GRAMBLING STATE UNIVERSITY
WHERE EVERYBODY IS SOMEBODY

Big Data
Retrieving Required Information From Text Files

Desmond Hill
Yenumula B Reddy (Advisor)

GRAMBLING STATE UNIVERSITY
WHERE EVERYBODY IS SOMEBODY

OUTLINE

- Objective
- What is Big data
- Characteristics of Big Data
- Setup Requirements
- Hadoop Setup
- Word Count Importance
- The first approach uses the MapReduce techniques
- Mapp Reduce Experiment
- Results
- Conclusions
- ???

GRAMBLING STATE UNIVERSITY
WHERE EVERYBODY IS SOMEBODY

Objective

The objectives of this research related to Big Data is:

- Analysis of text files using Hadoop package and select the required document

GRAMBLING STATE UNIVERSITY
WHERE EVERYBODY IS SOMEBODY

What is Big data

Big data usually includes data sets with sizes beyond the ability of commonly used software tools to capture, curate, manage, and process data within a tolerable elapsed time.

Big data "size" is a constantly moving target, as of 2012 ranging from a few dozen terabytes to many petabytes of data.

Big data requires a set of techniques and technologies with new forms of integration to reveal insights from datasets that are diverse, complex, and of a massive scale

GRAMBLING STATE UNIVERSITY
WHERE EVERYBODY IS SOMEBODY

Characteristics of Big Data

The following are the Characteristics of Big Data

Volume: The quantity of generated and stored data. The size of the data determines the value and potential insight- and whether it can actually be considered big data or not.

Variety: The type and nature of the data. This helps people who analyze it to effectively use the resulting insight.

Velocity: In this context, the speed at which the data is generated and processed to meet the demands and challenges that lie in the path of growth and development.

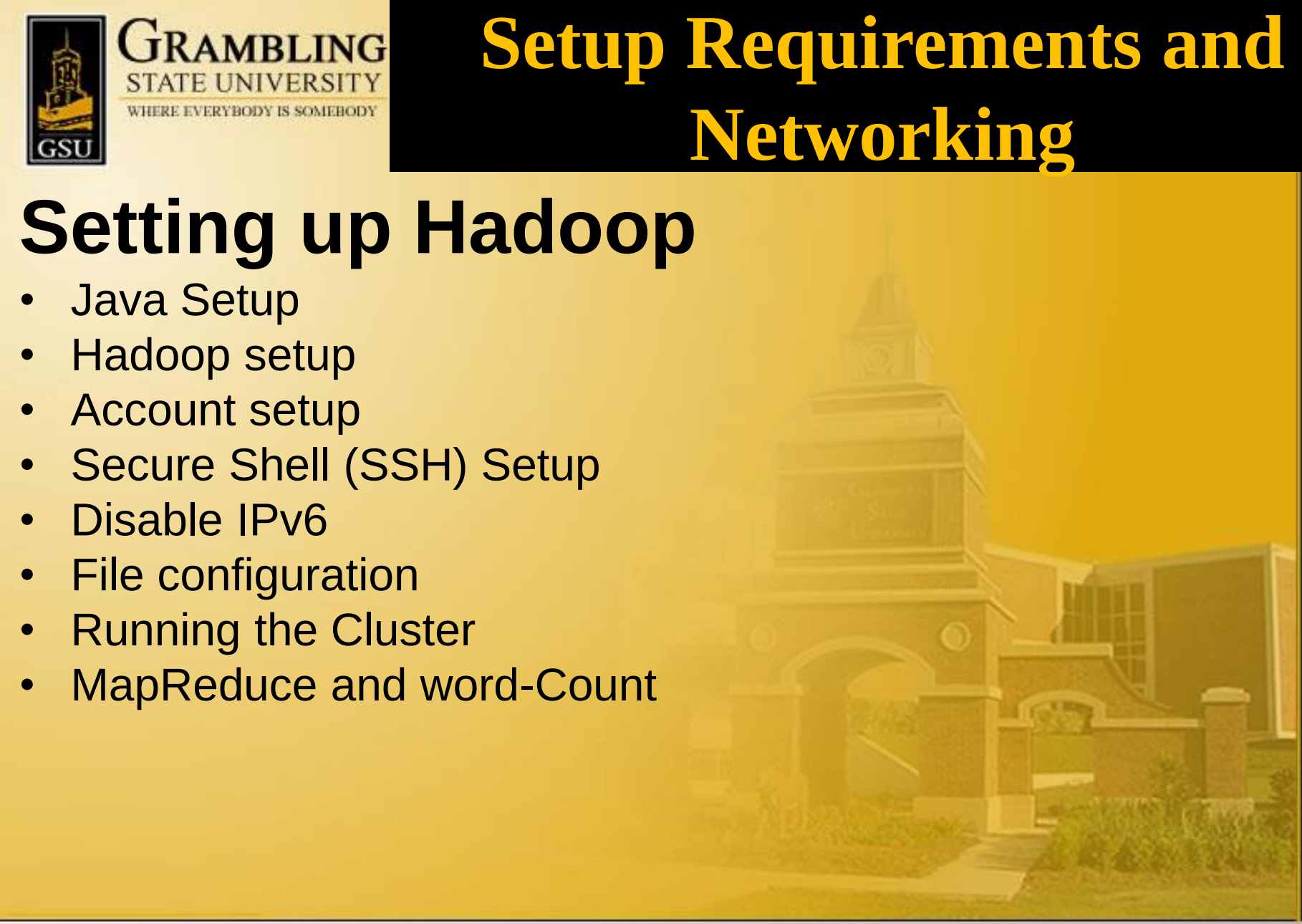
Variability: Inconsistency of the data set can hamper processes to handle and manage it.

Veracity: The quality of captured data can vary greatly, affecting accurate analysis.

Setup Requirements and Networking

Setting up Hadoop

- Java Setup
- Hadoop setup
- Account setup
- Secure Shell (SSH) Setup
- Disable IPv6
- File configuration
- Running the Cluster
- MapReduce and word-Count



GRAMBLING STATE UNIVERSITY
WHERE EVERYBODY IS SOMEBODY

Setup Hadoop

Java Setup

- installing the latest version of Java Development Kit using the following command:
- sudo apt-get install jdk-1.7.0
- Make sure to place this file in the /usr/lib directory

Hadoop setup

- http://apache.mirrors.tds.net/hadoop/common/hadoop-2.6.0/hadoop-2.6.0.tar.gz
- sudo tar xzf hadoop-2.6.0.tar.gz
- move it to the /usr/local directory

  sudo mv hadoop-2.6.0 /usr/local

Account setup

sudo adddgroup hadoop

sudo adduser –ingroup hadoop datauser

sudo adduser datauser sudo

After setting up the account we then changed the Hadoop files to be owned by the new Hadoop user.

Secure Shell (SSH) Setup

ssh-keygen –t rsa –P “”

cat $HOME/ .ssh/id_rsa.pub >> $HOME/ .ssh/authorized_keys

Setup Hadoop

File Configuration steps

- .bashrc
- sysctl.conf
- core-site.xml
- hdfs-site.xml
- mapred-site.xml
- hadoop-env.sh
- yarn-env.sh

Running the Cluster

After configuring all of the files we where able to start both of the Single – Node cluster.

Before starting we first deleted the 'data' directory, then we formatted the namenode, and last we started up the cluster. We used the JPS command to display all of the running nodes.

- rm –r data
- bin/hadoop namenode –format
- sbin/start-all.sh
- jps

Word Count Importance

- To select a required document Using MapReduce, we provided the keywords and their importance that varies between 0 and 1.
- We then take the important factor multiplied by the number of times keyword and sum the result of all keyword importance.
- If the sum is greater than or equal to threshold we conclude that the document is required.
- The algorithm was coded in two stages.
  - During the first stage the reputation of the words
  - In the second stage the importance factor and selection of the document were coded in Python

Single Node Setup

- Analysis of text files using Hadoop package and select the required document – completed with single node and multiple nodes
- Analysis of Big Data – Unstructured data needs to be analyzed to find the importance of that data from stream of data generated today (text, images, social media data, email data, etc.,), which is impossible with normal database models (SQL based)

GRAMBLING STATE UNIVERSITY
WHERE EVERYBODY IS SOMEBODY

The first approach uses the MapReduce techniques

- To select a required document Using MapReduce, we provided the keywords and their importance that varies between 0 and 1.
- We then take the important factor multiplied by the number of times keyword and sum the result of all keyword importance.
- If the sum is greater than or equal to threshold we conclude that the document is required.
- The algorithm was coded in two stages.
  - During the first stage the reputation of the words
  - In the second stage the importance factor and selection of the document were coded in Python
- We processed six text files to compare the results for the current experiment.

Parameters

- medicine (0.02) profession (0.025), disease (0.02), surgery (0.02), mythology (0.02), and cure (0.05).
- The amount of words per file, and there times measured in seconds to process and impact factors respectively are:

1. (128,729w; 0.1539s)
2. (128,805w; 0.1496s)
3. (266,017w; 0.13887s)
4. (277,478w; 0.1692s)
5. (330,582w; 0.1725s)
6. (409,113w; 0.2032s)

GRAMBLING STATE UNIVERSITY
WHERE EVERYBODY IS SOMEBODY

Mapper File

word_mapper.py (~/Documents/MRCode) - gedit

Open Save Undo

word_mapper.py × word_reducer.py ×

```
#!/usr/bin/python

import sys
import time


search_words = {'cure':0.05,'disease':0.02,'medicicne':0.02,
        'mythology':0.02,'surgery': 0.02,'mythology': 0.02}
#Note: Only search for existing words.
#Note: Currently case sensitive. ~Try .lower()

startTime = time.time()
wordTotal = 0
counter = 0


for line in sys.stdin:
        for word in line.strip().split():
                for counter in range(len(search_words)):
                        if word == search_words.keys()[counter]:
                                print "%s\t%d" % (word, 1)
```

Python 3 Tab Width: 8 Ln 11, Col 1 INS

GRAMBLING STATE UNIVERSITY
WHERE EVERYBODY IS SOMEBODY

Reducer File

word_reducer.py (~/Documents/MRCode) - gedit

Open Save Undo

word_mapper.py × word_reducer.py ×

```
#!/usr/bin/python

import sys
import time

current_word = None
current_count = 1
word_names = []
word_amounts = []
x = 0
j = 0
gate = False
importance = 0
search_importance = 0
totalTime = time.time()


print("\n-----------Word Occurance------------------------------------------------------\n")



for line in sys.stdin:
            word, count = line.strip().split('\t')
            if current_word:
                if word == current_word:
                    current_count += int(count)
                else:
                    word_names.append(current_word)
                    print "%s\t%d" % (current_word, current_count)
                    word_amounts.append(current_count)
                    j = j + 1
                    current_count = 1

            current_word = word




if current_count > 0:
    word_names.append(current_word)
    print "%s\t%d" % (current_word, current_count)
    word_amounts.append(current_count)

                #adds last searched word


print("\n-----------Word Importance---------- \n")
```

Python 3 Tab Width: 8 Ln 23, Col 23 INS

GRAMBLING STATE UNIVERSITY
WHERE EVERYBODY IS SOMEBODY

Reducer File

word_reducer.py (~/Documents/MRCode) - gedit

Open Save Undo

word_mapper.py × word_reducer.py ×

```
            word, count = line.strip().split('\t')
            if current_word:
                if word == current_word:
                    current_count += int(count)
                else:
                    word_names.append(current_word)
                    print "%s\t%d" % (current_word, current_count)
                    word_amounts.append(current_count)
                    j = j + 1
                    current_count = 1

            current_word = word



if current_count > 0:
    word_names.append(current_word)
    print "%s\t%d" % (current_word, current_count)
    word_amounts.append(current_count)

                #adds last searched word


print("\n-----------Word Importance---------- \n")



from word_mapper import search_words
from word_mapper import startTime

startTime = time.time() - startTime

for i in range(len(word_names)):
        importance = search_words[word_names[x]] * word_amounts[x]
        search_importance += importance
        print "%s\t%.2f" % (word_names[x], importance)
        x = x + 1

print("\n-------------Search Stats----------- ")

print "%s%.2f" % ("\nSearch importance: ", search_importance)
totalTime = ((time.time() - totalTime) + startTime)
print "%s %.23f %s" % ("Search time: ", totalTime, "seconds.")



print("\n-----------END--------------------------------------------------------------------")
```

Python 3 Tab Width: 8 Ln 23, Col 23 INS

Results

Results - 1

- An introduction to the history of medicine

Occurrence:

- cure 31
- disease 214
- medicine 428
- mythology 2
- surgery 222

Importance:

- cure 1.55
- disease 4.28
- medicine 8.56
- mythology 0.04
- surgery 4.44

- Search importance: 18.87

Results - 2

- Aristotle history

Occurrence:

- cure 4
- disease 21
- medicine 1
- mythology 0
- surgery 0

Importance:

- cure 0.20
- disease 0.42
- medicine 0.02
- mythology 0.00
- surgery 0.00

- Search importance: 0.64

Results - 3

- Emergency Medicine Secrets

Occurrence:

- cure 9
- disease 288
- medicine 55
- mythology 0
- surgery 31

Importance:

- cure 0.45
- disease 5.76
- medicine 1.10
- mythology 0.00
- surgery 0.62

- Search importance: 7.93

Results

Results - 4

- Sexual Life in Ancient India

Occurrence:

- cure 1
- disease 0
- medicine 2
- mythology 1
- surgery 0

Importance:

- cure 0.05
- disease 0.00
- medicine 0.04
- mythology 0.02
- surgery 0.00

- Search importance: 0.14

Results - 5

- The biochemical system of medicine

Occurrence:

- cure 63
- disease 81
- medicine 31
- mythology 0
- surgery 0

Importance:

- cure 3.15
- disease 1.62
- medicine 0.62
- mythology 0.00
- surgery 0.00

- Search importance: 5.39

Results - 6

- Avicennas Cannon of Medicine

Occurrence:

- cure 14
- disease 145
- medicine 121
- mythology 0
- surgery 1

Importance:

- cure 0.70
- disease 2.90
- medicine 2.42
- mythology 0.00
- surgery 0.02

- Search importance: 6.04

Conclusions

- In conclusion, we have successfully set up a single node cluster on two different machines.
- We have also constructed Python code that will use the machines clusters to take a text file, use MapReduce facility, and output the number of times each word repeats in the document
- We retrieved the important documents by providing the impact factor for the specific key words
- Depending upon the formula used for importance of key words the algorithm selects the displays the required document

GRAMBLING STATE UNIVERSITY
WHERE EVERYBODY IS SOMEBODY

Questions