Big Data: Machine Learning in Hadoop

Douglas Code
College of Wooster
Wooster, OH
dcode15@wooster.edu

Norman Chamusah
College of Wooster
Wooster, OH
nchamusah14@wooster.edu

Abstract

This project focuses on Hadoop, which is a platform used to process very large data sets. The goal of the project is to gain basic understanding of Hadoop functionality and how Hadoop can be applied to large data problems. Hadoop achieves its processing speed through a high level of a parallelism. MapReduce is a data processing model which acts as a framework for parallel processing . Hadoop was implemented on a virtual machine, VMware Fusion, and a demo Hadoop system available from Cloudera to simulate WordCount activity. Hadoops accessibility and simplicity gives it an edge over running and maintaing large high-end servers. Its robustness and scalability make it suitable for many demanding jobs.

Background

Today we are living in the data age. The flood of data is coming from many sources. For example, stock markets and social media are privy to stores of information well beyond what was imaginable a few decades ago. The good news is that big data processing capabilities have grown as well. However, storing and analyzing this wealth of data remains a significant problem. This project focuses on Hadoop, which is java-based software used to process very large data sets. Hadoop can also be thought of as a software framework for distributed processing of large data sets on clusters of hardware [1].

Hadoop is often used on datasets in the range of petabytes and run on clusters containing thousands of computers. A common use of Hadoop is in distributed computer platforms for analyzing or processing large amount of data. These data sets are usually too big for even high-end machines to manage efficiently due to input/output limitations which can eat up hours of processing time just reading and writing data. These large-scale computations are characterized by the need for large numbers of CPUs and large memory to store and process data. Hadoop's use of the MapReduce system provides a framework for implementing software for these large clusters of computers.

MapReduce

MapReduce is a data processing model which acts as a framework for parallel processing implemented on a large scale [2]. Several organization like IBM and Google use Hadoop as an open implementation of MapReduce which processes vast amounts of data in parallel on large clusters. The basic process behind MapReduce is summarized in Figure 1.

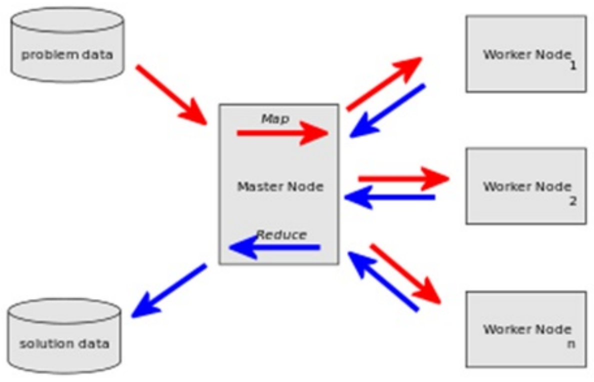


Figure 1: Diagram of basic MapReduce structure. Problem data is broken up into smaller subproblems and sent to worker nodes. The subsolutions to these problems are then reduced to a single solution[5].

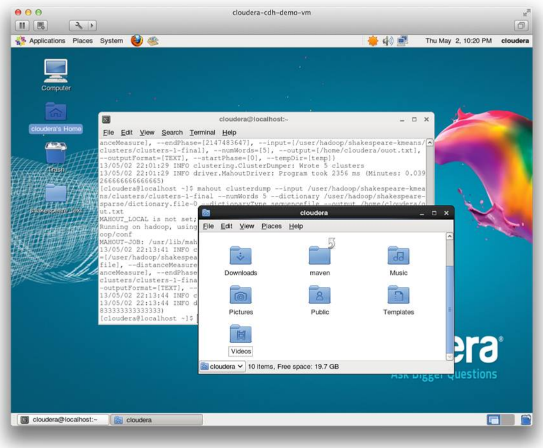
The problem is broken down into many sub-problems, which are mapped to worker nodes. The worker nodes perform the processing and pass back sub-solutions to the Master Node. The Master Node then reduces the solutions to a single data solution, which is passed as the output. The Master Node regularly checks on the functionality and progress of the worker nodes. If a single worker node experiences a failure and the Master Node consequently stops receiving updates from that node, the Master Node reassigns the sub-problem to another worker node, making sure that no subproblems are left incomplete. Due to this behavior, we are assured that every job will be completed regardless of worker node failure, giving Hadoop the trait of being particularly robust to failures in individual machines.

Hadoops accessibility and simplicity gives it any edge over writing and running large distributed problems and programs. Its robustness and scalability makes it suitable for even the more demanding jobs at Yahoo, Google, and Facebook etc. It scales linearly to handle larger data by adding more nodes to the cluster. Another advantage of using Hadoop is that different users can submit computing jobs from individual clients and desktops [3]. Our team was tasked with investigating big-data machine learning possibilities because of many of the above appealing advantages of using Hadoop. Therefore, the goal for this project is to gain basic understanding of Hadoop and its functions and explore possible real-world applications.

The Hadoop tutorial from Cloudera provides a short overview of Hadoop and MapReduce, discussing how MapReduce splits the data into individual chunks to be processed separately [2].It also describes the Mapreduce framework as a master Job Tracker that schedules jobs and a slave Tracker that monitors tasks. The tutorial continues with a sample implementation of the WordCount application for Hadoop. A walkthrough the guide provides the steps for using WordCount to count the frequency of each word in a large data set. This tutorial and several online resources helped us implement Hadoop on our machine.

Implementation

We successfully installed the virtual machine from Cloudera to run Hadoop. The implementation is run entirely on the virtual machine, which simulates a Hadoop cluster. This system contains a number of packages for Hadoop, each of which is meant to use Hadoop's large-scale processing power for a different type of task. For our implementation we used Mahout, which is one of Hadoops machine learning packages[3]. Figure 2 shows an example screen from the Cloudera Hadoop package.



```
anceMeasure], --endPhase=[2147483647], --input=[/user/hadoop/shakespeare-kmeans/
clusters/clusters-1-final], --numWords=[5], --output=[/home/cloudera/ouot.txt],
--outputFormat=[TEXT], --startPhase=[0], --tempDir=[temp]}
13/05/02 22:01:29 INFO clustering.ClusterDumper: Wrote 5 clusters
13/05/02 22:01:29 INFO driver.MahoutDriver: Program took 2356 ms (Minutes: 0.039
266666666666665)
[cloudera@localhost ~]$ mahout clusterdump --input /user/hadoop/shakespeare-kmea
ns/clusters/clusters-1-final --numWords 5 --dictionary /user/hadoop/shakespeare-
sparse/dictionary.file-0
ut.txt
MAHOUT_LOCAL is not set;
Running on hadoop, using
oop/conf
MAHOUT-JOB: /usr/lib/mah
13/05/02 22:13:41 INFO c
=[/user/hadoop/shakespea
file], --distanceMeasure
anceMeasure], --endPhase
clusters/clusters-1-fina
-outputFormat=[TEXT], --
13/05/02 22:13:44 INFO c
13/05/02 22:13:44 INFO d
8333333333333333)
[cloudera@localhost ~]$
```

Figure 2: Demonstration of the Cloudera Hadoop package running on a VMWare Fusion virtual machine. Both the terminal window and the file system can be seen.

Mahout

Mahout is a package of machine learning implementations currently in development. A large group of programmers are currently working on coding for Mahout. As a result of their efforts, implementations of k-means clustering, naive bayes classifiers, and random forest decision trees. Mahout is one of the most popular extensions of Hadoop and comes with the Cloudera virtual machine.

K-Means Clustering

Clustering is an example of unsupervised learning technique. The goal of clustering is to partition data into a number of clusters so that data in each cluster is more similar to each other than in other clusters. Our implementation calls on K means clustering capabilities of Mahout. K-means clustering classifies test data by assigning a set of k centroid points and adjusting those centroid points so that they correctly represent data clusters around them. Hadoops scalability offers the power to use k-means clustering on extremely large data sets with thousands of centroid points, however, due to limitations on time and the use of simulated cluster rather than a real cluster, we only perform the cluster on small data set.

N-Grams and Text Analysis

In our project we want to analyze frequently occurring text in a file. We used a collection of Shakespeares work and apply K-means clustering to find the most frequently occurring n-grams. N-grams are the sequences of a given size, which are made up of n adjacent words in a section of text. For example, The, is a possible n-gram where n=1, The dog, is an n-gram where n=2, and The dog ran is an n-gram where n=3.

N-grams are frequently used in speech recognition applications. By identifying commonly occuring phrases and combinations of words, programmers are able to create software which more effectively picks up human speech patterns. The use of n-grams is also common in sentiment classifiers, which examine text and assign a certain sentiment or mood to that section of text. This is helpful to social network and advertisers, as they can keep track of trends among their users over time and general responses to changes and advertisements. Google's n-gram viewer allows the user to perform a large-scale search of n-grams from over 30 million books. This sort of data analysis is a perfect example of the possible applications of Hadoop, and our implementations is similar to the Google n-gram viewer, albeit on a much smaller scale. Figure 3 shows an example result from the Google n-gram viewer.

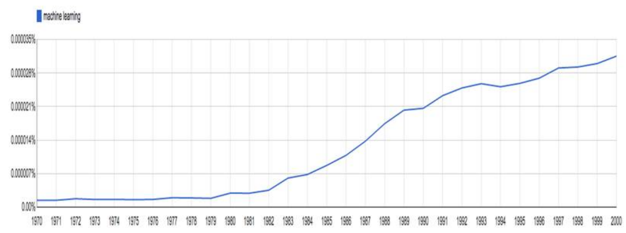


Figure 3: Sample n-gram from Google n-gram viewer. The frequency of the phrase ”machine learning” was plotted over the time period 1970-2000[4].

Hadoop Commands

Our K-means clustering was run on a set of 5 Shakespeare plays: Hamlet, Romeo and Juliet, King Lear, Macbeth, and Julius Caesar. In order to run the clustering, Apache Maven had to be installed onto the Hadoop cluster. Apache Maven is a tool which automatically builds Java programs and downloads necessary Java packages and plugins. Due to the fact that our software was written in Java, Maven was essential to an effective run. In total, five Hadoop commands needed to be run to complete the k-means clustering and output the data.

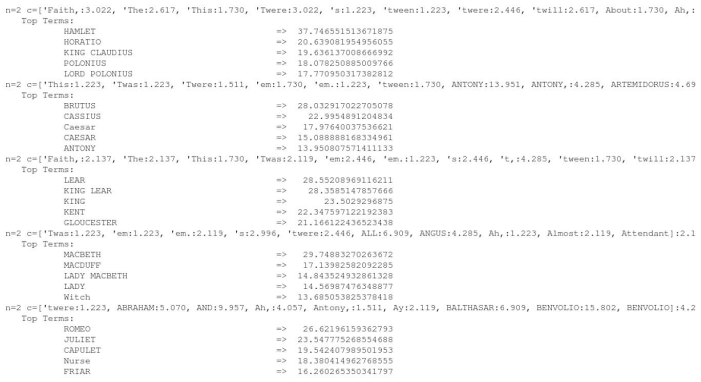
The first command is an *fs -put* command. This command takes the data files, in this case the folder of .txt files containing transcriptions of Shakespeare's plays, and places those files into the Hadoop file system. This lets Hadoop work in a consistent directory and easily operate on the needed files.

The next command is a *seqdirectory* command. The *seqdirectory* command converts the .txt files to SequenceFiles, which are the required format for the upcoming actions that Mahout and Hadoop will take. This command is parameterized by input and output commands which let the user decide which files to convert and where to place the converted files in the Hadoop file system.

Following *seqdirectory* is the *seq2sparse* command. This command processes the SequenceFiles into a vector of words, which is required for processing. A whitespace analyzer implementation from Apache is used, which creates breaks at each whitespace (space), breaking up each individual word. This lets the text be vectorized, and lets the software know where each individual word begins and ends. Many of the basic functions of the n-gram processing are parameterized in this command. *-ml* sets the minimum likelihood required, which can be adjusted to remove extremely low-frequency words from the final results. Conversely, -*x* sets the maximum frequency, which can be adjusted to remove extremely common and largely meaningless words like. The *-md* (minimum documents) parameter also excludes words occuring in only one or two documents, making it so that uncommon words with extremely high clustering in a single area aren't disproportionately represented in the results. Finally, the *-ng* parameter decides the size of the n-gram, letting the user control the type of final output they will receive.

After the data is vectorized, the *k-means* command is called, finally performing k-means on the now-ready data set. This creates cluster files in the Hadoop file system. Some k-means parameters are available, like *-maxIter* , which controls how many iterations the centroids will adjust, and -*numClusters*, which lets the user decide how many classification clusters to use.

Once the classifications have been made, the data is output using the *clusterdum* command. This command prints the data to a text file which can be used for analysis. The text file shows the frequency of all n-grams in the original works, and also shows a number of the most frequently occurring n-grams, which can be customized by using the *-numWords* parameter to choose the number that will be displayed. Figure 4 shows the format of the output data.



```
n=2 c=['Faith,:3.022, 'The:2.617, 'This:1.730, 'Twere:3.022, 's:1.223, 'tween:1.223, 'twere:2.446, 'twill:2.617, About:1.730, Ah,:
   Top Terms:
           HAMLET                                  =>  37.746551513671875
           HORATIO                                 =>  20.639081954956055
           KING CLAUDIUS                           =>  19.636137008666992
           POLONIUS                                =>  18.078250885009766
           LORD POLONIUS                           =>  17.770950317382812
n=2 c=['This:1.223, 'Twas:1.223, 'Twere:1.511, 'em:1.730, 'em.:1.223, 'tween:1.730, ANTONY:13.951, ANTONY,:4.285, ARTEMIDORUS:4.69
   Top Terms:
           BRUTUS                                  =>  28.032917022705078
           CASSIUS                                 =>    22.9954891204834
           Caesar                                  =>   17.97640037536621
           CAESAR                                  =>  15.088888168334961
           ANTONY                                  =>  13.950807571411133
n=2 c=['Faith,:2.137, 'The:2.137, 'This:1.730, 'Twas:2.119, 'em:2.446, 'em.:1.223, 's:2.446, 't,:4.285, 'tween:1.730, 'twill:2.137
   Top Terms:
           LEAR                                    =>   28.55208969116211
           KING LEAR                               =>    28.3585147857666
           KING                                    =>       23.5029296875
           KENT                                    =>  22.347597122192383
           GLOUCESTER                              =>  21.166122436523438
n=2 c=['Twas:1.223, 'em:1.223, 'em.:2.119, 's:2.996, 'twere:2.446, ALL:6.909, ANGUS:4.285, Ah,:1.223, Almost:2.119, Attendant]:2.1
   Top Terms:
           MACBETH                                 =>   29.74883270263672
           MACDUFF                                 =>   17.13982582092285
           LADY MACBETH                            =>  14.843524932861328
           LADY                                    =>   14.56987476348877
           Witch                                   =>  13.685053825378418
n=2 c=['twere:1.223, ABRAHAM:5.070, AND:9.957, Ah,:4.057, Antony,:1.511, Ay:2.119, BALTHASAR:6.909, BENVOLIO:15.802, BENVOLIO]:4.2
   Top Terms:
           ROMEO                                   =>   26.62196159362793
           JULIET                                  =>  23.547775268554688
           CAPULET                                 =>  19.542407989501953
           Nurse                                   =>  18.380414962768555
           FRIAR                                   =>  16.260265350341797
```

Figure 4: Sample output from Mahout's k-means classifier. The top results for each cluster are shown along with the total data from each cluster.

Conclusions

While Hadoop presents a significant learning curve to new users, it presents significantly more accessibility and data-crunching potential than the much more difficult task of manually programming highly parallelizable software. Much of the time we spent working with the software was taken up by the task of learning new commands and struggling with the command-line interface. However, both of these problems would fade quickly with more practice and exploration into the scripting of commands so that programs run more autonomously.

The MapReduce framework offers tremendous benefits from a programming perspective. By hiding most of the work of subdividing and managing huge distributed systems, MapReduce makes parallel programming accessible to programmers of all levels. The programmer merely has to tell the system how to subdivide and reduce the larger problem at hand, and the software handles all of the node micromanagement and failure handling. This also presents significant advantages to large companies, as it lets them implement custom systems and code without having to go through the added expense of hiring specialists.

Additionally, and partially as a result of this accessibility, a large number of public projects to provide implementations of software for Hadoop have sprung up around the world. Packages like Mahout offer relative ease of use and save a tremendous amount of time by letting users get all the benefits of machine learning without the expense and time of implementing these algorithms. Many packages like Mahout expand Hadoop's capabilities even further and well beyond the realm of machine learning. This once again offers significant benefits to businesses which decide to set up Hadoop clusters, as they don't have to pay programmers to implement often complex algorithms. Instead they can use premade packages which are often free and completely capable of solving the problem in question.

Future Work

Due to the learning curve of Hadoop and the tremendous scale of the project, we were only able to explore a tiny fraction of Hadoop's capabilities. Future projects would benefit from further exploration of Mahout's various packages including Naive Bayes Classifiers and random forest decision trees.

Another direction to take in the future is to start working on custom implementations of software for Hadoop. By directly working with the MapReduce framework, researchers could grasp the underlying structure and its implications for programmers much more quickly than solely through reading tutorials and research papers. Furthermore, many of the real-world applications of Hadoop require code to be modified or created to customize to a project's needs. Once more comfort programming for Hadoop is achieved, future work could lead to involvement in the communities developing packages like Hadoop.

Finally, the capabilities of Hadoop could be better explored by setting it up on a real cluster of computers rather than a simulated cluster on a virtual machine. This would give a firsthand look at the difficulties involved in large scale distributed computing, and let the user really enjoy the speed benefits of using Hadoop.

References

Cloudera, ”Hadoop Tutorial.” Accessed May 6, 2013. http://www.cloudera.com/content/cloudera-content/cloudera-docs/HadoopTutorial/CDH4/Hadoop-Tutorial.html.

Patil, Sarika, and Shyam Deshmukh. ”Survey on Task Assignment Techniques in Hadoop.” *International Journal of Computer Applications*. no. 14 (2012): 15-18.

Owens, Jonathan, Brian Femiano, and Jon Lentz. *Hadoop Real-World Solutions Cookbook*. Packt Publishing, 2013.

Google Inc, ”Google Ngram Viewer.” Accessed May 6, 2013. http://books.google.com/ngrams/.