Taking a Walk on the Wild Side: Teaching Cloud Computing on Distributed Research Testbeds

ABSTRACT

Distributed platforms are now a de facto standard in modern software and application development. Although the ACM/IEEE Curriculum 2013 introduces Parallel and Distributed Computing as a first class knowledge area for the first time, the right level of abstraction to teach these concepts is still an important question that needs to be explored. This work presents our findings in teaching cloud computing by exposing upper-level students to testbeds in use by the distributed systems research community. The possibility of giving students practical and relevant experience was explored in the context of new course assignment objectives. Furthermore, students were able to significantly contribute to a pilot class project with medium-scale computation based on satellite data. However, the software engineering challenges in these environments proved to be daunting. In particular, these challenges were exacerbated by a lack of debugging support relative to the environments students were more familiar with—requiring development practices that out-stripped typical course experiences. Our proposed set of experiments and project provide a basis for an evaluation of the trade-offs of teaching cloud and distributed systems *on the wild side*. We hope that these findings provide insight into some of the possibilities to consider when preparing the next generation of computer scientists to engage with software practices and paradigms that are already fundamental in today's highly distributed systems.

Keywords

Distributed systems, experimental facilities, course design

1. INTRODUCTION

Traditional simple client-server architecture supports many of the conveniences affordable to popular applications. More complex peer-to-peer style infrastructure not only lies at the heart of many content distribution networks, but also within all devices housing multiple processing elements. Over the past decade, computer systems have been dominated by large-scale, highly-parallel, loosely-coupled systems. Common examples include parallel computation frameworks such as MapReduce [17] and Hadoop [22], Distributed Hash Tables [20], key-value stores [21], to take a few examples among many.

A key challenge for computer science educators is teaching our students how to construct and use these systems. As an example, today's generation of students should be able to launch many types of MapReduce jobs, and help design servers such as Google's search, which processes vast quantities of information to service queries in milliseconds. There is tremendous commercial interest in systems like these; social-networking and Internet information provider firms (Google, Facebook, Amazon, Yahoo!, Twitter, Ebay, Steam, and Zynga, to name a few) are highly dependent on these systems, and therefore require new employees to either know them or learn them. But teaching students about these systems typically requires resources far beyond the scale of that available to an individual course or department; a *clique* of servers at Yahoo! – the unit of management for that ISP – is *20,000 servers*.

In order to cope with this requirement for resources at very large scale, computer science researchers have constructed a number of medium-scale testbeds over the past decade, with different capabilities and usage models. Emulab [18] is designed for short-duration, small-to-medium scale cluster and network experiments in laboratory conditions. PlanetLab [16] is designed for long-running services and network measurements in the wide area. Seattle [15] is designed as a very large, very lightweight services and experiment deployment platform at the network edge, with particular emphasis on mobile devices. GENICloud [5], as part of US National Science Foundation's Global Environment for Network Innovations (GENI) project, is a classic infrastructure-as-a-service (IaaS) cloud computing platform designed for multi-site federated deployment.

We designed a senior undergraduate/beginning graduate systems course which exploits the properties of each of these testbeds and platforms, to give students experience with the foundational technologies of large-scale distributed systems, and report on that experience in this work. In this course, we chose assignments which were tailored to give students a sense of designing wide-area measurement systems; parallel back-end computation; scalable networking systems; and issues of latency and bandwidth in the wide area. We used Emulab as our primary development platform; PlanetLab and Seattle as the deployment platform for wide-area systems and measurement; and finally used GENICloud as the deployment platform for cluster-based systems in a pilot project.

The contributions in this paper are twofold. First, we introduce a pedagogical approach that would be easy to adopt at any institution by taking the advantage of the above existing distributed testbeds, via the design of a series of assignment

tasks. Second, by incorporating students' contribution in a challenging pilot course project, our experience uncovered some of the possibilities to consider when preparing the next generation of computer scientists, and provided a basis for an evaluation of the trade-offs of teaching cloud and distributed systems *on the wild side*.

2. RELATED WORK

Parallel and cloud computing is an emerging new computing paradigm for delivering computing services. In recent years, an increasing number of educational establishments are adopting the newly emerging parallel and cloud computing techniques, including both industry and academia. In 2007, Google and IBM announced a cloud computing university initiative [6]. The purpose is to address the emerging paradigm of large-scale distributed computing. There are also a number of universities and colleges offering Computer Science courses in the domain of networking, distributed systems and cloud computing. St. Olaf College implemented *Hadoop at Home* [14], based on the observation that today's leading technology depends on programming models to deliver services based on data-intensive scalable computing (DISC). Tufts offered an introductory course on cloud computing [1] that taught students a broad view from how to program a cloud application, to the business cases for using clouds for common tasks.

University of Washington implemented Seattle [12,15], a surprisingly flexible platform that supports a variety of pedagogical uses for cloud computing, grid computing, peer-to-peer networking, and distributed systems. It makes programming assignments much easier, such as implementing a reliable protocol over UDP, building application-level services like a webserver, and building a chat server over HTTP, etc. UC Berkeley successfully taught MapReduce in a large undergraduate lecture course using public cloud services [19]. Using the cloud, every student could carry out scalability benchmarking assignments on realistic hardware, a large-scale demonstration that it is feasible to use pay-as-you-go billing in the cloud for a large undergraduate course. The course was designed intentionally to draw students' attention to the economic aspects of cloud computing, or cost management. The outcome of their course also demonstrated that using the cloud provided sufficiently consistent performance to allow students to see parallel speedup.

3. ASSIGNMENT & TESTBED FACILITIES

Motivated by these previous success stories, we decided to redesign our own course in distributed systems, which had previously focused on writing applications using Java Remote Method Invocation (RMI).

3.1 Testbed Overview

We designed the course assignments around the practical usage of the following four large-scale testbeds. Each assignment comes with conceptual questions as well as programming.

Emulab [3] is essentially a Hardware-as-a-Service platform, though users can now get "Containers" as services. Emulab is designed for testing distributed systems under controlled network conditions. To some extent, it can also be used as a development platform. Our students were instructed to design an experiment by hooking up nodes with a GUI, configuring the nodes by specifying an OS image (e.g., "FEDORA15-STD" for FedoraCore 15) for each node, then swapping the experiment in. Next they log in to each node, load any software they need, perform their experiment by running any scripts they need on the various nodes, then swap the experiment out. Emulab can also be used as a staging platform for PlanetLab.

PlanetLab [11] is a distributed Containers-as-a-Service platform. A container is largely indistinguishable from a virtual machine (VM); the principal difference is users don't have a choice in OS. In this case, we get a stripped Linux distribution. Stripped because PlanetLab is a deployment, not development platform, so the goal is that users bring up the environment needed in each container. There are various tools to help, including the one on Emulab. PlanetLab has about $1,000$ nodes at 300 sites worldwide. It is designed for both short-duration experiments and long-running services. Two services (Coral and CoDeeN) have been running on PlanetLab since 2002.

Seattle [12,15] gives the programmer a high-level programming environment on many devices of various sorts: everything from smartphones to servers. We primarily use it to test the infrastructure services students will be writing. It was used as a test environment because heavy weight computing is not permitted on the platform, due to the concerns of using community based and donated resources, such as people's home computers and smartphones. Programming in Seattle is done in Restricted Python (RePy); a form of Python with strongly constrained I/O and metered use of system resources, so that students can play safe on other people's systems. We used this chance to introduce Python to our students, and assigned several entry level programming tasks using RePy.

GENICloud [5] is a centralized infrastructure-as-a-service system based on containers. Up till today, GENICloud is a project that is a work in progress, as part of the larger GENI project [4]. It will build a GENI federation interface for compute clusters based on PlanetLab and running Openstack [10], a popular, open-source software for building a cloud computing infrastructure. This interface will allow Openstack clusters to federate via the Slice-based Federation Architecture (SFA) [13], and will enable experimenters to seamlessly deploy and configure slices spanning PlanetLab and multiple Openstack installations. Ultimately, we want to use GENICloud for centralized cloud services, such as MapReduce/Hadoop.

3.2 Assignment

This section presents the assignment results by students on different testbeds. We also present the insights we obtained, and the potential pitfalls of the current course design.

3.2.1 Emulab

In this experiment, students instantiate a small Emulab experiment with two nodes, a server and a client, and one link between them. Both the server and client will be running on FEDORA15-STD. By modifying the link delay and packet

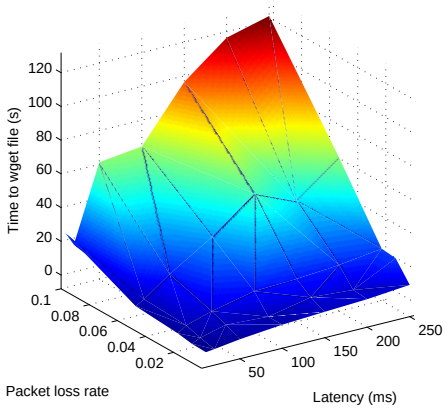

Figure 1: Download time vs. packet loss rate and client-server latency for an Emulab experiment.

loss rate between two nodes, both the available bandwidth and data transmission time will vary.

The reason for using Emulab in this experiment is that, part of Emulab's power is its ability to assume many different topologies. Emulab uses the "NS" ("Network Simulator") [9] format to describe network topologies. This is substantially the same Tcl-based format used by ns-2, although not identical. Although students can use a GUI interface on Emulab to generate the topology, it is highly recommended they use Tcl. Familiarity with the language allows for fine grained control over the Emulab experiment, and debugging opportunities if the topology is incorrect.

After uploading this script to Emulab, two physical nodes will be allocated to this experiment, i.e., *swapped in*. Then students install Apache on one node (the server) via remote login, and download files on the client node through HTTP. The link bandwidth is initialized to 2,000 Kbps and packet loss rate is 0, as defined in the Tcl script. To emulate a realistic network, students can dynamically create and modify node and link properties with `tevc` command to set different latency, and packet loss rate of the link between nodes.

In each experiment, the client uses `wget` to download a 1 MB text file hosted on the server, and measures the time it takes to get this file as well as the average bandwidth. The results are plotted in Fig. 1 by students in this course. The figure shows that the download time increases with the increasing link latency and packet loss rate. It is quite intuitive to observe such results, and students are expected to hand in their results shown in similar graphics.

Skill set: Students need to have the knowledge of SSH, Tcl and shell script (GNU core utilities such as `sed`, `awk`, and pipes). Also, plotting techniques such as Matlab, Gnuplot or any other preferred visualization tool are required. After this experiment, students will see how certain network parameters affect the systems performance quantitatively.

Pitfalls: As mentioned before, Emulab is designed for short-duration, small-scale testing of distributed systems under controlled network conditions. Therefore, all Emulab experiments will be *swapped out* once they have been idle for a certain period of time (2 hours by default). Once the experiment has been swapped out, students need to swap it in again and re-deploy the experiment. The reason is that the default Emulab images are stateless. Once an experiment gets swapped out, all previous changes are lost, with the exception of the content in the home directory. Although students learned this in a hard way, they came up with writing shell scripts to automate the configuration process.

3.2.2 Seattle

In this experiment, students instantiate a Seattle slice and add 10 nodes to the slice. These Seattle nodes are geographically distributed. Students are asked to write a short RePy program to access the 1 MB file they created from their Apache server at Emulab in the previous experiment, and record the download times from each slice, then use Google Maps [7] to plot their Seattle nodes and download times.

Acquiring resources in Seattle is straight-forward. VMs in Seattle are called vessels, which can be obtained on the Seattle Clearinghouse website. By a script `seash.py` students can launch their command line shell for Seattle, i.e., `seash`. `seash` allows students to easily run their RePy programs on a single vessel, or across multiple vessels simultaneously.

Skill set: Students need to have the knowledge of Python and RePy, socket programming and HTTP protocol, etc. After this experiment, students will experience the excitement of deploying programs in the wild, and see how location affects the systems performance. Students also get a chance to work with Google Map API and review knowledge on JavaScript and related topics.

Pitfalls: However, the downside of Seattle becomes obvious also in a world-wide setting. Seattle is very careful about using resources on things like people's home computers and smartphones, thus it limits the resources usage such as the bandwidth of each node. As a result, students can only modify the bandwidth on their local machine, but not remote vessels. This can be seen from Fig. 2, where the download times of the 1 MB file from all Seattle nodes at different locations (from the Apache server on Emulab deployed in the previous task) are very similar, varying between 102 to 105 seconds. These results are also plotted by students in this course. PlanetLab platform in Fig. 2 will be introduced next.

3.2.3 PlanetLab

In this experiment, students join the PlanetLab slice we created on the site at our institution, and add 10 nodes (slivers) that are geographically distributed to the slice. Then they start an Apache server on each PlanetLab node. Students need to modify their Seattle clients as they previously set up, so that each Seattle client pulls the 1 MB file from the PlanetLab node closest to them. Finally, they are asked to use Google Map to plot the Seattle clients, PlanetLab servers, and download times.

There are a number of ways students can accomplish this task. Examples include analyzing physical locations of nodes

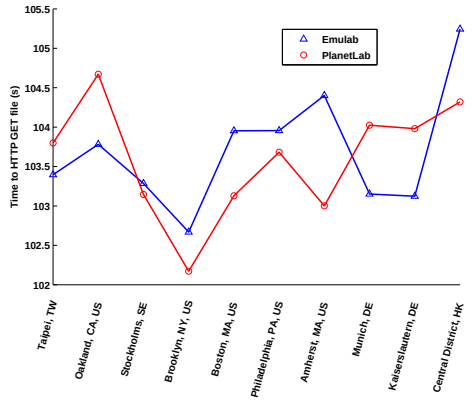

Figure 2: Seattle clients vs. download times for Emulab server and closest PlanetLab server.

via latitude and longitude provided by a geolocation service, or by using network tools such as `ping` and `traceroute` to compare the round trip time and hop number, etc. Fig. 2 also shows the results of the above PlanetLab experiment generated by students.

Skill set: Students need to have basic grounding in networks and systems knowledge in order to accomplish this task. Based on previous two experiments, they experience a gradual process of applying their knowledge to solve a real problem.

Pitfalls: While PlanetLab is a highly flexible testbed, the pitfalls of this experiment set still comes from Seattle. The download times from different clients still do not vary much given their different geographic locations, due to the bandwidth limit.

4. PROJECT IMPLEMENTATION & STUDENT CONTRIBUTION

In this course we design a medium-scale, computation-intensive project in the form of a simple geographic information system (GIS). This GIS system uses images captured by Landsat series of satellites [8] as the data set, and cloud computing services as its storage and computation platform. Given the large amount of data, the computations must be distributed. This project is of enough size that the students must deal with many of the problems they would experience in a real-world application. The computing platform we chose is GENICloud [5]. The project had a hard deadline, and was demonstrated at a conference demo session.

4.1 Project Design & Software Deployment

The task of the project system is calculating the amount of *greenspace* contained within a city using satellite imagery, doing so for all cities on earth, and finding out the *greenest* cities in the world. The greenspace calculation itself is simple. Given a set of satellite images, the algorithm generates a pseudo-color landscape image, and count what percentage of pixels are *green* inside a defined city area. The dataset for images which cover most of the world in 3 bands of light (one visible bands, two non-visible bands) is about 2 TB of zipped tiff files. The largest VMs on GENICloud only have 500 GB of disk space, so the dataset has to be distributed across machines. Each greenspace calculation requires between 3 and 12 images, which are about 100 MB each when unzipped. Those images have to be combined into a single large image before the calculation starts. On a small VM, each city takes between 2 and 10 minutes to process, resulting in a calculation that would take in the ballpark of 50 days to complete if left single threaded.

The students were given a preexisting single threaded, single machine example of how to do the greenspace calculation written in Python. The students were instructed to make it distributed and parallel. We broke the class up into two teams, a storage team and a computation team. The computation team was tasked with building the computation part of the calculation using a MapReduce based technology. The storage team was instructed to split the data across the sites and store it using the OpenStack Swift object storage technology, a open source alternative to Amazon's S3 service.

4.2 Storage

Multi-Site Swift Storage. Our first version of image storage was in a single Swift cluster at one site: one proxy and several storage nodes. To parallelize the data storage, as well as the computation, the storage team decided to transfer data from the main Swift cluster to the distributed Swift servers at multiple locations.

The storage team first divided the entire land area of the world into four groups of continents, which were then input into the database as a table *continents*, with multi-polygons delineating each of the groups. The strategy was to run a different continent group on each of the Swift clusters in a distributed network. To transfer the data to different clusters, the students wrote a PostGIS query that joined the data in *continents* table with the data in the *tiff* table to find which images corresponded to each cluster. To transfer data, students divided the process in two steps: temporarily store a list of the images to be transferred from the main Swift cluster, and then transfer the images. With such a process, whenever the transfer process encountered errors, the program could simply checkpoint the process where there was an error in transfer and continue from that point, reducing the amount of redundant data to be transferred to the distributed nodes.

4.3 Computation

Image Crawler. The image crawler is a Python script that extracts the geographic information from a Landsat Geotiff image, and then inserts the information into the *tiff* table of the PostGIS database. The most important geographic information extracted from the image is a polygon representing the area covered by the Landsat image, represented by a bounding box.

The crawler works by downloading representative files (one band for each Landsat image) from Swift and using `gdalinfo` to extract the geographic information. `gdalinfo` is part of the open source Geospatial Data Abstraction Library (GDAL) which performs various functions on raster geospatial data. From the output of gdalinfo we then construct a polygon using well known text (WKT), which is then inserted into the *tiff* table if the image has not yet been

processed by the crawler. Multiple instances of the crawler can be run concurrently, as all they need to function is a connection to the Swift proxy and the PostGIS/PostgreSQL database.

Greenspace Calculation. The greenspace calculation is a Python script which assigns a value to a city to represent how much greenspace the city contains. The calculation script first gets a city that does not have a calculated greenspace value, then pulls that city's bounding box from the *map* table in the PostGIS database. From the city boundary we then query the *tiff* table of the database to find all the images that intersect the city, which are then downloaded from Swift.

The script crops each band to contain only what lies inside the city using the `ogr2ogr` tool included in the GDAL library, then merges the cropped images into a single pseudocolor image with `gdal_merge`. To calculate the greenspace we iterate over each pixel checking if it is within the city boundary. If so, it is counted as a *greenspace pixel* if the value from band 4 (green) is greater than the other two bands. Finally the greenspace value is returned as the number of greenspace pixels over all pixels within the city. Multiple instances of the greenspace calculator can also run simultaneously as the computation for each image is independent.

Web Interface. The demo was presented through a web interface. The database tables were exposed through a Web Mapping System (WMS) interface, and a OpenLayers web frontend was provided to the students. The webmap showed each of the cities and color coded them based on their green value. If a city is clicked on by the user of this web interface, a pop-up with the imagery used for that calculation is shown. The map also shows the footprints of all the images, and how work was partitioned between sites. For a demo of this web interface, please refer to [anonymous].

Disco. The computation team elected to use a MapReduce framework for Python called Disco [2]. Disco has many of the features of other common MapReduce frameworks like Hadoop: it has features for defining custom input streams, multistage computations, and its own distributed file system. Disco suited the team's familiarity with Python, and made porting the old code easier since it was written in Python.

Disco was installed on several machines in each cluster. The students found it hard to install. Disco is a relatively young project, its error handling is a little weak. The course TAs helped the students get through the installation issues; however, this did eat up a lot of time in lab sessions. Almost two full 3 hour labs were spent dealing with Disco related issues. The MapReduce job used is very simple. There is no reduce phase, and the map phase is passed a city, which triggers the normal calculation to take place and store the result in the database. At its height, Disco allowed us to run 256 jobs in parallel on one cluster—a rewarding result for the students.

5. STUDENT FEEDBACK

In this section we introduce student evaluation on this course, from their expectation, experience to their feedback on different platforms we used.

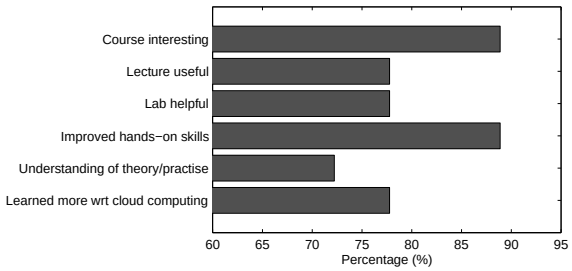

Figure 3: Student experience.

5.1 Student Experience

We are glad to see that, students' feedback on their experience in class is aligned with our expectations. In an anonymous class survey, students stated that they have improved their skills in: Linux basic knowledge, scripting languages (such as perl, python, etc.), knowledge about networking, distributed storage and concurrency.

5.1.1 Expectation & Overall Impression

Students have expectations on us the same way as we have expectations on them. They come to the class not only hoping for gaining knowledge in parallel and distributed computing, but also applying the theoretical foundations to systems *on the wild side*. In the same survey, students stated that they feel closer to the real world by doing assignments and projects that are hands-on and system based. The course is more enjoyable especially it is more "learning what you want" and less "memorizing information for your final". Close to 90% of students think the course is interesting, and helped them improved their hands-on and programming skills. Almost 80% of them have learned more about cloud computing. Other survey data and student opinions are plotted in Fig. 3.

However, students also expressed certain level of frustration because of the various testbeds and software involved. One student wrote his course experience in the anonymous survey: *"Swift installation instructions provided by the developers were certainly adequate, but tedious when installing on a large number of nodes. By scripting the installation steps, deployment was able to be done rapidly and accurately across multiple nodes, which freed up human capital."*

The observation from students revealed that current Computer Science curriculum lacks the much needed development practices for students to prepare for the real world. The testbed environments and software tools also lack debugging support for the installation and configuration. On the other hand, we believe that the experiences in this course requiring development and software automation practices are, in fact beneficial to the students. Despite various levels of difficulties, taking a walk *on the wild side* and teaching cloud computing on distributed research testbeds prepare the next generation of computer scientists to engage with software practices.

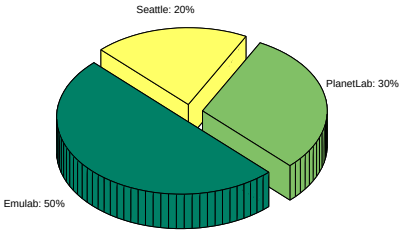

Figure 4: Student feedback on testbed facilities.

5.2 Testbed Facilities

We also conducted in the same survey about students' opinions about different cloud computing platforms used in this course. Figure 4 shows the percentage breakdown of the platform that each student liked the most. Since GENICloud was only used in the pilot project for experimental purpose, we did not include it in Fig. 4. As the most stable testbed, Emulab was voted by 50% of all the students in the course. The next highest is PlanetLab, which is both stable and flexible. However, only Principal Investigator (PI) has the privilege to create PlanetLab slices. Such management of PlanetLab makes it difficult for students to fully control the testing facilities.

Last but not least, Fig. 4 reveals that the students had unrealistic expectations for Seattle, thus this is not a fair comparison in terms of the result reported. Though they knew that security is the primary goal of Seattle, the ramifications of running experiments on donated machines and devices did not become clear until after they had tried to surpass the constraints of their permissions. Students often become frustrated when encountering scenarios where operations are not supported or permitted. However, in our case, these students are starting to write their own libraries, such as Trigonometric functions, which can be integrated in the Seattle base.

6. DISCUSSION AND CONCLUSIONS

Over the past decade, practical considerations when building distributed systems have changed dramatically. Today's students have opportunities to genuinely create computations that span the globe. In this paper we introduced the research testbeds used in our exploratory course, details of the assignment and project used, along with initial student feedback about their perspectives on this experience. We have demonstrated that using distributed research test facilities could give students the opportunity to experience distributed systems in a realistic way.

While getting first-hand experience with distributed application deployment in the cloud is an important step in career preparation, we found the raw exposure to research infrastructure revealed that, when students were striped of the luxury of higher level debugging facilities, they found the experience to be daunting. In some cases, the battles with the infrastructure actually prevented students from focusing on the underlying technologies. We do, however, believe this kind of exposure is necessary and ultimately sharpens problem solving abilities once the students survive the initial shock. We hope our findings provide insight into software practices and paradigms that are fundamental to modern Computer Science education.

7. REFERENCES

[1] Clouds and power-aware computing, `http://www.cs.tufts.edu/comp/150CPA/`.
[2] The disco project, `http://discoproject.org/`.
[3] Emulab—network emulation testbed home, `https://www.emulab.net/`.
[4] GENI: Exploring networks of the future, `http://www.geni.net/`.
[5] GENICloud, `http://groups.geni.net/geni/wiki/GENICloud`.
[6] Google and IBM announce university initiative to address internet-scale computing challenges, `http://www-03.ibm.com/press/us/en/pressrelease/22414.wss`.
[7] Google map API, `https://developers.google.com/maps/`.
[8] Landsat then and now, `http://landsat.gsfc.nasa.gov/about/`.
[9] The network simulator - ns-2, `http://www.isi.edu/nsnam/ns/`.
[10] Openstack: Open source software for building private and public clouds, `http://www.openstack.org/`.
[11] PlanetLab, `http://www.planet-lab.org/`.
[12] Seattle—open peer-to-peer computing, `https://seattle.cs.washington.edu/html/`.
[13] SFA overview, `http://svn.planet-lab.org/wiki/SFAGuide`.
[14] R. Brown. Hadoop at home: large-scale computing at a small college. In *ACM SIGCSE Bulletin*, volume 41, pages 106–110. ACM, 2009.
[15] J. Cappos, I. Beschastnikh, A. Krishnamurthy, and T. Anderson. Seattle: a platform for educational cloud computing. In *ACM SIGCSE Bulletin*, volume 41, pages 111–115. ACM, 2009.
[16] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: An overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, pages 3–12, 2003.
[17] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.
[18] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau. Large-scale virtualization in the Emulab network testbed. In *ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 113–128, Berkeley, CA, USA, 2008. USENIX Association.
[19] A. Rabkin, C. Reiss, R. Katz, and D. Patterson. Experiences teaching mapreduce in the cloud. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pages 601–606. ACM, 2012.
[20] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.
[21] T. Schütt, F. Schintke, and A. Reinefeld. Scalaris: reliable transactional p2p key/value store. In *ERLANG '08: Proceedings of the 7th ACM SIGPLAN workshop on ERLANG*, pages 41–48, New York, NY, USA, 2008. ACM.
[22] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.