Exploring Large-Data Issues in the Curriculum: A Case Study with MapReduce

Jimmy Lin
The iSchool, College of Information Studies
Laboratory for Computational Linguistics and Information Processing
University of Maryland, College Park
jimmylin@umd.edu

Abstract

This paper describes the design of a pilot research and educational effort at the University of Maryland centered around technologies for tackling Web-scale problems. In the context of a "cloud computing" initiative lead by Google and IBM, students and researchers are provided access to a computer cluster running Hadoop, an open-source Java implementation of Google's MapReduce framework. This technology provides an opportunity for students to explore large-data issues in the context of a course organized around teams of graduate and undergraduate students, in which they tackle open research problems in the human language technologies. This design represents one attempt to bridge traditional instruction with real-world, large-data research challenges.

1 Introduction

Over the past couple of decades, the field of computational linguistics, and more broadly, human language technologies, has seen the emergence and later dominance of empirical techniques and data-driven research. Concomitant with this trend is the requirement of systems and algorithms to handle large quantities of data. Banko and Brill (2001) were among the first to demonstrate the importance of dataset size as a significant factor governing prediction accuracy in a supervised machine learning task. In fact, they argue that size of training set is perhaps more important than the choice of machine learning algorithm itself. Similarly, experiments in question answering have shown the effectiveness of simple pattern-matching techniques when applied to large quantities of data (Brill et al., 2001). More recently, this line of argumentation has been echoed in experiments with large-scale language models. Brants et al. (2007) show that for statistical machine translation, a simple smoothing method (dubbed *Stupid Backoff*) approaches the quality of Kneser-Ney Smoothing as the amount of training data increases, and with the simple method one can process significantly more data.

Given these observations, it is important to integrate discussions of large-data issues into any course on human language technology. Most existing courses focus on smaller-sized problems and datasets that can be processed on students' personal computers, making them ill-prepared to cope with the vast quantities of data in operational environments. Even when larger datasets are leveraged in the classroom, they are mostly used as static resources. Thus, students experience a disconnect as they transition from a learning environment to one where they work on real-world problems.

Nevertheless, there are at least two major challenges associated with explicit treatment of large-data issues in an HLT curriculum:

- The first concerns resources: it is unclear where one might acquire the hardware to support educational activities, especially if such activities are in direct competition with research.
- The second involves complexities inherently associated with parallel and distributed processing, currently the only practical solution to large-data problems. For any course, it is diffi-

Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics (TeachCL-08), pages 54–61,
Columbus, Ohio, USA, June 2008.

cult to retain focus on HLT-relevant problems, since the exploration of large-data issues necessitates (time-consuming) forays into parallel and distributed computing.

This paper presents a case study that grapples with the issues outlined above. Building on previous experience with similar courses at the University of Washington (Kimball et al., 2008), I present a pilot "cloud computing" course currently underway at the University of Maryland that leverages a collaboration with Google and IBM, through which students are given access to hardware resources. To further alleviate the first issue, research is brought into alignment with education by structuring a team-oriented, project-focused course. The core idea is to organize teams of graduate and undergraduate students focused on tackling open research problems in natural language processing, information retrieval, and related areas. Ph.D. students serve as leaders on projects related to their research, and are given the opportunity to serve as mentors to undergraduate and masters students.

Google's MapReduce programming framework is an elegant solution to the second issue raised above. By providing a functional abstraction that isolates the programmer from parallel and distributed processing issues, students can focus on solving the actual problem. I first provide the context for this academic–industrial collaboration, and then move on to describe the course setup.

2 Cloud Computing and MapReduce

In October 2007, Google and IBM jointly announced the Academic Cloud Computing Initiative, with the goal of helping both researchers and students address the challenges of "Web-scale" computing. The initiative revolves around Google's MapReduce programming paradigm (Dean and Ghemawat, 2004), which represents a proven approach to tackling data-intensive problems in a distributed manner. Six universities were involved in the collaboration at the outset: Carnegie Mellon University, Massachusetts Institute of Technology, Stanford University, the University of California at Berkeley, the University of Maryland, and University of Washington. I am the lead faculty at the University of Maryland on this project.

As part of this initiative, IBM and Google have dedicated a large cluster of several hundred machines for use by faculty and students at the participating institutions. The cluster takes advantage of Hadoop, an open-source implementation of MapReduce in Java.[1] By making these resources available, Google and IBM hope to encourage faculty adoption of cloud computing in their research and also integration of the technology into the curriculum.

MapReduce builds on the observation that many information processing tasks have the same basic structure: a computation is applied over a large number of records (e.g., Web pages) to generate partial results, which are then aggregated in some fashion. Naturally, the per-record computation and aggregation function vary according to task, but the basic structure remains fixed. Taking inspiration from higher-order functions in functional programming, MapReduce provides an abstraction at the point of these two operations. Specifically, the programmer defines a "mapper" and a "reducer" with the following signatures:

$$\text{map: } (k_1, v_1) \rightarrow [(k_2, v_2)]$$
$$\text{reduce: } (k_2, [v_2]) \rightarrow [(k_3, v_3)]$$

Key/value pairs form the basic data structure in MapReduce. The mapper is applied to every input key/value pair to generate an arbitrary number of intermediate key/value pairs. The reducer is applied to all values associated with the same intermediate key to generate output key/value pairs. This two-stage processing structure is illustrated in Figure 1.

Under the framework, a programmer need only provide implementations of the mapper and reducer. On top of a distributed file system (Ghemawat et al., 2003), the runtime transparently handles all other aspects of execution, on clusters ranging from a few to a few thousand nodes. The runtime is responsible for scheduling map and reduce workers on commodity hardware assumed to be unreliable, and thus is tolerant to various faults through a number of error recovery mechanisms. The runtime also manages data distribution, including splitting the input across multiple map workers and the potentially very large sorting problem between the map and reduce phases whereby intermediate key/value pairs must be grouped by key.

[1] http://hadoop.apache.org/

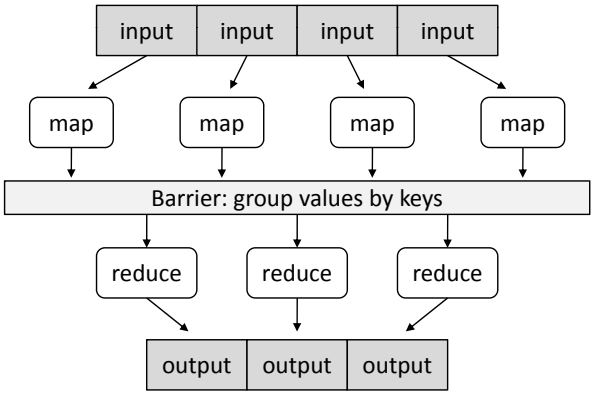

Figure 1: Illustration of the MapReduce framework: the "mapper" is applied to all input records, which generates results that are aggregated by the "reducer".

The biggest advantage of MapReduce from a pedagogical point of view is that it allows an HLT course to retain its focus on applications. Divide-and-conquer algorithms running on multiple machines are currently the only effective strategy for tackling Web-scale problems. However, programming parallel and distributed systems is a difficult topic for students to master. Due to communication and synchronization issues, concurrent operations are notoriously challenging to reason about—unanticipated race conditions are hard to detect and even harder to debug. MapReduce allows the programmer to offload these problems (no doubt important, but irrelevant from the perspective of HLT) onto the runtime, which handles the complexities associated with distributed processing on large clusters. The functional abstraction allows a student to focus on problem solving, not managing the details of error recovery, data distribution, etc.

3 Course Design

This paper describes a "cloud computing" course at the University of Maryland being offered in Spring 2008. The core idea is to assemble small teams of graduate and undergraduate students to tackle research problems, primarily in the areas of information retrieval and natural language processing. Ph.D. students serve as team leaders, overseeing small groups of masters and undergraduates on topics related to their doctoral research. The roles of "team leader" and "team member" are explicitly assigned at the beginning of the semester, and are associated with different expectations and responsibilities. All course material and additional details are available on the course homepage.[2]

3.1 Objectives and Goals

I identified a list of desired competencies for students to acquire and refine throughout the course:

- Understand and be able to articulate the challenges associated with distributed solutions to large-scale problems, e.g., scheduling, load balancing, fault tolerance, memory and bandwidth limitations, etc.
- Understand and be able to explain the concepts behind MapReduce as one framework for addressing the above issues.
- Understand and be able to express well-known algorithms (e.g., PageRank) in the MapReduce framework.
- Understand and be able to reason about engineering tradeoffs in alternative approaches to processing large datasets.
- Gain in-depth experience with one research problem in Web-scale information processing (broadly defined).

With respect to the final bullet point, the students are expected to acquire the following abilities:

- Understand how current solutions to the particular research problem can be cast into the MapReduce framework.
- Be able to explain what advantages the MapReduce framework provides over existing approaches (or disadvantages if a MapReduce formulation turns out to be unsuitable for expressing the problem).
- Articulate how adopting the MapReduce framework can potentially lead to advances in the state of the art by enabling processing not possible before.

I assumed that all students have a strong foundation in computer science, which was operationalized in having completed basic courses in algorithms, data structures, and programming languages

[2] http://www.umiacs.umd.edu/~jimmylin/cloud-computing/

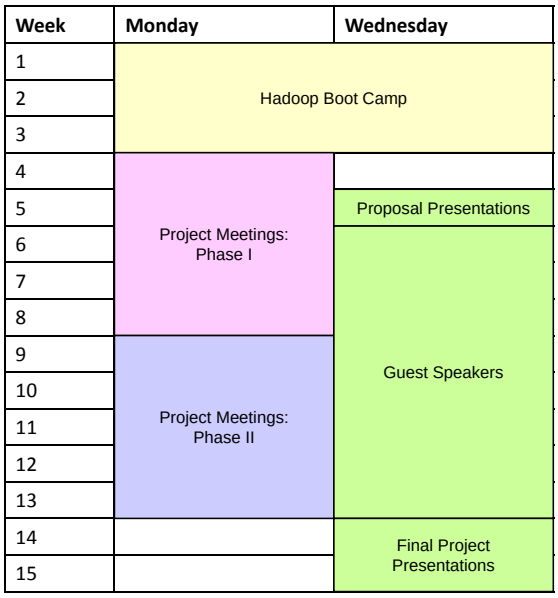

Figure 2: Overview of course schedule.

(in practice, this was trivially met for the graduate students, who all had undergraduate degrees in computer science). I explicitly made the decision that previous courses in parallel programming, systems, or networks was not required. Finally, prior experience with natural language processing, information retrieval, or related areas was not assumed. However, strong competency in Java programming was a strict requirement, as the Hadoop implementation of MapReduce is based in Java.

In the project-oriented setup, the team leaders (i.e., Ph.D. students) have additional roles to play. One of the goals of the course is to give them experience in mentoring more junior colleagues and managing a team project. As such, they were expected to acquire real-world skills in project organization and management.

3.2 Schedule and Major Components

As designed, the course spans a standard fifteen week semester, meeting twice a week (Monday and Wednesday) for one hour and fifteen minutes each session. The general setup is shown in Figure 2. As this paper goes to press (mid-April), the course just concluded Week 11.

During the first three weeks, all students are immersed in a "Hadoop boot camp", where they are introduced to the MapReduce programming framework. Material was adapted from slides developed by Christophe Bisciglia and his colleagues from Google, who have delivered similar content in various formats.[3] As it was assumed that all students had strong foundations in computer science, the pace of the lectures was brisk. The themes of the five boot camp sessions are listed below:

- Introduction to parallel/distributed processing
- From functional programming to MapReduce and the Google File System (GFS)
- "Hello World" MapReduce lab
- Graph algorithms with MapReduce
- Information retrieval with MapReduce

A brief overview of parallel and distributed processing provides a natural transition into abstractions afforded by functional programming, the inspiration behind MapReduce. That in turn provides the context to introduce MapReduce itself, along with the distributed file system upon which it depends. The final two lectures focus on specific case studies of MapReduce applied to graph analysis and information retrieval. The first covers graph search and PageRank, while the second covers algorithms for information retrieval. With the exception of the "Hello World" lab session, all lecture content was delivered at the conceptual level, without specific reference to the Hadoop API and implementation details (see Section 5 for discussion). The boot camp is capped off with a programming exercise (implementation of PageRank) to ensure that students have a passing knowledge of MapReduce concepts in general and the Hadoop API in particular.

Concurrent with the boot camp, team leaders are expected to develop a detailed plan of research: what they hope to accomplish, specific tasks that would lead to the goals, and possible distribution of those tasks across team members. I recommend that each project be structured into two phases: the first phase focusing on how existing solutions might be recast into the MapReduce framework, the second phase focusing on interesting extensions enabled by MapReduce. In addition to the detailed research

[3] http://code.google.com/edu/parallel/

plan, the leaders are responsible for organizing introductory material (papers, tutorials, etc.) since team members are not expected to have any prior experience with the research topic.

The majority of the course is taken up by the research project itself. The Monday class sessions are devoted to the team project meetings, and the team leader is given discretion on how this is managed. Typical activities include evaluation of deliverables (code, experimental results, etc.) from the previous week and discussions of plans for the upcoming week, but other common uses of the meeting time include whiteboard sessions and code review. During the project meetings I circulate from group to group to track progress, offer helpful suggestions, and contribute substantially if possible.

To the extent practical, the teams adopt standard best practices for software development. Students use Eclipse as the development environment and take advantage of a plug-in that provides a seamless interface to the Hadoop cluster. Code is shared via Subversion, with both project-specific repositories and a course-wide repository for common libraries. A wiki is also provided as a point of collaboration.

Concurrent with the project meetings on Mondays, a speaker series takes place on Wednesdays. Attendance for students is required, but otherwise the talks are open to the public. One of the goals for these invited talks is to build an active community of researchers interested in large datasets and distributed processing. Invited talks can be classified into one of two types: infrastructure-focused and application-focused. Examples of the first include alternative architectures for processing large datasets and dynamic provisioning of computing services. Examples of the second include survey of distributed data mining techniques and Web-scale sentiment analysis. It is not a requirement for the talks to focus on MapReduce *per se*—rather, an emphasis on large-data issues is the thread that weaves all these presentations together.

3.3 Student Evaluation

At the beginning of the course, students are assigned specific roles (team leader or team member) and are evaluated according to different criteria (both in grade components and relative weights).

The team leaders are responsible for producing the detailed research plan at the beginning of the semester. The entire team is responsible for three checkpoint deliverables throughout the course: an initial oral presentation outlining their plans, a short interim progress report at roughly the midpoint of the semester, and a final oral presentation accompanied by a written report at the end of the semester.

On a weekly basis, I request from each student a status report delivered as a concise email: a paragraph-length outline of progress from the previous week and plans for the following week. This, coupled with my observations during each project meeting, provides the basis for continuous evaluation of student performance.

4 Course Implementation

Currently, 13 students (7 Ph.D., 3 masters, 3 undergraduates) are involved in the course, working on six different projects. Last fall, as planning was underway, Ph.D. students from the Laboratory for Computational Linguistics and Information Processing at the University of Maryland were recruited as team leaders. Three of them agreed, developing projects around their doctoral research—these represent cases with maximal alignment of research and educational goals. In addition, the availability of this opportunity was announced on mailing lists, which generated substantial interest. Undergraduates were recruited from the Computer Science honors program; since it is a requirement for those students to complete an honors project, this course provided a suitable vehicle for satisfying that requirement.

Three elements are necessary for a successful project: interested students, an interesting research problem of appropriate scope, and the availability of data to support the work. I served as a broker for all three elements, and eventually settled on five projects that satisfied all the desiderata (one project was a later addition). As there was more interest than spaces available for team members, it was possible to screen for suitable background and matching interests. The six ongoing projects are as follows:

- Large-data statistical machine translation
- Construction of large latent-variable language models
- Resolution of name mentions in large email archives

- Network analysis for enhancing biomedical text retrieval
- Text-background separation in children's picture books
- High-throughput biological sequence alignment and processing

Of the six projects, four of them fall squarely in the area of human language technology: the first two are typical of problems in natural language processing, while the second two are problems in information retrieval. The final two projects represent attempts to push the boundaries of the MapReduce paradigm, into image processing and computational biology, respectively. Short project descriptions can be found on the course homepage.

5 Pedagogical Discussion

The design of any course is an exercise in tradeoffs, and this pilot project is no exception. In this section, I will attempt to justify course design decisions and discuss possible alternatives.

At the outset, I explicitly decided against a "traditional" course format that would involve carefully-paced delivery of content with structured exercises (e.g., problem sets or labs). Such a design would perhaps be capped off with a multi-week final project. The pioneering MapReduce course at the University of Washington represents an example of this design (Kimball et al., 2008), combining six weeks of standard classroom instruction with an optional four week final project. As an alternative, I organized my course around the research project. This choice meant that the time devoted to direct instruction on foundational concepts was very limited, i.e., the three-week boot camp.

One consequence of the boot-camp setup is some disconnect between the lecture material and implementation details. Students were expected to rapidly translate high-level concepts into low-level programming constructs and API calls without much guidance. There was only one "hands on" session in the boot camp, focusing on more mundane issues such as installation, configuration, connecting to the server, etc. Although that session also included overview of a simple Hadoop program, that by no means was sufficient to yield in-depth understanding of the framework.

The intensity of the boot camp was mitigated by the composition of the students. Since students were self-selected and further screened by me in terms of their computational background, they represent the highest caliber of students at the university. Furthermore, due to the novel nature of the material, students were highly motivated to rapidly acquire whatever knowledge was necessary outside the classroom. In reality, the course design forced students to spend the first few weeks of the project simultaneously learning about the research problem and the details of the Hadoop framework. However, this did not appear to be a problem.

Another interesting design choice is the mixing of students with different backgrounds in the same classroom environment. Obviously, the graduate students had stronger computer science backgrounds than the undergraduates overall, and the team leaders had far more experience on the particular research problem than everyone else by design. However, this was less an issue than one would have initially thought, partially due to the selection of the students. Since MapReduce requires a different approach to problem solving, significant learning was required from everyone, independent of prior experience. In fact, prior knowledge of existing solutions may in some cases be limiting, since it precludes a fresh approach to the problem.

6 Course Evaluation

Has the course succeeded? Before this question can be meaningfully answered, one needs to define measures for quantifying success. Note that the evaluation of the course is distinct from the evaluation of student performance (covered in Section 3.3). Given the explicit goal of integrating research and education, I propose the following evaluation criteria:

- Significance of research findings, as measured by the number of publications that arise directly or indirectly from this project.
- Placement of students, e.g., internships and permanent positions, or admission to graduate programs (for undergraduates).
- Number of projects with sustained research activities after the conclusion of the course.

- Amount of additional research support from other funding agencies (NSF, DARPA, etc.) for which the projects provided preliminary results.

Here I provide an interim assessment, as this paper goes to press in mid-April. Preliminary results from the projects have already yielded two separate publications: one on statistical machine translation (Dyer et al., 2008), the other on information retrieval (Elsayed et al., 2008). In terms of student placement, I believe that experience from this course has made several students highly attractive to companies such as Google, Yahoo, and Amazon—both for permanent positions and summer internships. It is far too early to have measurable results with respect to the final two criteria, but otherwise preliminary assessment appears to support the overall success of this course.

In addition to the above discussion, it is also worth mentioning that the course is emerging as a nexus of cloud computing on the Maryland campus (and beyond), serving to connect multiple organizations that share in having large-data problems. Already, the students are drawn from a variety of academic units on campus:

- The iSchool
- Department of Computer Science
- Department of Linguistics
- Department of Geography

And cross-cut multiple research labs:

- The Institute for Advanced Computer Studies
- The Laboratory for Computational Linguistics and Information Processing
- The Human-Computer Interaction Laboratory
- The Center for Bioinformatics and Computational Biology

Off campus, there are ongoing collaborations with the National Center for Biotechnology Information (NCBI) within the National Library of Medicine (NLM). Other information-based organizations around the Washington, D.C. area have also expressed interest in cloud computing technology.

7 Conclusion

This paper describes the design of an integrated research and educational initiative focused on tackling Web-scale problems in natural language processing and information retrieval using MapReduce. Preliminary assessment indicates that this project represents one viable approach to bridging classroom instruction and real-world research challenges. With the advent of clusters composed of commodity machines and "rent-a-cluster" services such as Amazon's EC2,[4] I believe that large-data issues can be practically incorporated into an HLT curriculum at a reasonable cost.

Acknowledgments

I would like to thank the generous hardware support of IBM and Google via the Academic Cloud Computing Initiative. Specifically, thanks go out to Dennis Quan and Eugene Hung from IBM for their tireless support of our efforts. This course would not have been possible without the participation of 13 enthusiastic, dedicated students, for which I feel blessed to have the opportunity to work with. In alphabetical order, they are: Christiam Camacho, George Caragea, Aaron Cordova, Chris Dyer, Tamer Elsayed, Denis Filimonov, Chang Hu, Greg Jablonski, Alan Jackoway, Punit Mehta, Alexander Mont, Michael Schatz, and Hua Wei. Finally, I would like to thank Esther and Kiri for their kind support.

References

Michele Banko and Eric Brill. 2001. Scaling to very very large corpora for natural language disambiguation. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL 2001)*, pages 26–33, Toulouse, France.

Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean. 2007. Large language models in machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 858–867, Prague, Czech Republic.

Eric Brill, Jimmy Lin, Michele Banko, Susan Dumais, and Andrew Ng. 2001. Data-intensive question answering. In *Proceedings of the Tenth Text REtrieval*

[4] http://aws.amazon.com/ec2

Conference (TREC 2001), pages 393–400, Gaithersburg, Maryland.

Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI 2004)*, pages 137–150, San Francisco, California.

Chris Dyer, Aaron Cordova, Alex Mont, and Jimmy Lin. 2008. Fast, easy, and cheap: Construction of statistical machine translation models with MapReduce. In *Proceedings of the Third Workshop on Statistical Machine Translation at ACL 2008*, Columbus, Ohio.

Tamer Elsayed, Jimmy Lin, and Douglas Oard. 2008. Pairwise document similarity in large collections with MapReduce. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL 2008), Companion Volume*, Columbus, Ohio.

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google File System. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP-03)*, pages 29–43, Bolton Landing, New York.

Aaron Kimball, Sierra Michels-Slettvet, and Christophe Bisciglia. 2008. Cluster computing for Web-scale data processing. In *Proceedings of the 39th ACM Technical Symposium on Computer Science Education (SIGCSE 2008)*, pages 116–120, Portland, Oregon.