MOVING CS50 INTO THE CLOUD

David J. Malan
Harvard University
School of Engineering and Applied Sciences
malan@post.harvard.edu

ABSTRACT

In Fall 2008, we moved Harvard College's introductory computer science course, CS50, into the cloud. Rather than continue to rely on our own instructional computing infrastructure on campus, we created a load-balanced cluster of virtual machines (VMs) for our 330 students within Amazon Elastic Compute Cloud (EC2). Our goals were both technical and pedagogical. As computer scientists, we wanted more control over our course's infrastructure (*e.g.*, root access), so that we ourselves could install software at will and respond to students' needs at any hour without an IT department between us and our systems. As teachers, we wanted easier access to our students' work (as via `su`) as well as the ability to grow and shrink our infrastructure as problem sets' computational requirements demanded. But we also wanted to integrate into the course's own syllabus discussion of scalability, virtualization, multi-core processing, and cloud computing itself. What better way to teach topics like those than to have students actually experience them. Although Amazon supported our experiment financially with credits, it was not without costs. Serving as our own system administers cost us time, as did some self-induced late-night technical difficulties. But the upsides proved worth it, as we accomplished our goals. We present in this paper what we did right, what we did wrong, and how we did both so that others can more easily build their own home in the cloud.

INTRODUCTION

Computer Science 50 is Harvard College's introductory course for majors and non-majors alike, a one-semester amalgam of courses generally known as CS1 and CS2. Most of the course's students (94%) have little or no prior programming experience. Although the course introduces some fundamentals of programming in Week 0 by way of Scratch [20], a graphical programming environment developed by MIT's Media Lab, the course immediately transitions to C in Week 1, where it spends much of the semester. Among the topics covered during that time are abstraction, encapsulation, data structures, memory management, and software development. The semester concludes with an introduction to web programming by way of PHP, SQL, and JavaScript plus XHTML and CSS.

With so much material to cover, the course itself moves quickly, demanding of students upwards of 15 hours per week outside of class, much of which is spent on programming assignments. With the exception of Problem Set 0, much of the course's workload involves a command line. Students write C code with their choice of text editor (*e.g.*, `emacs`, `nano`, or `vim`), compile that code using `gcc`, and then debug that code using `gdb` and `valgrind`. Toward term's end, `httpd` and `mysqld` enter the picture as well.

The course's problem sets, meanwhile, vary in their computational and I/O needs. Whereas Problem Set 1 has students implement some bite-sized C programs, Problem Set 2 tasks them with implementing ciphers and cracking DES-encrypted passwords. Problem Set 5 has students recover megabytes of photos from a `dd` image of an "accidentally formatted' CompactFlash card. And Problem Set 6 hands them a 143,091-word dictionary that they need to load efficiently into a speller-checker.

With 330 undergraduates and 30 teaching fellows, the course tends to consume its share of cycles. Over the years, the course has relied for its needs on a load-balanced cluster of servers comprising Alphas running Digital (Tru64) UNIX or, more recently, Xeons running Ubuntu Linux. Via SSH (or, formerly, Telnet) can students log into that cluster, where a shell (`tcsh`) and NFS-mounted home directory await. Once done with their homework, students submit it electronically to the course's own account on that same cluster via a setuid script.

Unfortunately, those home directories include only 100 MB of storage. Though plenty for source code, it has proved insufficient for students' inevitable core dumps as well as for problem sets involving large datasets. Because we, the course, do not manage the cluster, having those quotas increased is never as easy as would be ideal.

Quite often, too, has the cluster suffered technical difficulties that get resolved no sooner than next business day. Unfortunately, our students tend to work 24/7, and prolonged technical difficulties not only induce stress, they affect the syllabus's tight schedule if we resort to extensions.

Because this cluster is used not only by our students but also by other students, faculty, and staff, we do not have root access. When students need help, they must email us their code or go through the process of submitting electronically, neither of which is ideal for efficiency. System-wide installation of software, meanwhile, requires

the help of IT, who often prefer not to install or upgrade software, lest it affect other users. And so we often resort to installing (sometimes with ease, sometimes with difficulty) software we need in the course's own account (*e.g.*, within `/home/cs50/pub/`), thereafter altering students' `$LD_LIBRARY_PATH`, `$LD_RUN_PATH`, `$LIBRARY_PATH`, `$MANPATH`, and `$PATH` via a `.cshrc` that their shells source upon login.

And so we have long craved our own cluster of systems that we could optimize for the course's technical and pedagogical goals. But with that vision comes a need not only for cash but also for space, power, and cooling, none of which we, as a course, have in any supply. In fact, the university's own supply of those resources is increasingly limited. Moreover, we would prefer not to assume responsibility for technical problems (*e.g.*, failed hardware) that might otherwise distract us from our students. And we simply do not need an entire cluster of systems every day of the year, as our computational needs vary with problem sets' deadlines.

We thus moved CS50 into the cloud, building within Amazon Elastic Compute Cloud (EC2) [3] our own load-balanced cluster of x86-based virtual machines (VMs) running Fedora Core 8. Not only did we gain the autonomy we sought, we also gained pedagogical conveniences, every one of which helps with 330 students. The ability to see and help students debug code prior to submission *in situ* (as via `su`), by itself, proved invaluable, eliminating unnecessary exchanges of code via email and premature electronic submissions.

In the section that follows, we elaborate on what it means to move into the cloud and provide background on EC2 specifically. We then offer implementation details on how we built our new home in the cloud. Thereafter, we present our results and admit to mistakes we made during the term. We then conclude and reveal our plans for next year.

BACKGROUND

Although it once represented, often in cartoon form, any network beyond one's own LAN, "the cloud" now refers to on-demand computational resources whose provision usually relies on virtualization. In more real terms, cloud computing means that courses like ours (or, more generally, users) can pay for access to servers when and only when we actually need them. Those servers just so happen to be VMs, otherwise known as virtual private servers (VPSes), that live alongside other customers' VPSes on hardware that we ourselves do not own.

Amazon EC2 offers precisely this service, as does an exploding number of competitors (*e.g.*, Linode [16], unixshell# [24], and VPSLAND [25]). One of the largest providers, though, EC2 offers multiple *availability zones*, "distinct locations that are engineered to be insulated from failures," in two regions (US and Europe). In the event that some availability zone becomes, if ironically, unavailable, an EC2 *instance* (Amazon's parlance for a virtual machine) can be re-launched in another zone altogether. Users need not even be informed of such changes thanks to EC2's *elastic IP addresses*, which can be re-mapped within minutes from unavailable instances to available ones.

An instance, meanwhile, can boot any one of several operating systems by loading at startup an *Amazon Machine Image* (AMI). Amazon currently offers AMIs pre-configured with Debian, Fedora, Gentoo Linux, OpenSolaris, openSUSE Linux, Red Hat Enterprise Linux, Oracle Enterprise Linux, Ubuntu Linux, and Windows Server [3], but customers can create [7] and share [6] their own too. Instances' (virtual) disks are typically ephemeral, though, whereby any changes made to an instance's filesystem (vis-á-vis some AMI) are lost upon shutdown (though not upon restart). But EC2 provides persistent storage via a service it calls *Elastic Block Store* (EBS) [2], whereby an EBS *volume* (which can be 1 GB to 1 TB in size) can be attached as a block device to any one instance. Changes to that volume persist even after an instance is shut down, and that volume can be re-attached later to another instance altogether. EBS also supports snapshots, whereby copies of volumes can be archived using *Amazon Simple Storage Service* (S3) [4].

Instances can be created and managed from any networked computer using EC2's Web-based management console, Java-based command-line tools [10], or SOAP- and HTTP-based APIs [7]. A number of third-party tools facilitate management too [8]. We ourselves are fans of the Elasticfox Firefox Extension [9]. Links to the EC2 resources we found most helpful appear in this paper's appendix.

Although "cloud" computing is perhaps the buzzword *du jour*, we have found that few, if any, services offer precisely what EC2 does: an ability to provision on demand within seconds any number of (Linux) VMs without human assistance or additional contracts. Perhaps related in spirit to EC2, though, are "grid" frameworks like Google's App Engine [14], Microsoft's Windows Azure [19], University of Chicago's Globus Toolkit [13], University of California's BOINC [11], Princeton's PlanetLab [23], and University of Washington's Seattle [12]. But frameworks like these, inasmuch as they provide APIs and managed environments for execution of distributed applications and services more than they do familiar command lines and binaries, represent solutions to problems that we did not have. We simply wanted some VMs that we could wire together in order to implement a topology and workflow already familiar to us, per Figure 1.
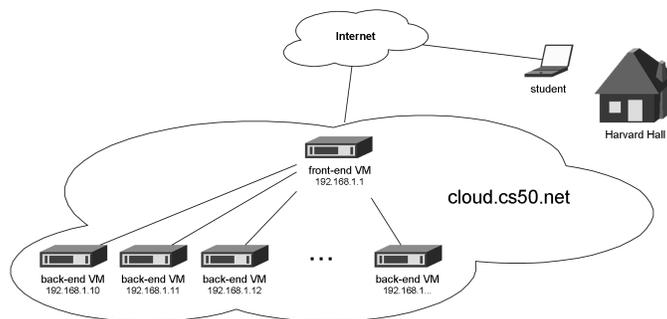
Figure 1: Our cluster in the cloud comprised one "front-end" VM providing LDAP, NAT, and NFS services to multiple "back-end" VMs. From their laptops on campus or elsewhere, students SSHed throughout the semester to `cloud.cs50.net`, which mapped via DNS to an elastic IP address bound to our front-end VM, which routed each student to the back-end VM with the fewest connections.

IMPLEMENTATION DETAILS

For an introductory course like ours, it was not our goal to provide every student with his or her own VM but, rather, to provide, for the sake of uniform experience, a managed environment in which all students could implement problem sets. Although our own local infrastructure was imperfect, its overarching design had always worked well: a load-balanced cluster on which each student had his or her own shell account and home directory. And so we sought to replicate that design inside the cloud, thereafter improving upon it.

Although EC2 provides firewalling services between instances and the Internet at large (via what it calls *security groups*), it does not offer load balancing (except on high-numbered ports) for non-HTTP traffic, which we very much needed for SSH's sake. And so we had to implement our own load balancer. (In the interests of simplicity for students, we were not willing to run `sshd` on a port other than 22.) We were determined to avoid a DNS-based solution (whereby a fully qualified domain name resolves, via an A record, to multiple IP addresses), as DNS caching by servers and clients can induce dead ends lasting hours or days.[1] Prior to term's start, we knew of no non-DNS solutions in use within EC2's cloud, and so we set out to implement our own.[2]

After much trial and error, we came up with a solution. We first spawned two instances using EC2's AMI for a 32-bit installation of Fedora Core 8 (ami-2b5fba42).[3] We then created a TCP tunnel between those two instances using VTun [18].[4] We assigned one end of the tunnel an IP address of 192.168.1.1 (much like a home router) and the other an IP address of 192.168.1.10. We henceforth referred to 192.168.1.1 as our "front-end" VM and 192.168.1.10 as a "back-end" VM, per Figure 1. We next enabled IP forwarding on the front-end:

```
echo "1" > /proc/sys/net/ipv4/ip_forward
echo "0" > /proc/sys/net/ipv4/conf/all/send_redirects
echo "0" > /proc/sys/net/ipv4/conf/default/send_redirects
echo "0" > /proc/sys/net/ipv4/conf/eth0/send_redirects
echo "0" > /proc/sys/net/ipv4/conf/tun0/send_redirects
```

Meanwhile, we told the back-end to route any traffic from the front-end back through the front-end:

```
echo 80 lvs >> /etc/iproute2/rt_tables
ip route add default via 192.168.1.1 dev tun0 table lvs
ip rule add from 192.168.1.10 table lvs
```

We repeated this process for additional back-ends (with IP addresses 192.168.1.11, 192.168.1.12, *etc.*). Next, using Linux Virtual Server (LVS) [26], we told the front-end to route incoming SSH connections to one of the back-ends according to LVS's *Weighted Least-Connection* heuristic, whereby the back-end with the fewest existing connections wins the next one:[5,6]

---

[1]In the event a host in a cluster goes down, its IP address can certainly be removed from a load-balancing A record. If some client (or some other DNS server between it and the host) has cached that address, though, the client may continue to look for the host at that address until the cache actually expires.

[2]We are indebted to M. David Peterson [17] for his help with our quest.

[3]We avoided EC2's 64-bit AMIs lest we discover mid-semester (*i.e.*, too late) that some software we want only exist in a 32-bit flavor.

[4]We also considered OpenVPN [22] but preferred the simplicity of VTun.

[5]We utilized Weighted Least-Connection instead of LVS's *Least-Connection* heuristic so that we could take back-ends out of rotation temporarily simply by setting their weights to 0.

[6]So that the course's staff could still SSH to the front-end itself, we ran its instance of `sshd` on a TCP port other than 22.

	CPU	Disk	I/O Requests	Bandwidth	Cost
Sep	2,275 Hrs	125 GB	45,348	14 GB	**$274**
Oct	3,425 Hrs	108 GB	93,257,314	191 GB	**$657**
Nov	5,484 Hrs	199 GB	337,019,916	239 GB	**$1,252**
Dec	5,206 Hrs	300 GB	427,639,962	52 GB	**$1,184**
Jan	5,208 Hrs	300 GB	1,502,614,186	62 GB	**$1,298**

Table 1: **Resources consumed by our virtual cluster between September 2008 and January 2009. CPU refers to the number hours our instances collectively ran; between October and November, we transitioned from single-core instances to dual-core instances. Disk refers to space used (for home directories and snapshots) on EBS volumes. I/O Requests refer to our I/O activity on our EBS volumes, as measured by** `iostat`. **Bandwidth refers to network traffic to and from our cluster. Per the upward trends over time, our students' utilization of cloud resources rose with problem sets' computational needs.**

```
ipvsadm -C
ipvsadm -A -t 10.253.131.161:22 -s wlc
ipvsadm -a -t 10.253.131.161:22 -r 192.168.1.10:22 -m -w 1
ipvsadm -a -t 10.253.131.161:22 -r 192.168.1.11:22 -m -w 1
ipvsadm -a -t 10.253.131.161:22 -r 192.168.1.12:22 -m -w 1
```

At the time, 10.253.131.161 was the private IP address that EC2 itself had assigned to our front-end instance. Finally, we enabled network address translation (NAT) on our front-end:

```
iptables -t nat -F
iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to 10.253.131.161
```

And we mapped `cloud.cs50.net` via DNS to an elastic IP address bound to our front-end.

Not only did our front-end ultimately serve as a load-balancing NAT router, it also provided LDAP and NFS services to our back-ends as well. Usernames and groups lived in a tree on the front-end, and home directories lived on an EBS volume attached to the front-end that was then exported to each of the back-ends. Backups, meanwhile, were managed by `rsnapshot` [21]. In fact, even though we preached the virtues of source control to our students, we took it upon ourselves to take snapshots of their home directories every 5 minutes. Thanks to its use of `rsync` and hard links, `rsnapshot` maintains snapshots efficiently; it does not maintain multiple copies of files that have not changed. The sheer number of panic attacks we averted among students by empowering them to access so many versions of their code easily justified the additional gigabytes.

We installed on each of our back-ends `gcc`, `gdb`, `valgrind`, and more.[7] On all of our instances did we also install Webmin [15], a web-based GUI that facilitates system administration. Ultimately, we relied entirely on freely available software and tools, almost all of it installed via `rpm` and `yum`.

It's worth noting that EC2 offers different *types* of instances (*i.e.*, classes of virtual hardware). We began Fall 2008 running 3 "small instances" (m1.small), each of which had 1 (virtual) core rated at 1 *EC2 Compute Unit* (the equivalent of a 1.0 – 1.2 GHz 2007 Opteron or 2007 Xeon processor) and 1.7 GB of RAM. Because problem sets' computational needs rose over time, we ended the term running as many as 6 *high-CPU medium instances* (c1.medium) at once, each of which had 1.7 GB of RAM and 2 (virtual) cores, each rated at 2.5 EC2 Compute Units (*i.e.*, 2.5 – 3.0 GHz).

Our front-end was indeed a potential single point of failure in our cluster. However, because an EBS volume (*e.g.*, students' home directories) can only be attached to one instance anyway, we considered that weakness an acceptable risk, particularly since we could re-spawn an identical instance within minutes if necessary, in a different availability zone no less. Anecdotally, we're happy to say that our front-end never actually went down.

RESULTS

At the very start of Fall 2008, we briefly relied, as usual, on our local cluster of systems. On 3 October 2008, though, `cloud.cs50.net` debuted among a subset of students. Two weeks later, we moved all 330 students into the cloud, where they remained until term's end in January 2009. Table 1 details our cluster's consumption of resources during that time.

We ultimately judged our new home in the cloud a success, as we gained precisely the technical autonomy and pedagogical conveniences that we had set out to claim. Not only did the cloud empower us to install software and change settings at will, it allowed us to examine students' code and reproduce errors therein *in situ* via `su` and `sudo`, thereby expediting a common scenario. To be sure, we could have had these same powers on campus had we run our own cluster with actual hardware. But `cloud.cs50.net` required no space, no power, no cooling from us. Someone else (*i.e.*, Amazon) kept an eye on the hardware's hard drives and fans. And provisioning more capacity

---

[7]For efficiency's sake, we actually configured just one back-end initially, thereafter burning our own AMI based on it, and then spawning additional instances of that new AMI.

for students was as simple as clicking a button. In a word, reliability and scalability proved easy (or, at least, easier) in this cloud. In fact, EC2's own Service Level Agreement (SLA) [1] commits to 99.95% uptime, which, we daresay, is even higher than we've experienced on campus in terms past.

But our experience was not without hiccups, most the result of mistakes made by this paper's author. Simply preparing the cluster took multiple weeks (perhaps 80 hours in total), largely because we tackled at once so many packages unfamiliar to us. Our quest to implement a load-balancing NAT router within EC2's confines took particularly long but proved well worth the journey. We temporarily ran out of disk space on our EBS volume midway through term, the result of our having been conservative with space. And we experienced I/O delays around some problem sets' deadlines until we realized that OpenLDAP's `slapd` was keeping open too many file descriptors.

Although throughput between our virtual cluster and computers on campus was high, we did suffer latency, whereby X applications (*e.g.*, `emacs`) performed poorly. We are not yet confident that we can eliminate that particular problem next time around.

Our time in the cloud cost us less than $5,000 (roughly $15 per student), although Amazon defrayed those costs with EC2 credits. In fact, Amazon now offers *Teaching Grants* "supporting free usage of [EC2] for students in eligible courses" [5], so our arrangements are by no means exclusive. Had we been more conservative with VMs and more attentive to usage, scaling our cluster's size up and down more frequently, we are confident that we could have accomplished our goals for less, perhaps even half this amount.

CONCLUSION

In Fall 2008, we moved Harvard College's CS50 into the cloud via Amazon EC2. We turned to the cloud in search of technical autonomy and pedagogical conveniences, both of which we realized in our new home. We even added capacity to our virtual cluster as the semester progressed. Yet we did not need to find space, power, or cooling on campus for `cloud.cs50.net`. And we did not even need to find dollars, thanks to support Amazon has since institutionalized for educators at large.

To be sure, building this new home did cost us time, but we think it time very well spent. Not only did we accomplish our goals, we introduced 330 students first-hand to the cloud. We not only told students about scalability, virtualization, multi-core processing, and cloud computing, we had them experience each. Ironically, had we not told them that `cloud.cs50.net` lived somewhere other than campus, most would not ever have known. But that in itself perhaps speaks to the potential of cloud computing itself.

In Fall 2009, not only will we continue to spend time in the cloud, we will also make available a stand-alone replica of `cloud.cs50.net`, a virtual machine that students can download and run on their own Macs and PCs, even offline. Our goal now is to distribute a whole "course in a box."

APPENDIX

To get started with EC2, we recommend these resources:

Amazon Elastic Compute Cloud
`http://aws.amazon.com/ec2/`

Getting Started Guide
`http://docs.amazonwebservices.com/AWSEC2/latest/GettingStartedGuide/`

Developer Guide
`http://docs.amazonwebservices.com/AWSEC2/latest/DeveloperGuide/`

ACKNOWLEDGEMENTS

References

[1] Amazon EC2 Service Level Agreement. `http://aws.amazon.com/ec2-sla/`.

[2] Amazon Elastic Block Store. `http://aws.amazon.com/ebs/`.

[3] Amazon Elastic Compute Cloud. `http://aws.amazon.com/ec2/`.

[4] Amazon Simple Storage Service. `http://aws.amazon.com/s3/`.

[5] AWS in Education. `http://aws.amazon.com/education/`.

[6] Amazon EC2. Amazon Machine Images. `http://developer.amazonwebservices.com/connect/kbcategory.jspa?categoryID=171`.

[7] Amazon EC2. Developer Guide. `http://docs.amazonwebservices.com/AWSEC2/latest/DeveloperGuide/`.

[8] Amazon EC2. Developer Tools. `http://developer.amazonwebservices.com/connect/kbcategory.jspa?categoryID=88`.

[9] Amazon EC2. Elasticfox Firefox Extension for Amazon EC2. `http://developer.amazonwebservices.com/connect/entry.jspa?externalID=609&categoryID=88`.

[10] Amazon EC2. Getting Started Guide. `http://docs.amazonwebservices.com/AWSEC2/latest/GettingStartedGuide/`.

[11] D. P. Anderson. BOINC: A System for Public-Resource Computing and Storage. pages 4–10, 2004.

[12] J. Cappos, I. Beschastnikh, A. Krishnamurthy, and T. Anderson. Seattle: A Platform for Educational Cloud Computing. *SIGCSE Bull.*, 41(1):111–115, 2009.

[13] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. In *IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779*, pages 2–13, 2005.

[14] Google Inc. Google App Engine. `http://code.google.com/appengine/`.

[15] Jamie Cameron. Webmin. `http://www.webmin.com/`.

[16] Linode, LLC. Linode. `http://www.linode.com/`.

[17] M. David Peterson. `http://mdavid.name/`.

[18] Maxim Krasnyansky. VTun. `http://vtun.sourceforge.net/`.

[19] Microsoft Corporation. Windows Azure. `http://www.microsoft.com/azure/`.

[20] MIT Media Lab. Scratch. `http://scratch.mit.edu/`.

[21] Nathan Rosenquist and David Cantrell *et al.* rsnapshot. `http://rsnapshot.org/`.

[22] OpenVPN Technologies, Inc. OpenVPN. `http://openvpn.net/`.

[23] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the internet. *SIGCOMM Comput. Commun. Rev.*, 33(1):59–64, 2003.

[24] TekTonic. unixshell#. `http://www.unixshell.com/`.

[25] VPSLAND.com, LLC. `http://vpsland.com/`.

[26] Wensong Zhang. Linux Virtual Server. `http://www.linuxvirtualserver.org/`.