Book Reviews

Data-Intensive Text Processing with MapReduce

Jimmy Lin and Chris Dyer
(University of Maryland)

Morgan & Claypool (Synthesis Lectures on Human Language Technologies, edited by Graeme Hirst, volume 7), 2010, xi+165 pp; paperbound, ISBN 978-1-60845-342-9, $40.00; ebook, ISBN 978-1-60845-343-6, $30.00 or by subscription

Reviewed by
Peng Xu
Google Inc.

The world has been blessed by the ever-growing World Wide Web and the associated vast amount of information available through commercial search engines. Many subfields of computational linguistics, such as speech recognition, machine translation, summarization, coreference resolution, question answering, word sense disambiguation, and so on, are playing increasingly important roles in information extraction from the Web. At the same time, the Web itself is also providing more and more data for computational linguists to study. For example, in the past decade, the amount of text available for speech recognition and machine translation research has increased by several orders of magnitude. Corpora consisting of hundreds of millions of words are not uncommon anymore. This clearly poses serious challenges in terms of computation for traditional text processing approaches using a single computer. As a result, efficient distributed computing has become more crucial than ever.

This book, *Data-Intensive Text Processing with MapReduce*, written by Jimmy Lin and Chris Dyer, is an excellent source for computational linguists to start the paradigm shift. The book focuses on algorithm design with MapReduce—a programming model for distributed computations on massive data sets on clusters of commodity servers. With text processing algorithms at the core, the book provides easy-to-follow MapReduce design patterns with reusable solutions to common problems in natural language processing, information retrieval, and machine learning. The entire book is based on the authors' excellent experiences in using MapReduce on Hadoop, the most well-known open source implementation of MapReduce and a proven commodity easily available to academia. It is a must-read for computational linguists who might be interested in taking advantage of billion-word level corpora in their research. Given the increasing importance of large data processing both in research and in industry, this book can also serve as an excellent supplementary textbook in modern computer science education.

Chapter 1: Introduction

The first chapter of the book is really fun to read. The authors first give convincing facts about the importance and availability of large data corpora. It follows naturally that large-scale data processing using computer clusters is inevitable. With the authors' interesting view of MapReduce as a service in the cloud (cloud computing), it becomes clear that the MapReduce programming model is a powerful abstraction that separates

the *what* from the *how* of data-intensive processing. What's more revealing and interesting are the "Big Ideas" that the authors summarize for MapReduce:

- Scale "out," not "up."
- Assume failures are common.
- Move processing to the data.
- Process data sequentially and avoid random access.
- Hide system-level details from the application developer.
- Incorporate seamless scalability.

These Big Ideas behind MapReduce are excellent motivations for readers to move on to the rest of the book. They are the secret sauce that makes large-scale data processing a commodity to regular users, including computational linguists.

Chapter 2: MapReduce Basics

Starting from the functional programming roots of MapReduce, the authors present a precise picture of how MapReduce works. They not only show the fundamental concepts in programming with MapReduce, but also describe the execution framework and the highly optimized distributed file systems backing the computation and storage needs of MapReduce. Fortunately, as the authors point out, these details in execution framework and distributed file systems are integrated components of MapReduce, but they are not part of the programming model. Undoubtedly, MapReduce would not work well without the execution framework and the supporting distributed file system, but the fact that users of MapReduce do not need to address them directly makes it even more appealing. The most important concepts from this chapter are mappers, reducers, partitioners, and combiners, all of which are concisely and precisely described with a simple example. A smart reader could already start designing MapReduce programs!

Chapter 3: MapReduce Algorithm Design

Diving deeper into the MapReduce world, the authors explain several important and useful design patterns in this chapter. The focus is on controlling code execution and data flow—to design algorithms that are both scalable and efficient. On the surface, it may seem that users have to express algorithms in terms of a small set of rigidly defined components in the MapReduce environment, but as the authors show with concrete examples, there exist many controls at the users' disposal to shape the flow of computation.

It is particularly nice that the authors introduce the design patterns through a series of tasks that are simple yet challenging when the amount of data is large. These tasks and example algorithms guide the readers to gain a better understanding of the basics from the previous chapter. Each design pattern is presented to solve or alleviate a particular performance issue in real examples. Advantages and disadvantages of each design pattern are explained at length. Through the examples and the design patterns, it becomes increasingly clear that scalable and efficient algorithms can be achieved by controlling synchronization in the MapReduce programming model.

Chapters 4–6: Inverted Web Indexing, Graph Algorithms, EM Algorithms

In these chapters, three classes of important real-world problems are studied in detail within the MapReduce world. The first problem is to build a Web index—one of the core components of Web search. The design patterns from the previous chapter are reinforced here, either to solve a scalability bottleneck or to properly set parameters for compressing the Web index.

The second class of problems is related to graph algorithms. Graph algorithms can be applied to many real-world problems, such as ranking in Web search, friend suggestion in social networks, and product promotion in advertising, to name a few. Two such problems, breadth-first search and PageRank, are discussed in detail with an iterative design which employs a driver program to chain multiple MapReduce tasks.

What would concern computational linguists the most is the last class of problems: expectation maximization (EM) algorithms or gradient-based optimization techniques. The authors show that the EM algorithms can be expressed naturally in the MapReduce programming model. Many EM algorithms make independence assumptions that permit a high degree of parallelism in both the E- and M-steps. Furthermore, EM algorithms are unsupervised learning algorithms, which makes them much more often exposed to large data sets. A generic EM algorithm in the MapReduce programming model is discussed in Chapter 6, together with a case study for the word alignment problem in machine translation. The authors show that the MapReduce implementation of the EM algorithm can achieve significant speed-up compared to a highly optimized single-core implementation. When the computation is expensive, such as in the HMM word alignment model, near perfect speed-up can be achieved because the overhead associated with distributing and aggregating data in MapReduce is almost negligible compared to the real computation. The discussion demonstrates that not only does MapReduce provide a means for coping with ever-increasing amount of data, but it is also useful for parallelizing expensive computations. The ability to leverage clusters of commodity hardware to parallelize computationally expensive algorithms is an important use case of MapReduce.

Conclusion

Jimmy Lin and Chris Dyer have done a fantastic job in demonstrating the beauty and simplicity of the MapReduce programming model in this well-written book. With solid data, simple algorithms, and convincing reasoning, the authors show us how low the barriers are to adopting MapReduce. The capabilities necessary to tackle large-data problems are already within reach of many and will continue to become accessible over time. For our fellow computational linguists, it is easy to imagine the possibility of Web-scale unsupervised natural language grammar induction, Web-scale machine reading and text understanding, and machine translation systems using parallel text from the whole world. As the authors say, "The golden age of massively distributed computing is finally upon us." This reviewer believes this book could potentially help computational linguistics to reach the "golden age of unsupervised natural language processing."

This review was edited by Pierre Isabelle.

Peng Xu is a staff research scientist at Google Inc. He has been working on large-scale language modeling using MapReduce and other aspects of large-scale machine translation systems. Peng's address is: 1600 Amphitheatre Parkway, Mountain View, CA 94043; e-mail: xp@google.com.