Bringing Big Systems to Small Schools: Distributed Systems for Undergraduates *

Jeannie R. Albrecht
Williams College
Williamstown, MA 01267
jeannie@cs.williams.edu

ABSTRACT

Distributed applications have become a core component of the Internet's infrastructure. However, many undergraduate curriculums, especially at small colleges, do not offer courses that focus on the design and implementation of distributed systems. The courses that are offered address the theoretical aspects of system design, but often fail to provide students with the opportunity to develop and evaluate distributed applications in real-world environments. As a result, undergraduate students are not as prepared as they should be for graduate study or careers in industry. This paper describes an undergraduate course in Distributed Systems that not only studies the key design principles of distributed systems, but also has a unique emphasis on giving students hands-on access to distributed systems through the use of shared computing testbeds, such as PlanetLab and GENI, and open-source technologies, such as Xen and Hadoop. Using these platforms, students can perform large-scale, distributed experimentation even at small colleges.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications; K.3.2 [**Computer and Information Science Education**]: Computer Science Education

General Terms

Design, Experimentation

Keywords

Distributed Systems, PlanetLab, Undergraduate Education

1. INTRODUCTION

As the number of Internet users continues to rise, Internet-based companies, such as Google, Amazon, and Yahoo, are leveraging the aggregate computing power of *distributed systems* to satisfy the demands of their users. In general terms, distributed systems are collections of independent networked computers that function as single coherent systems. These systems offer many advantages over non-distributed systems and applications, such as enhanced performance and better resilience to failure. However, in addition to the advantages of distributed systems, they also introduce many new challenges to developers. Configuring and maintaining a distributed set of computers for hosting an application is a tedious task. Detecting and recovering from bugs in applications that are running on hundreds of machines worldwide is much more challenging than debugging code locally. These challenges are overwhelming to developers without prior programming experience in distributed environments.

Since distributed computing is quickly becoming the defacto way to accomplish large-scale tasks on the Internet, one would think that the skills required to design, implement, and evaluate distributed systems would be available to students at the college and university level. Unfortunately this is frequently not the case, since most colleges and universities do not offer undergraduate Distributed Systems courses. Particularly at small colleges, the problem is often due to a lack of computing resources. Without access to a dedicated computing cluster that permits students to run code on distributed resources, it is difficult to expose students to the challenges of implementing and evaluating distributed systems. At large universities, although computing clusters are present, they are typically reserved for research purposes and are not readily available for use in the classroom. As a result, undergraduates are not receiving the training necessary to become good programmers in distributed environments.

In Spring 2008, I designed and taught a new Distributed Systems course for undergraduates at Williams College. The purpose of the course was to introduce students to the key design principles of distributed systems, help students understand how large-scale computational systems are built, and ultimately teach students the skills necessary to be successful distributed programmers. In addition to the more conventional textbook-oriented topics of distributed systems, I incorporated several cutting-edge technologies—that are both used in the real-world and have resulted in high impact research publications—into the course assignments and readings. The course also included four programming projects that served two key purposes: first, the projects exposed students to state-of-the-art tools that are commonly used in distributed application development; second, the projects

*This material is supported by the NSF grant CNS-0834243.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGCSE'09, March 3–7, 2009, Chattanooga, Tennessee, USA.
Copyright 2009 ACM 978-1-60558-183-5/09/03 ...$5.00.

gave students hands-on access to distributed computing platforms, allowing them to design and evaluate large-scale systems running on computers worldwide.

This paper describes my newly developed undergraduate Distributed Systems course in detail. To summarize, the course makes the following contributions: (i) the course combines classical, theoretical concepts in distributed systems with practical, hands-on programming projects that incorporate several new widely-used technologies; (ii) the assignments assume no prior knowledge in networks or distributed systems, making the course accessible to a range of students; (iii) the course leverages the use of shared computing platforms and open-source software packages to allow students to develop large-scale systems using a minimal amount of local computing resources, making the course well-suited for both large and small schools.

2. RELATED WORK

Throughout the past two decades, several educators have proposed ways to introduce students to distributed computing concepts. In the 1990s, the key challenges were developing courses that emphasized "practical case studies" without requiring expensive equipment and high performance machines [8, 10, 25]. In recent years, the need for expensive supercomputers has diminished, and the emphasis of many distributed computing courses now focuses on realistically simulating and emulating large-scale distributed systems using local resources [14, 16, 20].

The aforementioned previous work had two ideas in common: (i) a general dissatisfaction with the lack of undergraduate distributed computing courses; and (ii) an acknowledgement of the importance of integrating theory and practice to give students exposure to realistic systems. While the state-of-the-art of distributed computing has changed significantly over the past few decades, these two ideas are still relevant today. However, the emergence of shared distributed computing platforms, such as PlanetLab [19] and GENI [11], has created new opportunities for educators to give students access to diverse sets of distributed resources, even when budgetary constraints are present.

3. COURSE DESIGN

This section describes the design of a new undergraduate Distributed Systems course. The projects and topics covered in this course were inspired by assignments in related courses at various research institutions [22, 24, 26]. This course was taught in the Spring 2008 semester at Williams College, and consisted of 36 classes spread over approximately 14 weeks of instruction. A total of 14 students were enrolled in the class, ranging from sophomores to seniors. Prerequisites included Data Structures and Computer Organization. The two main components of the course were reading assignments and programming projects, as discussed in the following sections. Additional information is available on the course webpage [28].

3.1 Reading Assignments

The course included daily reading assignments from a textbook [7] in addition to several high-impact research publications. Since a basic understanding of networks is essential in distributed systems, and because many students in the class did not have a background in computer networks, the course started with a two week review of basic networking concepts such as TCP, UDP, IP, and socket programming. After the networks review, we spent approximately four weeks covering conventional distributed systems topics, including communication protocols, naming, synchronization, coordination, consistency, and replication. At this point in the semester, we shifted the focus away from the theoretical aspects of systems design, and spent two weeks investigating specific examples of real distributed systems. In particular, we studied several Google services that have resulted in research papers, including the Chubby Lock Service [5], MapReduce framework [9], and BigTable database system [6], in addition to different distributed file and storage system implementations. The remaining four weeks of the course included discussions related to replication schemes, fault tolerance, and security, in addition to more wide-area computing case studies in overlay networks, peer-to-peer systems, and finally, sensor networks.

Throughout the semester, the students read eight research publications that supplemented the textbook reading assignments. The goal of the research papers was twofold: to help students develop the skills necessary to understand and critique other people's research projects, and to help students hone their technical writing abilities. Thus, for each assigned paper, students submitted a two-page evaluation that summarized the key contributions of the paper, described any problems with the work, and highlighted the key design principles in use. These evaluations (unintentionally) ended up being one of the most beneficial aspects of the course. At the beginning of the semester, students struggled to think critically, and few students were able to find problems in the research papers. By the end of the semester, the students were not only finding problems in the papers, but some were even able to compare and contrast related projects and discuss the relative merits of each approach. In addition, by exposing students to several well-written papers, they were able to learn by example and improve their own technical writing skills.

3.2 Programming Projects

The course consisted of four programming assignments spread out evenly throughout the semester. The first two projects were traditional assignments that taught students the basic skills required to develop distributed applications. The last two projects introduced students to new technologies and computing paradigms, and allowed the students to experiment with different distributed environments. For all projects, students had the option of working alone or in pairs. They were given approximately two weeks to work on each of the first two assignments, and three weeks to work on each of the last two. The details of each project are discussed in this section.

3.2.1 Project 1: Web Server

The first project was the design and implementation of a basic web server written in C. The project was intended to teach students the basics of distributed network programming and client-server architectures. Since a web server and browser-based web client are arguably the simplest example of a distributed system involving only two computers, the project eased students into the challenges of distributed sys-

tems. None of the students had any prior experience with socket programming, and for many students, this was their first program developed completely in C, which added to the difficulty of the assignment.

The project required students to implement a fully functional web server that supported HTTP/1.0 and HTTP/1.1 GET requests. They were required to choose a multi-threaded, multi-process, or event-driven architecture for their server, and they had to provide enough support so that basic HTML pages including images were successfully returned to the clients. Support for parsing scripts and POST requests was not required, although accurate error pages with properly formatted message headers were mandatory. In addition to the implementation of the web server, students also submitted a write-up that described their server's design. The write-up included a basic overview of their implementation, as well as a discussion of the performance of HTTP/1.0 versus HTTP/1.1 from the perspective of both the web client and server.

3.2.2 Project 2: Online Bookstore

The second project in the course involved the implementation of a distributed bookstore. The goals of the project were to introduce students to client- and server-side remote procedure calls and multi-tier distributed systems. The students chose between Java RMI or XML-RPC to implement their store. The store carried only four (fake) books, since the focus of the assignment was not large-scale data-management, but instead was intended to help students understand the design challenges associated with large-scale multi-tier distributed services. In this assignment the students ran their code across three separate computers.

The bookstore employed a two tier (front-end and back-end) design consisting of three total components. The front-end tier was a server that accepted customer requests for books, performed initial request processing, and interacted with the back-end components. Three key operations were supported by the front-end server: search, lookup, and buy. The back-end tier included a catalog server and an order server. The catalog server was similar to a database. It maintained a list of all books in stock in the store, the cost of each book, and the general topic area of each book for searching purposes. The catalog server responded to search and lookup queries from the front-end server. The order server maintained a master record of all orders received, and was responsible for updating the catalog server when new shipments of books arrived. Every time a buy operation was issued from a customer, the order server was contacted by the front-end server to complete the transaction. The order server frequently interacted with the catalog server to ensure that the catalog server correctly responded to queries from customers regarding the price and quantity of books in stock. Issues related to synchronization among the servers and concurrent requests had to be considered in the design of the order server.

As part of the write-up for Project 2, students also performed an evaluation of the performance of their system. They measured the average response time for customer requests under different levels of server load. The evaluation allowed students to appreciate the importance of making good design decisions, especially when customer satisfaction was at stake. The students also described the design of their system, and reflected on potential performance bottlenecks in their write-ups.

3.2.3 Project 3: Inverted Index with Hadoop

After the completion of Project 2, we were approximately halfway through the semester, and the students had developed a basic understanding of the key design principles of distributed systems. Project 3 moved away from the more conventional programming projects, and introduced students to new technologies and concepts. This assignment also exposed students to cluster computing for the first time. The project was loosely based on a new course developed at the University of Washington that leveraged the recent Google and IBM initiative for addressing Internet-scale computing challenges [12]. However, since Williams College was not part of the initiative, we were unable to gain access to the large clusters provided by IBM and Google for hosting distributed computations. Instead, I reconfigured our local cluster of 14 computers to create mini-clusters of Xen virtual machines (VMs) [3] for students to host their projects. Each mini-cluster consisted of six VMs. While several tools are available to simplify virtual cluster management [17, 27], I used Orca [15] to create approximately 65 VMs in total. The students then built an inverted index of several classic eBooks using Apache's Hadoop framework [13] (which is an open-source implementation of Google's MapReduce) on their mini-clusters. eBooks were obtained from the Project Gutenberg collection [23].

The project was designed to help students become familiar with the Hadoop/MapReduce distributed programming model. In addition, since each team of students managed their own mini-clusters, it also taught students basic system administration skills. For example, although I created the mini-clusters, I did not configure the VMs with anything aside from a basic Linux installation. Thus the students were given root access to their VMs, and they learned to install software and configure their clusters. To assist in this process, I held class in the computer lab one day to make sure all teams were able to get a trivial MapReduce example up and running on their mini-clusters. Students with prior system administration experience enjoyed the opportunity to tinker on their own clusters without causing problems for our system administrator, and students with no prior experience benefited from the chance to learn the basic skills required to maintain their own working environments. As in the previous two projects, students submitted a write-up that described their design, implementation, and results (*i.e.*, sample output from inverted index).

3.2.4 Project 4: P2P Computing

For their fourth project, the students designed their own peer-to-peer file distribution service and evaluated its performance on PlanetLab [19]. This project was designed to introduce students to the fully distributed, peer-to-peer computing paradigm, which is typically more complicated than client-server architectures. As the final project in the course, this project was significantly more difficult and less structured than the first three projects. Students submitted proposals and progress reports to help them focus their ideas and keep them on task. A "default" system design was also posted on the course web page to help students who struggled with the open-endedness of this project. On the last

day of class, the students gave short presentations to their classmates describing their systems, and had the option of submitting an extended write-up that included extra performance evaluations and a detailed related work section in lieu of a written final exam.

This project had several goals. First, I wanted students to have the freedom to be creative and design their own system from scratch. Since P2P computing is a topic that they all had some familiarity with (thanks to BitTorrent [4]), they were excited to build their own architectures. Second, this project moved students off of our local network, and allowed them to experience the challenges and measure the effects of large-scale, wide-area computing. PlanetLab is a publicly available, shared testbed that consists of over 800 Linux computers spread around the world. By joining the PlanetLab Consortium, which mostly involved connecting three computers on the Williams campus to the worldwide testbed, my students were able to run experiments on the remaining 800 PlanetLab computers across the globe. In the future, NSF initiatives such as GENI [11] will continue to create new opportunities for students to experiment with large-scale, wide-area computing platforms. While GENI is still in its early stages of development, within the next few years GENI plans to provide researchers and educators with access to a diverse set of distributed resources, including sensor, mobile, and wireless devices. Shared platforms like PlanetLab and GENI enable students and researchers at small schools to experience the technical richness of large research institutions.

Advancing from six VMs in Project 3 to 800+ machines in the final project was understandably a bit overwhelming for some students. In particular, students with no prior system administration experience struggled to manage code running across hundreds of machines worldwide. To make this task easier, students were encouraged to use application management tools like Plush [1, 21]. Plush was designed to simplify the complex tasks associated with running applications on distributed sets of resources. They were also encouraged to use resource discovery and monitoring tools such as SWORD [2] and CoMon [18] to find "good" machines to host their applications. In the end, all students were able to run their applications on at least 20 machines, and the most successful group evaluated their service running across 450 PlanetLab computers.

4. EXPERIENCES

This section summarizes my opinions and student feedback received about the course.

4.1 Student Feedback

Student feedback for the course was largely positive. The students liked learning about technologies that are actually being used by companies such as Google and Yahoo, and they seemed to enjoy building their own P2P systems. They found it advantageous to learn practical programming skills that would be useful after they graduated. I was surprised by how many students commented on the writing-intensive aspects of the course. Several students mentioned the benefits of learning to read and write technical papers. They also liked critically evaluating recent research results. In terms of improvement, the majority of them noted that they needed extra time for the final project, which I think is a good suggestion. To summarize, the following comments appeared on end-of-the-semester student evaluations.

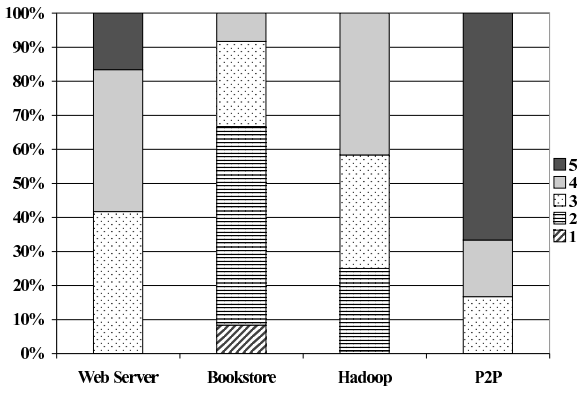

Figure 1: Student evaluation of project difficulty. 1 was the easiest, and 5 was the most difficult.

- *I loved the papers! This was the first class that required critical responses to papers like that and I was surprised by how much I enjoyed it.*
- *Evaluating the papers, while kind of a pain sometimes, was actually quite valuable in retrospect; I learned a lot about distributed systems that way, and I'm glad we did them.*
- *Labs were actually fun to work on.*
- *[The P2P project] was one of the hardest and most rewarding projects I've done at Williams.*
- *I really felt like this was one of the most real-life applicable CSCI courses I took at Williams.*

4.2 Project difficulty

As part of their end-of-semester evaluation, I asked the students to evaluate the difficulty of the projects. The scores ranged from 1 to 5, with 1 being the easiest score, and 5 being the most difficult. The results are shown in Figure 1. Based on these results, the students felt that Project 4 was the most difficult, followed by Project 1. The general consensus was that Project 2 was too easy, while Project 3 was moderately difficult. Ironically, when asked whether or not they would recommend these projects for future use, Projects 1 and 4 received unanimous "yes" votes. Project 2 ranked the lowest, and only received seven (out of twelve total) recommendations. Project 3 was recommended by ten students. This indicates that while the students found Projects 1 and 4 to be the most difficult, they also thought they were the most beneficial and should definitely be included in future version of the class.

4.3 Instructor Comments

Overall, I was pleased with the design of the course. The students seemed to enjoy the class. They liked learning about recent research and using new technologies. I will likely include additional research papers next time, since that seemed to be one aspect of the class that received

positive comments from almost everyone. I think the students benefited from learning to technically write about their projects, and all of them showed significant improvement throughout the semester. Out of the four projects, I think Projects 1 and 4 were the most effective. Project 1 provided a good introduction to networks and distributed systems, and despite their initial concerns, eventually the students all agreed that learning about sockets in C was good for them. Project 4 was a bit of a struggle for some students, but in the end, they all benefited from the experience. I need to allow at least an extra week for this project next time, and I plan to drop the written final exam option entirely to allow them to focus on this final project. Progress reports were a good way to keep them on track and avoid end-of-semester procrastination.

Out of the four projects, I was the least satisfied with Projects 2 and 3. I believe that both projects accomplished my goals, but ultimately, I think these projects were actually a bit too easy and boring for the students. For Project 2, I would like to add a bit more complexity next time. Perhaps the addition of an actual database or some basic security measures would make the project more interesting. Also, I allowed the students to choose between Java RMI and XML-RPC. XML-RPC is slightly more difficult, and only one student chose XML-RPC over Java RMI. In future iterations of the class I will consider requiring both implementations. For Project 3, understanding Hadoop and getting the mini-clusters configured was challenging, but building an inverted index was easy. Next time I plan to add additional components to this project to make it more challenging.

5. CONCLUSIONS

In conclusion, this paper presents the design of a Distributed Systems course for undergraduates that combines theoretical "textbook" concepts with practical real-world applications to give students the hands-on experience necessary to be successful distributed programmers. The course incorporates an introduction to networks and eases students into the complexities associated with designing distributed systems, making it accessible to a range of students with different backgrounds. Perhaps most importantly, the course leverages the increasing availability of shared distributed computing platforms to bring large-scale systems development to even small schools with limited local resources.

6. REFERENCES

[1] J. Albrecht, R. Braud, D. Dao, N. Topilski, C. Tuttle, A. C. Snoeren, and A. Vahdat. Remote Control: Distributed Application Configuration, Management, and Visualization with Plush. In *USENIX Large Installation System Administration Conference (LISA)*, 2007.

[2] J. Albrecht, D. Oppenheimer, D. Patterson, and A. Vahdat. Design and Implementation Tradeoffs for Wide-Area Resource Discovery. *ACM Transactions on Internet Technology (TOIT)*, 8(2), 2008.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *ACM Symposium on Operating System Principles (SOSP)*, 2003.

[4] BitTorrent. `http://www.bittorrent.com/`.

[5] M. Burrows. The Chubby Lock Service for Loosely-coupled Distributed Systems. In *ACM/USENIX Symposium on Operating System Design and Implementation (OSDI)*, 2006.

[6] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A Distributed Storage System for Structured Data. *ACM Transactions on Computer Systems (TOCS)*, 26(2), 2008.

[7] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design (4th Edition)*. Addison-Wesley Longman Publishing Co., Inc., 2005.

[8] J. C. Cunha and J. Lourenço. An Integrated Course on Parallel and Distributed Processing. In *ACM Technical Symposium on Computer Science Education (SIGCSE)*, 1998.

[9] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters . In *ACM/USENIX Symposium on Operating System Design and Implementation (OSDI)*, 2004.

[10] E. Dillon, C. G. D. Santos, and J. Guyard. Teaching an Engineering Approach for Network Computing. *ACM SIGCSE Bulletin (inroads)*, 29(1), 1997.

[11] GENI. `http://www.geni.net`.

[12] Google and IBM Announce University Initiative to Address Internet-Scale Computing Challenges. `http://www.google.com/intl/en/press/pressrel/20071008_ibm_univ.html`.

[13] Hadoop Project. `http://hadoop.apache.org/core/`.

[14] V. Y. Hnatyshin and A. F. Lobo. Undergraduate Data Communications and Networking Projects Using OPNET and Wireshark Software. In *ACM Technical Symposium on Computer Science Education (SIGCSE)*, 2008.

[15] D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, and K. G. Yocum. Sharing Networked Resources with Brokered Leases. In *USENIX Annual Technical Conference (USENIX)*, 2006.

[16] W. D. Laverell, Z. Fei, and J. N. Griffioen. Isn't It Time You Had An Emulab? In *ACM Technical Symposium on Computer Science Education (SIGCSE)*, 2008.

[17] M. McNett, D. Gupta, A. Vahdat, and G. M. Voelker. Usher: An Extensible Framework for Managing Clusters of Virtual Machines. In *USENIX Large Installation System Administration Conference (LISA)*, 2007.

[18] K. Park and V. S. Pai. CoMon: A Mostly-Scalable Monitoring System for PlanetLab. *ACM Operating Systems Review (OSR)*, 40(1), 2006.

[19] L. L. Peterson, A. C. Bavier, M. E. Fiuczynski, and S. Muir. Experiences Building PlanetLab. In *ACM/USENIX Symposium on Operating System Design and Implementation (OSDI)*, 2006.

[20] C. Pheatt. An Easy To Use Distributed Computing Framework. In *ACM Technical Symposium on Computer Science Education (SIGCSE)*, 2007.

[21] Plush Webpage. `http://plush.cs.williams.edu`.

[22] Problem Solving on Large Scale Clusters. `http://www.cs.washington.edu/education/courses/490h/`.

[23] Project Gutenberg. `http://www.gutenberg.org/wiki/Main_Page`.

[24] P. Shenoy. Distributed Operating Systems. `http://lass.cs.umass.edu/~shenoy/courses/spring08/`.

[25] C. Stewart. Distributed Systems in the Undergraduate Curriculum. *ACM SIGCSE Bulletin (inroads)*, 26(4), 1994.

[26] A. Vahdat. Networked Services. `http://www-cse.ucsd.edu/classes/fa07/cse124/`.

[27] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *ACM/USENIX Symposium on Operating System Design and Implementation (OSDI)*, 2002.

[28] Williams College CS339 Course Website. `http://www.cs.williams.edu/~jeannie/cs339/`.