A

Using Clouds for MapReduce Measurement Assignments[1]

ARIEL RABKIN, CHARLES REISS, RANDY KATZ and DAVID PATTERSON, UC Berkeley

We describe our experiences teaching MapReduce in a large undergraduate lecture course using public cloud services and the standard Hadoop API. Using the standard API, students directly experienced the quality of industrial big-data tools. Using the cloud, every student could carry out scalability benchmarking assignments on realistic hardware, which would have been impossible otherwise. Over two semesters, over 500 students took our course. We believe this is the first large-scale demonstration that it is feasible to use pay-as-you-go billing in the Cloud for a large undergraduate course. Modest instructor effort was sufficient to prevent students from overspending. Average per-pupil expenses in the Cloud were under $45. Students were excited by the assignment: 90% said they thought it should be retained in future course offerings.

Categories and Subject Descriptors: K.3.2 [**Computer and Information Science Education**]: Computer science education—*Human Factors*

General Terms: Economics, Management, Measurement

Additional Key Words and Phrases: Cloud computing, education, MapReduce

1. INTRODUCTION

Cloud computing is a major and disruptive change in how computing services are delivered. The term "cloud" has unfortunately been used for several different related concepts. In this paper, we refer to what is sometimes called a "public cloud" — a service that allows large quantities of computational resources to be allocated in a pay-as-you-go manner with minimal prior arrangement [Armbrust et al. 2009]. Thanks to the cloud, users can conveniently obtain access to thousands of cores and terabytes of memory at a moderate cost.

Modern software can put these resources to use. Cluster computing has matured to the point where large-scale analysis can relatively easily use commodity machines like those provided by public cloud providers. The most prominent example of such software is MapReduce [Dean and Ghemawat 2008], a data-processing framework that handles load balancing and faults with little programmer intervention. In exchange, the programmer must fit their data-processing task into a limited API. This programming model has proved useful and expressive enough that a large ecosystem has evolved around MapReduce and similar frameworks, including SQL-like high-level languages [Thusoo et al. 2009; Gates et al. 2009] and language-integration for writing workflows of many MapReduce programs [Chambers et al. 2010; Cloudera 2012].

With these large-scale data processing systems, the public cloud now makes it practical for anyone to quickly do complex analyses of large datasets. This change is beginning to percolate into the undergraduate curriculum. Many universities have begun offering courses that cover MapReduce and related distributed execution systems [Kim-

[1]A preliminary version of this paper appeared at the SIGCSE 2012 conference.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

ball et al. 2008; Johnson et al. 2008; Brown 2009]. These courses prepare students for a world in which large-scale distributed "big data" processing is routine.

This paper describes an effort to integrate the cloud into the lower-division machine structures course at UC Berkeley (CS 61C, titled "Great Ideas of Computer Architecture") in the Fall 2010 - Spring 2011 academic year. This course is typically taken in the third or fourth semester, after some prior introductory programming courses.

There were two distinctive aspects of our course design. First, we used the public cloud. Second, we used Hadoop, the standard open-source MapReduce implementation, in a lower-division course without giving students a more beginner-friendly interface than the Hadoop Java API. Both these decisions are unusual; we believe we are the first to make the former choice. Our intent in this paper is to supply guidance to practitioners seeking to incorporate either or both elements into their own courses.

To make the course more relevant to current computing challenges, we made parallelism a central theme. We wanted to emphasize parallelism at all levels, from the inherently parallel nature of hardware logic blocks through multicore systems and all the way up to warehouse-sized shared-nothing clusters. Data-flow frameworks like MapReduce are the most successful parallel programming model for commodity hardware clusters. These frameworks embody solutions to many of the fundamental challenges of coarse-grain parallelism: handling failures, dividing up work into independent chunks, handling inconsistent performance across the cluster, and so on.

Cloud computing makes the cost of computing explicit. For many instructors and pupils, this will be a novel environment. For others, though, it will be seen as a reversion to an era before the personal computer, in which students were given quotas for the computing resources available to them. There is an important distinction, however: in the cloud, users choose whether to consume resources sequentially or in parallel. This choice offers a tradeoff between efficiency and time-to-completion, which was not generally visible in the mainframe era but which is a significant aspect of parallel computing, and an important lesson we sought to teach.

1.1. Instructional Goals

Our interest in teaching MapReduce was primarily because we wanted to teach our students about datacenters. We wanted to teach students about the aspects of MapReduce that are well-suited to datacenter environments: the framework handles fault-tolerance and load-balancing at large scale and without programmer intervention. These features only make sense in the context of an execution distributed across several machines. Such an execution environment is also necessary to motivate the tradeoffs MapReduce makes so that fault tolerance and load balancing are simpler. For example, without the distributed context, the lack of access to shared state and serialization of all intermediate values cannot be explained.

We also wanted to use the student's experience to help explain the economics of datacenters. The design of MapReduce can be explained by the challenges modern datacenter designs create: their cheaper hardware is unreliable and does not achieve consistent performance (especially when it is heavily shared). MapReduce illustrates the tradeoff that this cheaper hardware reflects: datacenter operators compensate for less reliable hardware with more reliable software. Datacenter operators further achieve economies of scale in the support for this hardware (buildings, cooling, staffing, etc.). We wanted students to understand these costs.

We also wanted to teach about cloud computing itself. We believe the principal difference (versus traditional colocation) arises from the payment model. With cloud computing providers, one has *cost associativity*[Armbrust et al. 2009]: k hours of time on one computer costs essentially the same as one hour of time on k computers. Cost associativity is important because it allows users with modest resources to occasionally

run large parallel computations. Since cost associativity makes the largest difference for end users, we felt that this was the most important concept for our students to understand about cloud computing.

1.2. Use of the Cloud

We were driven to use a public cloud by the conjunction of our instructional goals. First, we thought the experience of using public cloud infrastructure and seeing the actual dollar cost of compute resources would be a valuable one for students. This would illustrate cost associativity to our students. Additionally, since the prices cloud computing providers offer tend to reflect their real costs, experiencing these costs also helped emphasize our unit on datacenter economics. We believed that seeing the costs of their example computations on this infrastructure would help them understand these economics.

Additionally, focusing on cost in combination with runtime in this contexts gave a natural context for students to understand the incentives behind parallelization. By using cost as an evaluation metric, students would have a way to see that their more parallelized computations were less efficient even though they were usually substantially faster.

Second, we wanted students to experience running and debugging distributed MapReduce jobs on a significantly-sized cluster. Our ultimate pedagogic goal was to emphasize the aspects of MapReduce that we believe have made it more successful than competing distributed processing layers. The natural way for students to observe MapReduce's automatic load balancing is to see how a MapReduce implementation can take advantage of additional resources. Unfortunately, with current implementations of MapReduce, this advantage does not occur until relatively large data sizes. Since Hadoop was designed for large, long jobs, it has relatively large startup overheads. Thus, on a typical simple MapReduce task, one needs at least hundreds of megabytes and probably gigabytes of data to observe substantial parallel speedups.

For students to observe speedup consistently, their programs must be isolated from other students'. Public cloud providers such as Amazon Web Services offer virtualized platforms with consistent performance. This gives customers predictable value-for-money. While the consistency is not perfect, some degree of performance variability is a fact of life in many execution environments.

Machine structures is a large course at our institution: The Fall '10 term had 170 students; the Spring '11 offering had 320. Given the size of our student body, using purely university IT resources would have imposed unacceptable infrastructure costs. To achieve the isolation we required would require several machines for each student, even though the students would only use these resources for a brief period of time. Even if we had a large budget to purchase computing resources and support at our institution, we would not be able to justify purchasing such a large cluster which would ordinarily be at such low utilization.

1.3. Research Questions and Methods

The focus of this paper is primarily on the opportunities and challenges offered by the cloud, rather than on teaching big data, parallelism, or MapReduce. We answer four research questions:

— Can we effectively manage cloud costs in a large introductory class, given current billing models?
— Can we use rented cloud hardware to demonstrate principles of parallelism using MapReduce?

— What difficulties do lower-division students confront when using current industry-grade tools like Hadoop on Amazon's Elastic Compute Cloud (EC2)?
— What challenges do instructors face in teaching these topics?

We use instructional costs, student survey results, and the quality of graded student work to evaluate the success of our course. As we will show, using the cloud worked well. Costs were moderate, students were happy. We made mistakes along the way, particularly in terms of preparing students for "big data" programming. We describe the lessons we learned and what we intend to do differently going forward. We focus on the second offering of the course, in the Spring, which was larger, better documented, and benefited from the experience of the first offering.

During the Spring semester, we administered three surveys to students: one over the Winter break before the course, once in the middle of the semester, and once at the end of the term. The first and last surveys are relevant to this paper. At the time of the opening survey, we had email addresses for two-thirds of the students who would ultimately enroll. Of these, 80% responded. The final survey had responses from half the class enrollment.

2. RELATED WORK

Several schools are now teaching MapReduce as part of their undergraduate curricula. Here, we summarize these efforts and contrast them with our own.

In the Spring of 2007, the University of Washington taught an experimental course covering Hadoop for upper-division undergraduates [Kimball et al. 2008]. (This course has since become a regular offering.) The emphasis was on the use of Hadoop (the standard open-source MapReduce implementation) to solve practical problems at large scale. Students spend several weeks on a large project. In contrast, we are teaching Hadoop to lower-division undergraduates, in a machine structures course that includes MapReduce as just one unit.

Both Tufts and the University of Maryland have offered "big data" courses with a strong focus on MapReduce and related technologies such as Pig [Gates et al. 2009], a higher-level parallel programming framework atop MapReduce [Couch 2011; Lin 2011]. Maryland has made use of EC2 for this purpose. As with UW, this was in the context of a small upper-division class, with MapReduce, not pricing or parallelism, being the focus.

UC Berkeley has experimented with integrating Hadoop MapReduce very early into the curriculum, covering it briefly in the initial programming course. This course is taught in Scheme, not Java, using a custom-written Scheme-to-Map-Reduce glue library [Johnson et al. 2008]. Performance tuning is not a goal, nor is understanding the mechanisms behind parallel execution.

Even small schools have been able to cover Hadoop. St. Olaf College has a Hadoop cluster, managed by student volunteers and used in several courses [Brown 2009]. The cited paper is notable for the suggestion that schools should explore "obtaining 'cloud' resources on demand." Our paper represents a large-scale demonstration and evaluation of that possibility. Subsequent effort at St. Olaf has been directed towards simplifying Hadoop, hiding the details from students [Garrity et al. 2011]. As at Berkeley, considerable effort was made to have students write MapReduce programs in Scheme, hiding the Hadoop Java APIs and command-line job submission. In contrast, we took the opposite approach, exposing students to the real industrial tools and documentation.

Harvard has experimented with using EC2 to provide compute resources to their introductory CS course [Malan 2010]. Students, however, were completely insulated from provisioning and billing. Effectively, the Cloud was used as a scalable replace-

ment for local IT infrastructure, rather than to allow assignments that could not have been taught otherwise.

3. COURSE DESIGN

As we mentioned, our overall goal was to re-orient our machine structures course around the theme of parallelism. (As part of this reorientation, the course title was changed from "Machine Structures" to "Great Ideas of Computer Architecture." We took a top-down approach, starting with MapReduce, which we presented as an example of coarse-grained and high-level task parallelism. We went into less detail about the hardware support used to implement MapReduce than future parallelism assignments would. Instead, the focus of the MapReduce assignments was illustrating the economics behind datacenters and introducing students to parallelism.

The MapReduce assignment also serves as an introduction to performance measurement of parallel programs that students will do later in the course. Later assignments, mostly labs, in the course included examples meant to demonstrate shared-memory multithreading (including false sharing), vector instructions, and cache-aware optimizations. Students would implement the changed version of the code and compare its performance to the original on our lab machines, similar to the performance experiments they previously performed for the MapReduce assignment. Generally, these examples would be smaller than our MapReduce assignment but more closely affected by the microarchitecture and thus tied to core machine structures material.

We also had two large parallelism-related projects later in the course. One of these was an optimization assignment for a C program. The optimization assignment included a similar component where students would compare the speedup they experienced (on a multicore machine) to a serial version, similar to what they did previously for the MapReduce assignment.

Our projects, the three parallelism-related projects and an additional processor simulation project were each two-week assignments. Projects had an intermediate milestone deadline to help students start early. Projects were supported by weekly labs, most of which served as tutorials for the project assignments. For the MapReduce project we had one (Fall semester) or two (Spring) such labs for the MapReduce and cloud computing unit. For the labs, students were permitted partners, but the projects were done individually. Additionally, we had three hours of lecture per week, assigned some short problem sets, and gave two written exams. Students were assessed primarily on the projects and the exams.

3.1. Assignment context

We wanted students to understand the sort of data processing that cloud computing makes available. Accordingly, we designed our assignments to have sufficiently high resource needs that it would be obvious why one would want to rent computing resources rather than use one's own machines. We choose data processing tasks (rather than, for example, a web application use case) because the cost and time benefits of parallelizing are more apparent. Data processing assignments gave students a clear metric for efficiency: cost per gigabyte of input.

It was important to us that, as much as practical, students used the same tools that professionals use. One reason was permitting more direct application of the assignment to student's own processing tasks. Another is that we did not want to give students the impression that performance anomalies and debugging problems were the consequence of us providing only a "student-quality" framework. We wanted students to understand that these are pervasive issues in dealing with real distributed systems. Further, using off-the-shelf tools will reduce the maintenance burden on future course staff.

Table I. Assignments used. F= Fall, S=Spring, FS = both

Sem.	Lines of Code	Description	Dataset Size (bytes)	Corpus
FS	0	Lab: performance measurement, introduction to EC2 tools	20,000 MB (F); 8,000 MB (S)	Wikipedia (F); Usenet (S)
F	~ 50	Project: PageRank [Page et al. 1998] in C, using Hadoop Streaming locally and then on EC2	650 MB	Web graph
S	~ 10	Lab: writing MapReduce programs in Java, running locally	34 MB	Usenet
S	~ 50	Project: computing a text-comparison metric locally and then on EC2	8,000 MB	Usenet

3.2. Instruction

Due to time pressure, we had only limited opportunities to explain Hadoop and MapReduce to students. This was unfortunate and required students to do significant work on their own to catch up.

We spent two hours of lecture on MapReduce and Cloud computing. The MapReduce portions concentrated on the purpose and mechanisms of MapReduce. We noted several ways in which Hadoop differs from the earlier Google MapReduce implementation, such as that Hadoop does not require Reducers to be idempotent, and that Google MapReduce does. However, we did not describe the specifics of the Hadoop Java API. The Cloud portions concentrated on the scale, components, and cost distribution of a modern datacenter: how many nodes, what hardware per node, what costs per node.

We were able to devote only one discussion to Java and MapReduce. This was intended both as a Java catch-up for new students as well as quick introduction to the details of the Hadoop API. We especially focused on the type-signature for a Hadoop MapReduce program: how the Java program encodes the types for the Map and Reduce functions as both template parameters and as configuration method calls.

In the Spring, we added a two-hour lab practice session where students developed some small MapReduce programs (using the Hadoop Java API) and ran them locally using Hadoop MapReduce's single-process mode. We provided students with a complete word count implementation and instructed them to use it as a template for the two MapReduce programs assigned in the lab: counting the number of documents containing each word (instead of the number of times each word appears) and constructing an inverted index from the source text.

Given the limited instructional time available, students had to figure things out using our Lab examples and Hadoop tutorials available online. One of the strengths of using a standard tool like Hadoop is that resources intended for professional practitioners are available as supplemental materials for students. With a custom-designed interface, used exclusively in instructional contexts, no such resources are likely to be available.

3.3. Assignments

We designed our MapReduce unit around a set of related text-processing assignments, characterized by simple algorithms to be executed on big data. We strove for assignments that would be slightly more complex than the usual "word count"-style examples for MapReduce, but that would not overly tax students' nascent Java programming skills. Table I summarizes these assignments. All these assignments are available online from our course webpages [2].

[2]These can be obtained via `http://www.cs.berkeley.edu/~randy/Courses/toce12-mr-assignment/`

In both semesters, our assignments culminated with the project. This project required students to write a small data processing program and run it on a large data set. We had students run their program in the cloud so that they could vary the number of (virtual) machines their program used and measure the scale-up. The data set their programs ran across was chosen to be as large as practical while requiring less than a half-hour of time waiting for programs to complete.

In the Fall semester, the text corpus was the link graph (pairs of source and destination web pages) from a circa 1999 crawl of CS department webpages, derived from the Stanford WebBase project [Hirai et al. 2000]. In the Spring, we used subsets of the Westbury lab corpus of Usenet messages [Shaoul and Westbury 2011]. In both cases, we pre-loaded a copy of the data in the cloud (specifically, Amazon's S3 storage service) where it could be quickly accessed by student code running in EC2.

MapReduce relies on record-level parallelism in the Map phase. For Hadoop, this is arranged by using data in a particular format that the framework can split correctly. Given a large dataset not designed for processing with Hadoop, some work is required to put it into the appropriate format. In each semester, staff built special tools for this. We have found that it is valuable to share these conversion tools with students so that they could create their own test data.

We had students produce both the program for their project and answer some questions about their experiences. Some of these questions were merely intended to verify that the students ran their program at scale. We asked students for part of the output of their program over one of our large datasets. We also asked students to measure the efficiency of their programs. For two different numbers of virtual machines, we had students compute the processing rate. In order to verify their calculations, we asked students to provide their raw runtime measurements.

To draw attention to the economic aspects, we also asked students more conceptual questions. We had them express their processing rate both in terms of data per unit time and in terms of cost (if they were paying the advertised commercial rate) per unit data. We had students compare the costs when running with different degrees of parallelism. We asked them to explain the difference qualitatively; in most cases, the speedup would illustrate Amdahl's Law naturally. Due to the pay-as-you-go billing model, increased cost per unit of data corresponds to a less-than-ideal speedup.

Originally, our MapReduce assignments used C to match other assignments in the course. This relied on Hadoop's language-neutral "streaming" mode [3], which executes an external program for each Map and Reduce task. This caused several problems; as we discuss below, debugging in this environment was a significant challenge. While not a formal prerequisite, most students taking our course had some experience in Java or a C-like language. Hence, we switched from C to Java for the Spring offering of the course. Switching to Java permitted our assignments to use the better supported Java API to Hadoop. It also meant that we could cover MapReduce early in the course, without needing to first teach students string manipulation in C.

3.4. Payment Models

Amazon and other Infrastructure-as-a-service cloud providers use a post-paid billing model. Users supply a credit card number when they create their accounts. Users then use the provider's API, command line tools, or web interface to request resources, which are then made available. At the end of every month, users' credit cards are billed for the previous month's usage. There is no way for users to impose a cutoff beyond which further requests will be denied. The Cloud unit of our course was funded by

[3] `http://hadoop.apache.org/common/docs/r0.20.0/streaming.html`

an education grant from Amazon Web Services. Even so, Amazon requires a payment card for each account, which will be billed if usage exceeds the amount of the grant.

Our grant was for $100 per student per semester, the standard size offered by Amazon. From our research experience, we knew that cloud usage could easily exceed expectation. We anticipated that most of our usage would occur shortly before the deadline. We wanted to have enough slack in our budget to ensure that we would not exhaust the budget in this pre-deadline peak. We also wanted to reserve some funds so that students who made mistakes and needed extra resources could still complete the assignment.

We had a choice of whether to have a separate grant per student or a large lump-sum for the course. The former would have required students to supply their own credit card, in case they exceeded the grant. We thought this was inappropriate. Students agreed: In the Fall semester, we surveyed the students on whether they would be comfortable signing up for an EC2 account with their credit card for backup billing.[4]

Approximately a third of students responded; half of these said they would not be willing to risk their own money. Consequently, we chose to have a single Amazon account for the class, with subaccounts for each student. An advantage of this lump-sum approach is that the savings from frugal students could be directly applied to their more profligate or unfortunate classmates. However, this required us to create wrapper scripts for all the tools that we expected students to need rather than allowing students to use existing commercial tools.

In both semesters, we provided students with scripts to launch a Hadoop MapReduce cluster on Amazon's Elastic Compute Cloud (EC2) and run programs on it. These scripts were responsible for the necessary accounting, access control, and billing. They were based on Cloudera's Hadoop scripts [Cloudera, inc. 2011], modified to access our shared account and integrated with the local user account structure in our environment for accounting.

4. EVALUATION

This section evaluates our experience in several dimensions. We begin by describing the quality of student work. Next, we describe the parallel speedups that students saw. Last, we report student satisfaction; not only did students learn, but they were excited by it.

4.1. Assessing Student Learning

We did not perform explicit assessment of our curricular innovation. Instead, we use the grades and grading rubric from the course to measure our success.

Grading had three components. As noted above, submission was in two parts: a checkpoint requiring students to submit working code (in the single machine case), and the final submission requiring evaluation of working code in the cloud. A quarter of the points were assigned by autograding the checkpoint, another 45% from having working code at the final submission, and the remaining 30% from manual grading of student short-form responses.

[4]The wording of the question was as follows: *Amazon Web Services is set up for companies rather than for students, so the way you get an account is to get an activation key (which we have) and then supply a credit card number as a backup in case you exceed your initial allocation. We think you'll only use 20% of your allocation for both your lab and your project, so we think no one will get close to exhausting their allocation. (The AWS model is to let you do your work and then charge, rather than to terminate your work in mid flight once the account is empty.) We're trying to find another solution, but it may be that we need to supply a credit card number. Please tell us your (anonymous) opinion about using your credit card as backup:*
- I willing to supply a personal credit card number
- I am NOT willing to supply personal a credit card number

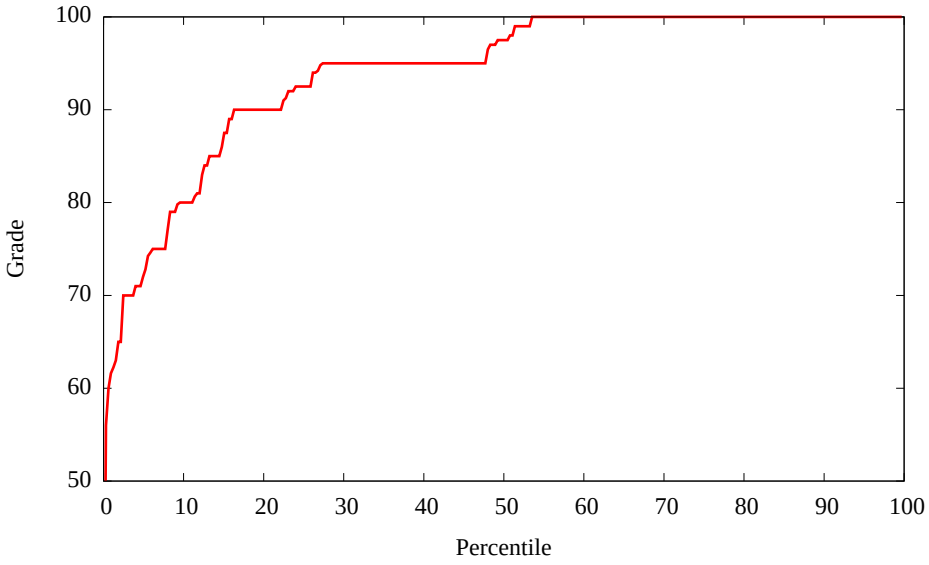

Fig. 1. Distribution of student grades. More than 80% of students had working code and sensible answers to questions. Nearly half did perfectly.

To get full credit, students needed to measure runtime, compute speedup, explain what they saw, determine the total cost and evaluate cost-per-gigabyte. We intended these questions to make sure students thought about the economics of cloud computing: they should see the approximate scale of costs involved in a sizable computation.

The results are shown in Figure 1. As can be seen, the distribution skews high. The median was 97.5% – nearly perfect – and even the 20th percentile was 90 points out of 100. Based on our grading rubric, this implies a substantial level of proficiency. In particular, it implies that student code worked correctly and that the students had good answers to most or all of our questions.

From these grades we can draw some inferences about student learning. More than 80% of students had working MapReduce programs. The students were able to run those programs, and had a good understanding of what those programs cost to run.

4.2. Experiencing Parallelism and Big Data

Figure 2 plots the reported parallel speedups observed by students. Most students saw more than a speedup of 1.5 times when they increased their cluster sizes by a factor of 1.8; few saw more than 2. Seeing this speedup was a good result. Most students saw a sizable but sublinear speedup, the right expectation to have for parallelism in general.

That some students saw anomalous speedup is also useful pedagogically. Performance variation in clouds (or other distributed systems) is an important and inevitable fact. Students benefit from seeing that it happened to them or their classmates at least some of the time. Variation in performance appeared to come from two sources in our students' experiences: from variation in I/O speeds and from spurious failures.

Although the performance of CPU and memory resources is quite consistent, I/O isolation is much harder, especially for less powerful virtual machines, which must share NICs and disks with other virtual machines. Since data-intensive MapReduce programs necessarily perform a lot of I/O, some students reported substantially different runs of the same program.

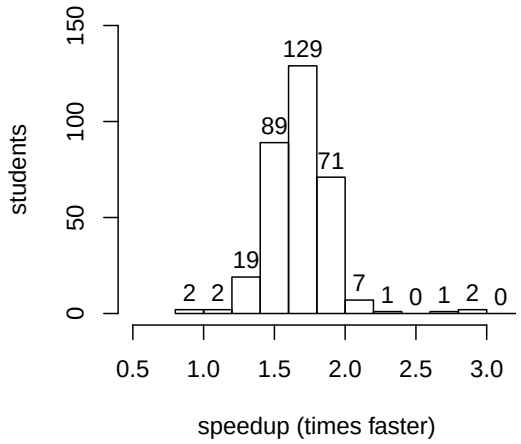

Fig. 2. Speedups observed by students. Most students saw a substantial but sub-linear speedup, as desired.

The cause of the most substantial performance anomalies students observed was failures. Hadoop MapReduce usually recovers automatically from failures, but naturally, the computation completes more slowly. A small number of our students reported very long run times from one node of their cluster failing in such a way that Hadoop needed the node to timeout to recover.

We asked students to take the median of three measurements. This meant that their results would not be seriously affected by one-time anomalies. Nevertheless, some these anomalies will persist for an entire session with multiple measurements. Slow I/O performance (by up to a factor of 2.5) can be caused by interference with other VMs [Zaharia et al. 2008], and failed VMs are unlikely to recover without manual intervention. Consequently, only students ran measurement collection run multiple times (for example, because their code had a bug the first time) were likely to experience performance variation as variance within measurements on the same size cluster. Most students observed the effects of performance variation in an anomalous speedup number or through very different runtimes versus their peers.

4.3. Student Satisfaction

We now turn to quantitative evaluation of how well our course ran in terms of student satisfaction. At the end of the Spring semester, we administered a survey to our class. We asked students to rank the four class projects in terms of value. The four projects were MapReduce, writing a MIPS emulator in C, writing an optimized matrix-multiply program, and designing a pipelined processor at the logic-gate level. The second and fourth of these are routine and well-debugged class projects, with the last of these usually being very popular.

Thirty percent of students thought MapReduce was the most valuable, and another thirty percent listed it as second-most-valuable. The MapReduce assignment came in second, overall. We conclude that students are enthusiastic about the assignment, even given its rough edges and challenges.

We asked students explicitly whether they would advocate keeping or replacing the project. 45% of students suggested keeping unequivocally, and another 47% marked "There are pros and cons, but better to keep it." Only 8% marked "better to drop" or "definitely drop."

Several students thought that the project was valuable to them professionally. One student commented "Too many employers are looking at Cloud Computing for it to be dropped from the curriculum - it's good to give students at least an intro to the subject." Another noted that "employers at job fairs really seemed to like it!"

The students who were dissatisfied primarily focused on programming language issues. A minority of students struggled because they had only minimal Java proficiency coming in. This was not a universal problem; many students had comments like "I didn't know Java, and though I missed points for a few small things, the project was overall definitely still worth it."

5. EXPERIENCES

Above, we presented an evaluation of our educational outcomes. We now broaden our scope to assess additional aspects of our innovation. We emphasize cost management, since being able to manage machines effectively is necessary for using the cloud for lower-division instruction. When using the public cloud, the cost per student becomes visible to both instructors and pupils in a way that is not currently common in computer science.

We start by describing our cost-control measures and what was required for provisioning and management; we follow by exploring the influence of staff quality and describing the aspects that do and do not require MapReduce or cloud expertise.

5.1. Provisioning

Using a cloud computing provider was effective for supplying students with large compute resources for short periods. Even though we had many hundreds of students and most students did their work close to deadlines, no one ran out of machines. The cost was relatively modest, around $45 per student in the Spring and around $25 per student in the Fall. In the Fall, with limited experience, we opted for medium-size instances, with 2 cores each. In the Spring, we decided that it was worth the expense to give students experience with modern datacenter-standard hardware, and used 8-core machines.

In the Spring, peak instantaneous usage was around 500 8-core virtual machines: we instructed students to time their programs on two cluster sizes, 5 and 9 machines, and compare the timings. Obtaining and managing a similarly capable physical cluster would have been much more expensive.

Although Amazon could handle the load of our course, obtaining the computing resources was not simply a matter of signing up online as cloud computing is traditionally imagined. All our usage went through one account with Amazon. Amazon requires approval for an account to run more than twenty virtual machines, and these approvals are usually granted as a matter of course.

Amazon is much more reluctant to approve large requests over a thousand machines, especially without strong evidence of an ability to pay for sustained usage at that scale. The potential peak for our assignment was approximately 3000 virtual machines (9 VMs per student and 320 students). We worried that if most of the class worked on the assignment simultaneously near the deadline, we might actually hit the limit. We asked Amazon to increase our limit, but we only received capacity for about a third of our maximum usage. So we crossed our fingers, hoped that a sufficient number of students would do the assignment early, and accepted the thousand-instance limit.

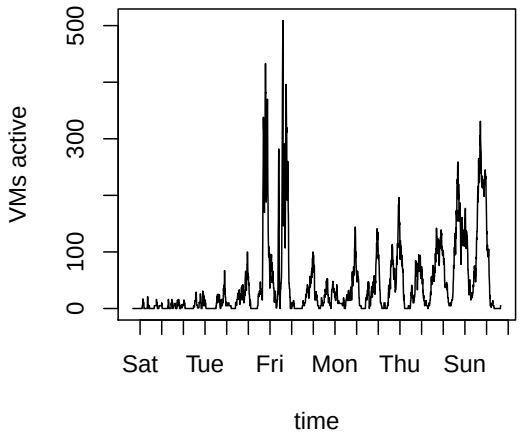

Fig. 3. Number of active virtual machines on our course account in the Spring semester over time. The peak (over 500 virtual machines) occurs during the day the lab assignment was given. The second peak is the project due date.

Happily, this posed no problem in practice, as our peak usage of 'only' 500 machines and Figure 3 shows.

5.2. Staffing

In developing our MapReduce unit, we had the advantage of a highly MapReduce-literate course staff. The high-level direction of the MapReduce unit was set by the last two authors of this paper (Katz and Patterson) both of whom have done significant research on clouds and MapReduce. The details of the MapReduce assignments were fleshed out by the two student authors of this paper (Rabkin and Reiss) – both of whom have substantial experience with clouds and Hadoop, and both of whom have done significant implementation work on the underlying framework.

This invites the question of how similar assignments would work with staff whose expertise is otherwise. We can offer two lines of evidence on this question. During the 2010-2011 academic year, our primary focus, our course had five additional teaching assistants. These staffers did not have a deep background in Hadoop or in cloud computing. Even so, they were able to give useful aid to the students. One TA commented that "most of the projects had their unique hangups and this one wasn't noticeably worse than, say, no one having a clue how to start optimizing linear algebra code." (This was another new course project that was prompted by our emphasis on parallelism.)

Subsequent course offerings had a course staff with comparatively weak background for a big data unit. Even so, the staff was able to devise new big-data problems for students to work on. They used PageRank for the project. This worked well. The assignment used a comparatively small dataset (less than a gigabyte), resulting in low speedup ratios (below 1.5). This is a pedagogically useful experience for students, since poor parallel scaling on small problem instances is a common phenomenon.

Most of the difficulties the Fall 2011 staff experienced were related to the cloud-instance launching and accounting scripts, not the underlying Hadoop or cloud technologies. This reassures us that incorporating cloud-based MapReduce assignments can be done without especially expert staff. Infrastructure is required, but, while building this infrastructure is potentially time-consuming, it does not rely on deep expertise. Over time, we believe improvements by cloud computing providers are likely to mitigate these infrastructure requirements.

One aspect does require significant expertise, however: picking a project assignment. This must satisfy several constraints. There must be a suitable data-set available. The assignment must have a moderate runtime, large enough to amortize scheduling overhead and demonstrate parallel speedup, small enough not to be overwhelming. It must also be different enough from standard examples that students cannot easily find a solution online.

6. LESSONS LEARNED

On the whole, our experiment with using the public cloud for undergraduate core courses worked well. Even so, we learned several negative lessons. This section describes those lessons and what we intend to do differently in future offerings of the course.

6.1. Billing and Account Support

Using a commercial cloud system forced us to consider the properties that shared cloud-like infrastructure should ideally have for academic instructional use.

— **Student work is private.** The cloud provider should provide per-student access control so that students do not have automatic access to one another's work.
— **Students are resource limited.** No student should be able to consume an excessive quantity of computational resources.
— **Billing is flexible.** One might imagine having student costs paid directly by the student, paid out of instructional funds, or implicitly supported by overhead funds. The cloud system should not impose a specific model that may not work for all institutions.
— **Costs are centralized.** While each student should be resource-limited, staff should not have to manually parcel out funds to each student.

Today's cloud providers do not completely meet these goals. Amazon for instance currently does minimal resource limitation; the system will not prevent a careless user from incurring a vast bill. However, Amazon meets most of the rest of these requirements, and we were able to write custom code to make the system usable.

Many different models for cloud services could meet these requirements. Commercial vendors may adapt their products for academic customers. Alternatively, we might see institutional clouds, perhaps without fine-grained accounting. We might see cloud resources treated similarly to studio materials in art classes, with students partly or fully responsible for their own costs.

6.2. Debugging

One of our pedagogic goals was to have students experience real-world big data programming and debugging. This has two distinctive challenges. First, big data is noisy data, and this means that some thought is required to determine what correct behavior will be. Second, distributed debugging is harder than local debugging.

At one point, we asked students to run various text-processing programs on a few tens of megabytes of sample data derived from the Usenet corpus. Students were alarmed to discover that the first page of output was exclusively words like `0`, `00`, and `000`. After some thought, the staff and students realized that this was not evidence of

a bug. The assignment had defined a word as “any text separated by whitespace”; this meant that a number with spaces around it would qualify as a word. Given a large enough text corpus, such “words” would inevitably appear. Given the usual ASCII sort order, such words would consume the first few pages of output.

In one sense, this experience was caused by the staff’s failure to carefully test the assignment on a large dataset. In another sense, this was a routine part of working with large data sets and therefore an educationally valuable experience. A large dataset will often have unexpected elements or elements that interact unexpectedly. Students benefit from learning that “big data” is usually “messy data.”

The pain of distributed debugging was another characteristic experience of real-world MapReduce programming at scale. In the Fall 2010, using C, several students received a very pointed lesson in the importance of local testing and defensive programming. Buggy C programs often fail with segmentation faults. Locally, these can be diagnosed with a debugger such as GDB. But when the C program is run inside Hadoop, there is no opportunity to do so. Hadoop Streaming’s diagnostics for programs that fail without outputting an error message themselves were very poor. Hence, troubleshooting can be difficult, even for experienced course staff.

In the Spring, using Java, debugging was easier. Typically, when Map or Reduce tasks fail, a stack trace is recorded in the logs. This meant that staff, at least, could work out where student programs had gone awry. Students had difficulty debugging, however, since reading and understanding stack traces is not a skill that is emphasized or taught in our lower-division data structures course, where most students learned Java. Nor had students ever had significant practice coping with test cycles measured in minutes, rather than seconds. Forcing students into a more demanding (and realistic) debugging scenario thus exposed a gap in their previous CS education that we tried to remedy as best we could.

Subsequent offerings of the machine structures course have had students do the MapReduce assignment in pairs. Course staff advise students that at least one member of each partnership should have Java expertise. This seems to have worked well.

6.3. Using the Hadoop API

A distinctive aspect of our course was that we used the standard industrial MapReduce implementation with lower-level undergraduates, without any attempt to wrap it in a simpler, student-friendly interface. This appears to be an unusual educational decision. It certainly caused some unexpected difficulties for us.

At our university, the first two semesters of the core computer science sequence teach students the building blocks of programs — notions like iteration, recursion, data structures, and so forth. Using system libraries is not emphasized. Learning to use a large complex software system is completely unexplored. Being confronted with the Hadoop API may have been the first time students ever saw a realistic API or had to read documentation for a large software system.

The Hadoop API has some idiosyncrasies that would be absent from a ‘teaching’ MapReduce API. Probably the most notable of these are the datatypes supported for keys and values. Hadoop requires these values to be efficiently serializable. Hadoop can use Java serialization, but this has a substantial performance penalty. Consequently, Hadoop’s API implements a parallel hierarchy of ‘Writable’ types. These types are required to implement in-place serialization and deserialization functions. To avoid object creation and garbage collection overhead, Hadoop reuses objects of these types – for example, map() is likely to be repeatedly called with the same object with different values loaded and reduce()’s iterator is likely to return the same object over and over again.

This serialized object API caused some confusion for students. We mitigated this confusion by providing templates that named usable datatypes and suggesting conversions to 'native' Java types within map() and reduce() functions. Nevertheless, several students experienced problems from placing Writable types in containers (for example, attempting to use them as a key for a Hashtable). Since Hadoop reused Writable objects, such programs would almost always produce the wrong answer without other indications of error.

Many students complained about less subtle errors when they tried to change the type of values produced by their mapper or reducer to a non-Writable type. We only introduced students to a subset of the types and suggested solutions using strings to represent multiple values (where required). A few students wanted to define custom types and use them as map or reduce outputs, but we did not teach them how to create a custom Writable type. We were reluctant to mention the ability to do this because debugging serialization is extremely difficult: Hadoop implicitly relies on deserialization functions reading the correct amount of input; if they do not, Hadoop may read other data sent alongside the serialized values from the wrong place in its input stream. Consequently, serialization bugs can easily lead to both incorrect values for serialized types and crashes within Hadoop internal code far from when the deserialization function ran.

6.4. Efficiency

In both semesters, the staff coded and tested a simple implementation before releasing the assignment to students. We used this to estimate performance, both in choosing the data set size to assign students as well as to announce the performance they should expect. In the Spring, many students produced solutions that were more than a factor of two slower. Although our reference implementation was not specially optimized, the simple MapReduce programs we had the students write are sensitive to overheads from object creation, autoboxing (Java's automatic wrapping of primitive values in objects), string parsing, and string/number conversions. Students whose programs did more of these than our reference implementation (for example, keeping a counter in an Integer instead of an int or splitting a string in an inner loop) were correct but much slower. A small number of students made poor data-structure choices, resulting in run-times quadratic in the number of words in an input record (representing a Usenet message). Since we were targeting a run time of around 15 minutes, the financial effect of students' programs running even two or three times slower than our reference implementation was substantial.

Here again, a large-scale assignment in the cloud showed both us and the students a previously unsuspected gap in their prior experience. After we became aware of the slow program problem, we supplied a non-trivial local test dataset and told students what run time they should expect on it. We also gave a brief list of ways to make programs faster for students who were interested in doing so: avoiding object creation, reducing the number of generated map output keys, and so on.

In future offerings, we will spend more time advising students on how to debug in distributed contexts and how to program in a performance-conscious manner. Performance issues fit naturally into a computer organization course, so this strikes us as a good use of class time.

6.5. Waste

Inefficiency was not the major driver of costs; waste was. Some students accidentally left virtual machines running idle. We had mitigated the risk of this by configuring our scripts to automatically shutdown virtual machines after a time delay, though initially, we did not always enable this feature. Early versions of the staff-supplied

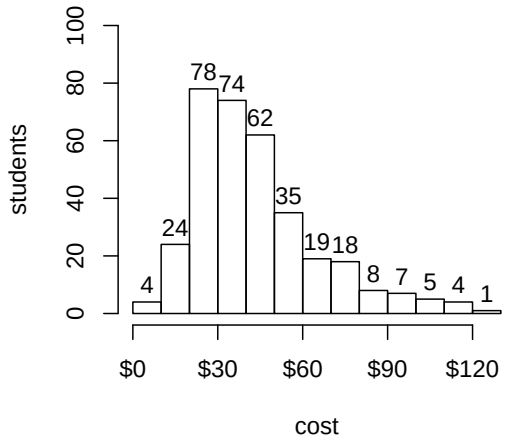

Fig. 4. Histogram of estimated Spring semester EC2 usage per student.

wrapper scripts had a bug and would occasionally fail to terminate virtual machines when students requested it.

A much larger problem was that some students would re-run their entire experiment after any technical glitch or mistake. This repetition was responsible for the outliers seen in Figure 4. Our management scripts were only available on the campus hosts, so students seeking to use the cloud from home would log into campus, rather than running directly from home. These students were generally not aware of tools like GNU `screen` that would let them maintain their session on the instructional machines after a network failure. Also, students assumed incorrectly that they could not continue using their existing EC2 session when they reconnected.

Consequently, many students would start a fresh set of virtual machines (killing the old ones) each time they experienced a problem. Since EC2 charges for at least an hour each time a virtual machine is started, this could get expensive quickly. Some students who did not experience these glitches made different errors that led them to restart machines unnecessarily — for example, assuming that each run of their program required a fresh cluster.

This is by no means a fundamental problem. For research uses, our lab has developed a sophisticated set of deployment tools. These are able to compensate for failures of a particular node to boot. There are open-source projects devoted to managing EC2 instances, such as Apache Whirr. As these tools mature, we expect to use them for instructional purposes. (At present, they are unsuitable since they do none of the logging and accounting that we require.)

7. CONCLUSIONS

Overall, our experience was a success. Our course let every student in a large course run their programs on comparatively large clusters of modern hardware, without scheduling or resource contention. This experience would have been infeasible without the public cloud. Students enjoyed the ability to run at a “real” scale with real

tools. Student evaluation was very positive of using Hadoop, even despite rough edges and difficulty.

Using an industrially-developed tool in a lower-level course offered advantages but posed problems. Using a standard tool meant that standard online tutorials could supplement instruction. It also exposed students to implementation aspects that were irrelevant to our primary educational goals and that caused some confusion.

Though current cloud billing models posed some difficulties, per-pupil costs were not a problem. In the Spring version of our course, we only spent $45 per student on average, less than half the standard grant, or $15000 total. This was small compared to the overall instructional cost of the course. The per-pupil cost was comparable to a textbook.

The vast majority of students were conscientious about costs. They were generally careful to not waste computing resources. The ones who were careless were usually quick to respond when notified that they had accidentally kept instances running. Cost management was a small fraction of the staff time for the assignment.

The course also succeeded in pedagogic terms. Students had the opportunity to exercise their programming and debugging skills in a new and challenging environment. They were able to experience parallel performance at a scale that would have been unthinkable without the cloud. And they were able to experience cutting-edge tools that helped them grow professionally.

ACKNOWLEDGMENTS

We are grateful to Brian Harvey, Dan Garcia, and Colleen Lewis for their advice in writing this article. Several past CS61C staff members were very helpful in describing their experiences, particularly Andrew Waterman and Brian Gawalt. The experiences described in this article were funded by an Amazon Web Services instructional grant.

REFERENCES

ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A., KATZ, R., KONWINSKI, A., LEE, G., PATTERSON, D., RABKIN, A., ET AL. 2009. Above the Clouds: A Berkeley View of Cloud Computing. Tech. Rep. 2009-28, UC Berkeley, http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html.

BROWN, R. A. 2009. Hadoop at home: large-scale computing at a small college. In *Proceedings of the 40th ACM technical symposium on Computer science education*. SIGCSE '09. ACM, New York, NY, USA, 106–110.

CHAMBERS, C., RANIWALA, A., PERRY, F., ADAMS, S., HENRY, R. R., BRADSHAW, R., AND WEIZENBAUM, N. 2010. FlumeJava: easy, efficient data-parallel pipelines. In *Proceedings of the 2010 ACM SIGPLAN conference on Programming language design and implementation*. PLDI '10. ACM, New York, NY, USA, 363–375.

CLOUDERA. 2012. Crunch. https://github.com/cloudera/crunch.

CLOUDERA, INC. 2011. Configuring and Running CDH Cloud Scripts. Retrieved August 31, 2011 from https://ccp.cloudera.com/display/CDH2DOC/Configuring+and+Running+CDH+Cloud+Scripts.

COUCH, A. 2011. Comp150 CPA. Retrieved August 21, 2011 from http://www.cs.tufts.edu/comp/150CPA/.

DEAN, J. AND GHEMAWAT, S. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM Volume 51,* Issue 1, 107–113.

GARRITY, P., YATES, T., BROWN, R., AND SHOOP, E. 2011. WebMapReduce: an accessible and adaptable tool for teaching map-reduce computing. In *Proceedings of the 42nd ACM technical symposium on Computer science education*. SIGCSE '11. ACM, New York, NY, USA, 183–188.

GATES, A., NATKOVICH, O., CHOPRA, S., KAMATH, P., NARAYANAMURTHY, S., OLSTON, C., REED, B., SRINIVASAN, S., AND SRIVASTAVA, U. 2009. Building a high-level dataflow system on top of MapReduce: the Pig experience. *Proceedings of the VLDB Endowment 2,* 2, 1414–1425.

HIRAI, J., RAGHAVAN, S., GARCIA-MOLINA, H., AND PAEPCKE, H. 2000. WebBase: A repository of web pages. In *Proceedings of the 9th international World Wide Web conference on Computer networks*. North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands, 277–293.

JOHNSON, M., LIAO, R. H., RASMUSSEN, A., SRIDHARAN, R., GARCIA, D. D., AND HARVEY, B. 2008. Infusing Parallelism into Introductory Computer Science Curriculum using MapReduce. Tech. Rep. EECS-2008-34, UC Berkeley, http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-34.html.

KIMBALL, A., MICHELS-SLETTVET, S., AND BISCIGLIA, C. 2008. Cluster computing for web-scale data processing. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education*. SIGCSE '08. ACM, New York, NY, USA, 116–120.

LIN, J. 2011. Data-Intensive Information Processing Applications. Retrieved August 21, 2011 from http://www.umiacs.umd.edu/~jimmylin/cloud-2010-Spring/info.html.

MALAN, D. J. 2010. Moving cs50 into the cloud. *J. Comput. Small Coll. 25*, 111–120.

PAGE, L., BRIN, S., MOTWANI, R., AND WINOGRAD., T. 1998. The PageRank Citation Ranking: Bringing Order to the Web. Tech. rep., Stanford Digital Library Technologies Project.

SHAOUL, C. AND WESTBURY, C. 2011. A usenet corpus. Retrieved August 21, 2011 from http://www.psych.ualberta.ca/~westburylab/downloads/usenetcorpus.download.html.

THUSOO, A., SARMA, J. S., JAIN, N., SHAO, Z., CHAKKA, P., ANTHONY, S., LIU, H., WYCKOFF, P., AND MURTHY, R. 2009. Hive: a warehousing solution over a map-reduce framework. *Proc. VLDB Endow. 2,* 2, 1626–1629.

ZAHARIA, M., KONWINSKI, A., JOSEPH, A. D., KATZ, R., AND STOICA, I. 2008. Improving MapReduce performance in heterogeneous environments. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*. OSDI'08. USENIX Association, Berkeley, CA, USA, 29–42.