Experiences Teaching MapReduce in the Cloud

Ariel Rabkin, Charles Reiss, Randy Katz, David Patterson
{asrabkin,charles,randy,pattrsn}@cs.berkeley.edu
Electrical Engineering and Computer Science Department
University of California, Berkeley

ABSTRACT

We describe our experiences teaching MapReduce in a large undergraduate lecture course using public cloud services. 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, less than half our available grant funding. 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

Keywords

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 [2]. This change is begining to percolate into the undergraduate curriculum. Many universities are now offering courses that cover MapReduce [5] and related distributed execution systems [9, 8, 3]. These courses prepare students for a world in which large-scale distributed "big data" processing is routine. In contrast, the change in billing models caused by cloud computing has not yet had overt impact on undergraduate instruction.

This paper describes an effort to integrate the cloud into lower-division courses, allowing new kinds of assignments. For the last year, we have used Amazon's public Elastic Compute Cloud, EC2, as a platform for the MapReduce unit in our introductory machine organization course at UC Berkeley. This course is ordinarily taken by students in their third or fourth semester. The Fall term had 170 students; the Spring had 320.

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.

Clouds make 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 a pre-PC era, 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

We were driven to use a public cloud by the conjunction of several 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. Second, we wanted students to experience running and debugging distributed MapReduce jobs on a significantly-sized cluster. Given the size of our student body, using purely university IT resources would have imposed unacceptable infrastructure costs. Last, we wanted students to evaluate the performance of MapReduce jobs and see the speedup from parallelism. This requires isolation from other students, which is difficult to achieve without

SIGCSE'12, February 29–March 3, 2012, Raleigh, North Carolina, USA.

the public cloud. (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.)

1.2 Research Questions and Methods

The focus of this paper is on the opportunities and challenges offered by the cloud, rather than on teaching big data, parallelism, or MapReduce. We answer three 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 EC2?

We use instructional costs, student survey results, and the average 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 [9]. (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, a higher-level parallel programming framework atop MapReduce [4, 10]. 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-MapReduce glue library [8]. Performance tuning is a non-goal, as 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 [3]. 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 [6]. As with 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 [11]. Students, however, were completely insulated from provisioning and billing. Effectively, the Cloud was used as a scalable replacement for local IT infrastructure, rather than to allow assignments that could not have been taught otherwise.

3. COURSE DESIGN

We wanted students to understand the sort of data processing that cloud computing makes available. Accordingly, we designed our assignments to be sufficiently compute-intensive 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 here. 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.

Originally, our MapReduce assignments used C to match other assignments in the course. This relied on Hadoop's language-neutral "streaming" mode [1], 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. It also permitted our assignments to use Hadoop's better-supported Java API.

Our course had four two-week project assignments, a two-hour lab and three hours of lecture per week. Projects had an intermediate milestone deadline to help students start early. We assigned short problem sets for homework and had two written exams. Students were assessed primarily on their projects and the exams. Most labs 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, stu-

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

Sem.	Lines of Code	Description	Dataset Size (bytes)	Corpus
FS	0	Lab: performance measurement, introduction to EC2 tools	20,000 MB (F); 8GB (S)	Wikipedia (F); Usenet (S)
F	$\sim$ 50	Project: PageRank [12] in C, using Hadoop Streaming locally and then on EC2	650 MB	Web graph
S	$\sim$ 10	Lab: writing MapReduce programs in Java, running locally	34 MB	Usenet
S	$\sim$ 50	Project: computing a text-comparison metric locally and then on EC2	8,000 GB	Usenet

Table 1: Assignments used. F= Fall, S=Spring, FS = both

dents were permitted partners, but the projects were done individually.

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 1 summarizes these assignments.

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 CS department webpages, derived from the Stanford WebBase project [7]. In the Spring, we used subsets of the Westbury lab corpus of Usenet messages [13]. In both cases, we pre-loaded a copy of the data in the cloud where it could be quickly accessed.

All assignments, both lab and project, came with conceptual questions as well as coding. We asked students, for instance, to compute how much money they had spent, and what price this was per unit of work. This was intended to draw students' attention to the economic aspects of cloud computing. Since cloud computing costs scale linearly with computational resources, increased cost makes it clear when students are seeing less than ideal speedup.

In the Spring, we added a lab 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. We supplied a subset of the Usenet corpus for both cases.

4. EVALUATION

This section evaluates our experience in several dimensions. We emphasize cost management, not because it is the most important goal, but because it is tied directly to our core goal of 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 also evaluate the quality of student work and reported student satisfaction. These demonstrate that using the cloud provided sufficiently consistent performance to allow students to see parallel speedup.

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 [1], modified to access our shared account and integrated with the local user account structure in our environment for accounting.

4.1 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 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.

One approach would have been to have a separate Amazon grant allocation per student. Doing this would have required students to supply their own credit card, if 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.

The full 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*

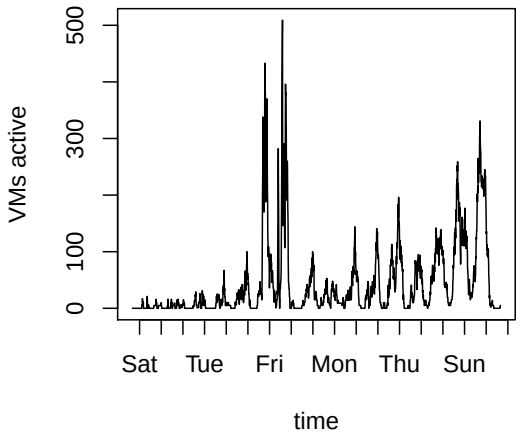

Figure 1: Number of active virtual machines on our course account in the Spring semester over time. The peak (over 500 virtual machines) occurs during the day the lab assignment was given. The second peak is the project due date.

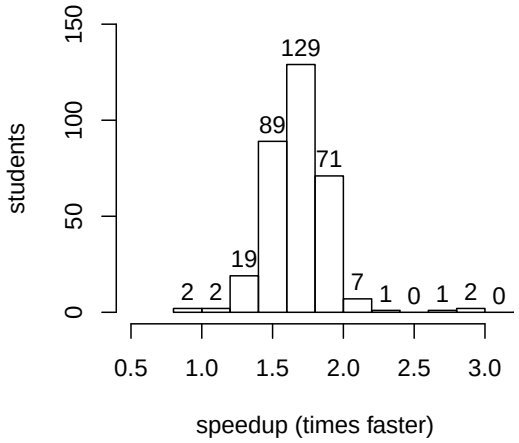

Figure 2: Speedups observed by students. Most students saw a substantial but sub-linear speedup, as desired.

your project, so we think no one will get close to exhausting their allocation.

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. This decision required us to create our own wrapper scripts for all the tools we expected students to need.

4.2 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 who do the assignment early, and accepted the thousand-instance limit. Happily, this posed no problem in practice, as Figure 1 shows.

4.3 Experiencing Parallelism and Big Data

Figure 2 plots the reported parallel speedups observed by students. Most students saw more than a 1.5 speedup 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.

The overall quality of student work impressed us. In the Spring, 75% of student projects passed all our test cases; many more failed only due to minor bugs that could have been caught with more testing. This shows that students were able to cope with the MapReduce programming model.

4.4 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 pro-

gram, 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. 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.

Today's cloud billing options are not well-adapted to academic needs. Ideally, each student would have a balance of cloud credits issued by staff, and their accounts would be suspended if the limit is exceeded. Students should be unable to view each others' work or terminate virtual machines belonging to other students. Course staff should be able to use standard industry tools, without needing to modify them to prevent abuse. We understand that vendors, particularly Amazon, are working to introduce new billing models more appropriate for classroom use. This issue may therefore be resolved soon.

Another major inconvenience for our course was the linguistic fragmentation of the early undergraduate curriculum in our school: there is no imperative language that students were required to know, coming in to the class. As of Fall 2011, our introductory course has switched to Python, which we could reasonably use for this assignment. This switch would also reduce the confusion caused by Hadoop's sometimes-idiosyncratic and unevenly-documented Java APIs.

5.1 Debugging

One of our pedagogic goals was to have students experience real-world big data programming and debugging. 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. 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 did students ever 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. We expect the difficulty of Python debugging to be comparable to Java.

5.2 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.

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

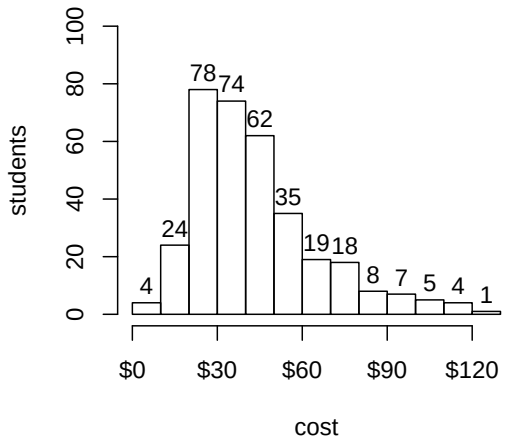

Figure 3: Histogram of estimated Spring semester EC2 usage per student.

A much larger problem was that some students would rerun their entire experiment after any technical glitch or mistake. This repetition was responsible for the outliers seen in Figure 3. 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 lead them to restart machines unnecessarily — for example, assuming that each run of their program required a fresh cluster.

6. CONCLUSIONS

Overall, our experience was a success. 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 is small compared to the overall instructional cost of the course. The per-pupil cost is 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.

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.

Acknowledgements

Our course was funded by an Amazon Web Services instructional grant.

7. REFERENCES

[1] Configuring and Running CDH Cloud Scripts. Retrieved August 31, 2011 from `https://ccp.cloudera.com/display/CDH2DOC/Configuring+and+Running+CDH+Cloud+Scripts`.

[2] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, et al. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report 2009-28, UC Berkeley, 2009.

[3] R. A. Brown. Hadoop at home: large-scale computing at a small college. In *SIGCSE*, 2009.

[4] A. Couch. Comp150 CPA. Retrieved August 21, 2011 from `http://www.cs.tufts.edu/comp/150CPA/`.

[5] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, Volume 51(Issue 1):107–113, 2008.

[6] P. Garrity, T. Yates, R. Brown, and E. Shoop. WebMapReduce: an accessible and adaptable tool for teaching map-reduce computing. In *SIGCSE*, 2011.

[7] J. Hirai, S. Raghavan, H. Garcia-Molina, and H. Paepcke. WebBase: A repository of web pages. In *WWW*, May 2000.

[8] M. Johnson, R. H. Liao, A. Rasmussen, R. Sridharan, D. D. Garcia, and B. Harvey. Infusing Parallelism into Introductory Computer Science Curriculum using MapReduce. Technical Report EECS-2008-34, UC Berkeley, 2008.

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

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

[11] D. J. Malan. Moving cs50 into the cloud. *J. Comput. Small Coll.*, 25:111–120, June 2010.

[12] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.

[13] C. Shaoul and C. Westbury. A usenet corpus. Retrieved August 21, 2011 from `http://www.psych.ualberta.ca/~westburylab/downloads/usenetcorpus.download.html`.