5-2014

Teaching HDFS/MapReduce Systems Concepts to Undergraduates

Linh B. Ngo
Clemson University, lngo@clemson.edu

Amy W. Apon
Clemson University, aapon@clemson.edu

Edward B. Duffy
Clemson University

Teaching HDFS/MapReduce Systems Concepts to Undergraduates

Linh B. Ngo
School of Computing
Clemson University
Clemson, South Carolina
lngo@clemson.edu

Edward B. Duffy
Clemson Computing and Information Technology
Clemson University
Clemson, South Carolina
eduffy@clemson.edu

Amy W. Apon
School of Computing
Clemson University
Clemson, South Carolina
aapon@clemson.edu

***Abstract*—This paper presents the development of a Hadoop MapReduce module that has been taught in a course in distributed computing to upper undergraduate computer science students at Clemson University. The paper describes our teaching experiences and the feedback from the students over several semesters that have helped to shape the course. We provide suggested best practices for lecture materials, the computing platform, and the teaching methods. In addition, the computing platform and teaching methods can be extended to accommodate emerging technologies and modules for related courses.**

***Keywords*-HDFS; MapReduce; distributed systems; parallel computing education**

I. Introduction

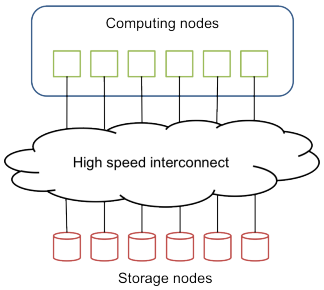
The open source Hadoop implementation of MapReduce [1], [2] has become an important technology to be included in the undergraduate computer science curriculum. Hadoop MapReduce may be taught in a variety of courses, including introductory parallel programming [3], natural language processing [4], data mining [5], and cloud computing [6].

In this paper, we describe our approach for teaching Hadoop MapReduce in a distributed computing course offered at the upper undergraduate level at Clemson University. In previous instances of the course we taught aspects of distributed computing that emphasized computationally intensive computing. MPI was used as the programming tool. By adding a Hadoop MapReduce module in the course's content, our learning objectives were to have the students:

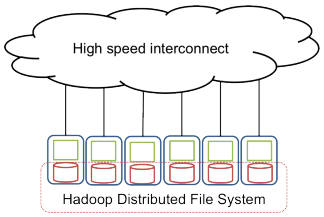
- Understand the challenge of data-intensive computing and why the typical computation/storage cluster architecture of supercomputing clusters sometimes fails to support data-intensive computing,
- Become familiar with the Hadoop Distributed File System (HDFS), a distributed system architecture that supports data locality for data-intensive computing, and understand the trade-offs of this architecture compared to the computation/storage architecture,
- Understand the underlying concepts of the MapReduce programming model and be able to design and implement MapReduce programs to analyze a large data set, and
- Understand the scalability and performance of MapReduce programs running on HDFS.

We also wanted students to have hands-on experience in observing and studying the behaviors of HDFS in detail, and to understand how these behaviors enable the performance and scalability of the MapReduce programming model and support of data-intensive computing through data locality. In pursuing these objectives, we were faced with three major challenges. The first challenge was to provide the students with a Hadoop computing platform powerful enough for students to observe scalability, and robust enough that errors from one student would not affect the work of the others. The second challenge was to balance the hours of lectures and labs within the limited total hours of the Hadoop MapReduce module. The third challenge was designing assignments that incorporate knowledge of both MapReduce programming and the HDFS system. Over the course of three consecutive semesters we refined our approach to address these challenges. For the computing platform, after investigating and experimenting with several approaches in building an educational Hadoop computing platform, we settled on a dynamic Hadoop environment running on the home institution's shared academic research supercomputer. The balance of lecture/lab hours and the design of lecture and assignment materials were determined through instructors' evaluation of students' progress as well as from students. The in-class experience and students' feedback over time demonstrated the maturity of our teaching methods and the computing platform for Hadoop environment. The platform can easily be extended to support emerging technologies and to create new modules in related courses.

The remainder of this paper is organized as follows. Section II provides background information on the course and describes in detail the development of the Hadoop MapReduce module. This section discusses how the technical and administrative challenges and the in-class reactions and students' feedback in successive offerings influence the development of the course materials. Section III describes the comprehensive lecture materials and tools for the MapReduce module in the most current version. Section IV concludes the paper and discusses future work.



(a) A typical HPC cluster with compute nodes separated from storage nodes



(b) A Hadoop cluster with storage on the compute nodes for data locality

Figure 1: Computing and storage placement design for a typical HPC cluster and Hadoop cluster.

II. Hadoop MapReduce Module Development

The target for the integration of Hadoop MapReduce is a course on the topic of distributed computing. The course is taught in a class format that is one hour and fifteen minutes of lecture twice a week. While this is a required junior-level course, most students in the class wait until the first or second semester of their senior year before taking the course. Typically, the enrollment of the class ranges between 35 and 40 students. The School of Computing at Clemson University maintains a computing cluster that serves as both distributed network lab resource and general purpose lab. The queuing model permits a student to use a compute node interactively while another student's job runs in the background. Clemson University also maintains a centralized supercomputer with more than 2000 compute nodes. The supercomputer uses the typical compute nodes/parallel storage setup, as shown in Figure 1(a). The students are allowed to use this supercomputer for their course work, but their jobs can be preempted from the system by higher priority research jobs asking for more computational resources. The supercomputer is upgraded regularly with a typical system utilization rate around 90%.

We first taught the Hadoop MapReduce module in Fall 2012. Since then, the Hadoop MapReduce lecture materials and tools have been taught in three standard full semesters (Fall 2012, Spring 2013, and Fall 2013) and one four-hour training session for undergraduate students who participated in the summer 2013 Research Experiences for Undergraduates (REU) program [7].

A. Version 1, Fall 2012

In Fall 2012, the first sixteen class lectures were spent on parallel programming concepts in MPI and shared memory programming while the last five lectures of the semester were spent on MapReduce. The lectures for Hadoop MapReduce module were arranged in the following order: one lecture on basic MapReduce concepts, one in-class lab, one lecture on HDFS, a second in-class lab, and one final lecture on advanced MapReduce optimization concepts. The three Hadoop MapReduce lectures follow the schedule from [8]. There were two assignments for the module. The first assignment reinforced concepts from the in-class lab, using a slight modification of the WordCount as a programming question (find the word with highest count in the complete Shakespeare collection). The second assignment asked the students to analyze the 171GB of a Google Data Center's system log [9] and find the computing job with largest number of task resubmissions.

Setting up a dedicated Hadoop platform for educational purposes is challenging [10]–[12], partly due to the differences in hardware architectures as shown in Figure 1. There are four typical approaches in setting up a computing platform for teaching Hadoop MapReduce:

- An interface for students to work on the MapReduce parallel programming model is provided that hides other elements of the Hadoop MapReduce ecosystem. An example is the WebMapReduce tool [13].
- The students execute their work on a dedicated shared Hadoop cluster with individual user accounts [14].
- The platform is a dedicated Linux cluster that supports virtualization. Hadoop/MapReduce is installed and configured on virtual machines, and students start up their individual virtual Hadoop clusters on a subset of nodes from the cluster [12], [15].
- Access to cloud resources are provided so that students can set up their own Hadoop clusters on the cloud [10].

We provided two Hadoop setups for the students. The first setup was a pseudo-distributed Hadoop setup inside a single virtual machine (VM). The second setup was a dedicated 8-node Hadoop cluster. Although Hadoop was not installed on the main supercomputer, the system administrators agreed to detach eight nodes from the supercomputer for us to set up this dedicated Hadoop cluster and to provide unlimited access to students enrolled in the class. Each node had dual 8-core CPUs, 64GB RAM, and 850GB HDD. The Google Trace Data was pre-loaded on the cluster.

Both of these setups turned out to be problematic for the class. In the first setup, since the VM was located on the supercomputer, the students needed to establish an SSH tunnel connection with the VM in order to access the virtual OS's GUI for Hadoop's web interfaces. A significant amount of time was spent by the students getting the VMs

up and running. The students' attempts to display GUI visualization of an entire operating system over their laptops' wireless connections only further complicated the problem. The second setup started out well during the in-class lab. Students were able to log in and execute the WordCount example on the dedicated Hadoop cluster. However, significant problems occurred near the due date for the second Hadoop MapReduce assignment. A large number of students waited until the last day before starting on the assignment. As a result, the Hadoop cluster began to slow down significantly. In addition, some of job submissions contained run time errors that created memory leaks on the Java heap memory and consequently crashed the task tracker and data node daemons. When the Hadoop cluster was restarted, it typically took at least fifteen minutes for all the Data Nodes to check for data integrity and report back to the Name Node. However, as soon as the cluster was up again, students continued to resubmit their jobs, hence creating additional under-replicated data blocks. Without prior experience in administrating Hadoop cluster, we ended up with a corrupted Hadoop cluster that stopped all the new jobs. By the end of the semester, only about one third of the students who had begun their work long before the deadline were able to complete the second assignment.

After the first semester, we concluded that maintaining a small centralized Hadoop cluster for a class of our size is not practical due to potential technical issues, many of which were exacerbated by the students. We recognize that the issues with the Hadoop virtual machine can be overcome using a more detailed step-by-step online guide. However, the configuration of the supercomputer prevented the virtual machines to fully utilize the physical network connection and limited the virtual network connection to roughly 1 MB/s. This eliminated the option of setting up virtual Hadoop clusters inside physical compute node in a manner similar to [12]. Financial investment in a cloud-based approach was not justified given the capabilities of the on-site supercomputer that was already available for students' use.

B. Version 2, Spring 2013

The following semester, we maintained the five-lecture plan from the previous semester but revised the setup of the Hadoop computing platforms. We redesigned the lectures, the assignments, and the Hadoop setup to highlight two separate aspects of the Hadoop ecosystem: the programming API libraries to support developing MapReduce programs and the middle infrastructure to support automated large scale data management and parallel execution of MapReduce programs. The MapReduce lecture focuses on the first aspect. The corresponding assignment only required the students to use Hadoop/MapReduce API libraries to develop and test MapReduce code on the standard Linux command line interface without using a supporting HDFS/MapReduce infrastructure. More specifically, this assignment required students to develop a number of MapReduce programs to answer analytical questions on the Airline Traffic database (11GB in size) [16]. With this assignment, we emphasized the fact that MapReduce was only a parallel programming technique that can take advantage of data-level parallelism through the underlying support of data locality due to Hadoop HDFS. The HDFS lecture, the second assignment, and the new Hadoop setup were designed to illustrate this support. Without a dedicated Hadoop cluster, we used the myHadoop scripts from the San Diego Supercomputing Center [17] with some modifications. This allowed students to have their own Hadoop clusters running on the supercomputer without any additional administrative support. In the second assignment, the students were asked to package their MapReduce programs into *jar* files and run them on their 8-node myHadoop clusters.

The separation of the programming interface and the supporting infrastructure concepts proved to be effective. The serial/stand-alone setup of MapReduce libraries allowed students to minimize configuration time and to have more time to understand the MapReduce programming concepts. The first in-class lab showed the students how to download and set up the class path for the Hadoop MapReduce libraries. They were also shown how to write the MapReduce code in Java, compile the code, and package the code into a jar for testing purposes. To help with code writing, students also learned how to use X Windows on different platforms (Windows, Linux, and Mac) to access the Eclipse IDE on the supercomputer. At a larger scale, the utilization of myHadoop enabled the students to have access to their own Hadoop cluster on the supercomputer. The second in-class lab showed the students how to modify the myHadoop scripts so that the paths worked correctly with individual students' accounts. In this semester, all of the students completed both MapReduce assignments on time.

There were a few glitches that remained. The Eclipse interface for Java MapReduce development suffered from connection problems due to high bandwidth requirements of visualization over the wireless network. A number of students did not like the burden of setting up a Java project inside Eclipse and preferred the straightforward command line editor for writing Java code. The utilization of myHadoop required intermediate Linux knowledge as well as knowledge about the layout of the supercomputer's storage in order to set up the HDFS data and MapReduce intermediate data directories properly. Among the errors that happened during the first time setting up the Hadoop cluster, the most common ones were incorrect paths to the Hadoop MapReduce installation directory, data nodes' local directory, and log directory. Another common issue was with ghost Hadoop daemons. If students exited from their reserved nodes without explicitly stopping Hadoop, the Hadoop daemons became orphaned while still bound to

the ports for Hadoop communication. If these nodes were assigned to other students immediately afterward (before the scheduler could run a clean-up script), myHadoop scripts would not be able to start a new Hadoop cluster due to required ports being blocked off. If the orphaned daemons belonged to the same student, they could be terminated individually before a manual startup of the Hadoop cluster. Otherwise, the student would have to wait 15 minutes for the scheduler to clean up these daemons.

In this semester, we also began to take feedback from the students through surveys to help to refine the Hadoop MapReduce module.

C. Version 3, Summer 2013

The distributed computing class was not offered during summer 2013. However, we packaged the module into a four-hour unit technical training session for undergraduate students participating in the Research Experience for Undergraduate program. Since less time was available, it was important that the existing technical issues from the previous semester were minimized. Within the allotted time, the two lectures on MapReduce and HDFS were compressed into two 45-minute segments. The remaining time was spent on teaching the students how to write Map Reduce programs and how to set up their Hadoop clusters on the supercomputer. All MapReduce programming steps were done on the command line terminal. Detailed steps provided in an accompanying tutorial. The myHadoop scripts were provided so that very few modifications from the students were needed. For the lab, the boot camp session covered two examples. The first was the Hadoop Word Count example, and the second was the calculation of average delay time for each airline from the Airline Traffic data.

The feedback comments were very useful. The students expressed positive sentiments about examples and the usefulness of a detailed tutorial handout while emphasizing the need for an easier Hadoop setup, more details included on the handout, and a slower lecturing pace. It should be noted that the students participating in the REU program were top quality students with recommendations from their home institutions' faculty. With the information from this feedback, we doubled the number of lab hours in the full semester module and also incorporated more details in the lecture notes and the lab tutorials.

D. Version 4, Fall 2013

By Fall 2013, the environments and tools for the Hadoop module had reached a mature state with minimal effort needed from students to get their Hadoop cluster up and running. The students were required to follow an exact directory structure when saving Hadoop MapReduce libraries and the myHadoop scripts and were provided with scripts to automatically compile and package their MapReduce codes into jar files. The number of class hours spent on Hadoop MapReduce was also increased from five to seven lectures. Based on the feedback from the previous offerings of the course, we maintained two lectures for MapReduce and HDFS but doubled the number of hands-on labs. We also spent one lecture introducing HBase/Hive to the students to provide a more comprehensive view of the Hadoop ecosystem.

Table I: Level of Proficiency (0 to 10 with 10 being highest)

Topic	Before	After
Java	6.6±1.2	7.3±1.1
Linux	5.86±1.7	7.1±1.7
Networking	4.38±1.6	6.29±1.5
Hadoop MapReduce	0.03±0.2	4.53±1.16

In this semester we executed a survey evaluation of the course module. The survey provided the students with a number of detailed questions on the module. The survey first asked the students about their levels of proficiency on Java, Linux, networking concepts, and Hadoop MapReduce, before and after the Hadoop MapReduce module. Next, the survey asked the students to estimate the time they spent completing each assignment and setting up the Hadoop cluster and to evaluate the usefulness of the lecture slides and tutorials. The survey also asked the students about whether they thought the coverage was enough and at what level of undergraduate study Hadoop MapReduce should be introduced. Finally, the survey had a section for students' comments and feedback.

Out of 39 students taking the class, 29 filled out and returned the survey forms. On proficiency level, besides the obvious improvements on Hadoop MapReduce, the students also recognized improvements in their Java, Linux, and networking proficiency, as shown in Table I. This was not surprising because a comprehensive understanding on the Hadoop ecosystem requires one to understand the programming aspects of MapReduce, the architectural aspects of how HDFS' data blocks are set up and located on the Linux file system for data locality, and the networking aspects of how mappers and reducers communicate to process data blocks and shuffle intermediate data.

We asked the students to report the time it took to complete the assignments so as to gauge whether the workloads of the assignments are appropriate. The results are summarized in Table II. On average, the students spent close to four hours on the first assignment. The second assignment, despite being twice as long as the first assignment, also only took close to four hours to complete on average. These times were acceptable given that these were two-week and three-week long assignments, respectively. The time it took to set up the Hadoop cluster was also within the range of two hours or less. In fact, the majority of the students were able to set up their Hadoop cluster within the HDFS in-class lab.

We were satisfied with the level of usefulness of the lectures, in-class labs, and tutorials. It should be noted that

Table II: Time to Complete (1: less than 30 minutes, 2: 30 minutes to 2 hours, 3: 2 hours to 4 hours, 4: more than 4 hours).

Activity	Time Taken
First Assignment	3.5±0.7
Second Assignment	3.1±0.9
Set up Hadoop cluster	2.5±1.1

Table III: Helpfulness of Lectures and Tutorials (1: not useful, 2: somewhat useful, 3: useful, 4: very useful).

Teaching Materials	Usefulness
Lecture	3±0.9
In-class lab	3.6±0.7
Hadoop cluster tutorial	2.9±0.82

the students favored the in-class labs over the lectures, as indicated in Table III. The survey shows that the current amount of materials and the number of lectures spent on Hadoop MapReduce were appropriate. One of the important statistics from the survey is the level of undergraduate study for which the students think the Hadoop MapReduce module is appropriate. Looking at Table IV, we observe that while the majority of the students chose junior year or higher, more than 25% of the responses still thought that this module could be taught at sophomore or freshman level. Given that most of the requests for assistance in this module come from working with HDFS and the supercomputer, we hypothesize that while the MapReduce programming aspects could be learned at a lower undergraduate level [13], materials on HDFS require some fundamental understanding of the system-level aspects of cluster computing. Besides courses on distributed computing, this knowledge is typically picked up in operating systems, networking, or network programming courses, which are taught at the late sophomore or early junior level. We believe that the junior year or higher is the best time to teach Hadoop MapReduce.

On the written feedback, all students expressed satisfaction about how the Hadoop MapReduce module was designed and taught. Comments to improve the module included requests for even more detailed tutorials and guidance along with explanations on the purpose of each command and more in-class labs for hand-on experience and discussion. Given that the students also thought that the amount of time on the module was appropriate, this suggests that they want to be able to better understand what has been taught, and not to be taught more materials.

Table IV: Lowest level of CS course that Hadoop MapReduce should be introduced at Clemson University.

Year to teach Hadoop/MapReduce	Survey Counts
Senior	7
Junior	14
Sophomore	6
Freshman	2

III. Teaching Materials

The teaching materials for the Hadoop MapReduce module are grouped into four categories: lecture notes and example codes, assignments, data sources, and tools to set up Hadoop platforms [18]. The lectures and assignments on MapReduce and HDFS with the guiding theme of data intensive computing familiarize the students with concepts of knowledge units in the area of Parallel and Distributed Computing as defined by the ACM/IEEE curricula [19], as shown in Table V.

A. Lecture Notes and Example Codes

For the Hadoop MapReduce module, we have lectures on the three topics: MapReduce, HDFS, and HBase. There exists a large number of reference lectures from academic and industry sources for these topics. We follow the set of lecture notes from [4]. However, we enhance the MapReduce/HDFS lectures with additional visualizations, for example, Figure 2, that demonstrate the HDFS-MapReduce integration and how this integration lies on top of the physical computing cluster and the Linux file system. The first coding example accompanying the MapReduce lecture is the standard WordCount example which illustrates the basic concepts of mapping and reducing. This is followed by another WordCount example that uses the reducer as a combiner. The students observe the tradeoff between increased map task run time (observed through Hadoop's JobTracker's web interface) versus reduced network traffic (observed through final MapReduce job report). In the MapReduce lab, the students goes through another set of MapReduce examples analyzing the Airline Traffic data. The analysis is to find out the average delay time for each individual airline on the entire data set. Three examples of code are provided which implement different algorithmic choices described in [20]. These examples emphasize the usage of MapReduce's combiner, the customized MapReduce's Value classes, and the trade-off in memory and network traffic due to different implementations of the combiner. The first example code is a standard MapReduce program whose mappers emit the airline code and the delay time as a key-value pair and reducers calculate the average delay time for all the values of the same key. The second example code implements a combiner, which also requires the implementation of a customized Hadoop Value class. The third example code utilizes global memory on each node to implement a combining mechanism without implementing a combiner class.

B. Assignments

Assuming that the students are familiar with the basics of writing code in a Linux environment, the process of compiling and executing a simple MapReduce program is automated through a scripting template for MapReduce without HDFS. The first assignment requires the student to recall the lab materials by implementing a number of

Table V: Parallel and Distributed Computing Learning Outcomes through Hadoop MapReduce lectures and assignments.

Level	Knowledge Area	Knowledge Unit	Learning Outcome
Familiarity	Parallel & Distributed Computing	Parallelism Fundamentals	Distinguishing using computational resources for a faster answer from managing efficient access to a shared resource
Familiarity	Parallel & Distributed Computing	Parallel Architecture	Describe the key performance challenges in different memory and distributed system topologies
Usage	Parallel & Distributed Computing	Parallel Performance	Explain performance impacts of data locality
Familiarity	Information Management	Distributed Databases	Explain the techniques used for data fragmentation, replication, and allocation during the distributed database design process
Assessment	Parallel & Distributed Computing	Parallel Algorithms, Analysis, and Programming	Decompose a problem via map and reduce operations
Usage	Parallel & Distributed Computing	Parallel Performance	Observe how data distribution/layout can affect an algorithm's communication costs

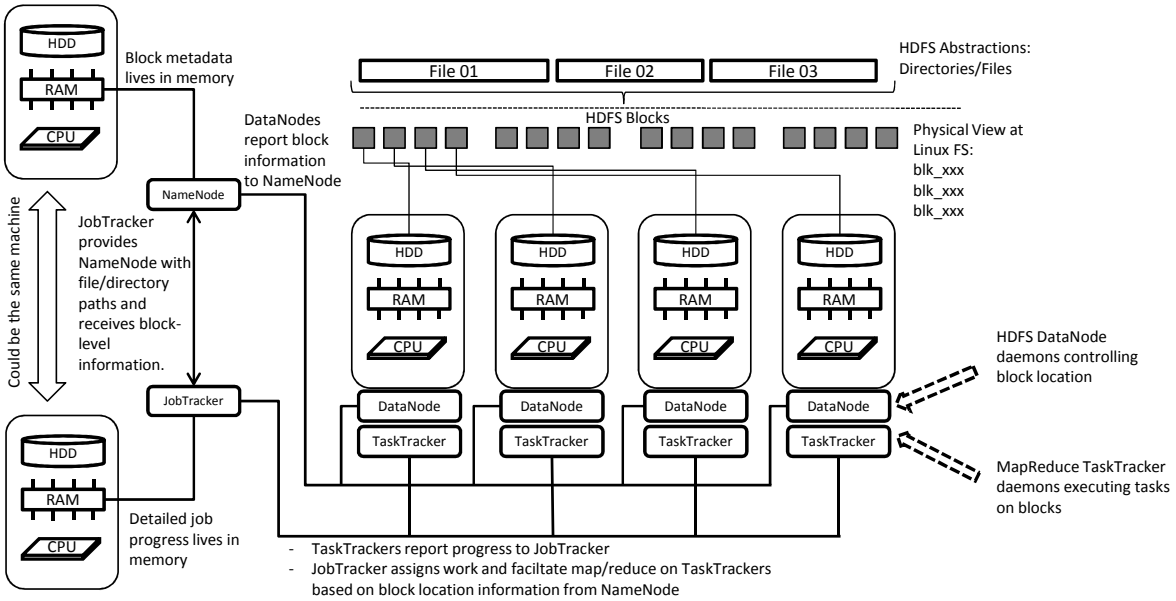


Figure 2: Illustration to emphasize the relationship between components of the base Hadoop ecosystem and the underlying hardware and the Linux file system

descriptive statistics calculations on the rating of individual movie genres from the movie rating database [21]. While the calculations are relatively straightforward, the matching of the ratings for individual movies into the relevant genres that the movies belong to requires the map tasks to interact with an additional data file. As the students later learned, the optimized implementation of this external access with respect to the map tasks can make the program run one order of magnitude faster. The easiest, but inefficient approach, is to read the additional file from inside each mapper. An alternative and more efficient approach is to implement a Java object that reads the additional file once and stores the content in memory. The mappers can then access this object as needed. The second part of the assignment requires the students to implement a single MapReduce program to identify the user that provides the most ratings and that user's favorite movie genre. In order to answer this question, the students need to implement a customized Hadoop output value class, as the information needed in the reduce step requires several values for each key. The first assignment has the students run their final jars using only serial Java commands without any HDFS support.

The second assignment familiarizes student with the underlying HDFS and how it supports the MapReduce programming model. The first part of this assignment takes the jar files from the first assignment and reruns them on the data on HDFS. The goal of this part is to demonstrate the ease in which Hadoop MapReduce can immediately speed up the application without having to worry about parallel workload division, process' ranks, etc. This part also requires students to execute and record the output of a number of Hadoop shell commands to observe how HDFS transforms, stores, replicates, and abstracts the actual data to support data intensive computing. The second part of this assignment

asks the students to analyze the Yahoo song database (10GB) [22] and identify the album that has the highest average rating using MapReduce and HDFS. Again, this requires the students to access the list of songs in each album to support the main rating data files.

In both assignments, while the students can follow different approaches in designing the MapReduce programs in order to find the answers to the problems, the differences between runtimes of efficient and inefficient approaches can be observed. Having individual mappers reading from the same additional data file increases runtimes to several hours, and implementing a customized Java object to preprocess the additional data can reduce the runtimes to minutes.

C. Data

To make the assignments realistic, the data set needs to have some aspects that are "big" as compared to a typical data table. Utilizing full text data such as Wikipedia is a convenient way to have a big data set. However, without getting into complex text analysis methods, full text analysis is limited to counting words and recognizing phrases. The Google Trace data [9] is large enough and contains complex and interesting information to be analyzed. However, as the size of the Google Trace data is relatively large (171GB), it can take over an hour for students to stage the data into the temporary Hadoop cluster. This trace data is more appropriate for semester projects rather than weekly assignments. The Airline Traffic data [16] has a reasonable size (12GB) with a straightforward single-table data schematic. We utilize the Airline Traffic data in our examples so that we can spend more time on the programming techniques and less time for data movement.

The first assignment uses the Movie Rating dataset [21], which is 250MB in size and contains 10 million ratings for 10,000 movies by 72,000 users. This provides the students with a significant number of records on a more complex data structure that can still be processed serially within a reasonable amount of time. The results from the students' assignments show that the best implementation of the first assignment can run as fast as several minutes, while the worst implementation takes a little over half an hour to run. The second assignment uses the Yahoo Music Rating dataset [22]. This dataset is similar in size to the Airline Traffic data with a complex set of tables that is similar to the Movie Rating dataset. The data is large enough to be impractical on a serial execution yet small enough so that it takes less than five minutes to load the data into the HDFS file system.

D. Tools for setting up the Hadoop platform

To ensure consistent configurations for the Hadoop platforms, regardless of whether they are set up on the supercomputer or on personal machines, we provided students with a package of Java 1.7.0_25 and Apache Hadoop 1.2.1. The package is decompressed into students' Linux home directories. The students only needed to modify their *.bashrc* to source a script that automatically identifies the paths for JAVA_HOME and HADOOP_HOME. Another script is provided with commands to clean up temporary output directories, to compile and compress MapReduce source codes into *jar* files, and to execute these *jar* files in serial mode on the Linux file system without HDFS support.

For the dynamic Hadoop platform on the supercomputer with HDFS, the myHadoop scripts were modified so that the only changes the students needed to make were changes to the Hadoop platform's physical configurations (number of nodes, size of RAM, and expected run time) on the scheduler's submission script. The submission script also includes Hadoop commands to automatically create HDFS directories, load data from the Linux file system, check HDFS' health status, execute an example MapReduce job, and export output data back to students' home directories on the supercomputer's Linux file system. With a batch submission, the scheduler will record all outputs from these commands, so that the students can review and analyze the performance of their Hadoop platforms. As the large scale parallel storage of Clemson's supercomputer is not configured with file-locking support, we cannot use the persistent storage option of myHadoop, and all Hadoop data storage must reside on the local hard drive of the scheduled compute nodes. In order to analyze MapReduce output file on HDFS in a batch execution, the students must export the desired HDFS data to the Linux file system by placing either *hadoop fs -copyToLocal* or *hadoop fs -get* after the command to execute the MapReduce job. The students can also insert a *sleep* command into the submission script and turn the dynamic Hadoop platform into an interactive platform for the duration of the *sleep* command.

IV. Conclusion and Future Work

We have developed a Hadoop MapReduce module and taught it extensively in a course on Distributed Computing. The complex architecture of the Hadoop ecosystem and its departure from the typical high performance computing cluster design with a parallel storage file system have pushed us to experiment with a number of different approaches to teach the materials as well as to set up a Hadoop environment on which students can experiment and learn. The feedback from the students indicates a high level of satisfaction with the module, and the modifications on the myHadoop scripts allow instructors to take advantage of a centralized shared computing resource to allow students to set up individual Hadoop clusters.

The Hadoop MapReduce ecosystem is a prime example of a significant area of interest which is "continually evolving as topics mature and even newer topics appear on the scene" [23]. This is made evident by recent developments of the Hadoop ecosystem that have moved Hadoop beyond

MapReduce's limitations in order to support additional capabilities such as cluster resource manager [24], in-memory distributed computing [25], interactive programming [26], and distributed data store [27]. While not all of these concepts are relevant to the subject of distributed computing, they are certainly desirable skills for an employee or a graduate student to have and can be incorporated into other courses. Future work includes developing the myHadoop scripts to continue to support these new components of the Hadoop ecosystem. Efforts should also be spent on matching the materials used for teaching elements of the Hadoop ecosystem to the knowledge units within the computer science curriculum so that the students are taught the fundamental concepts in computer science but are also familiar with the frontier of computing technologies.

Teaching materials from all versions of the course can be accessed at [18].

V. Acknowledgments

We are grateful for the support from NSF MRI Award #1228312 and NSF CI-SEEDS Award #1212680. We would like to acknowledge the assistance from Randy Martin and Corey Ferrier, Clemson Computing and Information Technology, in setting up and maintaining the dedicated Hadoop cluster, and other assistance from Michael E. Payne, Jason Anderson, and Pengfei Xuan.

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *ACM SIGOPS Operating Systems Review*, 2003.

[2] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communication of the ACM*, vol. 51, no. 1, Jan. 2008.

[3] M. Johnson, R. H. Liao, A. Rasmussen, R. Sridharan, D. Garcia, and B. K. Harvey, "Infusing parallelism into introductory computer science curriculum using MapReduce," EECS Department, University of California, Berkeley, Tech. Rep., 2008.

[4] J. Lin, "Exploring large-Data issues in the curriculum: A case study with MapReduce," in *Workshop on Issues in Teaching Computational Linguistics*, 2003.

[5] Stanford University, "CS246: Mining massive data sets," http://www.stanford.edu/class/cs246/info.html, 2014.

[6] University of Pennsylvania, "NETS212: Scalable and cloud computing," http://www.cis.upenn.edu/ nets212/, 2013.

[7] Clemson University, "Data-Intensive Computing Research Experiences for Undergraduates," 2013. [Online]. Available: http://people.cs.clemson.edu/ bcdean/reu/

[8] University of Maryland, "INFM71G/CMSC838G: Data-intensive information processing applications," http://www.umiacs.umd.edu/j̃immylin/cloud-2010-Spring/, 2010.

[9] J. Wilkes, "More Google cluster data," googleresearch.blogspot.com/2011/11/more-google-cluster-data.html, 2011.

[10] A. Rabkin, C. Reiss, R. Katz, and D. Patterson, "Experiences teaching MapReduce in the cloud," in *Proceedings of the ACM SIGCSE*, 2012.

[11] W. Moody, L. Ngo, E. Duffy, and A. Apon, "JUMMP: Job uninterrupted maneuverable mapreduce platform," in *Proceedings of the 2013 IEEE Cluster*, 2013.

[12] R. Brown, "Hadoop at home: Large-scale computing at a small college," in *Proceedings of the ACM SIGCSE*, 2009.

[13] P. Garrity, T. Yates, R. Brown, and E. Shoop, "WebMapReduce: An accessible and adaptable tool for teaching MapReduce Computing," in *Proceedings of the ACM SIGCSE*, 2011.

[14] A. Kimball, S. Michels-Slettet, and C. Bisciglia, "Cluster computing for web-scale data processing," in *Proceedings of the ACM SIGCSE*, 2008.

[15] J. R. Albrecht, "Bringing big systems to small schools: Distributed systems for undergraduates," in *Proceedings of the ACM SIGCSE*, 2009.

[16] American Statistical Association, "Airline on-time performance," http://stat-computing.org/dataexpo/2009/, 2009.

[17] S. Krishnan, M. Tatineni, and C. Baru, "myHadoop - Hadoop-on-Demand on traditional HPC resources," San Diego Supercomputing Center, Tech. Rep., 2011.

[18] Clemson University, "CPSC3620: Parallel and Cluster Computing," https://sites.google.com/site/clemsondagoba/teaching, 2014.

[19] ACM/IEEE Joint Task Force on Computing Curricula, ai.stanford.edu/users/sahami/CS2013/, 2013.

[20] J. Lin, "Monoidify! Monoids as a design principle for efficient MapReduce algorithms," http://arxiv.org/pdf/1304.7544v1.pdf, 2013.

[21] GroupLens, http://grouplens.org/datasets/movielens/, 2013.

[22] Yahoo! Labs, "Yahoo! Music user ratings," https://webscope.sandbox.yahoo.com/index.php, 2013.

[23] S. K. Prasad *et al.*, "NSF/IEEE-TCPP Curriculum initiative on parallel and distributed computing - Core topics for undergraduates," http://www.cs.gsu.edu/ tcpp/curriculum/sites/default/files/NSF-TCPP-curriculum-version1.pdf, 2012.

[24] Apache Hadoop 2.0: YARN (Yet Another Resource Negotiator), hadoop.apache.org/docs/current2/hadoop-yarn, 2013.

[25] Apache Spark, "Lightning-fast cluster computing," http://spark.incubator.apache.org, 2013.

[26] Apache Tez, "A framework for near real-time big data processing," http://hortonworks.com/hadoop/tez, 2013.

[27] Apache HBase, http://hbase.apache.org, 2013.