MapReduce modules for undergraduate instruction

Weiwei Ge
Health Informatics
UNC Charlotte
Charlotte, NC
wge@uncc.edu

David J. John
Computer Science
Wake Forest University
Winston-Salem, NC
djj@wfu.edu

Stan J. Thomas
Computer Science
Wake Forest University
Winston-Salem, NC
sjt@wfu.edu

ABSTRACT

Recent curriculum recommendations have underscored the importance of introducing parallel and distributed computing (PDC) in the undergraduate curriculum. Implementing these recommendations is challenging for many computer science (CS) programs. Our approach has been to develop PDC modules for use within existing courses. In this poster we introduce two modules which focus on the *MapReduce* paradigm. The first is an introductory module, appropriate for beginning CS students. The second is an intermediate module targeted to more advanced students. The modules will be described by diagrams and pseudocode in the poster format. Pre- and post-tests to determine the short-term effectiveness of these modules will be described.

Keywords

MapReduce, instructional modules, undergraduate education, Hadoop

1. INTRODUCTION

The ACM/IEEE Joint Curriculum (CS2013) recommendation [2] urges the inclusion of parallel and distributed computing, as well as other timely topics, in the undergraduate CS major. The challenge for CS programs is to determine which of these topics will be included, and how they will be integrated into densely packed curricula.

One approach to addressing the recommendations is to develop PDC modules that can be integrated into the existing curriculum. Several groups, *CSinParallel* [4], *LittleFE* [1], and *Shodor* [7], are spearheading this effort and developing and distributing instructional modules that focus on parallel and distributed computing.

Our goal is to integrate PDC modules across the core curriculum. Focusing on core courses guarantees that PDC topics and concepts will reach every major. To this end we have previously reported [6] on instructional modules based on message passing, specifically MPI. In this poster we describe two modules based on distributed computing using the *MapReduce* paradigm within the Hadoop framework.

For our purposes, a module consists of three components: a written document, an instructor, and student work. Each module is designed to be completed in three contact hours plus student work time. A written document is distributed to each student that states the goals of the module, introduces the PDC algorithm, including examples, and includes executable source code.. Student work, from reading and understanding the written document, to implementing source code to solve a problem (likely similar to an example in the written document), is crucial to the goals of the module.

For this poster two *MapReduce* modules are presented. The first is designed for a Java-based CS1 course. Here the emphasis is on understanding the algorithm and its implementation. The second module is designed for CS students in an algorithms course. These students will be expected to implement both a *mapper* and *reducer*. In the future, for CS0 and CS1 level courses, we plan to experiment with the WebMapReduce interface [5]. *WebMapReduce* provides a simple web-based interface for creating and submitting *MapReduce* jobs in several programming languages.

2. MAPREDUCE

MapReduce provides a powerful distributed computing approach to processing large data sets over multiple computers. The *MapReduce* paradigm is often implemented within the Apache Hadoop framework [3], although proprietary implementations exist.

In our experience, the greatest challenge for students upon encountering the *MapReduce* paradigm is often not the mapreduce data flow itself but rather the details of its implementation within the Hadoop framework. Most students have not (knowingly) worked with a distributed file system, and many students at this level have not worked with serialized data, two key concepts in the *MapReduce* environment. Another issue for students is the question of what is a "large" data set. Simple examples based on data sets appropriate for processing on a single computer may not convince students of the power of the *MapReduce* paradigm. On the other hand, processing genuine "big" data sets often requires significant time, even in a cluster environment. The combination of such factors creates a challenging environment for software development and debugging by students.

3. THE MODULES

The tasks chosen for the two modules both involve text processing. The first problem asks "for a large set of docu-

EduHPC-15 Austin, Texas USA

ments, can we produce an inverted index to associate words with the documents in which they occur?" As will be discussed in Section 3.1, this problem lends itself readily to one application of *MapReduce*. The second problem asks "for a large set of documents, which of these is most similar to a known document?" By design, the solution to the second problem builds upon the solution of the first. All students working on the second module must review the concepts and algorithms in the first module. The solution of this problem uses four applications of *MapReduce*.

3.1 Introductory Module

Module one is designed to be used in the last quarter of a Java-based CS1 class. This module requires students to be able to understand *MapReduce* and to read its implementation in Java and perhaps to modify the code provided. There is no expectation that the students will be able to create their own *MapReduce* implementation after this module but they should be able to describe the process.

The goals of the module are to develop a basic understanding of the *MapReduce* paradigm, to contrast *MapReduce* with other programming paradigms, to illustrate the *map* and *reduce* processes, and to discuss under what conditions *MapReduce* is applicable. The terms *key* and *value* are most important, and care must be taken to insure each student understands them. The *MapReduce* paradigm is explained as the input is read by mappers that produce intermediate <*key, value*> pairs which undergo shuffling and sorting; the resulting <*key, value*> pairs are the input to the reducers which produce the final output, as <*key, value*> pairs.

After illustrating a MapReduce solution for the classic word frequency example on a large data set, students are then asked to consider the following problem: the *Old Gold and Black*[1] has thousands of news articles in their file system, and they want to know which files contain important words. Of course, the students are being asked to develop an inverted list while omitting stop words. Again, the instructor works with the students to create, execute, and understand a working MapReduce solution.

The most challenging aspect for CS1 students is the abundant use of various specialized Java classes and associated methods for data serialization and processing in *MapReduce*. The instructor must take care to explain the purpose of these classes without getting mired in the implementation details.

3.2 Intermediate Module

The intermediate module is for students in a more advanced course focusing on algorithms. It is assumed that each student is comfortable with and reasonably proficient using an object oriented programming language (ideally Java). Furthermore, it is assumed that most have previously completed the first module in their CS1 course. An assignment for all students before commencing this module is to review module one.

The goals of this module are to understand the notion of document similarity, to understand *MapReduce*, the understand the benefit of *MapReduce*, and to be able to write a MapReduce program in the Hadoop environment. In this module the problem to solve is to determine which previous article in the *Old Gold and Black* is most similar to a targe article. Cosine similarity is introduced as the mechanism for measuring document similarity.

[1]Weekly student newspaper at Wake Forest University

4. EVALUATION

A small set of questions is used with each module as a pre- and post-test. The purpose of these questions is to ascertain if, in the short-term, the goals of the module are being achieved. The same questions appear in both instruments, which gives a straightforward measurement of the impact of the module.

5. CONCLUSIONS

The *MapReduce* modules can be challenging for students and instructors. For the students, an additional topic is being introduced into a course that already has a large set of goals. *MapReduce*, as well as most other PDC paradigms, requires the students and instructors to incorporate new and perhaps unfamiliar computing environments, such as the *Hadoop* framework, into the course's computing toolset. For the instructors, intentionality is required to include a PDC module. Often when teaching a course there are various pressures that demand extra, and unexpected, instructor and student time. There is a temptation to omit a PDC module to free up needed time.

For those creating PDC modules, the challenge is to design compact instruction units that fit naturally into existing courses. Successful modules will complement the course content while interjecting PDC content appropriate to the course and skill level of the students.

Integrating PDC modules into core undergraduate courses can effectively meet curriculum goals such as those suggested in CS2013. A tremendous advantage of the modular approach is the lack of a need of an additional core course; however, to add modules to an existing core course does require modification of the course topics.. Of course, module integration does not preclude separate elective courses focusing on specific aspects of parallel and distributed computing in depth.

6. REFERENCES

[1] Acme project. LittleFE: Parallel and Cluster Computing Education On The Move. `http://littlefe.net`.

[2] ACM/IEEE-CS Joint Task Force on Computing Curricula. Computer science curricula 2013. Technical report, ACM Press and IEEE Computer Society Press, December 2013. URL: `http://dx.doi.org/10.1145/2534860`.

[3] Apache Software Foundation. Welcome to Apache Hadoop. `https://hadoop.apache.org`.

[4] CSinParallel. CSinParallel: Parallel Computing in the Computer Science curriculum. `http://serc.carlton.edu/csinparallel/index.html`.

[5] CSinParallel. WebMapReduce. `http://serc.carleton.edu/csinparallel/ppps/wmr.html`.

[6] D. J. John and S. J. Thomas. Parallel and distributed computing across the computer science curriculum. In *Parallel Distributed Processing Symposium Workshops (IPDPSW), 2014 IEEE International*, pages 1085–1090, May 2014. URL: `http://dx.doi.org/10.1109/IPDPSW.2014.121`.

[7] Shodor. SHODOR: a national resource for computational science education. `http://shodor.org`.