Virtualized Games for Teaching About Distributed Systems*

Joel Wein
Polytechnic Institute of NYU
Brooklyn, NY
wein@poly.edu

Kirill Kourtchikov
Polytechnic Institute of NYU
Brooklyn, NY

Yan Cheng
Polytechnic Institute of NYU
Brooklyn, NY

Ron Gutierez
Polytechnic Institute of NYU
Brooklyn, NY

Roman Khmelichek
Polytechnic Institute of NYU
Brooklyn, NY

Matthew Topol
Polyetchnic Institute of NYU
Brooklyn, NY

ABSTRACT

Complex distributed systems are increasingly important in modern computer science, yet many undergraduate curricula do not give students the opportunity to develop the skill sets necessary to grapple with the complexity of such systems. We have developed and integrated into an undergraduate elective course on parallel and distributed computing a teaching tool that may help students develop these skill sets. The tool uses virtualization to ease the burden of resourcing and configuring complex systems for student study, and creates varied "firefighting" gaming scenarios in which students compete to keep the system up and running in the presence of multiple issues. Preliminary experience indicates that (1) students find the tool engaging and (2) it is a manageable way in which to give students a novel perspective on interaction with complex distributed systems.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Curriculum

General Terms

Management

Keywords

Distributed Systems, Gaming, Curricula

1. INTRODUCTION

Given the importance of distributed systems for the next decade of computing and beyond, it is critical that computer science educators train students who can understand such systems and contribute to their evolution. In fact, the Joint Task Force on Computing Curricula of the IEEE Computer Society and the ACM, in its 2001 report on curriculum guidelines for undergraduate degree programs in computer science, [1] identified the increasing importance of the world-wide web and networking technology as one of the key drivers for the need for revisiting computing curricula. Unfortunately, it is perceived that the rate of development of curricula in distributed systems has lagged behind the (very rapid) rate of deployment of these technologies in practice. For example, IBM and Google recently lamented this situation and launched a significant initiative to help universities develop advanced curricula for large scale distributed computing, in order to "prepare students to harness the potential of modern computing systems." They have made a multi-year commitment to providing universities with hardware, software, and services to advance training in large-scale distributed computing [7].

We believe that there is much interesting work to be done by computer science educators in developing curricula that more fully introduce students to distributed systems at the undergraduate level. One important direction in this vein is developing techniques to give students the opportunity to experiment with, appreciate, and even enjoy the complexity and power of large and complex distributed systems. In practice, such systems, such as a supply-chain application, span multiple languages and platforms at multiple sites and interact via one or more sorts of middleware. Producing students who can work at higher levels of such large systems is a substantial challenge. In this paper we describe one approach to improving the distributed systems curricular experience in support of this larger goal.

While many institutions teach undergraduates about distributed systems, to date many or most projects in these courses are of a very limited scope. This is primarily for two reasons. First, it is a major effort to get a nontrivial distributed system up and running correctly; it requires significant attention to configuration issues, support staff that is knowledgeable about the middleware and participating systems, and substantial dedicated hardware. Second, assuming such an environment is available, it is difficult for students to manage the development, deployment and testing of code on anything but a small number of nodes.

Distributed projects of a limited scope are sufficient for teaching students certain concepts in distributed systems. If one wants to teach students how to get two nodes to communicate over a socket or a remote procedure call, experiment with simple concurrency, or experience a three-machine deadlock, a simple application on 2-3 machines is more than sufficient. Our hypothesis is, however, that

SIGCSE'09, March 3–7, 2009, Chattanooga, Tennessee,USA

many concepts – such as scalability, heterogeneity, robustness, fault-tolerance, and the ability to think about composing a distributed systems out of a number of smaller pieces – can be taught much more effectively through interaction with larger and more complex systems. In addition, we hypothesize that students need a different set of skills to work with such systems, which includes the ability to think about the system at multiple levels of detail, from the broad system architecture to the details of individual components, and to learn how to navigate between these levels and chose the appropriate one at the appropriate time.

In this paper we present the rationale behind, design of and preliminary experience with a teaching tool that potentially can help students develop the necessary skill sets. This tool draws upon two important elements. First, it uses a *virtualization* layer to overcome many of the obstacles in resourcing, setup and configuration that make it difficult for instructors to work with substantial distributed system projects. Second, on top of this virtualization layer we build a novel *gaming environment* in which students work in teams to understand and keep running smoothly a nontrivial distributed system.

The basis of our game is a barebones but fully functional integrated website that combines features of social networking and video sharing. The site displays (mock) advertisements when different pages are viewed. The game generates artificial traffic against the site, and the overall system metric of health is the number of advertisements served and resulting "revenue." The game – a sort of firefighting exercise – begins when the game administrator "breaks" part of the system in some way. Students notice that the system's performance has degraded because ads and revenue have dropped off. They then work, typically in teams, to figure out what is wrong with the system and fix it.

We believe that this approach has several possible advantages, both in terms of engaging students and enabling them to learn about distributed systems in different ways. It should enable them to understand scalability issues when they see how a complex system scales and does not scale, understand by experience both the diversity of failure modes in a distributed system, and the important subtleties in designing distributed algorithms that can handle those failures. It should also help them gain skills in plunging into and understanding a complex system. This is an important point – most students who work with distributed systems will not start programming from scratch but rather need to understand and contribute to a pre-existing system. Our game can introduce them to this (sometimes daunting) undertaking in an engaging way.

We note that this paper is *not* a full-fledged research study as we are not yet at the point of conducting a rigorous evaluation of the game's educational impact. Rather, this paper should be viewed as a cross between an experience report, in which we share our experience in designing the game and deploying it in a limited way, and a philosophical paper which argues that more creative work is required to teach students about the reality of distributed systems, and that virtualization and gaming provide one interesting direction to pursue.

2. BACKGROUND

2.1 Why Virtualization?

Virtualization can be conceived of as an abstraction layer that separates the physical hardware from the operating system. In this model one physical machine can support multiple *virtual machines*, potentially each running different operating systems. Virtualization technology is having an enormous impact in the corporate environment because it enables organizations to simplify their infrastructure on fewer physical processors at substantial cost savings. As profound as its impact in the corporate environment, virtualization has potential for similar impact on the educational landscape. Virtualization technology enables us to create much richer and realistic networks for educational experiences, and support far greater numbers of nodes for the purposes of experimenting with distributed systems. For example, in one configuration of our gaming environment, a server purchased for less than $3000 easily supported 3 teams each with 6 servers, for a total of 18 virtual machines, by using VMWare virtualization software [17].

2.2 Why Gaming?

There is a growing body of literature on the potential of digital games to be an important teaching tool in a wide variety of disciplines. One example is a recent report by the American Federation of Scientists [13] in which they argued that the United States should prioritize the study of digital games for learning, because "(1) Many videogames require players to master skills in demand by today's employers, such as strategic and analytical thinking, problem solving, planning and execution, decision-making, and adaptation to rapid change. (2) They can be used to practice practical skills and important skills that are rarely used, to train for high-performance situations in a low-consequence-for-failure environment, and for team building. (3) Games offer attributes important for learning: clear goals, lessons that can be practiced repeatedly until mastered, monitoring learner progress and adjusting instruction to learner level of mastery, closing the gap between what is learned and its use, motivation that encourages time on task, personalization of learning, and infinite patience. (4) Today's students, the so-called digital natives, are poised to take advantage of educational games." Finally, stating the obvious, we believe that it is possible to develop games that will be **fun and engaging**; anything that is fun and engaging for the students has substantial educational potential.

2.3 Related Work

There has been a great deal of work on using *game development* in computer science curricula as a way to increase student engagement and retention, e.g. [8, 15]. There has been relatively less work, however, on developing gaming environments within which computer science concepts can be taught. Educators in the field of information security education have developed a number of capture-the-flag style games, many of which use virtualization to create the individual nodes e.g. [18]. The SimSE project [11] designed a simulation game for teaching about the software engineering process in a more realistic way. Game2Learn is a project that teaches introductory programming by using a multiplayer online role-playing game in which players need to carry out elementary programming activities to advance through the game[2]. Virtualization is also increasingly enabling innovation in CS curricula [5].

Until recently there has been relatively little work on innovative methodologies to teach distributed systems when

compared to other areas of computer science – overall there are 10-15 papers out of 1000+ published in the 2000-2008 SIGCSE and ITiCSE conferences. These include several papers on innovative ways to conceptualize distributed algorithms, [14, 16, 12], or on ways to teach specific distributed technology such as grid computing methodology [6]. There are several attempts at environments for building small-scale distributed projects such as DPLab[3], or for visualizing communication patterns in student programs [4]. None of these attempt to give students the opportunity to work with a large-scale integrated project of any sort.

As noted in the introduction, recently Google and IBM announced an initiative to help train the next generation of students to be equipped to build internet-scale distributed systems [7]. This effort has led to the devlopment of curricula focused on large-scale data processing using Hadoop, a powerful high-level tool for parallelizing large-scale computations over large collections of servers, e.g. [9]. The Hadoop/MapReduce model provides a very clean and highly abstract way to think about parallelizing large data-intensive applications. Infrastructure issues of fault tolerance, processor allocation, etc. are entirely hidden from the application programmer. In contrast, we are interested in training students who can think about that infrastructure layer and build the software necessary to support such high-level abstractions.

An assumption that underlies our work is that most institutions that offer degree programs in computer science give students at best experience with distributed systems of relatively limited scope. We attempted to validate this assumption by carrying out a limited survey of computing curricula in the 52 private campuses in New York Stat that offered bachelors degrees in computer science. Based on the websites of the institutions, we determined that 18 had dedicated elective courses in distributed systems and that only 7 of those 19 had significant projects. While our survey methodology is *quite rough*, we view it as some validation for our assumption.

3. GAME DESCRIPTION

3.1 The Distributed System

Our development goal was to build a distributed system of nontrivial complexity and then build a gaming environment that would present students with various failure modes that they would have to hunt down. The educational goals of this effort were several. First, to give students experience investigating and understanding complex distributed systems systems with which they are not intimately familiar, and provide experience in evaluating a large number of possible causes in order to find root cause and fix it; we believe this is a necessary skill set when dealing with real-world distributed systems. Second, to drive home in a concrete way the notion that in realistic distributed systems anything that can go wrong will go wrong (see [10] for a case study), and thus to teach in concrete ways different approaches to insure reliability. Third, to make concrete the notions of scalability by providing scenarios in which systems are pushed to the limits of their ability to scale. Fourth, and importantly, to create an engaging and fun experience to facilitate learning.

We thus needed to build a distributed system that was complex enough to be of interest, but at the same time was understandable by our students. At our institution we teach an elective undergraduate course in parallel and distributed systems that combines experience with the basic programming tools necessary to build simple distributed systems (threads, sockets, webservices), case studies of important real-world systems (e.g. distributed file systems) , and core algorithmic issues (e.g. time, fault tolerance, and replication). Each year the course has a different implementation project, which typically involves three major elements. First, a web site developed in PERL or PHP that interfaces to a backend database; second, a multithreaded server that communicates over sockets in order to accomplish various tasks necessary to the application, often storing results directly in the database; and third some sort of fault-tolerance achieved by implementation of variants of traditional distributed algorithms such as leader election, two-phase commit or replication protocols. The underlying application changes each year and has included a stock-trading application and a supply-chain management application.

In 2007 and 2008 the applications were mock social networking and video sharing applications. To develop our distributed system we chose to utilize these two applications, basing our codebase on some of the better assignments students turned in during the semester. As a result, we had confidence that the students had familiarity with appropriate concepts to understand the codebase of the project; on the other hand, our lab sessions were structured so that the students playing the game based on a certain application had not participated in that development project.

The combined system currently consists of a number of virtual machines: one DNS server, one MySQL Server, and multiple machines each of which hosts an Apache web server, as well as separate processes to handle a chat application, a video upload application, and a replication server that handles the "distributed systems logic" for fault tolerance and replication amongst the multiple servers. Each of these latter processes are coded in C++ based on the afore-mentioned student projects, and all utilize multiple threads and communicate over sockets. An end user who would interact with the system is directed by DNS to one of the web servers, and views content, uploads content and chats with other users. The Apache front end redirects the end users to the upload and chat servers, which invoke the replication servers to insure that they stay consistent in the presence of any faults and outages. Each time a web page is displayed (mock) advertisements are embedded in the page. These advertisements are critical to the game, because advertisements displayed and clicked on have revenue values associated with them; and revenue earned is the metric for game success.

3.2 Gameplay

We now explain how a student playing with the game interacts with it. A player is cast in the role of the systems administrator/engineer responsible for maintaining the distributed system. Players sit in teams of 2-3 in front of the web-based game console, which we call PDConsole. Currently the game flow is not automated but is managed by a game administrator who begins the game by generating artificial web traffic to the social networking sites and applications. This traffic generator causes web pages and their embedded advertisements to be transmitted and then follows embedded links and advertisements as well to generate additional traffic and revenue.

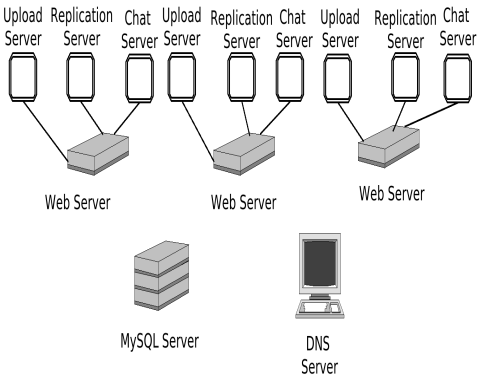

Figure 1: Sample System Diagram

The metric for the game is revenue from advertisements served. The PDConsole displays near-realtime graphs of revenue earned and advertisement traffic. It also provides status information on each of the constituent virtual machines and the applications running on it, including application state, application logs, and virtual machine statistics such as memory, CPU and disk usage. These real-time indicators are complemented by links to system documentation, windows to communicate with the game administrator and gain ssh access to the underlying virtual machines, and options to spend some revenue to upgrade the system by buying additional virtual machines or upgrading CPU/memory of the current virtual machines.

Once the system is in a steady state the game administrator modifies something in the system which will decrease revenue – we refer to these modifications as "System Issues." The players will likely first become aware of an issue when the revenue graph starts to drop off, and may start flashing yellow or red with captions indicating *Caution* or *Danger*. The players must then try to discover and fix the problem in order to return revenue to a healthy state. The speed with which they do this determines how quickly revenue is restored and how much revenue they gain over the course of the game.

Figure 2 illustrates the player console during a system issue. View and revenue graphs have dramatically fallen off, and the graphs are flashing Danger warnings. The player has opened up one view of various system statistics to try to troubleshoot the problem.

3.3 System Issues

We have implemented a number of system issues scenarios, classified by the categories of concepts we wish students to experience. We enumerate them here, noting that each system issue may touch on several categories.

We call the first category "Exploring a large and unfamiliar system – anything that might break in a large distributed system can and will." The issues in this category require the students to dig around the system to find relatively simple problems – a process is down, a rogue process is running and chewing up CPU, or a disk partition has filled up. Issues in these categories would as well serve as good warmups for students to develop familiarity with the overall system.

The second category has to do with scalability and covers two sorts of issues: what happens when the traffic levels rise

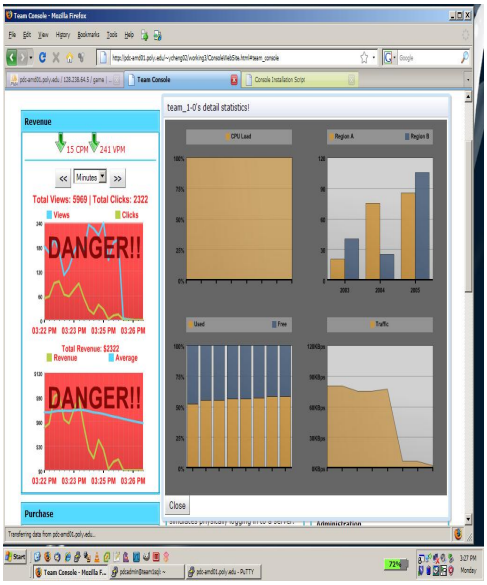

Figure 2: PDConsole during a System Issue

to levels that overwhelm some aspect of the system, and how to make sure that the underlying system works well as additional elements are added to it. If, for example, we raise the artificial traffic levels to a point beyond what the system can handle, the student may chose to expend revenue to upgrade the current machines or buy additional ones and add them to the pool of webservers serving traffic, or alternatively reconfigure Apache to limit the number of connections it can handle. If the students chose to grow the system by adding additional servers, the system must be architected to work well with more servers, and so potentially we will introduce other issues that only arise at higher numbers of servers.

The third category deals with distributed algorithms. Even simple distributed algorithms, because they need to respond gracefully to a multitude of failure scenarios, can be a bit subtle in how they are stated, and students often may not appreciate the subtleties of how the failure modes are handled. We therefore implement variants of the distributed algorithms with which the students are familiar that do not handle a failure case appropriately. For example, if we are using two-phase commit to insure that uploads of videos are appropriately replicated, we may induce some communications failure that is not handled well by the algorithm and that brings the video upload system to a grinding halt. In these cases the students need to look through the source code of the underlying distributed algorithm to figure out what is wrong.

Other categories include *Communications Failures*, which includes trying to fix situations in which two virtual nodes can not communicate and making sure that the system behaves appropriately in the presence of such situations, *Sockets and Threads* in which we introduce code that does not handle some aspect of sockets and threads correctly, and *Systems Administration*, in which students encounter problems that ultimately can be traced to configuration issues in Apache and MySQL for which the students need to dig through the online documentation of these third party packages to solve configuration problems.

4. EXPERIENCE AND FUTURE WORK

All development on this project has been carried out by undergraduates who are alumni of our elective course on parallel and distributed systems. Development started in the Fall of 2007, and an initial implementation was used in the Spring 2008 version of the course taking the place of one of the regularly scheduled laboratories for a subset of the students in the class. A small number of additional play/evaluation sessions were carried out in the Summer of 2008 with additional students who were alumni of the course. Overall, approximately 15 students participated, in teams of 2 or 3 individuals at a time. As a result, our experience with student interaction with the game is extremely preliminary at this point, and conclusions are at best anecdotal. Nonetheless, the sessions were valuable in developing some preliminary impressions (based on observations and subsequent discussions) and suggesting additional necessary work.

First, we observed that students responded well to the game, finding it fun and engaging. The energy level in the room was very high as students worked together to solve problems. Second, overall the students were capable of learning how to find their way around the system quickly enough so that they could work through an introduction and approximately 3 system issues in a 2 hour session. Third, as the two hour sessions progressed, the students developed enhanced awareness of the wide variety of the sorts of problems they needed to consider.

Virtualization proved to be an extremely valuable tool in designing the gaming environment. In addition to the simple advantage of supporting large numbers of virtual servers on one physical server, it enabled us to copy machines in order to provide uniform environments for different machines, and allowed us to let students modify their environment on the fly by upgrading machines or purchasing new ones.

Our most immediate future goal is to take this implementation and preliminary experiences and design experiments to make the effort a real contribution to the science of learning through games. In order to do this we need to define more precisely the skill sets that we hope students will develop through interaction with the game, and design pre and post evaluations to measure the impact of the game. Simultaneously, and guided by our experimental design, we need to refine our collection of system issues to precisely target the types of learning we hope to encourage. We also need to do additional development to support more varied types of game play flow. Certain types of problems lend themselves to solution in a high-energy group competition setting, but other issues will require deeper thought spaced out over a period of time. To facilitate this the game needs to support more automation, including pause and snapshot functionality so students can play at their pace.

5. ACKNOWLEDGMENTS

We gratefully acknowledge the input and encouragement of Julia Austin and Carl Skelton. Partially supported by NSF CPATH Grant 0722279, VMWare through the VMWare Academic Program, the Othmer Institute of the Polytechnic Institute of NYU, and the Polytechnic Institute of NYU's Summer Undergraduate Research Program.

6. ADDITIONAL AUTHORS

Additional authors: Chris Sherman (Polytechnic Institute of NYU).

7. REFERENCES

[1] ACM/IEEE. Computing curricula: Computer science. http://www.acm.org.

[2] T. Barnes, H. Richter, A. Chaffin, A. Godwin, E. Powell, T. Ralph, P. Matthews, and H. Jordan. Game2learn: A study of games as tools for learning introductory programming concepts. In *Proceedings of the ACM SIGCSE '07 Conference*, 2007.

[3] M. Ben-Ari and S. Silverman. Dplab: an environment for distributed programming. In *Proceedings of the ACM ITiCSE conference*, 1999.

[4] C. Brown and C. McDonald. Visualizing berkely socket calls in students' programs. In *Proceedings of ITiCSE 2007*, pages 101–105, 2007.

[5] W. I. Bullers, S. Burd, and A. F. Seazzu. Virtual machines - an idea whose time has returned: application to network, security and database courses. In *Proceedings of SIGCSE 2007*, pages 102–206, 2007.

[6] M. Holliday, B. Wilkinson, J. House, S. Daoud, and C. Ferner. A geographically-distributed, assignment-structured undergraduate grid computing course. In *SIGSCE 2005*, 2005.

[7] IBM. Google and ibm look to next generation of programmers. ibm.com/ibm/ideasfromibm/us/google/index.shtml.

[8] R. Jones. Design and implementation of computer games: A capstone course for undergraduate computer science education. In *Proceedings of the ACM SIGCSE 200 Conference*, pages 260–264, 2000.

[9] A. Kimball, S. Michels-Slettvet, and C. Bisciglia. Cluster computing for web-scale data processing. In *Proceedings of SIGCSE 2008*, 2008.

[10] S. Muir. The seven deadly sins of distributed systems. In *Proceedings of USENIX 1st Workshop on Real Large Distributed Systems*, 2004.

[11] E. O. Navarro and A. van der Hoek. Simse: an educational simulation game for teaching the software engineering process. *StIGCSE Bulletin*, 36(3), 2004.

[12] R. Oechsle and T. Gottwald. Disaster(distributed algorithms simulation terrain): A platform for the implementation of distributed algorithms. In *Proceedings of ITiCSE 2005*, pages 44–48, 2005.

[13] F. of American Scientists. Harnessing the power of video games for learning. http://fas.org/gamesummit/Resources/.

[14] W. Schreiner. A java toolkit for teaching distributed algorithms. In *Proceedings of ITiCSE 2002*, 2002.

[15] E. Sweedyk, M. de Laet, M. Slattery, and J. Kuffner. Computer games and cs education: why and how. In *Proceedings of SIGCSE 2005*, pages 256–257, 2005.

[16] A. Tikvati, M. Ben-Ari, and Y. B.-D. Kolikant. Virtual trees for the byzantine generals algorithm. In *SIGCSE 2004*, pages 392–396, 2004.

[17] VMWare. http://www.vmware.com.

[18] J. Walden. A real-time information warfare exercise on a virtual network. In *SIGCSE 2005*, pages 86–91, 2005.

name of the Bibliography in this case