7] M. Berman, J.S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, I. Seskar, GENI: A federated testbed for innovative network experiments, Comput. Netw. 61 (2014) 5–23.
[8] R. Brown, E. Shoop, J. Adams, C. Clifton, M. Gardner, M. Haupt, P. Hinsbeeck, Strategies for preparing computer science students for the multicore world. In: Proceedings of the 2010 ITiCSE Working Group Reports, 2010.
[9] R.M. Butler, R.E. Eggen, S.R. Wallace, Introducing parallel processing at the undergraduate level, in: ACM SIGCSE Bulletin, vol. 20, ACM, 1988, pp. 63–67.
[10] Clemson CITI, Introduction to Hadoop, 2016. http://clemsonciti.github.io/hadoop-python-01-workshop/.
[11] Cyberinfrastructure and Technology Integration, 2017. https://github.com/clemsonciti.
[12] L. Dalcin, MPI for Python 2.0.0 Documentation, 2016. https://pythonhosted.org/mpi4py/usrman/tutorial.html.
[13] J. Dean, S. Ghemawat, MapReduce: Simplified data processing on large clusters, Commun. ACM 51 (1) (2008).
[14] Y. Etsion, D. Tsafrir, A Short Survey of Commercial Cluster Batch Schedulers, School of Computer Science and Engineering, the Hebrew University of Jerusalem 44221, 2005, 2005–13.
[15] K.W. Eva, J.P. Cunnington, H.I. Reiter, D.R. Keane, G.R. Norman, How Can I Know What I Don't Know? Poor Self Assessment in a Well-defined Domain, Adv. Health Sci. Educ. 9 (3) (2004) 211–224.
[16] J. Fischer, D.Y. Hancock, J.M. Lowe, G. Turner, W. Snapp-Childs, C.A. Stewart, Jetstream: A cloud system enabling learning in higher education communities, in: Proceedings of the 2017 ACM Annual Conference on SIGUCCS, ACM, 2017, pp. 67–72.
[17] P. Garrity, T. Yates, R. Brown, E. Shoop, WebMapReduce: An accessible and adaptable tool for teaching map-reduce computing, in: Proceedings of the ACM SIGCSE, 2011.
[18] S. Ghemawat, H. Gobioff, S.-T. Leung, The Google File System, in: ACM SIGOPS Operating Systems Review, 2003.
[19] R.L. Henderson, Job scheduling under the portable batch system, in: Workshop on Job Scheduling Strategies for Parallel Processing, Springer, 1995, pp. 279–294.
[20] IPython Parallel, Using IPython for Parallel Computing, 2009. https://github.com/ipython/ipyparallel.
[21] D.A. Joiner, P. Gray, T. Murphy, C. Peck, Teaching parallel computing to science faculty: best practices and common pitfalls, in: Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2006.
[22] M. Kathleen Dunaway, Connectivism: Learning theory and pedagogical practice for networked information landscapes, Ref. Serv. Rev. 39 (4) (2011) 675–685.
[23] S. Krishnan, M. Tatineni, C. Baru, myHadoop - Hadoop-On-Demand on Traditional HPC Resources, San Diego Supercomputing Center, 2011.
[24] M.D. Lew, W. Alwis, H.G. Schmidt, Accuracy of students' self-assessment and their beliefs about its utility, Assessment Eval. Higher Educ. 35 (2) (2010) 135–156.
[25] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, I. Stoica, Tachyon: Reliable, memory speed storage for cluster computing frameworks, in: Proceedings of the ACM Symposium on Cloud Computing, ACM, 2014, pp. 1–15.
[26] N. Marz, J. Warren, Big Data: Principles and Best Practices of Scalable Realtime Data Systems, Manning Publications Co., 2015.
[27] A. Middleton, P. Solutions, HPCC systems: Introduction to HPCC (high-performance computing cluster), White Paper, LexisNexis Risk Solutions, 2011.
[28] L.B. Ngo, CPSC 3620: Distributed and Cluster Computing, 2017. https://github.com/linhbngo/cpsc-3620.
[29] L.B. Ngo, E.B. Duffy, A.W. Apon, Teaching HDFS/MapReduce systems concepts to undergraduates, in: Parallel & Distributed Processing Symposium Workshops, IPDPSW, 2014 IEEE International, IEEE, 2014, pp. 1114–1121.
[30] L.B. Ngo, M.E. Payne, F. Villanustre, R. Taylor, A.W. Apon, Dynamic provisioning of data intensive computing middleware frameworks: A case study, in: Proceedings of the 1st Workshop on the Science of Cyberinfrastructure: Research, Experience, Applications and Models, ACM, 2015, pp. 3–10.
[31] E. Orozco, R. Arce-Nazario, J. Ortiz-Ubarri, H. Ortiz-Zuazaga, A curricular experience with parallel computational thinking: A four years journey, in: EduPDHPC. Workshop on Education for High Performance Computing, 2013.
[32] J. Ortiz-Ubarri, R. Arce-Nazario, Modules to teach parallel computing using python and the littlefe cluster, in: The Int. Conference for High Performance Computing, Networking, Storage and Analysis, 2013.
[33] S.K. Prasad, A. Chtchelkanova, F. Dehne, M. Gouda, A. Gupta, J. Jaja, K. Kant, A.L. Salle, R. LeBlanc, A. Lumsnaide, D. Padua, M. Parashar, V. Prasanna, Y. Robert, A. Rosenberg, S. Sahni, B. Shirazi, A. Sussman, C. Weems, J. Wu, NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing - Core Topics for Undergraduates, 2017 http://www.cs.gsu.edu/~tcpp/curriculum/index.php.
[34] Project Jupyter, IPython Kernels for Other Languages, 2016. https://github.com/ipython/ipython/wiki/IPython-kernels-for-other-languages.
[35] Project Jupyter, JupyterHub, 2016. https://github.com/jupyterhub/jupyterhub.
[36] C. Reiss, J. Wilkes, J.L. Hellerstein, Google Cluster-usage Traces: Format+ Schema, Google Inc., White Paper, 2011, pp. 1–14.
[37] R. Ricci, E. Eide, Introducing cloudlab: Scientific infrastructure for advancing cloud architectures and applications, Login: Mag. USENIX & SAGE 39 (6) (2014) 36–38.
[38] B. Saha, H. Shah, S. Seth, G. Vijayaraghavan, A. Murthy, C. Curino, Apache Tez: A unifying framework for modeling and building data processing applications, in: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, ACM, 2015, pp. 1357–1369.
[39] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The hadoop distributed file system, in: Mass Storage Systems and Technologies, MSST, 2010 IEEE 26th Symposium on, IEEE, 2010, pp. 1–10.
[40] T. Sterling, D. Becker, M. Warren, T. Cwik, J. Salmon, B. Nitzberg, An Assessment of beowulf-class computing for NASA requirements: Initial findings from the first NASA workshop on beowulf-class clustered computing, in: Aerospace Conference, 1998 IEEE, vol. 4, IEEE, 1998, pp. 367–381.
[41] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G.D. Peterson, et al., XSEDE: Accelerating scientific discovery, Comput. Sci. Eng. 16 (5) (2014) 62–74.
[42] G. Wolffe, C. Trefftz, Teaching parallel computing: New possibilities, J. Comput. Sci. Coll. 25 (1) (2009) 21–28.
[43] A.B. Yoo, M.A. Jette, M. Grondona, Slurm: Simple linux utility for resource management, in: Workshop on Job Scheduling Strategies for Parallel Processing, Springer, 2003, pp. 44–60.
[44] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, Spark: Cluster computing with working sets, HotCloud 10 (2010) 10–10.

Linh Bao Ngo received the Ph.D. degree in Computer Science & Engineering from University of Arkansas of Fayetteville in 2012. After graduation, Dr. Ngo was a Research Associate at the Big Data Systems Lab, Computer Science Division at Clemson University, where he also served as the Deputy Lab Director until 2015 and subsequently became a research assistant professor for the Computer Science Division. Dr. Ngo's research focus is on the design, deployment, analysis and evaluation of distributed systems and big data system infrastructures and computer science education. Since November 2015, Dr. Ngo has also been appointed as the Director of Data Science at the Cyberinfrastructure and Technology Integration Group at Clemson University.

Ashwin Shrinath holds a Master degree in Mechanical Engineering from Clemson University. He is also a certified instructor for Software Carpentry and Data Carpentry. Shrinath is currently working as a research facilitator for the Cyberinfrastructure and Technology Integration (CITI group) at Clemson University.

Jeffrey Denton holds a Bachelor of Science degree in Computer Science from Clemson University. Since 2015, Denton has worked as a research facilitator for the Cyberinfrastructure and Technology Integration (CITI group) at Clemson University. He also works as a consultant for Omnibond, the company behind the open-source parallel file system OrangeFS.

Marcin Ziolkowski received his Ph.D. degree in Chemistry from Northwestern University in 2013. He became a research facilitator at the Cyberinfrastructure and Technology Integration group at Clemson University in 2014 and was appointed Interim Director in 2016. Dr. Ziolkowski is interested in advancing computational sciences via design and installation of advanced cyberinfrastructure and support of high performance computing education.