Teaching Map-reduce Parallel Computing in CS1

Richard Brown, Patrick Garrity,
Timothy Yates
Mathematics, Statistics, and
Computer Science
St. Olaf College
Northfield, MN
rab@stolaf.edu

Elizabeth Shoop
Mathematics, Statistics, and
Computer Science
Macalester College
Saint Paul, MN
shoop@macalester.edu

Abstract

Cluster-based map-reduce programming frameworks such as Apache Hadoop make compelling pedagogical examples of parallel computing in undergraduate CS education. Although the basic notions of map-reduce computation are at an appropriate level for introductory CS students, directly programming Hadoop is too difficult for most beginning students. However, the strategically simplified WebMapReduce system makes map-reduce computation accessible for CS1 students, using the programming language they study for their course. Modular materials for teaching map-reduce computing with WebMapReduce are cited, and experience using them are discussed.

Introduction

Because of recent changes in computer architecture brought on by physical engineering limitations, multi-core processors have now become standard in commodity computers, and computers will have more and more cores per CPU in the future[3][2]. Thus, programmer-level multi-core parallelism will inevitably be part of our students' careers in computing for years to come, which leads to the current national call for teaching our CS undergraduate students more principles of parallel computing [17].

The multi-core imperative is not the only reason to teach more parallelism to undergraduates. Some of the most exciting emerging applications of computing power involve *data-intensive scalable computing (DISC)*, in which computational frameworks such as Google's proprietary MapReduce [10] and the open-source Apache Hadoop [1] are applied to data sets that may have the scale of terabytes or even petabytes [12]. Such data no longer appears only in scientific fields such as astronomy, but has now become the purview of cybersecurity, web search, social computing, and web 2.0 applications. Furthermore, the well-known map-reduce problem solving techniques employed in these framework systems can equally well be applied to large data sets from sources that may have nothing to do with web computing, such as academic disciplines in the sciences, social sciences, humanities, etc. The availability of cloud computing platforms such as Amazon EC2 [4] at reasonable rates, and of open-source tools such as Hadoop and the Cloudera support tools [9], makes it feasible for colleges or individuals to explore DISC without extensive local or personal investment in cluster computing. Alternatively, even small colleges can provide practical and significant cluster computational systems through local computing laboratory resources and virtual-machine technologies [7], and Hadoop map-reduce computing can be demonstrated on even a single computer [1]. Furthermore, the map-reduce model exemplifies the combination of programmer productivity, scalability, and efficient performance that parallel programming models of the future must provide [3].

Can parallel computing concepts appear at the level of CS1? We hold that parallelism can and should appear as early as CS1. First, various authors such as [11], [16], and [8] have shown that parallel computing principles and practices can be introduced in CS1 or earlier. These examples teach level-appropriate concepts of parallelism. As members of the NSF/TCPP Curriculum Initiative on Parallel and Distributed Computing say, it's the faculty who must shift paradigms to include parallelism, not the students [15]. Second, recent proposals for adding more parallelism to CS curricula favor introducing the concepts and practices of parallel computing in an incremental, integrated fashion throughout undergraduate CS curricula. For example, the NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing document [14] states that "Students will more naturally start to think in parallel and distributed terms when they sense that their professors are always conscious of the implications of parallelism and distribution of computation with respect to each topic in their courses." Also, [6] recommends that professors "[t]each parallelism early and often, at all levels of undergraduate CS curricula. Both CS majors and non-majors need background in parallelism, and the material merits ongoing vertical study during a major's undergraduate career," and "[u]se a spiral approach for presenting substantial concepts and issues in parallelism" that revisits substantial principles of parallelism at various points throughout a student's coursework. Beginning the study of parallelism in CS1 helps to achieve these goals, and provides at least some desirable background in parallel computing to those non-majors who take CS1 as their only CS course.

Assuming that we intend to teach parallelism in the introductory course, what concepts and practices of parallelism shall we teach, and how shall we teach it? In [11], CS1 students program with Java threads to parallelize data-parallel computations, involving images and cryptography on data sets that can be partitioned without overlapping, leading to observable performance speedup. The authors of [8] used role-playing games derived from [13] to teach essential concepts of parallelism in multiple "thinking parallel" course segments. These supplemented parallel programming exercises with C++ in a three-day camp experience for talented high school students, although a course might include effective active-learning exercises that teach concepts of parallel computing whether or not they accompany parallel programming exercises.

What about map-reduce computing? Because of its importance to highly prominent technology companies such as Google, Yahoo!, and Facebook, map-reduce computing has the potential to capture the natural interest of students. Also, the basic idea of map-reduce computation is simple enough for beginning students who can write functions in their programming language: a *mapper* function decomposes chunks of data into labeled bits (key-value pairs); and a *reducer* function combines all those bits that share common labels. (More will be explained in the next section). This two-function strategy proves to be quite flexible and powerful for processing large data sets. However, the Hadoop programming interface assumes a level of sophistication in Java programming that few beginning programmers would find comfortable.

However, we have produced a strategically simplified web interface, called WebMapReduce, that enables CS1 students to carry out map-reduce computations at an appropriate level, using the programming languages they are learning for their introductory course. After presenting some further background in map-reduce computation, we describe the WebMapReduce tool and the teaching materials we have been using to teach map-reduce computing to beginners.

1 Map-reduce Computing Background

The staged use of mapper and reducer functions (or methods) described above represents a functional programming strategy that LISP programmers have used for decades. Thus, the Google engineers who created the MapReduce parallel programming framework were starting with a well-known algorithmic pattern. They were not first to apply these ideas to parallel computations; for example, the *reduction* strategy is common in parallel programming for accumulating distributed results and avoiding data-race conditions in shared memory systems [MPI website], [OpenMP website]. However, [Dean & Ghemawat 2004] describes a practical and useful system that reuses low-level code for fault tolerance, network communication, and optimized performance, freeing the programmer to solve problems by providing high-level algorithms expressed as functions or methods (along with job configuration parameters). Many experts anticipate that similar uses of the functional programming paradigm will lead to a new generation of programming tools for parallel computations (see, for example, [17]).

Map-reduce programmers often think of their algorithms in read-write cycles, as illustrated by Figure 1. Computations such as the celebrated PageRank algorithm [5] may require multiple read-write cycles to accomplish their goals.

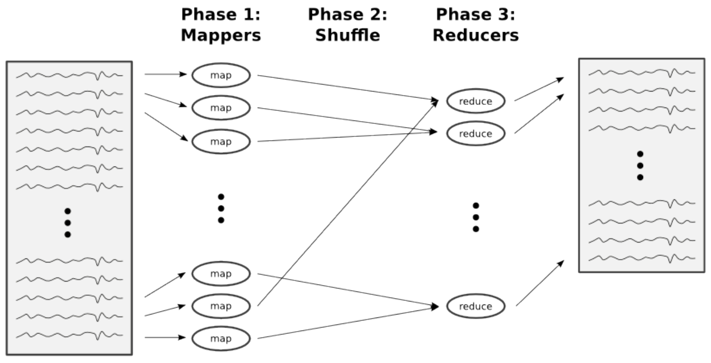


Figure 1: A simplified view of the map-reduce process.

Here, in a first phase, the data is read by multiple mapper functions executing in parallel. *After all mappers have completed their work,* the key-value pairs they produce are organized by key as a second "shuffle" phase; *thereafter*, calls to reducer functions executing in parallel each receive all values associated with a given key, combine those values, and produce results. We emphasize that map-reduce frameworks operate in three disjoint phases. This bypasses an opportunity for pipeline parallelism, but it enables a map-reduce framework to recover automatically from all but the most catastrophic faults, as described in [10]. This fault tolerance is essential for large DISC computations on clusters with thousands of nodes, because node failure is unavoidable in even the highest quality systems.

In addition, map-reduce frameworks provide various optimizations. For example, data is arranged so that mappers and reducers always perform local I/O during phases 1 and 3. Also, keys are automatically sorted during the shuffle phase, which efficiently reduces the need for such sorting operations in mappers and reducers.

2 WebMapReduce

Programming in the (cluster-based) map-reduce model exposes students to a number of aspects of parallel computation, including the following.

- *Data parallelism*, in which the same algorithm is applied to different data sets in parallel. Both the mapper phase and the reducer phase represent data parallelism, fitting the SPMD category (Single Program, Multiple Data) of computation that cluster computations frequently use.

- *Data locality.* Students may safely assume that only their mapper or reducer functions operate on those functions' arguments, freeing them to think functionally on local variables. Data locality also arises in discussions about the optimization in which mappers and reducers perform only local input/output.

- *Distributed computing.* The three-phase map-reduce computation illustrates some aspects of distributed computing systems, including parallel computations on multiple nodes and data transfer via point-to-point network communication. Also, the map-reduce fault-tolerance model depends on replication of data in a distributed file system, in order to recover data in case one or more cluster nodes crashes.

- *Scalability.* Larger clusters can effectively compute with larger data, which represents the notion of *scalability*, in which properties (such as effective computation) continue to hold in the presence of changes of scale. Also, consideration of the sizes of interesting data sets gives a natural opportunity to discuss the effects of scale. Scalability is a pivotal notion in parallel computation.

This is an illustrative list, not a complete one. Indeed, classroom discussions and student questions often provide openings for introducing additional aspects of parallel computing. For example, a comment about mapping, shuffling, and reducing taking place in three distinct phases could provide a natural opportunity for mentioning the alternative of pipeline (assembly-line) parallelism, a form of *task parallelism*. The rationale for that implementation choice is *fault tolerance*, an issue of great interest in distributed parallel computing. Also, the need for fault tolerance increases as the number of cluster nodes scales upward, a reliability issue that seldom arises in sequential computing. Although map-reduce computing represents only one pattern of parallel computation, any excursion into parallelism can serve as an entry point into parallel thinking.

Unfortunately, programming directly with Hadoop requires far more computational maturity than most CS1 students can muster. For example, the code in Figure 2 is a word-frequency count program for the Hadoop map-reduce framework in the system's native Java language. The use of types alone would probably leave most introductory students behind, even if they were learning Java rather than another programming language. In particular, the `map` method requires four arguments, each having a Hadoop-specific type with its own framework-specific role.

Yet the essential algorithm for computing word frequency is easily understood by a beginning programmer:

- Each mapper call to a map() method processes a single line of input. The mapper breaks its line of input into (space-delimited) words, and emits a key-value pair for each word consisting of that word (as key) and the integer 1 (as value).

```
// Java WordCount for Hadoop
// Based on Hadoop documentation

package edu.stolaf.cs;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class WordCount {

  public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("WordCount");

    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);

    conf.setMapperClass(Map.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);

    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);

    FileInputFormat.setInputPaths(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    JobClient.runJob(conf);
  }

  public static class Map extends MapReduceBase implements Mapper<LongWritable,
Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, OutputCollector<Text,
IntWritable> output, Reporter reporter) throws IOException {
      String line = value.toString();
      StringTokenizer tokenizer = new StringTokenizer(line);
      while (tokenizer.hasMoreTokens()) {
        word.set(tokenizer.nextToken());
        output.collect(word, one);
      }
    }
  }

  public static class Reduce extends MapReduceBase implements Reducer<Text,
IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException
{
      int sum = 0;
      while (values.hasNext()) {
        sum += values.next().get();
      }

      output.collect(key, new IntWritable(sum));
    }
  }

}
```

Figure 2. Direct Hadoop implementation of word frequency count (Java)

- Each reducer call to a reduce() method receives a particular key (i.e., a word) as well as all the values that were generated for that key (i.e., some number of integers 1). The reducer emits a single key-value pair, with that word as the key and the *sum* of all the integers 1 for that word as the value.

In order to enable introductory students to program Hadoop, we have created a strategically simplified interface for Hadoop computing, called *WebMapReduce (WMR)* (see webmapreduce.sourceforge.net). WMR contains features such as the following.

- Provides a convenient web-based graphics user interface rather than a command-line interface with explicit compilation and batch job submission. (See Figure 3.)

- Uses a form for job configuration, and translates entries to Hadoop's configuration structure.

- Exposes a simpler programming interface to students.

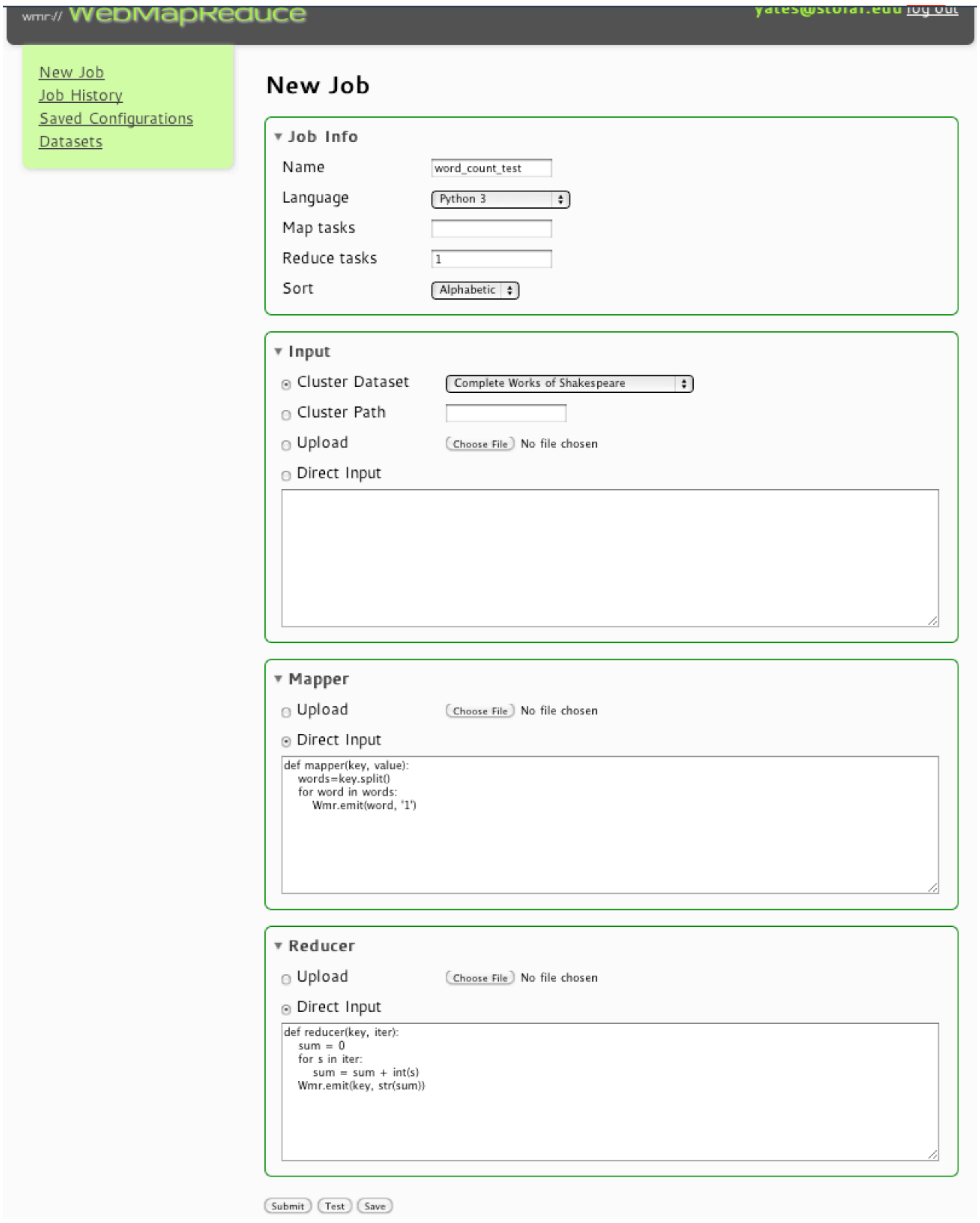
- Stores uploaded input and retrieves and displays output.

- Accepts mapper and reducer functions written in the programming language a student studies for the introductory course that student may be taking. (New languages can be added to the WMR system as needed.)

- Supports customization of the language interface according to local preference. For example, if a course using C++ employs a locally written library of string manipulation functions, that local library may be installed for use within the WMR system (for operations such as breaking a string up into words).

While the earlier points above make the WMR system accessible for beginning students, the latter two items make the WebMapReduce software usable in introductory courses at nearly any institution, employing nearly any programming language and local code libraries. WMR currently supports Python, C++, Java, and Scheme programming, with two varieties of Scheme interface (reducers written in an imperative vs. a functional style).

Figure 4 shows Java Mapper and Reducer classes for the word frequency count example in WebMapReduce. This code is much shorter than the corresponding Java code for a direct Hadoop solution of the same problem in Figure 2, and it is also far more accessible to beginners. A Python implementation solving the same word-frequency problem is even more transparent, due to Python's convenient features for splitting an input line into words and iterating over the result (illustrated in the WMR interface in Figure 3).


wmr:// WebMapReduce

yates@stolaf.edu log out

New Job

Job Info

Name	word_count_test
Language	Python 3
Map tasks	
Reduce tasks	1
Sort	Alphabetic

Input

[✓ Cluster Dataset] Complete Works of Shakespeare
Cluster Path
Upload — Choose File No file chosen
Direct Input

Mapper

Upload — Choose File No file chosen
[✓ Direct Input]

```
def mapper(key, value):
    words=key.split()
    for word in words:
        Wmr.emit(word, '1')
```

Reducer

Upload — Choose File No file chosen
[✓ Direct Input]

```
def reducer(key, iter):
    sum = 0
    for s in iter:
        sum = sum + int(s)
    Wmr.emit(key, str(sum))
```

Figure 3: WebMapReduce interface

```
  class Mapper
  {
      public void map(String key, String value)
      {
          String words[] = key.split(" ");
          for (int i = 0; i < words.length; i++)
          {
              Wmr.emit(words[i], "1");
          }
      }
  }

class Reducer
{
    public void reduce(String key, WmrIterator values)
    {
        int sum = 0;
        for (String value : values)
        {
            sum += Integer.parseInt(value);
        }
        Wmr.emit(key, Integer.valueOf(sum).toString());
    }
}
```

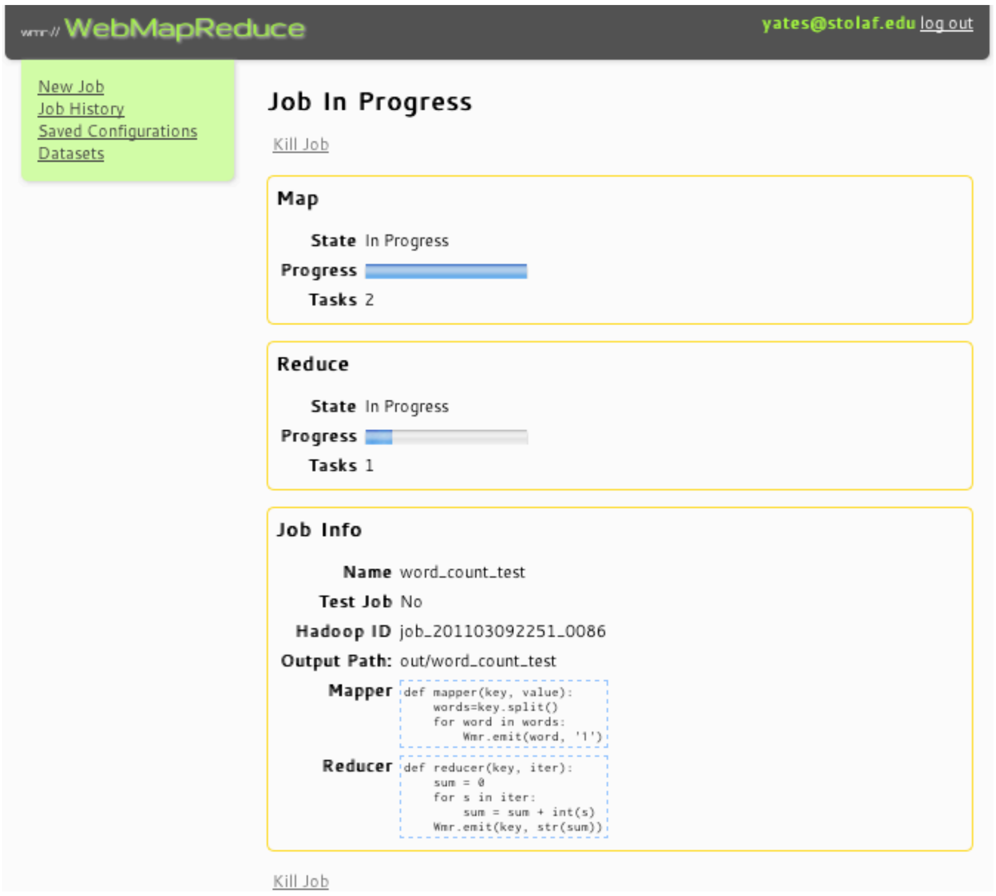
Figure 4. Mapper and Reducer classes for WMR in the Java language

Figure 5 depicts the progression of results that students see after choosing to submit their job to WMR to be run on the underlying Hadoop system. They receive an indication of the progress being made as the computation is progressing (in this case on the Complete Works of Shakespeare as an input file). When the job completes, the output is given, as seen in the lower half of Figure 5.

Finally, we note that a programmer can use WebMapReduce to perform data-intensive scalable computations, since the system provides access to the actual Hadoop computational framework. WebMapReduce can be set up to contain any type of large-scale input data, and many types of computational assignments can be devised for them.

3 A Map-reduce Module for CS1

We have produced modular materials for teaching map-reduce computing at the introductory level in the four languages listed above. These materials include learning objectives, contexts for use, readings for students, homework exercises, and notes for instructors. These modules are designed to require a day or two of class time, including an introduction to map-reduce computing and to WebMapReduce, discussion of homework questions (involving WMR and issues in parallelism), and a follow-up



Job In Progress

Map

State In Progress
Progress
Tasks 2

Reduce

State In Progress
Progress
Tasks 1

Job Info

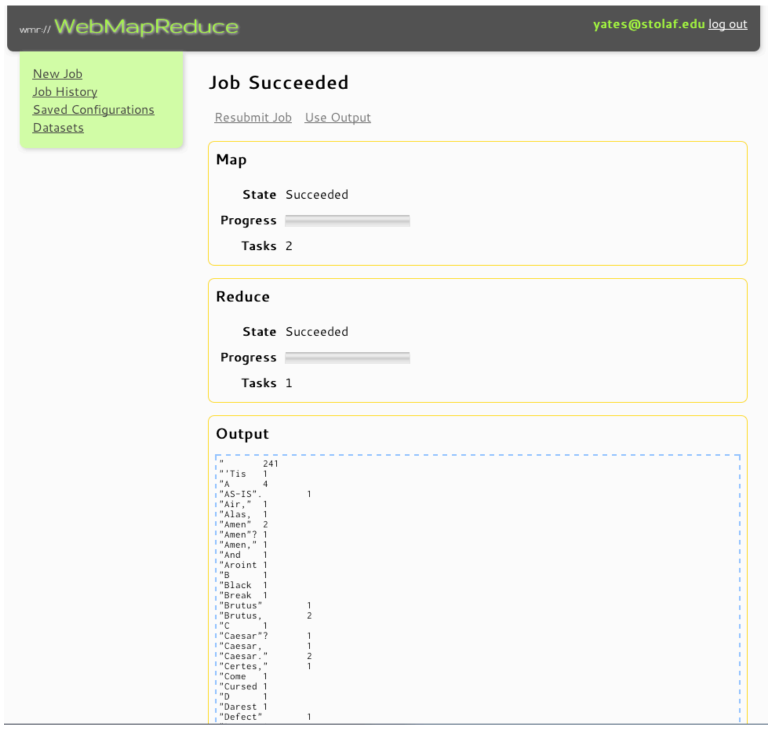
Name word_count_test
Test Job No
Hadoop ID job_201103092251_0086
Output Path: out/word_count_test

Mapper

```
def mapper(key, value):
    words=key.split()
    for word in words:
        Wmr.emit(word, '1')
```

Reducer

```
def reducer(key, iter):
    sum = 0
    for s in iter:
        sum = sum + int(s)
    Wmr.emit(key, str(sum))
```



Job Succeeded

Map

State Succeeded
Progress
Tasks 2

Reduce

State Succeeded
Progress
Tasks 1

Output

```
"       241
"'Tis   1
"A      4
"AS-IS".        1
"Air,"  1
"Alas,  1
"Amen"  2
"Amen"? 1
"Amen," 1
"And    1
"Aroint 1
"B      1
"Black  1
"Break  1
"Brutus"        1
"Brutus,        2
"C      1
"Caesar"?       1
"Caesar,        1
"Caesar."       2
"Certes,"       1
"Come   1
"Cursed 1
"D      1
"Darest 1
"Defect"        1
```

Figure 5. In-progess output of a WMR job, and final results

presentation on further strategies for map-reduce problem solving. These progressive exercises include the following problems.

- *Variations on the word-frequency count example.* For example, instead of whitespace-delimited words, a student may be asked to treat punctuation characters as word delimiters, or to modify the case of words, for instance, so that occurrences of the word "The" will not be tallied separately from "the". Of course, library functions that may be familiar to a particular group of students can be added to simplify these operations in cases where these operations are cumbersome.
- *Other symbolic computations.* These exercises include index generation and sorting a word-frequency list by frequency instead of by word. The latter problem may be solved by taking the output from a standard word-frequency count reducer (as in Figure 3 or 4) and using the key-value pairs produced by that reducer as input for a second mapper in a new round of map-reduce computation. In that second map-reduce cycle,
    - The second cycle's mapper call can simply interchange the elements of its key-value pair, producing a frequency as key and a word as value.
    - The second cycle's reducer call can emit the key-value pairs it receives without change; since the shuffle phase automatically sorts by the (frequency) key.
- *Numerical computations.* Given records whose fields are numbers separated commas, students determine the mean value of one of the columns of input.

We have provided our students with data sets from Project Gutenberg as larger-scale map-reduce data examples. These data sets range in scale from a single book (e.g., *War and Peace or The Complete Works of Shakespeare*) to folders of data representing hundreds of digitized books, one book per line. An example of numerical data is a dataset we derived from the former NetFlix Prize data.

4 Experience with the materials.

We have tested our module materials in introductory CS courses at both of our institutions, using Python and Scheme languages. (Other WMR language interfaces have been class tested in more advanced courses). Our experience leads us to make the following observations.

- The combination of the WebMapReduce software and our modular teaching materials enable our CS1 students to solve computational problems using the map-reduce framework for cluster computing, and to gain hands-on reinforcement of parallel computing principles.
- The ability to customize WebMapReduce libraries according to local practices may be especially important for introductory students. For example, in a Scheme based CS1 course that includes significant imperative programming, we found

that a Scheme-imperative configuration for WMR was difficult for those students, since they were not familiar with the input-output feel of the emit operation.

- Finding the right motivation for a group of students may require local adjustments. At one of our institutions, any mention of Google applications amplifies student interest; at the other institution, examples based on Facebook appear to be more likely to raise interest.
- Interesting and impressive data sets may help to demonstrate the value of parallel map-reduce computing. Determining data sets that will interest a particular student or group is sometimes a challenge, since young students may not understand the effects of scale, and they often take powerful applications of large-scale computing for granted. Significant speedup due to map-reduce parallel programs cannot readily be exhibited without direct computational comparisons to corresponding sequential programs.

Formal assessment of our modular teaching materials is just getting underway, since we are little more than half way through our two-year project.

5 Availability

The open-source WebMapReduce software may be downloaded from the site *webmapreduce.sourceforge.net*. Persons without a cluster for installing WMR and related software, or those seeking a quick test of the software, can use an inexpensive Amazon EC2 cloud image that has WebMapReduce installed.

The CS1 WebMapReduce modular teaching materials are available at *csinparallel.org*. Several other teaching modules for courses at every undergraduate level and various parallel programming systems are also available at the site. We are eager for feedback on any or all of our project software projects and teaching materials.

6 Conclusion

Map-reduce computing frameworks such as the open source Hadoop package provide a powerful and effective means for carrying out data-intensive scalable computing. The strategically simple WebMapReduce software supports map-reduce parallel problem solving that is accessible for introductory students, writing mappers and reducers in the programming language they are already learning in their CS1 course. Modular teaching materials that use WebMapReduce exercises for hands-on learning are available at *csinparallel.org*. The WebMapReduce software may be downloaded from its sourceforge site, and may be installed on a local cluster; alternatively, an Amazon EC2 image including WebMapReduce is available for free and inexpensive to run on that cloud.

Acknowledgments

Development of WMR was funded by NSF CCLI grant DUE- 0942190/0941962, an HHMI grant to St. Olaf College, and additional research support from St. Olaf College. Todd Frederick '09 and Nathaniel Meierpolys '09 implemented an early prototype of the WMR system. We would like to thank our colleagues at both St. Olaf and Macalester for generously making it possible for us to test our materials in their courses.

References

[1] Apache Hadoop Project. *http://hadoop.apache.org/*. Accessed: 06-28-2010.

[2] Asanovic, K., Bodik, B. et al. 2006. *The Landscape of Parallel Computing Research: A View from Berkeley*. Technical Report #UCB/EECS-2006-183. EECS Department, University of California, Berkeley.

[3] Asanovic, K., Bodik, R. et al. 2009. A view of the parallel computing landscape. *Commun. ACM*. 52, 10 (2009), 56-67.

[4] AWS in Education. *http://aws.amazon.com/education/*. Accessed: 05-18-2009.

[5] Brin, S. and Page, L. 1998. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*. 30, 1-7 (Apr. 1998), 107-117.

[6] Brown, R., Shoop, E. et al. 2010. Strategies for Preparing Computer Science Students for the Multicore Worl. *Proceedings of 15th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE)* (2010), to appear.

[7] Brown, R.A. 2009. Hadoop at home: large-scale computing at a small college. *Proceedings of the 40th ACM technical symposium on Computer science education* (Chattanooga, TN, USA, 2009), 106-110.

[8] Chesebrough, R.A. and Turner, I. 2010. Parallel computing: at the interface of high school and industry. *Proceedings of the 41st ACM technical symposium on Computer science education* (New York, NY, USA, 2010), 280–284.

[9] Cloudera. *http://www.cloudera.com/*. Accessed: 03-16-2011.

[10] Dean, J. and Ghemawat, S. 2004. MapReduce: Simplified Data Processing on Large Clusters. *OSDI* (2004), 137-150.

[11] Ernst, D.J. and Stevenson, D.E. 2008. Concurrent CS: preparing students for a multicore world. *Proceedings of the 13th annual conference on Innovation and technology in computer science education* (Madrid, Spain, 2008), 230-234.

[12] Gorton, I., Greenfield, P. et al. 2008. Data-Intensive Computing in the 21st Century. *IEEE Computer*. 41, 4 (Apr. 2008), 30-32.

[13] Kitchen, A.T., Schaller, N.C. et al. 1992. Game playing as a technique for teaching parallel computing concepts. *ACM SIGCSE Bulletin*. 24, (Sep. 1992), 35–38.

[14] NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing – Core Topics for Undergraduates. *http://www.cs.gsu.edu/~tcpp/curriculum/*. Accessed: 03-16-2011.

[15] Prasad, S.K., Chtchelkanova, A. et al. 2011. NSF/IEEE-TCPP curriculum initiative on parallel and distributed computing: core topics for undergraduates. *Proceedings of the 42nd ACM technical symposium on Computer science education* (New York, NY, USA, 2011), 617–618.

[16] Torbert, S., Vishkin, U. et al. 2010. Is teaching parallel algorithmic thinking to high school students possible?: one teacher's experience. *Proceedings of the 41st ACM technical symposium on Computer science education* (New York, NY, USA, 2010), 290–294.

[17] Wrinn, M. 2010. Suddenly, all computing is parallel: seizing opportunity amid the clamor. *Proceedings of the 41st ACM technical symposium on Computer science education* (Milwaukee, Wisconsin, USA, 2010), 560-560.