Implementing K-Means Clustering Algorithm Using MapReduce Paradigm

Botcha Chandrasekhara Rao[1], Medara Rambabu[2]

[1]M.Tech Scholar, Department of Computer Science & Engineering, Pydah Kaushik College of Engineering, Gambheeram Village, Anandapuram, Mandal-531163.Visakhapatnam, AP, India

[2]HOD &Associate Professor, Department of Computer Science & Engineering, Pydah Kaushik College of Engineering, Gambheeram Village, Anandapuram, Mandal-531163.Visakhapatnam, AP, India

Abstract: ***Clustering is a useful data mining technique which groups' data points such that the points within a single group have similar characteristics, while the points in different groups are dissimilar. Partitioning algorithm methods such as k-means algorithm is one kind of widely used clustering algorithms. As there is an increasing trend of applications to deal with vast amounts of data, clustering such big data is a challenging problem. Recently, partitioning clustering algorithms on a large cluster of commodity machines using the MapReduce framework have received a lot of attention. Traditional way of clustering text documents is Vector space model, in which tf-idf is used for k-means algorithm with supportive similarity measure. This project exhibits an approach to cluster text documents in which results obtained by executing map reduce k-means algorithm on single node cluster show that the performance of the algorithm increases as the text corpus increases.***

Keywords: Vector space model, map reduce, text, clustering, map reduce k-means, Hadoop

1. Introduction

Big Data[1] is the term applied to data sets whose size is beyond the ability of thecommonly used software tools to capture, manage, and process within a tolerableelapsed time. Big data can be analysed with the software tools commonly used as part of advanced analytics disciplines such as predictiveanalytics, datamining, textanalytics and statistica l analysis. The semi-structured and unstructured data may not fit well in traditional data warehouses based on relational databases.Hadoop is a platform that provides both distributed storage and computational capabilities. It is an open source software project that enables the distributed processing of large data sets across clusters of commodity servers. It is designed to scale up from a single server to thousands of machines, with a very high degree of fault tolerance. Hadoop is a distributed master-slave architecture that consists of Hadoop distributed file system (HDFS) for storage and Map-Reduce for computational capabilities.

Mathematical models like Boolean model, probabilistic model, vector space model are proposed to use in information retrieval systems. Vector space model (VSM) [2] is most popular and widely used model.

2. Literature Survey

A. Vector Space Model

In VSM [3], A term document matrix t X d is created, where t is the terms(words) of the documents and d represents documents and find the frequency of the terms in each and every document..This term frequency cannot able to find its importance in the whole corpus rather cannot compare with other documents. Term frequency -Inverse document frequency (TF-IDF) overcomes the limitations of term frequency and calculated term weighting to find how important a word is to a document in a collection or corpus.

The weight vector for document d is$V_d=[w_{1,d}, w_{2,d}, w_{3,d},\ldots, w_{N,d}]^T$, where

$$w_{t,d} = (1+\log \mathrm{tf}_{t,d}) \times \log \; N/\mathrm{df}_t$$

$\mathrm{tf}_{i,d}$ is the term frequency of term t in document d (a local parameter)

For any two documents d_j and d_k, their similarity is

$$sim(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{|\vec{d}_j||\vec{d}_k|} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2}\sqrt{\sum_{i=1}^{n} w_{i,k}^2}}$$

where w_i is the weight of the term (Here tf-idf value)
n is the number of dimensions of document vector

B. K-Means Clustering

K-Means is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters).The number of clusters should match the data. An incorrect choice of the number of clusters will invalidate the whole process. An empirical way to find the best number of clusters is to try K-means clustering with different number of clusters and measure the resulting sum of squares.

The basic K-means Algorithm is as follows:
Step 1: Select K points as initial centroids
Step2: repeat
Step 3: Form K clusters by assigning each point to its Closest centroid
Step 4: Recompute the centroid of each cluster
Step 5: Until centroids do not change.

The key step of Basic K-means algorithm is selection of proper initial centroids. Initial clusters are formed by random initialization of centroids.

C. Hadoop

Hadoop is a platform that provides both distributed storage and computational capabilities. It is an open source software project that enables the distributed processing of large data sets across clusters of commodity servers. It is designed to scale up from a single server to thousands of machines, with a very high degree of fault tolerance. Hadoop[7] is a distributed master-slave architecture that consists of Hadoop distributed file system (HDFS) for storage and Map-Reduce for computational capabilities. Hadoop can handle all types of data from disparate systems: structured, unstructured, log files, pictures, audio files, communications records, email – just about anything you can think of, regardless of its native format. Even when different types of data have been stored in unrelated systems, you can dump it all into your Hadoop cluster with no prior need for a schema.

Hadoop Distributed File System (HDFS)[5] is a file system that spans all the nodes in a Hadoop cluster for data storage. The HDFS splits large data files into chunks that are managed by different nodes in the cluster. Each chunk is replicated across several nodes to address single node outage or fencing scenarios.

D. MapReduce Programming

MapReduce[6] runs as a series of jobs, with each job essentially a separate Java application that goes out into the data and starts pulling out information as needed. Based on the MapReduce design, records are processed in isolation via tasks called Mappers. The output from the Mapper tasks is further processed by a second set of tasks, the Reducers, where the results from the different Mapper tasks are merged together. Using MapReduce instead of a query gives data seekers a lot of power and flexibility, but also adds a lot of complexity. The *Map* and *Reduce* functions of *MapReduce* are both defined with respect to data structured in (key, value) pairs. *Map* takes one pair of data with a type in one domain, and returns a list of pairs in a different domain:

$$\text{Map(k1,v1)} \rightarrow \text{list(k2,v2)}$$

The *Map* function is applied in parallel to every pair in the input dataset. This produces a list of pairs for each call. After that, the MapReduce framework collects all pairs with the same key from all lists and groups them together, creating one group for each key.
The *Reduce* function is then applied in parallel to each group, which in turn produces a collection of values in the same domain:

$$\text{Reduce(k2, list (v2))} \rightarrow \text{list(v3)}$$

Each *Reduce* call typically produces either one value v3 or an empty return, though one call is allowed to return more than one value. The returns of all calls are collected as the desired result list.

Thus the MapReduce framework transforms a list of (key, value) pairs into a list of values.

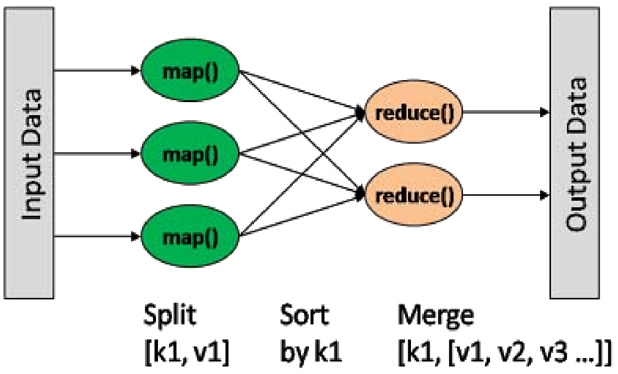


Figure 1: MapReduce Tasks

3. Mapreduce K-Means Clustering

The mapreduce k-means clustering approach for processing big text corpus [4] can be done by the following steps:
1) Give the Sequence file from directory of text documents as input.
2) Tokenize and generate TF-IDF vector for each document from sequence file.
3) Apply Map-Reduce K-Means algorithm to form k clusters.

A. Sequence File from Directory of Text Documents

Map reduce programming is coined to process huge data sets in parallel and distributed environment. Suppose, we select the input data from a document set, where the text files in the directory are small in size. Since HDFS and Mapreduce are optimized for large files, convert the small text files into larger file i.e., SequenceFile format. SequenceFile is a hadoop class, which allows us to write document data in terms of binary <key, value> pairs, where key is a Text with unique document id and value is Text content within the document in UTF-8 format. SequenceFile packs the small files and process whole file as a record. Since the SequenceFile is in binary format, we could not able to read the content directly but faster for read /write operations.

B. Creating TF-IDF Vectors

The sequence file from the previous step is fed as Input to create vectors. The TF-IDF vectors are calculated in Mapreduce by the following steps:

Step 1: Tokenization: The input fed to map function is in format of <key, value> pairs, where key is the document name and value as document content. The outcome of reduce function is also <key, value> pair where key is document name and value are tokens (words) present in that document.
Ex: Key: /acq1.txt: Value: [macandrews, forbes, holdings, bids, revlon, mcandrews, forbes, holdings, inc,said, offer, dlrs, per, share, all, revlon, group]

Step 2: Dictionary file: This step assign unique number to each token in all documents. The input format for the map function is <document name, wordlist> and the output of reduce function is <Word, uniqueid>.

Ex: Key: accounts: Value: 152.

Step 3: Frequency count: The number of times the word appears globally in all documents is calculated in this step. The input to this map function is <docname, words>and output format is <word id, 1>.The value of output of the map function is accumulated and find the sum in reduce function. The output format of reduce function is <word id, count>.
Ex: Key: 50: Value: 2

Step 4 : Calculate term frequency : The map reduce function in this step takes input as <docname, wordlist > and counts the number of times each word or term ti occurs in that document dj. The outcome of this step is in the format of <docname, {ti: count}>.

Ex: In the below example, acq1.txt is document name and the values are in the format of list of (wordid:count)
Key:/acq1.txt: Value: {3258:1.0, 3257:1.0, 157:2.0 ...}

Step 5 : Calculate tf-idf value : The output of step 4 is taken as input to map function of this step and calculates the weight vector as tf X tf-idf value of each term ti in each document dj as specified in equation(1). The output format of the result is <docname, {t1: tf-idf1, t2:tf-idf2 ...ti:tf-idfi}>.
For example: The output format of thisstep is as follows:
Key: acq1.txt: Value: {3258:0.12728, 3257:0.12728, 462:0.08060 ...}

C. MapReduce K-Means Algorithm

The implementation of map reduce k-means accepts two input files. One input file contains the documents with each term and its tf-idf values, and the second is k initial centroids file. The set of k initial centroids are selected randomly. In every iteration, the map reduce framework splits the input data into M splits and then processed in parallel as shown in Figure 2.

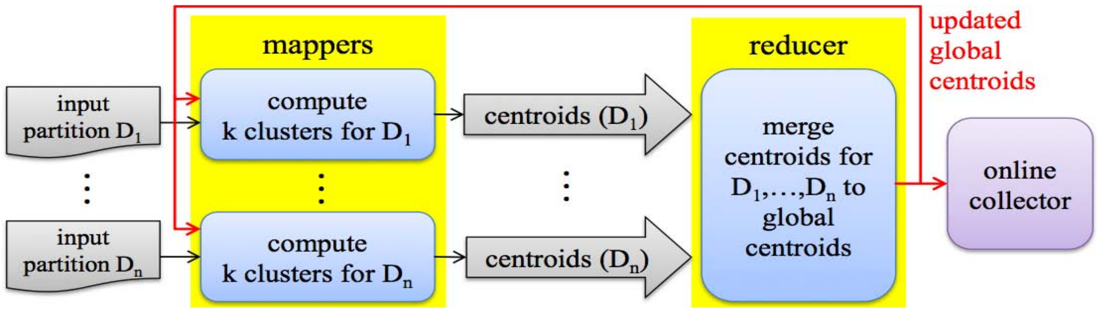


Figure 2: MapReduce Framework for K-Means Algorithm

The map function read the document in the format of <docname, {t1 :tf-idf1, t2:tf-idf2, ...ti:tf-idfi}> along with randomly selected set of k initial centroids. The map function determines the document set which are closer to the centroids by calculating cosine similarity measure and emits records containing all documents data with kcentroids in the format as <k-center, {docname, {t1 :tfidf1, t2:tf-idf2, ...ti: tf-idfi}>. The reducer function receives the output of map function k-centroids along with closest documents bound to it. and calculates new k-centroid. The mapper and reducer algorithm of k-means is explained as below:
Algorithm for Mapper

Input: A set of objects X = {x1, x2… xn}, A Set ofinitial Centroids C = {c1, c2, ,ck}
Output: An output list which contains pairs of (Ci, xj)where $1 \le i \le n$ and $1 \le j \le k$

```
Procedure
M1←{x1, x2… xm}
current_centroids←C
Distance (p, q) =$\sqrt{\Sigma^{d}_{i=1}(p_i - q_i)^2}$(where pi (or qi)is the coordinate of p (or q) in dimension i)
for all xi ϵ M1 such that 1≤i≤m do
    bestCentroid←null
    minDist←∞
    for all c ϵ current_centroids do
        dist← distance (xi, c)
        if (bestCentroid = null || dist<minDist)
        then
            minDist←dist
            bestCentroid ← c
        end if
    end for
    emit (bestCentroid, xi)
    i+=1
end for
return Outputlist
```

Algorithm for Reducer

Input: (Key, Value), where key = bestCentroid and Value =Objects assigned to the lpgr'; 1\] x centroid by the mapper

Output: (Key, Value), where key = oldCentroid and value = newBestCentroid which is the new centroid value calculated for that bestCentroid

```
Procedure

Outputlist←outputlist from mappers
ϑ ← { }
newCentroidList ← null
for all β outputlist do
    centroid ←β.key
    object ←β.value
    [centroid] ← object
end for
for all centroid ϵ ϑ do
    newCentroid, sumofObjects,
    sumofObjects← null
    for all object ϵ ϑ [centroid] do
```

```
                        sumofObjects += object
                        numofObjects += 1
                end for
newCentroid ← (sumofObjects +
numofObjects)
emit (centroid, newCentroid)
end for
end
```

The outcome of the k-means map reduce algorithm is the cluster points along with bounded documents as <key, value> pairs, where key is the cluster id and value contains in the form of vector: weight. The weight indicates the probability of vector be a point in that cluster. For Example: Key: 92: Value: 1.0: [32:0.127,79:0.114, 97:0.114, 157:0.148 ...].

The final output of the program will be the cluster name, file name: number of text documents that belong to that cluster.

4. Experimental Results

This section presents the results obtained by executing map reduce k-means clustering algorithm on cluster of machines. The experimentation with 20_newsgroups dataset is explained in detailed in below sections.

A. Environment setup

The experimentation is conducted in single node cluster. The single node is configured with I7processor, 4 GB memory, 64GB hard disk along with JDK 1.7.0 and hadoop 2.4.1 version. The Operating system used is Ubuntu 14.04LTS.

B. Dataset description

The 20_Newsgroups data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups.It was originally collected by Ken Lang, probably for his *Newsweeder: Learning to filter netnews* paper, though he does not explicitly mention this collection. The 20 newsgroups collection has become a popular data set for experiments in text applications of machine learning techniques, such as text classification and text clustering. The data is organized into 20 different newsgroups, each corresponding to a different topic. Some of the newsgroups are very closely related to each other (e.g. **comp.sys.ibm.pc.hardware / comp.sys.mac.hardware**), while others are highly unrelated (e.g. **misc.forsale / soc.religion.christian**).

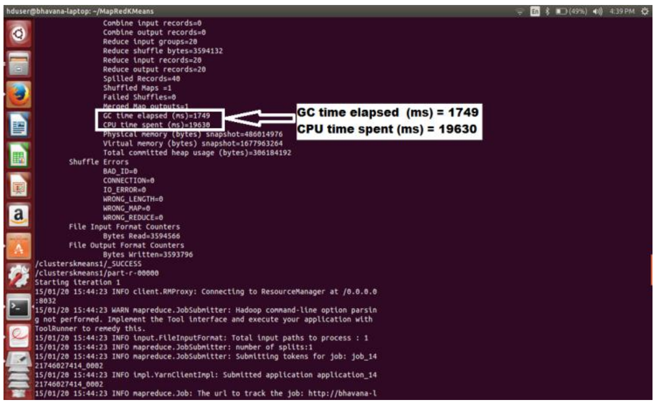
C. Data pre-processing

The data available here are in .tar.gz bundles. You will need tar and unzip to open them. Each subdirectory in the bundle represents a newsgroup; each file in a subdirectory is the text of some newsgroup document that was posted to that newsgroup. Create a sequence file from 20_newsgroups text documents. Sequence file is passed as input to find tf-idf value of each term in every text document. The tf-idf file is fed as input to map reduce k-means algorithm for form k number of clusters.

D. Results

In a single node cluster, the map reduce k-means algorithm is executed with 20_newsgroups dataset with selected number of text files of 20,000 from 20 different topics. The algorithm tested in single node cluster with 20_newsgroups database and the performance of the map reduce algorithm increases efficiently by increasing the number of input files and increasing number of nodes in the cluster. The runtime execution in terms of seconds can be given as the CPU time spent for each iteration.

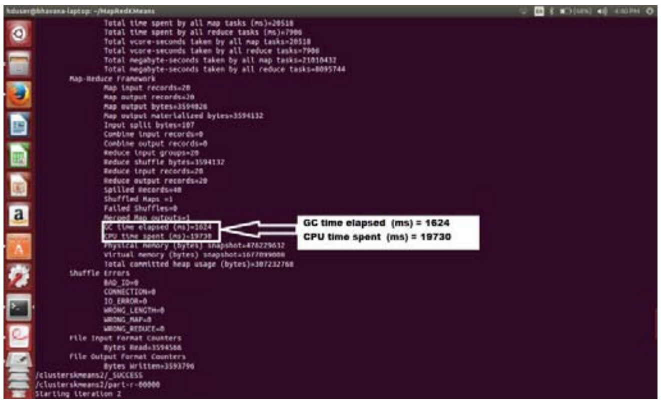
E. Output



```
                Combine input records=0
                Combine output records=0
                Reduce input groups=20
                Reduce shuffle bytes=3594132
                Reduce input records=20
                Reduce output records=20
                Spilled Records=40
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=1749
                CPU time spent (ms)=19630
                Physical memory (bytes) snapshot=486014976
                Virtual memory (bytes) snapshot=1677963264
                Total committed heap usage (bytes)=306184192
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=3594566
        File Output Format Counters
                Bytes Written=3593796
/clusterskmeans1/_SUCCESS
/clusterskmeans1/part-r-00000
Starting iteration 1
15/01/20 15:44:23 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0
:8032
15/01/20 15:44:23 WARN mapreduce.JobSubmitter: Hadoop command-line option parsin
g not performed. Implement the Tool interface and execute your application with
ToolRunner to remedy this.
15/01/20 15:44:23 INFO input.FileInputFormat: Total input paths to process : 1
15/01/20 15:44:23 INFO mapreduce.JobSubmitter: number of splits:1
15/01/20 15:44:23 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_14
21746027414_0002
15/01/20 15:44:23 INFO impl.YarnClientImpl: Submitted application application_14
21746027414_0002
15/01/20 15:44:23 INFO mapreduce.Job: The url to track the job: http://bhavana-l
```

GC time elapsed (ms) = 1749
CPU time spent (ms) = 19630

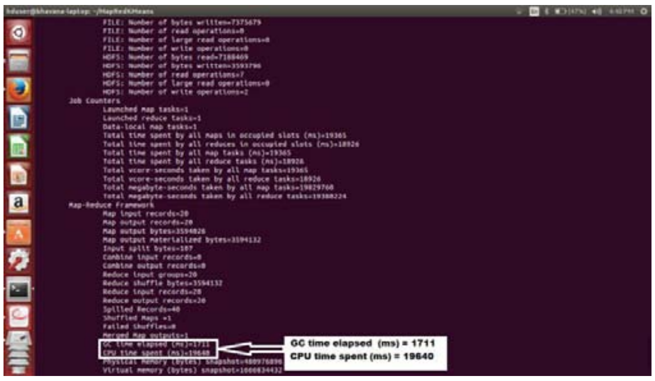
Time taken for iteration 0



```
                Total time spent by all map tasks (ms)=20518
                Total time spent by all reduce tasks (ms)=7906
                Total vcore-seconds taken by all map tasks=20518
                Total vcore-seconds taken by all reduce tasks=7906
                Total megabyte-seconds taken by all map tasks=21010432
                Total megabyte-seconds taken by all reduce tasks=8095744
        Map-Reduce Framework
                Map input records=20
                Map output records=20
                Map output bytes=3594026
                Map output materialized bytes=3594132
                Input split bytes=107
                Combine input records=0
                Combine output records=0
                Reduce input groups=20
                Reduce shuffle bytes=3594132
                Reduce input records=20
                Reduce output records=20
                Spilled Records=40
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=1624
                CPU time spent (ms)=19730
                Physical memory (bytes) snapshot=476229632
                Virtual memory (bytes) snapshot=1677099008
                Total committed heap usage (bytes)=307232768
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=3594566
        File Output Format Counters
                Bytes Written=3593796
/clusterskmeans2/_SUCCESS
/clusterskmeans2/part-r-00000
Starting iteration 2
```

GC time elapsed (ms) = 1624
CPU time spent (ms) = 19730

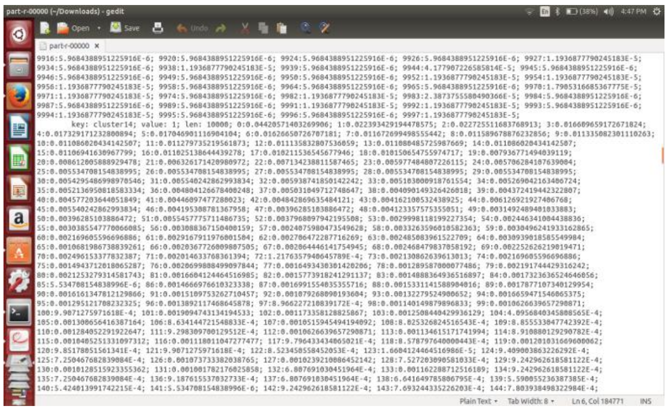
Time taken for iteration 1



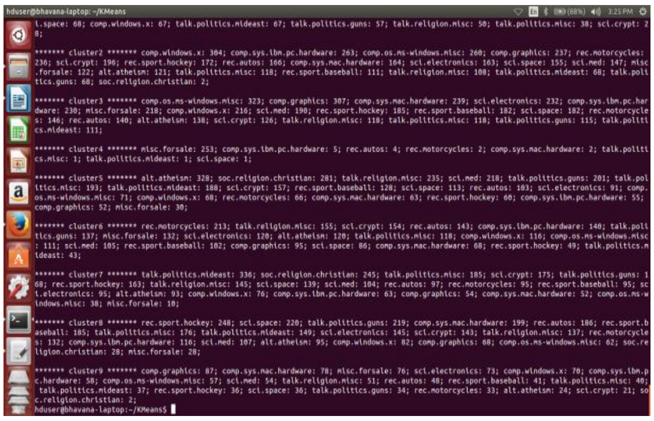
```
                FILE: Number of bytes written=7375679
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=7188469
                HDFS: Number of bytes written=3593796
                HDFS: Number of read operations=7
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=19365
                Total time spent by all reduces in occupied slots (ms)=18926
                Total time spent by all map tasks (ms)=19365
                Total time spent by all reduce tasks (ms)=18926
                Total vcore-seconds taken by all map tasks=19365
                Total vcore-seconds taken by all reduce tasks=18926
                Total megabyte-seconds taken by all map tasks=19829760
                Total megabyte-seconds taken by all reduce tasks=19380224
        Map-Reduce Framework
                Map input records=20
                Map output records=20
                Map output bytes=3594026
                Map output materialized bytes=3594132
                Input split bytes=107
                Combine input records=0
                Combine output records=0
                Reduce input groups=20
                Reduce shuffle bytes=3594132
                Reduce input records=20
                Reduce output records=20
                Spilled Records=40
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=1711
                CPU time spent (ms)=19640
                Physical memory (bytes) snapshot=480976896
                Virtual memory (bytes) snapshot=1666834432
```

GC time elapsed (ms) = 1711
CPU time spent (ms) = 19640

Time taken for iteration 2



```
part-r-00000 (~/Downloads) - gedit
9916:5.9684388951225916E-6; 9920:5.9684388951225916E-6; 9924:5.9684388951225916E-6; 9926:5.9684388951225916E-6; 9927:1.1936877790245183E-5;
9934:5.9684388951225916E-6; 9938:1.1936877790245183E-5; 9939:5.9684388951225916E-6; 9944:4.177907226585814E-5; 9945:5.9684388951225916E-6;
9946:5.9684388951225916E-6; 9949:5.9684388951225916E-6; 9950:5.9684388951225916E-6; 9952:1.1936877790245183E-5; 9954:1.1936877790245183E-5;
9956:1.1936877790245183E-5; 9958:5.9684388951225916E-6; 9964:5.9684388951225916E-6; 9965:5.9684388951225916E-6; 9970:1.7905316685367775E-5;
9971:1.1936877790245183E-5; 9974:5.9684388951225916E-6; 9982:1.1936877790245183E-5; 9983:2.3873755580490366E-5; 9984:5.9684388951225916E-6;
9987:5.9684388951225916E-6; 9989:5.9684388951225916E-6; 9991:1.1936877790245183E-5; 9992:1.1936877790245183E-5; 9993:5.9684388951225916E-6;
9994:1.1936877790245183E-5; 9995:5.9684388951225916E-6; 9996:5.9684388951225916E-6; 9997:1.1936877790245183E-5;
        key: cluster14; value: 1; len: 10000; 0:0.04420571403269906; 1:0.022393429194478575; 2:0.022725511683768913; 3:0.016609659172671824;
4:0.017329171232800894; 5:0.017046901116904104; 6:0.016266507267071181; 7:0.011672699498555442; 8:0.011589678876232856; 9:0.011335082301110263;
10:0.011086020434142507; 11:0.011279735219561873; 12:0.011135832807536059; 13:0.011080485725987669; 14:0.011086020434142507;
15:0.011069416309677799; 16:0.011025138644439278; 17:0.010211536545677946; 18:0.010150654755974717; 19:0.007936771494039119;
20:0.008612005888929478; 21:0.006326171420980972; 22:0.007134238811587465; 23:0.005977484807226115; 24:0.005706284107639004;
25:0.005534708154838995; 26:0.005534708154838995; 27:0.005534708154838995; 28:0.005534708154838995; 29:0.005534708154838995;
30:0.005429548699897054 6; 31:0.005540242862993834; 32:0.005938741850142242; 33:0.005103800918761554; 34:0.005269042163400724;
35:0.005213695081858 3334; 36:0.004804126678400248; 37:0.005031049712748647; 38:0.004090149326426018; 39:0.004372419442322807;
40:0.004577203644051849; 41:0.004460974772880023; 42:0.004842869635484121; 43:0.004162100532438925; 44:0.006126921927406768;
45:0.005540242862993834; 46:0.004195308781367958; 47:0.003962851038886472; 48:0.004123357575355051; 49:0.003149248948103388 3;
50:0.003962851038886472; 51:0.005545777571148673 5; 52:0.003796889794219550 8; 53:0.002999811819922735 4; 54:0.002446341084438836;
55:0.003038554777006608 5; 56:0.003088367150400159; 57:0.002407598047354962 8; 58:0.003326359601058236 3; 59:0.003049624193316286 5;
60:0.002169605596696886; 61:0.002916791197600150 4; 62:0.002706472287716269; 63:0.002485083961522709; 64:0.003093901858554998 4;
65:0.001068198673883926 1; 66:0.002036772600980750 5; 67:0.002064446141754945; 68:0.002468479837058192; 69:0.002252626219019471;
70:0.002496153377832387; 71:0.002014633768361394; 72:1.2176357940864578 9E-4; 73:0.002130862639613013; 74:0.002169605596696886;
75:0.001494371201806528 7; 76:0.002069988849909784 4; 77:0.001649343030142020 6; 78:0.001289587000077486; 79:0.002191744429316242;
80:0.002125327931458174 3; 81:0.001660412446451698 5; 82:0.001577391824129113 7; 83:0.001488836493651689 7; 84:0.001732363652464685 6;
85:5.534708154838996E-6; 86:0.001466697661032333 8; 87:0.001699155403535571 6; 88:0.001533114158890401 6; 89:0.001787710734012995 4;
90:0.001616134781212986 6; 91:0.001510975326271045 7; 92:0.001879268090193604; 93:0.001322795249006 52; 94:0.001665947154606537 5;
95:0.001295121708232325; 96:0.001389211746864587 8; 97:8.966227210839172E-4; 98:0.001140149879896833; 99:0.001062663965729087 1;
100:9.9871275971618E-4; 101:0.001909474313419453 3; 102:0.001173358128825867; 103:0.001250844042993612 9; 104:4.095684034588856 5E-4;
105:0.001300656416387164; 106:8.634144721540833E-4; 107:0.001051594549419409 2; 108:8.025326824516543E-4; 109:8.855338477423 92E-4;
110:0.001284052291922647; 111:9.298309700129512E-4; 112:0.001062663965729087 1; 113:0.001134615171741994; 114:8.910880129290782E-4;
115:0.001040525133109731 2; 116:0.001118011047277477; 117:9.796433434065021E-4; 118:8.578797640000443E-4; 119:0.001201031669600062;
120:9.851780515613 41E-4; 121:9.9871275971618E-4; 122:8.523458558452053E-4; 123:1.660412446451698 6E-5; 124:9.409003863226292E-4;
125:7.250467682839084E-4; 126:0.001073733382038765; 127:0.001023921000645214 2; 128:7.527203090581033E-4; 129:9.242962618581122E-4;
130:0.001012851592335536 2; 131:0.001001782176025858; 132:6.807691030451964E-4; 133:0.001162288712516109; 134:9.242962618581122E-4;
135:7.250467682839084E-4; 136:9.187615537032733E-4; 137:6.807691030451964E-4; 138:6.641649785806795E-4; 139:5.590055236387385E-4;
140:5.424013991742215E-4; 141:5.534708154838996E-6; 142:9.242962618581122E-4; 143:7.693244335226203E-4; 144:7.803938498322984E-4;
Plain Text ▾   Tab Width: 8 ▾   Ln 6, Col 184771   INS
```



```
hduser@bhavana-laptop: ~/KMeans
i.space: 68; comp.windows.x: 67; talk.politics.mideast: 67; talk.politics.guns: 57; talk.religion.misc: 50; talk.politics.misc: 38; sci.crypt: 2
8;

******* cluster2 ******* comp.windows.x: 304; comp.sys.ibm.pc.hardware: 263; comp.os.ms-windows.misc: 260; comp.graphics: 237; rec.motorcycles:
236; sci.crypt: 196; rec.sport.hockey: 172; rec.autos: 166; comp.sys.mac.hardware: 164; sci.electronics: 163; sci.space: 155; sci.med: 147; misc
.forsale: 122; alt.atheism: 121; talk.politics.misc: 118; rec.sport.baseball: 111; talk.religion.misc: 108; talk.politics.mideast: 68; talk.poli
tics.guns: 68; soc.religion.christian: 2;

******* cluster3 ******* comp.os.ms-windows.misc: 323; comp.graphics: 307; comp.sys.mac.hardware: 239; sci.electronics: 232; comp.sys.ibm.pc.har
dware: 230; misc.forsale: 218; comp.windows.x: 216; sci.med: 190; rec.sport.hockey: 185; rec.sport.baseball: 182; sci.space: 182; rec.motorcycle
s: 146; rec.autos: 140; alt.atheism: 138; sci.crypt: 126; talk.religion.misc: 118; talk.politics.misc: 118; talk.politics.guns: 115; talk.politi
cs.mideast: 111;

******* cluster4 ******* misc.forsale: 253; comp.sys.ibm.pc.hardware: 5; rec.autos: 4; rec.motorcycles: 2; comp.sys.mac.hardware: 2; talk.politi
cs.misc: 1; talk.politics.mideast: 1; sci.space: 1;

******* cluster5 ******* alt.atheism: 328; soc.religion.christian: 281; talk.religion.misc: 235; sci.med: 218; talk.politics.guns: 201; talk.pol
itics.misc: 193; talk.politics.mideast: 188; sci.crypt: 157; rec.sport.baseball: 128; sci.space: 113; rec.autos: 103; sci.electronics: 91; comp.
os.ms-windows.misc: 71; comp.windows.x: 68; rec.motorcycles: 66; comp.sys.mac.hardware: 63; rec.sport.hockey: 60; comp.sys.ibm.pc.hardware: 55;
comp.graphics: 52; misc.forsale: 30;

******* cluster6 ******* rec.motorcycles: 213; talk.religion.misc: 155; sci.crypt: 154; rec.autos: 143; comp.sys.ibm.pc.hardware: 140; talk.poli
tics.guns: 137; misc.forsale: 132; sci.electronics: 120; alt.atheism: 120; talk.politics.misc: 118; comp.windows.x: 116; comp.os.ms-windows.misc
: 111; sci.med: 105; rec.sport.baseball: 102; comp.graphics: 95; sci.space: 86; comp.sys.mac.hardware: 68; rec.sport.hockey: 49; talk.politics.m
ideast: 43;

******* cluster7 ******* talk.politics.mideast: 336; soc.religion.christian: 245; talk.politics.misc: 185; sci.crypt: 175; talk.politics.guns: 1
68; rec.sport.hockey: 163; talk.religion.misc: 145; sci.space: 139; sci.med: 104; rec.autos: 97; rec.motorcycles: 95; rec.sport.baseball: 95; sc
i.electronics: 95; alt.atheism: 93; comp.windows.x: 76; comp.sys.ibm.pc.hardware: 63; comp.graphics: 54; comp.sys.mac.hardware: 52; comp.os.ms-w
indows.misc: 38; misc.forsale: 10;

******* cluster8 ******* rec.sport.hockey: 248; sci.space: 220; talk.politics.guns: 219; comp.sys.mac.hardware: 199; rec.autos: 186; rec.sport.b
aseball: 185; talk.politics.misc: 176; talk.politics.mideast: 149; sci.electronics: 145; sci.crypt: 143; talk.religion.misc: 137; rec.motorcycle
s: 132; comp.sys.ibm.pc.hardware: 116; sci.med: 107; alt.atheism: 95; comp.windows.x: 82; comp.graphics: 68; comp.os.ms-windows.misc: 62; soc.re
ligion.christian: 28; misc.forsale: 28;

******* cluster9 ******* comp.graphics: 87; comp.sys.mac.hardware: 78; misc.forsale: 76; sci.electronics: 73; comp.windows.x: 70; comp.sys.ibm.p
c.hardware: 58; comp.os.ms-windows.misc: 57; sci.med: 54; talk.religion.misc: 51; rec.autos: 48; rec.sport.baseball: 41; talk.politics.misc: 40;
 talk.politics.mideast: 37; rec.sport.hockey: 36; sci.space: 36; talk.politics.guns: 34; rec.motorcycles: 33; alt.atheism: 24; sci.crypt: 21; so
c.religion.christian: 2;
hduser@bhavana-laptop:~/KMeans$
```

Clusters

5. Conclusions and Future Work

Information retrieval techniques are widely popular in most of the search engines to efficiently organize and retrieve information systems. Most of the data in internet is in the format of unstructured and semi structured. Currently clustering techniques are used to organize and group the similar data objects to retrieve search results faster. Traditional way of clustering text documents is Vector space model, in which tf-idf is used for k-means algorithm with supportive similarity measure. As the data is enormously increasing day by day, elastic resources are required to store and compute. Hadoop framework supports to store and compute big data in parallel and distributed platform with the help of HDFS and Map reduce.

This paper exhibits an approach to cluster text documents using vector space model. The results obtained by executing map reduce k-means algorithm on single node cluster shows that the performance of the algorithm increases as the text corpus increases. In the current approach of map reduce k-means algorithm, the initial centroids are selected randomly. In future, Canopy clustering algorithm can be used as initial step to get initial centroids and then fed this as input to map reduce k-means algorithm. In future, we can try to cluster text corpus by map reduced text clustering based on frequent itemsets.

References

[1] Hadoop The definitive guide.
[2] G. Salton, A. Wong, C. S. Yang.A vector space model for automatic indexing. Communications of the ACM, version.18 n.11, pages.613-620, Nov. 1975.
[3] GeorgeTsatsaronis and Vicky Panagiotopoulou. A generalized vector space model for text retrieval based on semantic relatedness. Proceedings of the EACL 2009 student research workshop, pages 70-78, April 2009.
[4] J Dittrich, JA Quiané-Ruiz. Efficient big data processingin Hadoop MapReduce. Proceedings of the VLDB Endowment, 2012 - dl.acm.org, Volume 5 Issue 12, August 2012 ,Volume 5 Issue 12, August 2012.
[5] Sanjay, G., G. Howard, and L. Shun-Tak, The Google file system, in Proceedings of the nineteenth ACM symposium on Operating systems principles. ACM: Bolton Lan,ding, NY, USA 2003,
[6] Jeffrey, D. and G. Sanjay,. MapReduce: simplified dataprocessing on large clusters. Commun. ACM, 51(1): pages 107-113 2008.
[7] Apache Lucene Hadoop[EB/OL]. http://hadoop.apache.org/.

Author Profile

Botcha Chandrasekhara raois currently pursuing his 2 years M.Tech in CSE at Pydah Kaushik College of Engineering, Gambheeram Village, Anandapuram Mandalam, Visakhapatnam. His area of interest includes Cloud Computing.

Medara Rambabu is currently working as HOD & Associate Professor in Department of Computer Science & Engineering at Pydah Kaushik College of Engineering, Gambheeram Village, Anandapuram Mandalam, Visakhapatnam. He completed his M.Tech from JNTUK in 2010, and he completed his B.Tech from AU, in 2004.He has more than 6 years of teaching experience in various engineering colleges. His research interest includes Computer Organization, Data Communication Systems, Computer Networks and DM & DW.