Data Analytics with Hadoop for Juniors

Lakshmi Prayaga* and Keerthi Devulapalli
University of West Florida
Pensacola, USA
*Email: lprayaga [AT] uwf.edu

Abstract—**The huge amount and types of data being generated every day is creating a dearth for data analysts and professionals in computational sciences. In response to this need there have been several attempts and approaches by academia including the use of popular tools and technologies such as 3D gaming, 3D printing, Robotics etc. to entice new recruits to computational sciences. This paper describes an effort on introducing Hadoop and data science to enthuse high school students to pursue computational sciences in higher education. We describe the method used and results obtained in this paper.**

Keywords- Computer and Information science education, Emerging technologies, Information systems education, Parallel processing

I. INTRODUCTION

The need for data analysts and computational scientists is increasing by the day to process the quintillion bytes [1] of data now known as Big Data [2]. This demand has given rise to the field of data science which blends the disciplines of statistics and computing. Though higher educational institutions have responded to this need by offering new degree programs in data science, few institutions in the K12 community have been introduced to this new field.

In an attempt to address the need of introducing data science to the k12 community, we present the design and implementation of a data science project using Big Data and Hadoop to juniors in high school. Results suggest that students were very engaged in this activity and responded with increased interest in pursuing fields such as data science as a career choice thus increasing the skilled workforce for the 21st century.

II. LITERATURE REVIEW

Given the need for students to develop computational thinking skills required to process the huge amount of data several researchers are proposing that computational thinking must be a part of the K12 curriculum ([3], [4], [5], [6]). "Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability" [7]. Research suggests that students demonstrate their attitudes towards their fields of study and career choices during their middle school and high school years ([8], [9], [10]). Researchers have therefore tried to attract students in this age group and shift their choice of study towards computational science disciplines using multiple strategies such as game programming, robotics, visual programming, multimedia etc. ([11], [12]). Current researchers are also exploring introducing data science as another channel to enthuse students to pursue computational sciences as their choice of post-secondary education [13]. In this paper we present an experimental project to introduce Hadoop [14] to juniors by demonstrating its power to efficiently and effectively process Big Data and extract meaning from this data. While recent studies [13] describe an effort to design CS curriculum for middle school students that include topics related to Big Data, we present the design and implementation of a complete project in Hadoop and Big Data that junior students can relate to and appreciate. The experiment suggests that the word count problem chosen for this project can indeed work as a "Hello world" to introduce the world of Hadoop to both the novices with an introductory programming background or experienced programmers wanting to venture into the world of Hadoop.

III. BACKGROUND

A. *Hadoop's MapReduce Architecture*

Hadoop is a Big Data technology which offers a distributed file system for storing Big Data called Hadoop's distributed File System (HDFS) and an execution framework called MapReduce for processing Big Data [15]. HDFS stores the data to be processed by MapReduce programs. It divides the data into data blocks and stores each data block on a different node of the Hadoop cluster. A MapReduce program has the following different phases as shown in Fig. 1:

1. Map phase
2. Shuffle and Sort phase
3. Reduce phase

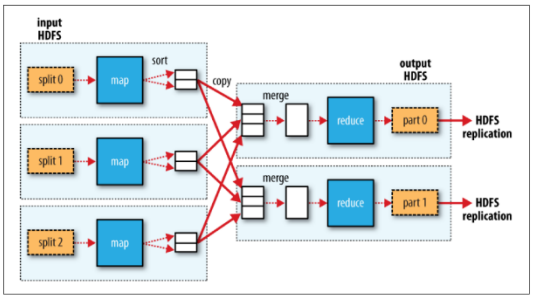

Figure 1 MapReduce flow with multiple reduce tasks [14]

In the map phase, the input data are read and processed. Hadoop's MapReduce execution framework works on the principle of "taking code to data", hence the operations to be performed on the data in the map phase are sent to the cluster nodes where the data block resides. The operations are performed on each data block in parallel and the results of the operations are collected and sent to the next phase of shuffle and sort. Thus MapReduce programs follow the divide-and-conquer strategy. The operations that have to be performed on one data block should be independent of the operations performed on another data block, such that the operations on different data blocks can be executed in parallel on different cluster nodes as different map tasks.

The intermediate results from the map phase are grouped together into different groups so that each group of intermediate data can be given to a different reduce task. In the reduce phase, the different groups of intermediate data are given to different reduce tasks. Reduce task usually perform aggregate or group-level operations on the grouped data and combine them to form the final result. Similar to map tasks, all operations performed by one reduce task on a group of data must be independent of the operations performed on another group of data. Hence, the reduce tasks can also be executed in parallel on different nodes of the cluster just like the map tasks. As map and reduce tasks run parallel on different cluster nodes, the overall execution time of the program is less than the time taken by the sequential program.

IV. DESIGN

Our primary focus in this project was to introduce students to the whole idea of data explosion and the necessity to be able to find effective and efficient tools and strategies to process, analyze and visualize data in a meaningful manner. The goals of this project were to

1. Prompt students to observe and find that the data that is out there is unstructured and is available in multiple formats.
2. Demonstrate the power of Hadoop's MapReduce programming model which uses distributed and parallel processing techniques to speed up the execution of a data intensive computer program.

The "Hello world program" is the standard used to introduce the basic elements of any programming language. This program generally includes an Input and output aspect requiring the programmer to tell the computer to take in an instruction and execute it. In general the programmer simply instructs the computer to print the word "Hello World" and the program displays these words on the computer screen. This technically demonstrates the basics of a computer program i.e. a computer program can send a set of instructions to the processor and the processor then processes the information and returns a result. For example, Hello World program of Java programming language looks as follows:

```
package default;

public class helloWorld {
        public static void main(String args[]){
                System.out.println("Hello World!");
        }
}
```

In this case, the programmer simply types the line "system.out.println("Hello world")" and the system displays the output on the screen. The computational problem is restricted to just taking one instruction and displaying the result as an output.

However, finding an example to introduce Hadoop's MapReduce programming paradigm is not that simple. The reason for this is that primarily the Hadoop's MapReduce framework is useful when the problem is complex and data intensive and typically involves independent operations that can be performed in parallel, grouping certain data items and performing aggregations on data. The problem under consideration for using MapReduce paradigm should typically exhibit the need for

1. Parallel processing of data
2. Grouping of data
3. Reducing or aggregating the data

To accommodate these requirements the researchers in this experiment used the word count problem as an introduction to MapReduce programming and it serves as the equivalent of the Hello world program in a standard programming language. The word count problem is stated as finding the number of words present in a given set of documents that have a given length. It can be observed that the word count problem accommodates all the three aspects- independent operations that can be processed in parallel, grouping and aggregations - which are the fundamental building blocks of a Hadoop solution.

To demonstrate the MapReduce programming model, the word count problem was first manually implemented in MapReduce model as a hands-on activity. The word count problem was later implemented as a sequential java program and as a Hadoop's MapReduce program. The execution time of the sequential program and the MapReduce program were compared.

V. WORD COUNT PROBLEM

The word count problem described below was used as the content to introduce the MapReduce technique from the Hadoop framework.

Word Count Problem Statement: A document with four paragraphs was chosen as the context for this Hadoop based experiment. The problem statement was to

1. Count the length of each word in the document

2. Display a list consisting of the length of the word and the number of instances a word with that specific length occurs in the document.

From the problem statement, it can be noted that for the word count problem, counting the words with different lengths in one paragraph is independent from counting the words with different lengths in another paragraph. Hence, the sample document can be divided into four different partitions, each containing one paragraph. This division of input data is similar to the data blocks formed by HDFS. The paragraphs can be processed in parallel, each independent of other paragraphs. The operation of counting the words with different lengths in one paragraph is suitable to be performed by map tasks in the map phase.

Also, it can observed that, the partial word counts obtained from different paragraphs can be aggregated together to obtain the total word counts for different word lengths. This aggregation can also be done independently for each word length. For example, the aggregation of the number of words for a word length of four can be performed independent of the aggregation for a word length of six, or fifteen, etc. This aggregation can be performed by the reduce tasks in the reduce phase. For reduce tasks to perform the aggregate operation, the partial counts from the map phase must be grouped based on the word length. This grouping is performed by the shuffle and sort phase. Hence, the word count problem is ideal to be solved by a MapReduce program.

VI. IMPLEMENTATION

A. Using Role-play for manual implementation

A total of eighteen high school (juniors) students from the Information technology elective participated in this experiment. The male to female ratio was close, 55% male students and 45% female students. The students had taken an introduction to programming course in java prior to participating in this experimental project. To understand the different phases of a MapReduce program, the word count MapReduce program was implemented manually as a role-play activity, where the students performed the map task, shuffle task and the reduce task manually.

Eight students were assigned the role of mappers. They were given one paragraph each from the sample document. Their task was to parse the paragraph word by word, find the length of each word and create a list of word length and count as their output. Hence, each mapper read the assigned paragraph and wrote the list of word lengths and counts on a paper. Five other students were assigned the role of groupers. Each of the grouper took a list from one of the mapper and grouped the counts based on the word lengths. The grouped word counts were written on the classroom board. The last five students were assigned the role of reducers. Each reducer picked one word length group counts at a time and calculated the total count for the given word length.

From this activity, the students observed that all mappers could parse their respective paragraphs in parallel and all reducers could calculate the total counts in parallel. The groupers could group the intermediate data from the mappers as they completed. The idea that computational tasks that are independent of each could be performed in parallel was the focus of this activity. From this observation, the students concluded that the word counts were obtained faster in the MapReduce model than in the sequential model because the mappers and reducers performed their tasks in parallel. They also noted that groupers could not start until at least one mapper completed and reducers could not start until all groupers completed their tasks. Hence, there are some dependencies between the mappers and reducers, but as such all mappers can run in parallel and all reducers can run in all parallel.

B. Programming Implementation

To compare the actual performance of a sequential program with a MapReduce program, the students implemented the word count problem's solution both as a sequential java program and as a MapReduce program.

In the sequential program, the input text file is opened and read line by line. An array of counts is maintained to store the word counts of different word lengths. In the counts array, the index represents the word length and the content represents the word count of that particular word length. The counts array is initialized to zero. Each line is split to obtain the individual words. For each word, the word length is obtained and then used as index into the counts array. The corresponding element in the counts array is incremented. After processing the complete file, the word counts are present in the counts array.

The algorithm for the sequential version is as follows:

```
Input: file, text file
Output: counts, array of word counts
counts[] = 0
Open file
while !end of file
begin
        line = readLine()
        words[] = line.split()
        for each word in words
                counts[word.length]++
end
```

It can be observed from the sequential program, that it requires two nested loops to determine the word counts. The MapReduce program splits the two nested loops across the map and reduce phases and introduces parallelism in the loop execution.

In Hadoop, the input file is split into multiple data blocks and each data block is stored on a different cluster node. Hadoop's MapReduce framework runs a map task on each cluster node that store the data blocks of the input file. For each map task, the Hadoop's MapReduce framework reads the

data block line by line and sends the line as input value to the map method. The map method implements the map phase algorithm and outputs the intermediate key-value pair. Hadoop's framework performs the role of the grouper in the shuffle and sort phase. It groups all intermediate key-value pairs returned as output by all map tasks based on the key. The grouped key and list of values are sent to the reduce method as input. The reduce method implements the reduce phase algorithm and outputs the final resulting key-value pair.

In the MapReduce version there are two algorithms, one for the map phase and other for the reduce phase. In the map phase, one line of the input text file is received as input value. As in the sequential program, the input line is split into words. For each word, the word length is obtained and outputted as the intermediate key along with 1 as value. The intermediate value '1' represents that 1 word of given length is found. The algorithm for the map phase is as follows:

```
map
input: value, a line of text from the input file
output: key, word length; value, the value 1
words[] = value.split()
for each word in words
        emit(word.length, 1)
```

As it can be seen from the map phase algorithm, the MapReduce program splits the outer loop of the sequential program across multiple map tasks and executes the loop in parallel. Though it retains the inner for loop, it does not perform the counting. The counting is performed by the reduce phase algorithm.

In the reduce phase, one word length and its partial counts are received as input key and list of values respectively. The list of partial counts are combined to obtain the final word count for the given word length and outputted as the final key-value pair. The reduce phase algorithm is as follows:

```
reduce
input: key, word length; list of values, partial word counts
output: key, word length; value, final word count
count = 0;
for each value in values
        count += value
emit(key, count)
```

From the reduce phase algorithm it can be observed, that the inner loop for counting the words in the sequential program is split across multiple reduce tasks which are executed in parallel. Each reduce task calculates the final word count for a given word length. Thus the MapReduce program has parallelized the two nested loops in the sequential program. It can also be noted that the counts array in the sequential program is indirectly maintained by the Hadoop framework via the intermediate and final key-value pairs.

VII. Results And Observation

The primary learning outcomes planned for this activity were the following:

a. Observe and describe that there is a huge amount of data being generated and
b. Recognize that the data available is in multiple formats
c. Parallel processing is a critical component of computation which increases efficiency in the computational process

It was found that the sequential java program ran for four minutes, but the MapReduce program ran less than a minute. This time difference helped students realize that distributed and parallel processing like Hadoop's MapReduce runs faster than sequential processing. The students observed that the two nested loops were the reason for the slow execution of the sequential program. As the MapReduce program splits the two nested loops and executes them in parallel, it runs faster than the sequential program.

As the students implemented both sequential java program and MapReduce, they observed that the MapReduce program needed lesser programming effort than the sequential program. The Hadoop framework not only co-ordinates the map and reduce tasks but it also takes the responsibility of file I/O for the input and output files.

Goals one and two were measured by the responses and discussions to following questions

a. Describe scenarios where you think huge data are being generated?
b. Describe scenarios where you think the MapReduce technique from Hadoop framework can be applied?

As a hint the researchers talked about face book, the massive amount of data generated by social media and how lists of "friends of friends" are constructed. This spurred a huge response and students in turn gave the following examples. These examples also met the critera required for a potential Hadoop problem and solution. The examples provided were

1. Dating search engines
2. Amazon product search
3. YouTube movies and other searches
4. Shopping sites

Students pointed out that in all these cases

a. There were huge data sets of multiple formats available from which data can/need to be extracted and aggregated (addressing the goal of recognizing that there is a huge amount of data being generated),
b. The datasets were suitable for using a MapReduce technique to speed up the processing due to the fact that the data set could be broken up into smaller segments and the same set of operations can be performed on each individual segment in a parallel fashion which speeds up the data

processing (addressing the goal that parallel processing increases efficiency of performance).

To address learning outcome c, (recognizing the need for and efficiency of parallel processing) the sequential java program and the Hadoop's MapReduce program implemented by the students were executed. Both the sequential program and the MapReduce program were run on the university XXX's Hadoop cluster. The Hadoop cluster consists of 3 management nodes and 6 data nodes. The cluster runs Hadoop 2.6.0-CDH5.5.1 with JDK 1.7. It uses MR2 (YARN) MapReduce framework where the 6 data nodes have the node managers and hence run the map and reduce tasks. The data nodes are identical in configuration where each node has two Xeon E5-2650 processors and 128GB RAM.

Additionally several students who typically did not like programming were very interested in this activity. Some of the student comments included:

" I get paid to do this analysis?"
"I get this"
"I would like to do more of these things"
"working in groups is useful"

VIII. Conclusion

This exercise clearly demonstrated that data science may be a good motivator for enticing students towards computational science, especially data analytics. We also believe that the contextual relevance might have played a role in generating the interest in data analytics since students could relate to social media and the vast amount data that is generated in that context. This experiment provides academia with a new avenue to motivate high school students to pursue computational disciplines in higher education and prepare to enter the IT work force as juniors.

Acknowledgment

The preferred spelling of the word "acknowledgment" in America is without an "e" after the "g". Avoid the stilted expression, "One of us (R. B. G.) thanks . . ." Instead, try "R. B. G. thanks". Put sponsor acknowledgments in the unnumbered footnote on the first page.

References

[1] VCLOUD news, Every Day Big Data Statistics – 2.5 Quintillion Bytes of Data Created Daily, http://www.vcloudnews.com/every-day-big-data-statistics-2-5-quintillion-bytes-of-data-created-daily/

[2] Steve Bryson, David Kenwright, Michael Cox, David Ellsworth, and Robert Haimes. Visually Exploring Gigabyte Datasets in Real Time. *Communications of the ACM*, 42.8 (1999): 82-90.

[3] Irene Lee, Fred Martin, and Katie Apone. 2014. Integrating computational thinking across the K--8 curriculum. *ACM Inroads* 5, 4 (December 2014), 64-71.

[4] Lindsey Ann Gouws, Karen Bradshaw, and Peter Wentworth. 2013. Computational thinking in educational activities: an evaluation of the educational game light-bot. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education* (ITiCSE '13). ACM, New York, NY, USA, 10-15.

[5] Ashok Ram Basawapatna, Alexander Repenning, and Clayton H. Lewis. 2013. The simulation creation toolkit: an initial exploration into making programming accessible while preserving computational thinking. In *Proceeding of the 44th ACM technical symposium on Computer science education* (SIGCSE '13). ACM, New York, NY, USA, 501-506.

[6] Flanery and Bers, 2014, Let's Dance the "Robot Hokey-Pokey!"

[7] Wing, J. 2006. Computational thinking. Communications of the ACM 49 (3): 33-35.

[8] Burke, Q. and Kafai, Y.B. 2012. The Writers' Workshop for Youth Programmers Digital Storytelling with Scratch in Middle School Classrooms. Proceedings of SIGCSE '12 (2012), 433–438.

[9] Masood Badri, Ali Alnuaimi, Jihad Mohaidat, Asma Al Rashedi, Guang Yang and Karima Al Mazroui, My science class and expected career choices—a structural equation model of determinants involving Abu Dhabi high school students, International Journal of STEM Education 2016

[10] Onen, D. (2016). Factors influencing student choice of the research supervisor: a study of doctoral students. Towards Excellence in Educational Practices.

[11] Lewis, C.M. 2010. How Programming Environment Shapes Perception, Learning and Goals: Logo vs. Scratch. Proceedings of SIGCSE '10 (2010), 346–350

[12] Webb, D.C. et al. 2012. Toward an Emergent Theory of Broadening Participation in Computer Science Education. Proceedings of SIGCSE '12 (2012), 173–178.

[13] Philip Sheridan Buffum, Allison G. Martinez-Arocho, Megan Hardy Frankosky, Fernando J. Rodriguez, Eric N. Wiebe, and Kristy Elizabeth Boyer. 2014. CS principles goes to middle school: learning how to teach "Big Data". In *Proceedings of the 45th ACM technical symposium on Computer science education* (SIGCSE '14). ACM, New York, NY, USA, 151-156. DOI: http://dx.doi.org/10.1145/2538862.2538949

[14] Apache Hadoop, Welcome to Apache™ Hadoop®!, http://hadoop.apache.org/

[15] Tom White. Hadoop: The definitive guide. O'Reilly Media, Inc., 3rd Edition, 2012.