NEHRU COLLEGE OF ENGINEERING AND RESEARCH CENTRE

(Accredited by NAAC, ISO 9001-2015 Certified Institution *Approved by AICTE New Delhi, Affiliated to APJKTU)*

Pampady, Thiruvilwamala(PO), Thrissur(DT), Kerala 680 588

DEPARTMENT OF COMPUTER SCEINCE AND ENGINEERING

LAB MANUAL

08CS6072 (P) DATAMINING AND ANALYTICS LAB

VISION OF THE INSTITUTION

To mould true citizens who are millennium leaders and catalysts of change through excellence in education.

MISSION OF THE INSTITUTION

NCERC is committed to transform itself into a center of excellence in Learning and Research in Engineering and Frontier Technology and to impart quality education to mould technically competent citizens with moral integrity, social commitment and ethical values.

We intend to facilitate our students to assimilate the latest technological know-how and to imbibe discipline, culture and spiritually, and to mould them in to technological giants, dedicated research scientists and intellectual leaders of the country who can spread the beams of light and happiness among the poor and the underprivileged.

ABOUT DEPARTMENT

- Established in: 2002
- Course offered : B.Tech in Computer Science and Engineering

  M.Tech in Computer Science and Engineering

  M.Tech in Cyber Security
- Approved by AICTE New Delhi and Accredited by NAAC
- Affiliated to the University of A P J Abdul Kalam Technological University.

DEPARTMENT VISION

Producing Highly Competent, Innovative and Ethical Computer Science and Engineering Professionals to facilitate continuous technological advancement.

DEPARTMENT MISSION

1. To Impart Quality Education by creative Teaching Learning Process
2. To Promote cutting-edge Research and Development Process to solve real world problems with emerging technologies.
3. To Inculcate Entrepreneurship Skills among Students.
4. To cultivate Moral and Ethical Values in their Profession.

PROGRAMME EDUCATIONAL OBJECTIVES

PEO1. Graduates will be able to Work and Contribute in the domains of Computer Science and Engineering through lifelong learning.

PEO2: Graduates will be able to analyse, design and development of novel Software Packages Web Services, System Tools and Components as per needs and specifications.

PEO3: Graduates will be able to demonstrate their ability to adapt to a rapidly changing environment by learning and applying new technologies.

PEO4: Graduates will be able to adopt ethical attitudes, exhibit effective communication skills, teamwork and leadership qualities.

PROGRAM OUTCOMES (POS)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSO)

PSO1: Ability to Formulate and Simulate Innovative Ideas to provide software solutions for Real-time Problems and to investigate for its future scope.

PSO2: Ability to learn and apply various methodologies for facilitating development of high quality
System Software Tools and Efficient Web Design Models with a focus on performance optimization.

PSO3: Ability to inculcate the Knowledge for developing Codes and integrating hardware/software
products in the domains of Big Data Analytics, Web Applications and Mobile Apps to create innovative career path and for the socially relevant issues.

PREPARATION FOR THE LABORATORY SESSION
GENERAL INSTRUCTIONS TO STUDENTS

1. You should come prepared for your lab session properly to utilize the maximum time of lab session

2. You should attempt all lab experiments given in the list

3. You may seek assistance in doing the lab experiments from the available lab Instructor.

4. There should be proper comments for description of the problem, requirement of class, function etc. Proper comments are to be provided as and when necessary in the programming. This will develop a good practice in you for the rest of your professional life
5. Your program should be interactive and properly documented with real input/output data.
6. Proper management of file of lab record is necessary. Completed lab experiments should be submitted in the form of lab records

7. Maintain silence, order and discipline inside the lab. Don't use cell phones inside the laboratory

AFTER THE LABORATORY SESSION

1 Shut down the system before you leave.

2. Arrange your chair.

3. Make sure you understand what kind of report is to be prepared and due submission of record is next lab class.

MAKE-UPS AND LATE WORK

Students must participate in all laboratory exercises as scheduled. They must obtain permission from the faculty member for absence, which would be granted only under justifiable circumstances. In such an event, a student must make arrangements for a make-up laboratory, which will be scheduled when the time is available after completing one cycle.

LABORATORY POLICIES

1. Food, beverages & mobile phones are not allowed in the laboratory at any time.

2. Do not sit or place anything on instrument benches.

3. Organizing laboratory experiments requires the help of laboratory technicians and staff. Be punctual.

SYLLABUS

Course No.	Course Name	L-T-P	Credits	Year of Introduction
08CS6072(P)	Data Mining and Analytics Lab	0-0-2	1	2015

Syllabus

Experiments are based on topics covered in Data Mining

To give the Student:-

- The ability to design and implement various algorithms related to information retrieval.
- Hands on experience with latest data storage and analytic infrastructure like Hadoop and R

08CS6072(P) - **Experiments**

Experiment No	Description
I	Study of WEKA / R Tool
II	Building an inverted index
III	Implementation of text classification techniques
IV	Implementation of text clustering techniques
V	Implementation of Page Rank Algorithm
VI	Setting up a Hadoop Cluster
VII	Developing a MapReduce Application
VIII	Working with a distributed data storage(Any NoSQL model)

INDEX

FINAL VERIFICATION BY THE HOD

EXPERIMENT – 1

STUDY OF NLTK AND SCIKIT-LEARN

OBJECTIVE

To study about Natural Language Toolkit and Scikit-Learn.

NATURAL LANGUAGE TOOLKIT (NLTK)

NLTK is a leading platform for building Python programs that work with human language data. It also provides easy-to-use interfaces to about 50 corporas and other lexical resources such as WordNet, along with a suite of text processing libraries for classification, , stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries. NLTK has been called "a wonderful tool for teaching, and working in, computational linguistics using Python," and "an amazing library to play with natural language."

The **Natural Language Toolkit**, or more commonly **NLTK**, is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English written in the Python programming language. It was developed by Steven Bird and Edward Loper in the Department of Computer and Information Science at the University of Pennsylvania. NLTK includes graphical demonstrations and sample data. It is accompanied by a book that explains the underlying concepts behind the language processing tasks supported by the toolkit, plus a cookbook.

Some simple things that we can do with NLTK are:

- Tokenize and tag some text
- Identify named entities
- Display a parse tree

NLTK was designed to satisfy the following goals:

- Simplicity
- Consistency
- Extensibility
- Modularity

INSTALLING NLTK

- Open Terminal
- Enter **sudo apt-get install python-numpy python-nltk**
- Press '**y**' when prompted. Wait for it to finish downloading and you're good to go.

Stemmers

Stemmers remove morphological affixes from words, leaving only the word stem.

```
>>> from __future__ import print_function
>>> from nltk.stem import *
Unit tests for the Porter stemmer
    >>> from nltk.stem.porter import
```

Create a new Porter stemmer.

```
    >>> stemmer = PorterStemmer()
```

Test the stemmer on various pluralised words.

```
    >>> plurals = ['caresses', 'flies', 'dies', 'mules', 'denied',
    ...            'died', 'agreed', 'owned', 'humbled', 'sized',
    ...            'meeting', 'stating', 'siezing', 'itemization',
    ...            'sensational', 'traditional', 'reference', 'colonizer',
    ...            'plotted']
    >>> singles = [stemmer.stem(plural) for plural in plurals]
    >>> print(' '.join(singles))  # doctest: +NORMALIZE_WHITESPACE
    caress fli die mule deni die agre own humbl size meet
    state siez item sensat tradit refer colon plot
```

Unit tests for Snowball stemmer

```
    >>> from nltk.stem.snowball import SnowballStemmer
```

See which languages are supported.

```
    >>> print(" ".join(SnowballStemmer.languages))
    danish dutch english finnish french german hungarian italian
    norwegian porter portuguese romanian russian spanish swedish
```

Create a new instance of a language specific subclass.

```
>>> stemmer = SnowballStemmer("english")
h
```

Stem a word.

```
>>> print(stemmer.stem("running"))

run
```

Decide not to stem stopwords.

```
>>> stemmer2 = SnowballStemmer("english", ignore_stopwords=True)
>>> print(stemmer.stem("having"))

have

>>> print(stemmer2.stem("having"))

having
```

Lemmatizing

Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma .

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

print(lemmatizer.lemmatize("cats"))
print(lemmatizer.lemmatize("rocks"))
print(lemmatizer.lemmatize("python"))
print(lemmatizer.lemmatize("better", pos="a"))
print(lemmatizer.lemmatize("best", pos="a"))
print(lemmatizer.lemmatize("run"))
print(lemmatizer.lemmatize("run",'v'))
```

.

TOKENIZATION

Tokenization is the task of cutting a string into linguistic units.The very simplest method for tokenizing is the usage of raw.split() method. Sentence tokenizer can be used to find the list of sentences and Word tokenizer can be used to find the list of words in strings.

SCI-KIT LEARN.

The scikit-learn project started as scikits.learn, a Google Summer of Code project by David Cournapeau. Its name stems from the notion that it is a "SciKit" (SciPy Toolkit), a separately-developed and distributed third-party extension to SciPy. Scikit-learn is largely written in Python, with some core algorithms written in cython to achieve performance. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy

RESULT AND DISCUSSION

The study of NLTK and Scikit-Learn was done successfully.

EXPERIMENT – 2

BUILDING AN INVERTED INDEX

OBJECTIVE

To build inverted index for a set of document collections using NLTK and Python.

TOOLS AND TECHNIQUES

The main tools used are NLTK, Python and Jupyter notebook.

NLTK-NATURAL LANGUAGE TOOLKIT

NLTK is a leading platform for building Python programs that work with human language data. It also provides easy-to-use interfaces to about 50 corporas and other lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries

PYTHON

Python is a powerful high-level, object-oriented programming language created by Guido van Rossum. It has simple easy-to-use syntax, making it the perfect language for someone trying to learn computer programming for the first time. It provides constructs that enable clear programming on both small and large scales. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Applications of python includes:

- A simple language which is easier to learn
- Free and open-source
- Portability
- Extensible and Embeddable
- A high-level, interpreted language
- Large standard libraries to solve common tasks

JUPYTER NOTEBOOK

Project Jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages. The Jupyter Notebook App is a server-client application that allows editing and running notebook documents via a web browser. The Jupyter Notebook App can be executed on a local desktop requiring no internet access or can be installed on a remote server and accessed through the internet**.**

INVERTED INDEX

In computer science, an inverted index (also referred to as postings file or inverted file) is an index data structure storing a mapping from content, such as words or numbers, to its locations in a database file, or in a document or a set of documents (named in contrast to a forward index, which maps from documents to content). The purpose of an inverted index is to allow fast full text searches, at a cost of increased processing when a document is added to the database. The inverted file may be the database file itself, rather than its index. It is the most popular data structure used in document retrieval systems, used on a large scale for example in search engines. Additionally, several significant general-purpose mainframe-based database management systems have used inverted list architectures, including ADABAS, DATACOM/DB, and Model 204.

There are two main variants of inverted indexes: A record-level inverted index (or inverted file index or just inverted file) contains a list of references to documents for each word. A word-level inverted index (or full inverted index or inverted list) additionally contains the positions of each word within a document. The latter form offers more functionality (like phrase searches), but needs more processing power and space to be created. The major steps of building an inverted index are:

- Collect the documents to be indexed.
- Tokenize the text, turning each document into a list of tokens.
- Do linguistic preprocessing, producing a list of normalized tokens, which are the indexing terms.
- Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and postings.

PROGRAM

```
import nltk

from collections import defaultdict

from nltk.stem.snowball import EnglishStemmer # Assuming we're working with English

  """ Inverted index datastructure """

  def __init__(self, tokenizer, stemmer=None, stopwords=None):

    """

    tokenizer -- NLTK compatible tokenizer
    function stemmer -- NLTK compatible stemmer

    stopwords  -- list of ignored words
```

```
        """
        self.tokenizer = tokenizer
        self.stemmer = stemmer
        self.index = defaultdict(list)
        self.documents = {}
        self.__unique_id = 0
        if not stopwords:
            self.stopwords = set()
        else:
            self.stopwords = set(stopwords)
    def lookup(self, word):
        """
        Lookup a word in the index
        """
        word = word.lower()
        if self.stemmer:
```

```
        word = self.stemmer.stem(word)

    return [self.documents.get(id, None) for id in
self.index.get(word)] def add(self, document):

    """

    Add a document string to the index

    """

    for token in [t.lower() for t in nltk.word_tokenize(document)]:

        if token in self.stopwords:

            continue

        if self.stemmer:

            token = self.stemmer.stem(token)

        if self.__unique_id not in self.index[token]:

            self.index[token].append(self.__unique_id)

    self.documents[self.__unique_id] = document

    self.__unique_id += 1
index = Index(nltk.word_tokenize,
        EnglishStemmer(),

        nltk.corpus.stopwords.words('english'))
```

INPUT

```
index.add('Achilles Last Stand')
index.add('Whole Lotta Love')
index.add('White Cat')

index.add('Over the Hills and Far Away') index.add('Dazed and Confused')


# queries:

print index.lookup('daze')

# ['Dazed and Confused']

print index.lookup('confusion')

# ['Dazed and Confused']


print index.lookup(‘White’)


#[‘White Cat’]
```

OUTPUT

['Dazed and Confused']

['Dazed and Confused']

['White Cat']

RESULT AND DISCUSSION

The Python program to build inverted index for a given set of document collection was executed successfully.

EXPERIMENT – 3

IMPLEMENTATION OF TEXT CLASSIFICATION TECHNIQUE

OBJECTIVE

To train a linear model to perform text classification using Scikit-learn.

TEXT CLASSIFICATION

Text classification is one of the important and typical task in *supervised* machine learning (ML). Assigning categories to documents, which can be a web page, library book, media articles, gallery etc. has many applications like e.g. spam filtering, email routing, sentiment analysis etc.

It is possible to divide the classification problem into below steps:

1. Prerequisite and setting up the environment.
2. Loading the data set in jupyter.
3. Extracting features from text files.
4. Running ML algorithms.

Step 1: Prerequisite and setting up the environment

Install anaconda that provides both python and jupyter notebook

Step 2: **Loading the data set in jupyter.**

The data set used here is “20 Newsgoup” data set which is built in sci-kit.

Loading the data set: (this might take few minutes, so patience)

```
from sklearn.datasets import fetch_20newsgroups
twenty_train = fetch_20newsgroups(subset='train',
...     categories=categories, shuffle=True, random_state=42)
```

Step 3:Extracting features from text files

Scikit-learn has a high level component which will create feature vectors for us 'CountVectorizer'.

```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(twenty_train.data)
```

TF (**Term Frequencies**): #count(word) / #Total words, in each document.

TF-IDF: Finally, we can even reduce the weightage of more common words like (the, is, an etc.) which occurs in all document.
We can achieve both using below line of code:

```
from sklearn.feature_extraction.text import TfidfTransformer
tf_transformer = TfidfTransformer(use_idf=False).fit(X_train_counts)
X_train_tf = tf_transformer.transform(X_train_counts)
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
X_train_tfidf.shape
```

Step 4:Running ML algorithm

Naïve-Bayes classifier

Naive Bayes classifiers are simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features. Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Naive Bayes models are known under a variety of names, including simple Bayes and independence Bayes. All these names reference the use of Bayes' theorem in the classifier's decision rule, but naive Bayes is not (necessarily) a Bayesian method.

It is easy build a NBclassifier in scikit using below 2 lines of code:

```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(X_train_tfidf, twenty_train.target)
```

Performance of NB Classifier: Now we will test the performance of the NB classifier on **test set**.

```
import numpy as np
twenty_test = fetch_20newsgroups(subset='test', shuffle=True)
predicted = text_clf.predict(twenty_test.data)
np.mean(predicted == twenty_test.target)
```

PROGRAM

```
 categories = ['alt.atheism', 'soc.religion.christian',
...              'comp.graphics', 'sci.med', 'rec.sport.football', 'sci.space']



 from sklearn.datasets import fetch_20newsgroups
 twenty_train = fetch_20newsgroups(subset='train',
...     categories=categories, shuffle=True, random_state=42)
 from sklearn.feature_extraction.text import CountVectorizer
 count_vect = CountVectorizer()
 X_train_counts = count_vect.fit_transform(twenty_train.data)
 from sklearn.feature_extraction.text import TfidfTransformer
 tf_transformer = TfidfTransformer(use_idf=False).fit(X_train_counts)
 X_train_tf = tf_transformer.transform(X_train_counts)
 tfidf_transformer = TfidfTransformer()
 X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
 X_train_tfidf.shape


from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(X_train_tfidf, twenty_train.target
 docs_new = ['Infections', 'sports"]
 X_new_counts = count_vect.transform(docs_new)
 X_new_tfidf = tfidf_transformer.transform(X_new_counts)
```

```
predicted = clf.predict(X_new_tfidf)

for doc, category in zip(docs_new, predicted):
...     print('%r => %s' % (doc, twenty_train.target_names[category]))
...
```

OUTPUT

'Infections' => sci.med

‘Sports’=> 'rec.sport.football'

RESULT AND DISCUSSION

The implementation of text classification using naïve-Bayes classifier was done successfully.

EXPERIMENT – 4

IMPLEMENTATION OF TEXT CLUSTERING TECHNIQUE

OBJECTIVE

To use Scikit-Learn to cluster documents by topics using a bag-of-words approach (k-means clustering technique).

TEXT CLUSTERING

Document clustering (or **text clustering**) is the application of cluster analysis to textual documents. It has applications in automatic document organization, topic extraction and fast information retrieval or filtering**.** Text clustering can be viewed as a task of grouping a set of texts in such a way that texts in the same group, called a cluster, are more similar to each other than to those in other clusters.

.

K-MEANS CLUSTERING ALGORITHM

k-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. *k*-means clustering aims to partition *n* observations into *k* clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.. The problem is computationally difficult (NP-hard); however, there are efficient heuristic algorithms that are commonly employed and converge quickly to a local optimum. These are usually similar to the expectation-maximization algorithm for mixtures of Gaussian distributions via an iterative refinement approach employed by both k-means and Gaussian Mixture Modeling. Additionally, they both use cluster centers to model the data; however, k-means clustering tends to find clusters of comparable spatial extent, while the expectation-maximization mechanism allows clusters to have different shapes. In cluster analysis, the *k*-means algorithm can be used to partition the input data set into *k* partitions (clusters).

Algorithmic steps for k-means clustering

1. Randomly select *'c'* cluster centers.
2. Calculate the distance between each data point and cluster centers.
3. Assign the data point to the cluster center whose distance from the cluster center is minimum of all the cluster centers.
4. Recalculate the new cluster center
5. Recalculate the distance between each data point and new obtained cluster centers.
6. If no data point was reassigned then stop, otherwise repeat from step 3.

PROGRAM

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score

documents = ["A little puppy came to play when I was  at a park.",
        "Merley has the best squooshy kitten belly.",
        "Google Music app is incredible.",
        "If you open 100 tab in google you get a smiley face.",
        "Best cat photo I've ever taken.",
        "Climbing ninja tiger.",
```

```
        "Impressed with google map feedback.",
        "Key promoter extension for Google Chrome.",
       ]

vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(documents)

true_k = 3
model = KMeans(n_clusters=true_k, init='k-means++', max_iter=100, n_init=1)
model.fit(X)

print("Top terms per cluster:")
order_centroids = model.cluster_centers_.argsort()[:, ::-1]
terms = vectorizer.get_feature_names()
for i in range(true_k):
    print("Cluster %d:" % i),
    for ind in order_centroids[i, :10]:
        print(' %s' % terms[ind]),
    print


print("\n")
print("Prediction")
 Y = vectorizer.transform(["Google is a search engine."])
prediction = model.predict(Y)
```

```
print(prediction)


Y = vectorizer.transform(["Best wishes"])

prediction = model.predict(Y)

print(prediction)
```

OUTPUT

```
Top terms per cluster:
Cluster 0: kitten belly squooshy merley eating park came play puppy little
Cluster
1: google feedback map app impressed incredible music key extension chrome
Cluster 2: tiger ninja climbing  photo taken best came belly chrome


Prediction
[1]
[2]
```

RESULT AND DISCUSSION

The implementation of text clustering technique using k-means algorithm was done successfully

EXPERIMENT – 5

IMPLEMENTATION OF PAGE RANK ALGORITHM

OBJECTIVE

To implement Page Rank algorithm using Scikit-learn.

PAGE RANK ALGORITHM

PageRank is an algorithm used by Google Search to rank websites in their search engine results. PageRank is a link analysis algorithm and it assigns a numerical weighting to each element of a hyperlinked set of documents, such as the World Wide Web, with the purpose of "measuring" its relative importance within the set. The algorithm may be applied to any collection of entities with reciprocal quotations and references. The numerical weight that it

assigns to any given element *E* is referred to as the *PageRank of E* and denoted by PR(E). Other factors like *Author Rank* can contribute to the importance of an entity. The PageRank algorithm outputs a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page. PageRank can be calculated for collections of documents of any size. It is assumed in several research papers that the distribution is evenly divided among all documents in the collection at the beginning of the computational process. The PageRank computations require several passes, called "iterations", through the collection to adjust approximate PageRank values to more closely reflect the theoretical true value.

A PageRank results from a mathematical algorithm based on the web graph, created by all World Wide Web pages as nodes and hyperlinks as edges, taking into consideration authority hubs. The rank value indicates an importance of a particular page. A hyperlink to a page counts as a vote of support. The PageRank of a page is defined recursively and depends on the number and PageRank metric of all pages that link to it ("incoming links"). A page that is linked to by many pages with high PageRank receives a high rank itself.

PROGRAM

```
import numpy as np

from scipy.sparse import csc_matrix

def pageRank(G, s = .85, maxerr = .001):

    """

    Computes the pagerank for each of the n states.

    Used in webpage ranking and text summarization using unweighted

    or weighted transitions respectively.

    Args

    ----------

    G: matrix representing state transitions

    Gij can be a boolean or non negative real number representing
    the transition weight from state i to j. Kwargs

    s: probability of following a transition. 1-s probability
    of teleporting to another state. Defaults to 0.85

    maxerr: if the sum of pageranks between iterations is
    bellow this we will have converged. Defaults to 0.001
```

```
"""
   n = G.shape[0]
   # transform G into markov
   matrix M M =
   csc_matrix(G,dtype=np.float
   ) rsums =
   np.array(M.sum(1))[:,0]

   ri, ci =
   M.nonzero()
   M.data /=
   rsums[ri]

   # bool array of sink
   states sink = rsums==0

   # Compute pagerank r until we
   converge ro, r = np.zeros(n),
   np.ones(n)

   while np.sum(np.abs(r-ro))
      >maxerr: ro = r.copy()

      # calculate each pagerank at a time

      for i in xrange(0,n):

         # inlinks of state i

         Ii = np.array(M[:,i].todense())[:,0]
```

```
        # account for sink
        states Si = sink /
        float(n)

        # account for teleportation to
        state i Ti = np.ones(n) / float(n)

        r[i] = ro.dot( Ii*s + Si*s + Ti*(1-s) )

    # return normalized pagerank

    return r/sum(r)
if __name__=='__main__':

    # Example extracted from 'Introduction to Information
    Retrieval' G = np.array([[1,0,1,0,1,0,0],

[0,1,1,0,0,0,0],

[1,0,1,1,0,0,0],

[0,0,0,1,1,0,0],

[0,0,0,0,0,0,1],

[0,0,0,0,0,1,1],

[0,0,0,1,1,0,1]])

    print pageRank(G,s=.86)
```

OUTPUT

[0.07625602 0.03093333 0.10222139 0.20927014 0.34157375 0.03974417 0.20000121]

RESULT AND DISCUSSION

The implementation of Page Rank algorithm was done successfully.

EXPERIMENT – 6

SETTING UP A HADOOP CLUSTER

This experiment illustrates how to set up a single node Hadoop cluster.

APACHE HADOOP

Apache Hadoop is a collection of open-source software utilities that facilitate using a network of many computers to solve problems involving massive amounts of data and computation. It provides a software framework for distributed storage and processing of big data using the Map Reduce programming model. Originally designed for computer clusters built from commodity hardware[1]—still the common use—it has also found use on clusters of higher-end hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common occurrences and should be automatically handled by the framework.

The core of Apache Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part which is a MapReduce programming mod…….el.
Hadoop splits files into large blocks and distributes them across nodes in a cluster. It then transfers packaged code into nodes to process the data in parallel. This approach takes advantage of data locality, where nodes manipulate the data they have access to. This allows the dataset to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking. The term Hadoop has come to refer not just to the aforementioned base modules and sub-modules, but also to the ecosystem, or collection of additional software packages that can be installed on top of or alongside Hadoop, such as Apache Pig, Apache Hive, Apache HBase, Apache Phoenix, Apache Spark, Apache ZooKeeper, Cloudera Impala, Apache Flume, Apache Sqoop, Apache Oozie, and Apache Storm.

Apache Hadoop's MapReduce and HDFS components were inspired by Google papers on their MapReduce and Google File System.

HADOOP ARCHITECTURE

Hadoop follows a master slave architecture design for data storage and distributed data processing using HDFS and MapReduce respectively. The master node for data storage is hadoop HDFS is the NameNode and the master node for parallel processing of data using Hadoop MapReduce is the Job Tracker. The slave nodes in the hadoop architecture are the other machines in the Hadoop cluster which store data and perform complex computations. Every slave node has a Task Tracker daemon and a DataNode that synchronizes the processes with the Job Tracker and NameNode respectively. In Hadoop architectural implementation the master or slave systems can be setup in the cloud or on-premise.

Hadoop consists of the Hadoop Common package, which provides file system and operating system level abstractions, a MapReduce engine (either MapReduce/MR1 or YARN/MR2) and the Hadoop Distributed File System (HDFS). The Hadoop Common package contains the Java ARchive (JAR) files and scripts needed to start Hadoop.

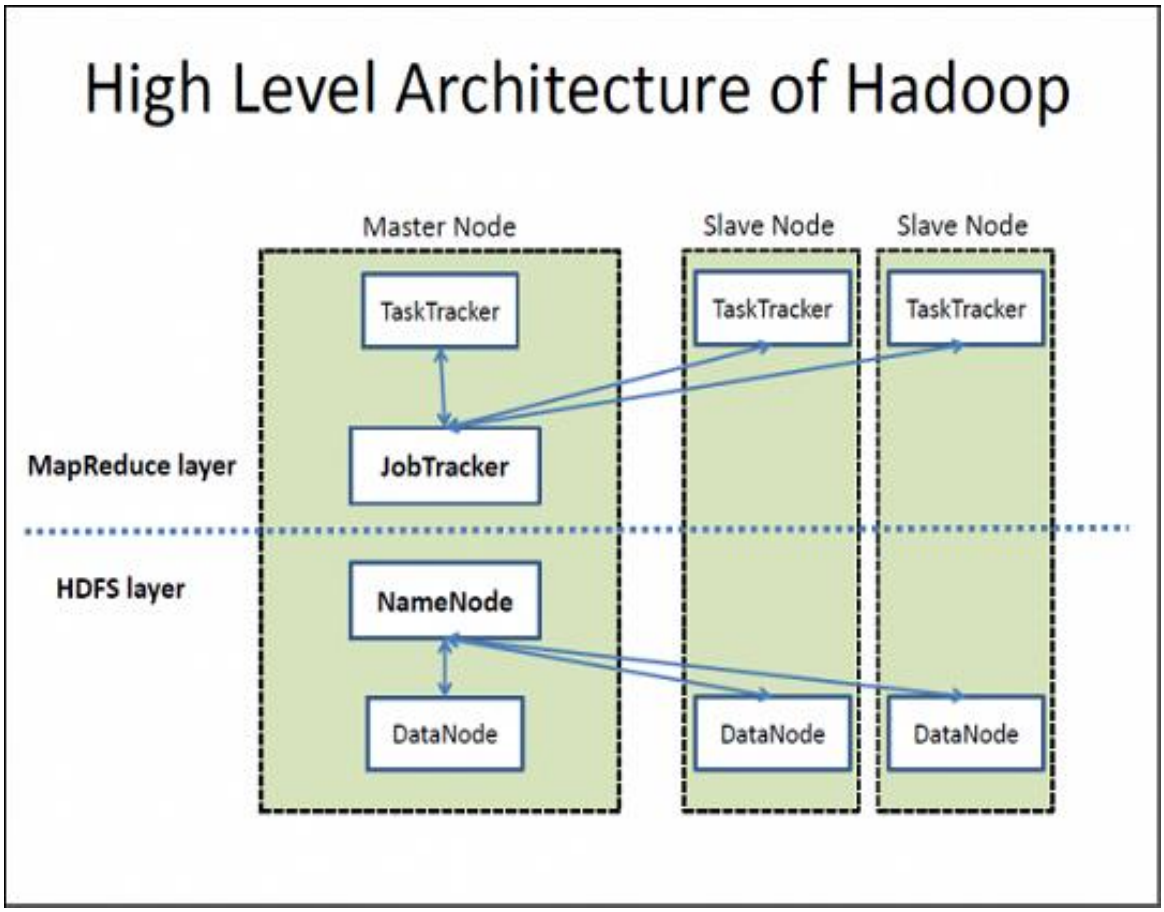

Fig. 1: Hadoop Architecture

HADOOP DISTRIBUTED FILE SYSTEM

A file on HDFS is split into multiple bocks and each is replicated within the Hadoop cluster. A block on HDFS is a blob of data within the underlying file system with a default size of 64MB.The size of a block can be extended up to 256 MB based on the requirements. It achieves reliability by replicating the data across multiple hosts, and hence theoretically does not require RAID storage on hosts. With the default replication value, 3, data is stored on three nodes: two on the same rack, and one on a different rack. Hadoop Distributed File System (HDFS) stores the application data and file system metadata separately on dedicated servers. NameNode and DataNode are the two critical components of the Hadoop HDFS architecture. Application data is stored on servers referred to as DataNodes and file system metadata is stored on servers referred to as NameNode. The NameNode and DataNode communicate with each other using TCP based protocols.

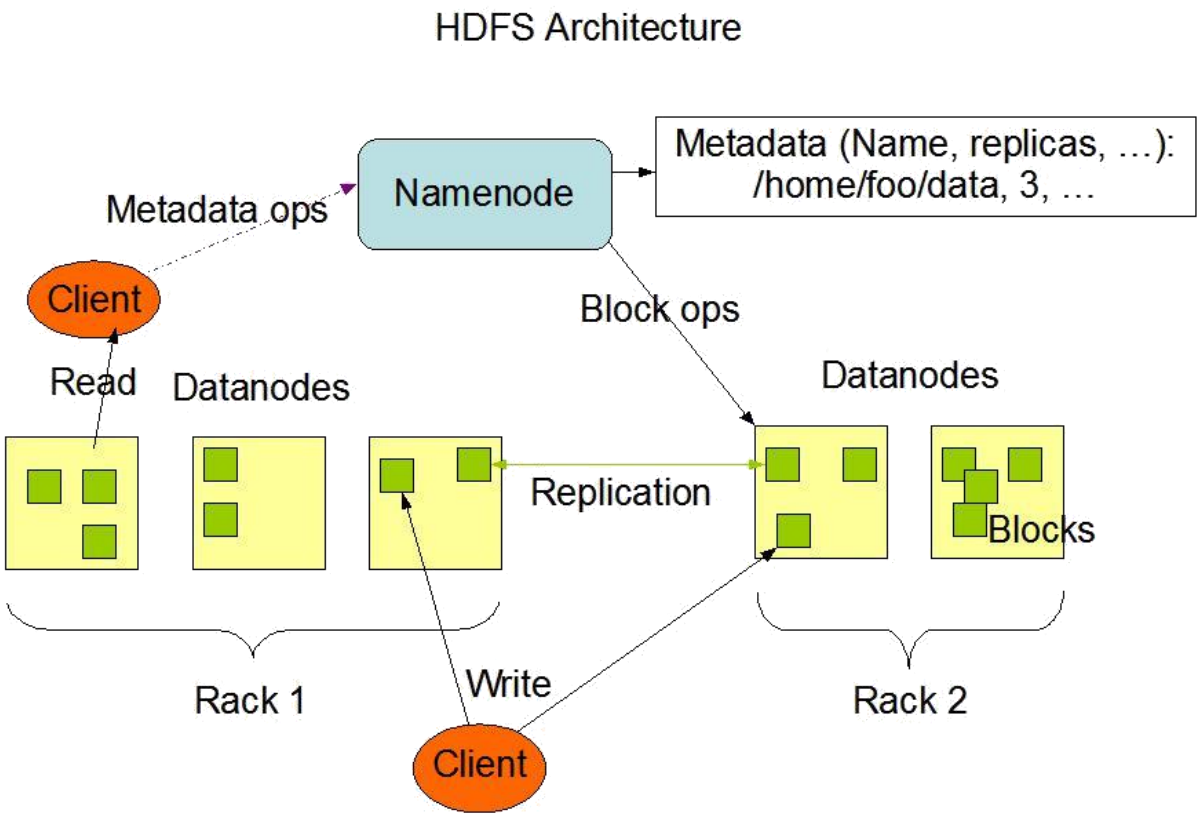

Fig. 2: HDFS Architecture

HOW TO SET UP A HADOOP CLUSTER

Step 1: Prerequisites

- **Sun Java 6**

  Hadoop requires a working Java 1.5+ installation. The following instructions are used for the installation purpose.

```
$ sudo apt-get update

$ sudo apt-get install sun-java6-jdk

$ sudo update-java-alternatives -s java-6-sun
```

  After installation, make a quick check whether Sun's JDK is correctly set up:

```
mtechstudents@ubuntu:~# java –version

java version "1.6.0_20"

Java(TM) SE Runtime Environment (build 1.6.0_20-b02)

Java HotSpot(TM) Client VM (build 16.3-b01, mixed mode, sharing)
```

- **Adding a dedicated Hadoop system user**

  Then we need to add a dedicated user and also create a group named hadoop because it helps to separate the Hadoop installation from other software applications.

```
$ sudo addgroup hadoop
```

```
$ sudo adduser --ingroup hadoop hduser
```

This will add the user hduser and the group hadoop to our local machine.

- **Configuring SSH**

Hadoop requires SSH access to manage its nodes, i.e. remote machines plus our local machine if we want to use Hadoop on it. For our single-node setup of Hadoop, we therefore need to configure SSH access to localhost for the hduser user.

First, we have to generate an SSH key for the hduser user.

```
mtechstudents@ubuntu:~$ su - hduser
hduser@ubuntu:~$ ssh-keygen -t rsa -P
"" Generating public/private rsa key
pair.
Enter file in which to save the key
(/home/hduser/.ssh/id_rsa):

Created directory '/home/hduser/.ssh'.

Your identification has been saved in
/home/hduser/.ssh/id_rsa.

Your public key has been saved in
/home/hduser/.ssh/id_rsa.pub.

The key fingerprint is:

9b:82:ea:58:b4:e0:35:d7:ff:19:66:a6:ef:ae:0e:d2
hduser@ubuntu

The key's randomart image is:
```

```
[...snipp...]

hduser@ubuntu:~$
```

Second, we have to enable SSH access to our local machine with this newly created key.

```
hduser@ubuntu:~$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

The final step is to test the SSH setup by connecting to your local machine with the hduser user.

Step 2: Hadoop Installation

Download Hadoop from the Apache Download Mirrors and extract the contents of the Hadoop package to a location of our choice(/usr/local/hadoop). Make sure to change the owner of all the files to the hduser user and hadoop group, for example:

```
$ cd /usr/local

$ sudo tar xzf hadoop-1.2.1.tar.gz

$ sudo mv hadoop-1.2.1 hadoop

$ sudo chown -R hduser:hadoop hadoop
```

Step 3: Configuration

- **hadoop-env.sh**

  Open conf/hadoop-env.sh in the editor and set the JAVA_HOME environment variable to the Sun JDK/JRE 6 directory.

Change

```
# export JAVA_HOME=/usr/lib/j2sdk1.5-sun
```

To

```
export JAVA_HOME=/usr/lib/jvm/java-6-sun
```

- **conf/*-site.xml**

  In this section, we will configure the directory where Hadoop will store its data files, the network ports it listens to, etc. Our setup will use Hadoop's Distributed File System, HDFS, even though our little "cluster" only contains our single local machine.

  Now we create the directory and set the required ownerships and permissions:

```
$ sudo mkdir -p /app/hadoop/tmp
```

```
$ sudo chown hduser:hadoop /app/hadoop/tmp
```

```
# ...and if you want to tighten up security, chmod from 755 to
  750...
```

```
$ sudo chmod 750 /app/hadoop/tmp
```

  Add the following snippets between the <configuration> ... </configuration> tags in the respective configuration XML file.

  In file **conf/core-site.xml**:

```
<property>

  <name>hadoop.tmp.dir</name>
```

```
  <value>/app/hadoop/tmp</value>

  <description>A base for other temporary
  directories.</description>

</property>

<property>

  <name>fs.default.name</name>

  <value>hdfs://localhost:54310</value>

  <description>The name of the default file system. A URI
  whose scheme and authority determine the FileSystem
  implementation. The uri's scheme determines the config
  property (fs.SCHEME.impl) naming the FileSystem
  implementation class. The uri's authority is used to
  determine the host, port, etc. for a
  filesystem.</description>

</property>
```

In file **conf/mapred-site.xml**:

```
<property>

  <name>mapred.job.tracker</name>

  <value>localhost:54311</value>

  <description>The host and port that the MapReduce job
  tracker runs at. If "local", then jobs are run in-process
  as a single map and reduce task.
```

```
  </description>

</property>
```

In file **conf/hdfs-site.xml**:

```
<property>

  <name>dfs.replication</name>

  <value>1</value>

  <description>Default block replication.

  The actual number of replications can be specified when
the file is created.

  The default is used if replication is not specified in
  create time. </description>

</property>
```

Step 4: Formatting the HDFS filesystem via the NameNode To format the filesystem, run the command

```
hduser@ubuntu:~$ /usr/local/hadoop/bin/hadoop namenode –format
```

The output will look like this:

```
hduser@ubuntu:/usr/local/hadoop$ bin/hadoop namenode -format
10/05/08 16:59:56 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
```

```
STARTUP_MSG: host = ubuntu/127.0.1.1

STARTUP_MSG: args = [-format]

STARTUP_MSG: version = 0.20.2

STARTUP_MSG: build =

https://svn.apache.org/repos/asf/hadoop/common/branches/branch-
0.20 -r

911707; compiled by 'chrisdo' on Fri Feb 19 08:07:34 UTC 2010
************************************************************/ 10/05/08
16:59:56 INFO namenode.FSNamesystem: fsOwner=hduser,hadoop 10/05/08
16:59:56 INFO namenode.FSNamesystem: supergroup=supergroup 10/05/08
16:59:56 INFO namenode.FSNamesystem: isPermissionEnabled=true

10/05/08 16:59:56 INFO common.Storage: Image file of size 96
saved in 0 seconds.

10/05/08 16:59:57 INFO common.Storage: Storage directory .../hadoop-
hduser/dfs/name has been successfully formatted.

10/05/08 16:59:57 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at ubuntu/127.0.1.1

************************************************************/
hduser@ubuntu:/usr/local/hadoop$
```

Step 5: Starting single-node cluster

Run the command:

```
hduser@ubuntu:~$ /usr/local/hadoop/bin/start-all.sh
```

This will startup a Namenode, Datanode, Jobtracker and a Tasktracker on your machine.

The output will look like this:

```
hduser@ubuntu:/usr/local/hadoop$ bin/start-all.sh

starting namenode, logging to
/usr/local/hadoop/bin/../logs/hadoop-hduser-namenode-ubuntu.out

localhost: starting datanode, logging to
/usr/local/hadoop/bin/../logs/hadoop-hduser-datanode-
ubuntu.out

localhost: starting secondarynamenode, logging to
/usr/local/hadoop/bin/../logs/hadoop-hduser-secondarynamenode-ubuntu.out

starting jobtracker, logging to
/usr/local/hadoop/bin/../logs/hadoop-hduser-jobtracker-
ubuntu.out

localhost: starting tasktracker, logging to
/usr/local/hadoop/bin/../logs/hadoop-hduser-tasktracker-ubuntu.out

hduser@ubuntu:/usr/local/hadoop$

hduser@ubuntu:/usr/local/hadoop$ jps

2287 TaskTracker

2149 JobTracker

1938 DataNode

2085 SecondaryNameNode
```

```
2349 Jps

1788 NameNode
```

OUTPUT

```
2287 TaskTracker

2149 JobTracker

1938 DataNode

2085 SecondaryNameNode

2349 Jps

1788 NameNode
```

RESULT AND DISCUSSION

The setting up of a single node Hadoop cluster was done successfully.

EXPERIMENT – 7

DEVELOPING A MAPREDUCE APPLICATION

OBJECTIVE

To develop a java MapReduce application to count the number of words in an input file in hadoop cluster.

MAPREDUCE FRAMEWORK

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

The Algorithm

- Generally MapReduce paradigm is based on sending the computer to where the data resides!
- MapReduce program executes in three stages, namely map stage, shuffle stage, and reduce stage.
    - **Map stage** : The map or mapper's job is to process the input data. Generally the input data is in the form of file or directory and is stored in

the Hadoop file system (HDFS). The input file is passed to the mapper function line by line. The mapper processes the data and creates several small chunks of data.

- **Reduce stage** : This stage is the combination of the **Shuffle**stage and the **Reduce** stage. The Reducer's job is to process the data that comes from the mapper. After processing, it produces a new set of output, which will be stored in the HDFS.

- During a MapReduce job, Hadoop sends the Map and Reduce tasks to the appropriate servers in the cluster.
- The framework manages all the details of data-passing such as issuing tasks, verifying task completion, and copying data around the cluster between the nodes.
- Most of the computing takes place on nodes with data on local disks that reduces the network traffic.
- After completion of the given tasks, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.

Inputs and Outputs

The MapReduce framework operates on <key, value> pairs, that is, the framework views the input to the job as a set of <key, value> pairs and produces a set of <key, value> pairs as the output of the job, conceivably of different types.

The key and the value classes should be in serialized manner by the framework and hence, need to implement the Writable interface. Additionally, the key classes have to implement the Writable-Comparable interface to facilitate sorting by the framework. Input and Output types of a MapReduce job: (Input) <k1, v1> -> map -> <k2, v2>-> reduce -> <k3, v3>(Output).

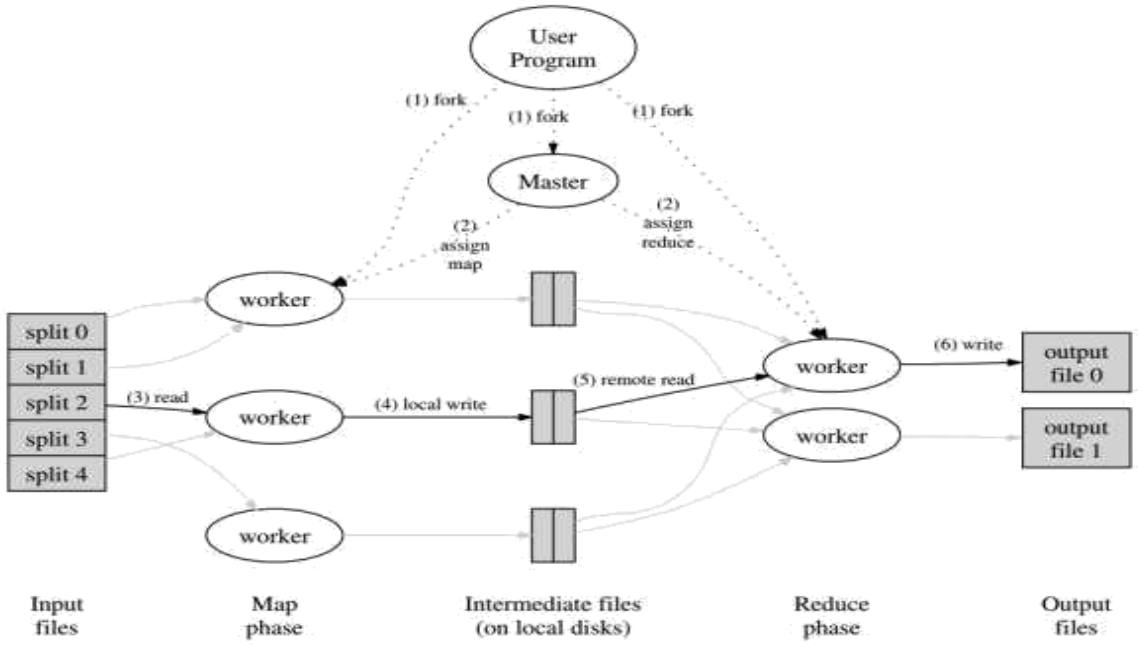

Fig. 1: MapReduce Framework

WORD COUNT EXAMPLE AS A MAPREDUCE APPLICATION

Word count program is a simple MapReduce application program which is intended to count the number of occurrences of each word in the provided input files. This works with a local-standalone, pseudo-distributed or fully-distributed Hadoop installation

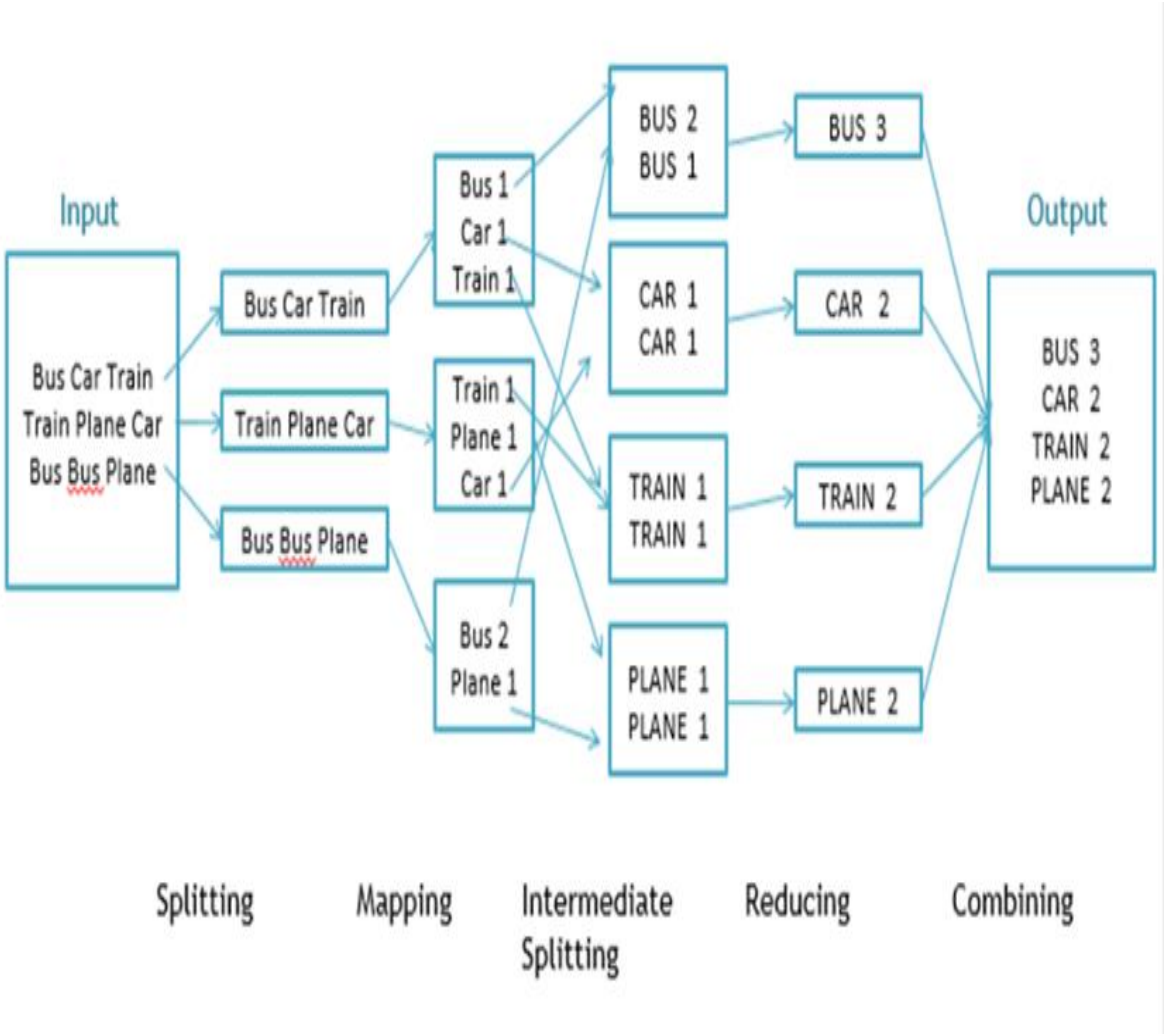

Figure Word count using map-reduce

Workflow of MapReduce consists of 5 steps

1. **Splitting** – The splitting parameter can be anything, e.g. splitting by space, comma, semicolon, or even by a new line ('\n').
2. **Mapping** – as explained above
3. **Intermediate splitting** – the entire process in parallel on different clusters. In order to group them in "Reduce Phase" the similar KEY data should be on same cluster.
4. **Reduce** – it is nothing but mostly group by phase
5. **Combining** – The last phase where all the data (individual result set from each cluster) is combine together to form a Result

PROGRAM

```
package PackageDemo;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
public class WordCount {
public static void main(String [] args) throws Exception
{
Configuration c=new Configuration();
String[] files=new GenericOptionsParser(c,args).getRemainingArgs();
Path input=new Path(files[0]);
Path output=new Path(files[1]);
Job j=new Job(c,"wordcount");
j.setJarByClass(WordCount.class);
j.setMapperClass(MapForWordCount.class);
j.setReducerClass(ReduceForWordCount.class);
j.setOutputKeyClass(Text.class);
j.setOutputValueClass(IntWritable.class);
```

```
FileInputFormat.addInputPath(j, input);

FileOutputFormat.setOutputPath(j, output);

System.exit(j.waitForCompletion(true)?0:1);

}

public static class MapForWordCount extends Mapper<LongWritable, Text, Text, IntWritable>{

public void map(LongWritable key, Text value, Context con) throws IOException, InterruptedException

{

String line = value.toString();

String[] words=line.split(",");

for(String word: words )

{

    Text outputKey = new Text(word.toUpperCase().trim());

 IntWritable outputValue = new IntWritable(1);

 con.write(outputKey, outputValue);

}

}

}


public static class ReduceForWordCount extends Reducer<Text, IntWritable, Text, IntWritable>

{

public void reduce(Text word, Iterable<IntWritable> values, Context con) throws IOException, InterruptedException

{

int sum = 0;

  for(IntWritable value : values)

  {

  sum += value.get();
```

```
  }
  con.write(word, new IntWritable(sum));
}
}
}
```

The program consist of 3 classes:

- Driver class (Public void static main- the entry point)
- Map class which **extends** public class Mapper<KEYIN,VALUEIN,KEYOUT,VALUEOUT> and implements the Map function.
- Reduce class which extends public class Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT> and implements the Reduce function**.**

INPUT

A file named wordcount.txt with the following contents:

OUTPUT

SUCCESS

BUS 7

CAR 4

TRAIN 6

RESULT AND DISCUSSION

A simple MapReduce application, Word count program was executed successfully.

EXPERIMENT 8

WORKING WITH A DISTRIBUTED DATA STORAGE (HBASE)

OBJECTIVE

To work with a distributed NoSQL data storage model, namely, HBase and to study its working in a standalone mode.

HBASE

HBase is a distributed column-oriented database built on top of the Hadoop file system. It is an open-source project and is horizontally scalable. HBase is a data model that is similar to Google's big table designed to provide quick random access to huge amounts of structured data. It leverages the fault tolerance provided by the Hadoop File System (HDFS). It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System. One can store the data in HDFS either directly or through HBase. Data consumer reads/accesses the data in HDFS randomly using HBase. HBase sits on top of the Hadoop File System and provides read and write access.

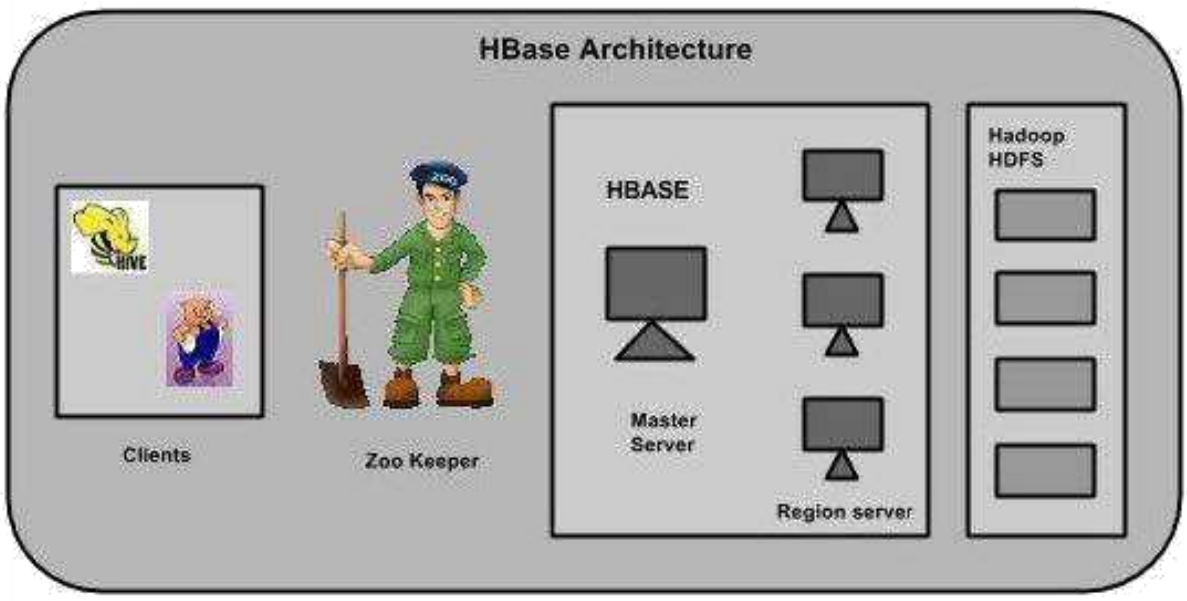

Fig. 1: HBase Architecture

HBASE INSTALLATION

Open the Linux terminal and type the following commands to create a user.

```
$ su
password:
# useradd hadoop
```

```
# passwd  hadoop
New      passwd:
Retype       new
passwd
```

SSH setup is required to perform different operations on the cluster such as start, stop, and distributed daemon shell operations. To authenticate different users of Hadoop, it is required to provide public/private key pair for a Hadoop user and share it with different users.

The following commands are used to generate a key value pair using SSH. Copy the public keys from id_rsa.pub to authorized_keys, and provide owner, read and write permissions to authorized_keys file respectively.

```
$ ssh-keygen -t rsa
```

```
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 0600 ~/.ssh/authorized_keys
```

Verify ssh using

```
ssh localhost
```

then install java and hadoop.

We can install HBase in any of the three modes: Standalone mode, Pseudo Distributed mode, and Fully Distributed mode.

Installing HBase in Standalone Mode

```
$cd usr/local/

$wget      http://www.interior-dsgn.com/apache/hbase/stable/hbase-
0.98.8-hadoop2-bin.tar.gz

$tar -zxvf hbase-0.98.8-hadoop2-bin.tar.gz
```

Shift to super user mode and move the HBase folder to /usr/local as shown below.

```
$su

$password:  enter  your  password
here mv hbase-0.99.1/* Hbase/
```

Configuring HBase in Standalone Mode

- **hbase-env.sh**

  Set the java Home for HBase and open **hbase-env.sh** file from the conf folder. Edit JAVA_HOME environment variable and change the existing path to the current JAVA_HOME variable as shown below.

```
cd
/usr/local/Hbase/conf
gedit hbase-env.sh
```

This will open the env.sh file of HBase. Now replace the existing **JAVA_HOME** value with a current value as shown below.

```
export JAVA_HOME=/usr/lib/jvm/java-1.7.0
```

- **hbase-site.xml**

This is the main configuration file of HBase. Set the data directory to an appropriate location by opening the HBase home folder in /usr/local/HBase. Inside the conf folder, we will find several files, open the **hbase-site.xml** file as shown below.

```
#cd /usr/local/HBase/

#cd conf
# gedit hbase-site.xml
```

Inside the **hbase-site.xml** file, we will find the <configuration> and </configuration> tags. Within them, set the HBase directory under the property key with the name "hbase.rootdir" as shown below.

```
<configuration>

  //Here we have to set the path where we want HBase to store its files.

  <property>

    <name>hbase.rootdir</name>

    <value>file:/home/hadoop/HBase/HFiles</value>
```

```
</property>
```

//Here we have to set the path where we want HBase to store its built in zookeeper files.

```
<property>
  <name>hbase.zookeeper.property.dataDir</name>
  <value>/home/hadoop/zookeeper</value>
</property>
</configuration>
```

With this, the HBase installation and configuration part is successfully complete. We can start HBase by using **start-hbase.sh** script provided in the bin folder of HBase. For that, open HBase Home Folder and run HBase start script as shown below.

```
$cd /usr/local/HBase/bin
$./start-hbase.sh
```

HBASE SHELL

HBase contains a shell using which we can communicate with HBase. HBase uses the Hadoop File System to store its data. It will have a master server and region servers. The data storage will be in the form of regions (tables). These regions will be split up and stored in region servers. The master server manages these region servers and all these tasks take place on HDFS.

Given below are some of the commands supported by HBase Shell.

General Commands

- **status** - Provides the status of HBase, for example, the number of servers. `hbase(main):009:0> status`
- **version** - Provides the version of HBase being used. `hbase(main):009:0> version`
- **table_help** - Provides help for table-reference commands. `hbase(main):02:0> table_help`
- **whoami** - Provides information about the user.

```
hbase(main):008:0> whoami
```

Data Definition Language

These are the commands that operate on the tables in HBase.

- **create** - Creates a table.
- **list** - Lists all the tables in HBase.

- **disable** - Disables a table.
- **is_disabled** - Verifies whether a table is disabled.
- **enable** - Enables a table.
- **is_enabled** - Verifies whether a table is enabled.
- **describe** - Provides the description of a table.
- **alter** - Alters a table.
- **exists** - Verifies whether a table exists.
- **drop** - Drops a table from HBase.
- **drop_all** - Drops the tables matching the 'regex' given in the command.
- **Java Admin API** - Prior to all the above commands, Java provides an Admin API to

  achieve DDL functionalities through programming. Under **org.apache.hadoop.hbase.client** package, HBaseAdmin and HTableDescriptor are the two important classes in this package that provide DDL functionalities.

Data Manipulation Language

- **put** - Puts a cell value at a specified column in a specified row in a particular table.
- **get** - Fetches the contents of row or a cell.
- **delete** - Deletes a cell value in a table.
- **deleteall** - Deletes all the cells in a given row.
- **scan** - Scans and returns the table data.
- **count** - Counts and returns the number of rows in a table.
- **truncate** - Disables, drops, and recreates a specified table.

- **Java client API** - Prior to all the above commands, Java provides a client API to achieve DML functionalities, **CRUD** (Create Retrieve Update Delete) operations and more through programming, under org.apache.hadoop.hbase.client package. **HTable Put** and **Get** are the important classes in this package.

HBASE - ADMIN API

HBase is written in java, therefore it provides java API to communicate with HBase. Java API is the fastest way to communicate with HBase. Given below is the referenced java Admin API that covers the tasks used to manage tables.

Class HBaseAdmin

HBaseAdmin is a class representing the Admin. This class belongs to the **org.apache.hadoop.hbase.client** package. Using this class, we can perform the tasks of an administrator. We can get the instance of Admin using **Connection.getAdmin()** method.

Methods and Description

S.No.	Methods and Description
1	**void createTable(HTableDescriptor desc)** Creates a new table.
2	**void createTable(HTableDescriptor desc, byte[ ][ ] splitKeys)** Creates a new table with an initial set of empty regions defined by the specified split keys.

3	**void deleteColumn(byte[] tableName, String columnName)** Deletes a column from a table.
4	**void deleteColumn(String tableName, String columnName)** Delete a column from a table.
5	**void deleteTable(String tableName)** Deletes a table.

Class Descriptor

This class contains the details about an HBase table such as:

- the descriptors of all the column families,
- if the table is a catalog table,
- if the table is read only,
- the maximum size of the mem store,
- when the region split should occur,
- co-processors associated with it, etc.

Constructors

S.No.	Constructor and summary
1	**HTableDescriptor(TableName name)**

Constructs a table descriptor specifying a TableName object.

Methods and Description

S.No.	Methods and Description
1	**HTableDescriptor addFamily(HColumnDescriptor family)** Adds a column family to the given descriptor

Creating a Table using HBase Shell

We can create a table using the **create** command, here we must specify the table name and the Column Family name. The **syntax** to create a table in HBase shell is shown below.

```
create '<table name>','<column family>'
```

Listing a Table using HBase Shell

list is the command that is used to list all the tables in HBase. Given below is the syntax of the list command.

```
hbase(main):001:0 > list
```

Disabling a Table using HBase Shell

To delete a table or change its settings, we need to first disable the table using the disable command. We can re-enable it using the enable command.

Given below is the syntax to disable a table:

```
disable ‘table name’
```

Enabling a Table using HBase Shell

Syntax to enable a table:

```
enable ‘table name’
```

HBase - Describe & Alter

1. **describe**

   This command returns the description of the table. Its syntax is as follows: `hbase> describe 'table name'`

2. **alter**

   Alter is the command used to make changes to an existing table.

   Given below is the syntax to change the maximum number of cells of a column family. `hbase> alter 't1', NAME ⇒ 'f1', VERSIONS ⇒ 5`

3. **Table Scope Operators**

   Below given is the syntax to make a table read only. `hbase>alter 't1', READONLY(option)`

   We can also remove the table scope operators. Given below is the syntax to remove

'MAX_FILESIZE' from emp table.

```
hbase> alter 't1', METHOD ⇒ 'table_att_unset', NAME ⇒ 'MAX_FILESIZE'
```

Using alter, we can also delete a column family. Given below is the syntax to delete a column family using alter.

```
hbase> alter ‘ table name ’, ‘delete’ ⇒ ‘ column family ’
```

Dropping a Table using HBase Shell

Using the **drop** command, we can delete a table. Before dropping a table, we have to disable it.

```
hbase(main):018:0> disable 'emp'
0 row(s) in 1.4580 seconds
```

```
hbase(main):019:0> drop 'emp'
0 row(s) in 0.3060 seconds
```

HBase - Shutting Down

1. **exit**

   We exit the shell by typing the exit command. `hbase(main):021:0> exit`

2. **Stopping HBase**

   To stop HBase, browse to the HBase home folder and type the following command.

```
./bin/stop-hbase.sh
```

HBASE - CLIENT API

The java client API for HBase is used to perform **CRUD** operations on HBase tables. HBase is written in Java and has a Java Native API. Therefore it provides programmatic access to Data Manipulation Language (DML).

Class HBase Configuration

Adds HBase configuration files to a Configuration. This class belongs to the **org.apache.hadoop.hbase** package.

S.No.	Methods and Description
1	**static org.apache.hadoop.conf.Configuration create()** This method creates a Configuration with HBase resources.

Class HTable

HTable is an HBase internal class that represents an HBase table. It is an implementation of table that is used to communicate with a single HBase table. This class belongs to the **org.apache.hadoop.hbase.client** class.

S.No.	Constructors and Description
1	**HTable()**
2	**HTable(TableName tableName, ClusterConnection connection, ExecutorService pool)** Using this constructor, we can create an object to access an HBase table.

S.No.	Methods and Description
1	**void close()** Releases all the resources of the HTable.
2	**void delete(Delete delete)**

:

3	Deletes the specified cells/row. **boolean exists(Get get)**

Using this method, we can test the existence of columns in the table, as specified by Get.

4

Result get(Get get)

Retrieves certain cells from a given row.

5

org.apache.hadoop.conf.Configuration getConfiguration()
Returns the Configuration object used by this instance.

6

TableName getName()

Returns the table name instance of this table.

7

HTableDescriptor getTableDescriptor() Returns the table descriptor for this table.

8

byte[] getTableName()
Returns the name of this table.

9

void put(Put put)

Using this method, we can insert data into the table.

Class Put

This class is used to perform Put operations for a single row. It belongs to the **org.apache.hadoop.hbase.client** package.

S.No.	Constructors and Description
1	**Put(byte[] row)** Using this constructor, we can create a Put operation for the specified row.
2	**Put(byte[] rowArray, int rowOffset, int rowLength)** Using this constructor, we can make a copy of the passed-in row key to keep local.
3	**Put(byte[] rowArray, int rowOffset, int rowLength, long ts)** Using this constructor, we can make a copy of the passed-in row key to keep local.
4	**Put(byte[] row, long ts)** Using this constructor, we can create a Put operation for the specified row, using a given timestamp.

S.No.	Methods and Description
1	**Put add(byte[] family, byte[] qualifier, byte[] value)** Adds the specified column and value to this Put operation.
2	**Put add(byte[] family, byte[] qualifier, long ts, byte[] value)** Adds the specified column and value, with the specified timestamp as its version to this Put operation.
3	**Put add(byte[] family, ByteBuffer qualifier, long ts, ByteBuffer value)** Adds the specified column and value, with the specified timestamp as its version to this Put operation.
4	**Put add(byte[] family, ByteBuffer qualifier, long ts, ByteBuffer value)** Adds the specified column and value, with the specified timestamp as its version to this Put operation.

Class Get

This class is used to perform Get operations on a single row. This class belongs to the **org.apache.hadoop.hbase.client** package.

S.No.	Constructor and Description

1	**Get(byte[] row)** Using this constructor, we can create a Get operation for the specified row.
2	**Get(Get get)**

S.No.	Methods and Description
1	**Get addColumn(byte[] family, byte[] qualifier)** Retrieves the column from the specific family with the specified qualifier.
2	**Get addFamily(byte[] family)** Retrieves all columns from the specified family.

Class Delete

This class is used to perform Delete operations on a single row. To delete an entire row, instantiate a Delete object with the row to delete. This class belongs to the **org.apache.hadoop.hbase.client** package.

S.No.	Constructor and Description
1	**Delete(byte[] row)**

Creates a Delete operation for the specified row.

2 **Delete(byte[] rowArray, int rowOffset, int rowLength)** Creates a Delete operation for the specified row and timestamp.

3 **Delete(byte[] rowArray, int rowOffset, int rowLength, long ts)**

	Creates a Delete operation for the specified row and timestamp.
4	**Delete(byte[] row, long timestamp)**

Creates a Delete operation for the specified row and timestamp.

S.No.	Methods and Description
1	**Delete addColumn(byte[] family, byte[] qualifier)** Deletes the latest version of the specified column.
2	**Delete addColumns(byte[] family, byte[] qualifier, long timestamp)**

Deletes all versions of the specified column with a timestamp less than or equal to the specified timestamp.

3 **Delete addFamily(byte[] family)**

Deletes all versions of all columns of the specified family.

4 **Delete addFamily(byte[] family, long timestamp)**

Deletes all columns of the specified family with a timestamp less than or equal to the specified timestamp.

Class Result

This class is used to get a single row result of a Get or a Scan query.

S.No.	Constructors
1	**Result()**

Using this constructor, we can create an empty Result with no KeyValue payload; returns null if we call raw Cells().

S.No.	Methods and Description
1	**byte[] getValue(byte[] family, byte[] qualifier)**

This method is used to get the latest version of the specified column.

2 **byte[] getRow()**

This method is used to retrieve the row key that corresponds to the row from which this Result was created.

Inserting Data using HBase Shell

To create data in an HBase table, the following commands and methods are used:

- **put** command,
- **add()** method of **Put** class, and
- **put()** method of **HTable** class.

Updating Data using HBase Shell

We can update an existing cell value using the **put** command. To do so, just follow the same syntax and mention the new value as shown below.

```
put ‘table name’,’row ’,'Column family:column name',’new value’
```

The newly given value replaces the existing value, updating the row.

Reading Data using HBase Shell

The **get** command and the **get()** method of **HTable** class are used to read data from a table in

HBase. Using **get** command, we can get a single row of data at a time. Its syntax is as follows:

```
get ’<table name>’,’row1’
```

Deleting a Specific Cell in a Table

Using the **delete** command, we can delete a specific cell in a table. The syntax of **delete** command is as follows:

```
delete ‘<table name>’, ‘<row>’, ‘<column name >’, ‘<time stamp>’
```

Deleting All Cells in a Table

Using the “deleteall” command, we can delete all the cells in a row. Given below is the syntax of deleteall command.

```
deleteall ‘<table name>’, ‘<row>’,
```

Scanning using HBase Shell

The **scan** command is used to view the data in HTable. Using the scan command, we can get the table data. Its syntax is as follows:

```
scan ‘<table name>’
```

HBase - Count & Truncate

We can count the number of rows of a table using the **count** command. Its syntax is as follows:

```
count ‘<table name>’
```

truncate command disables drops and recreates a table. The syntax of **truncate** is as follows:

```
hbase> truncate 'table name'
```

RESULT AND DISCUSSION

The study of HBase distributed NoSQL data storage model was done successfully.