Available online at www.sciencedirect.com

ScienceDirect

Future Computing and Informatics Journal 3 (2018) 200–209
http://www.journals.elsevier.com/future-computing-and-informatics-journal/

An analysis of MapReduce efficiency in document clustering using parallel K-means algorithm

Tanvir Habib Sardar, Zahid Ansari*

Computer Science and Engineering, P.A. College of Engineering, Mangalore, India

Received 28 October 2017; revised 18 February 2018; accepted 20 March 2018
Available online 17 May 2018

Abstract

One of the significant data mining techniques is clustering. Due to expansion and digitalization of each field, large datasets are being generated rapidly. Such large dataset clustering is a challenge for traditional sequential clustering algorithms due to huge processing time. Distributed parallel architectures and algorithms are thus helpful to achieve performance and scalability requirement of clustering large datasets. In this study, we design and experiment a parallel k-means algorithm using MapReduce programming model and compared the result with sequential k-means for clustering varying size of document dataset. The result demonstrates that proposed k-means obtains higher performance and outperformed sequential k-means while clustering documents.

Keywords: MapReduce; Hadoop; Parallel K-means; Document clustering; Distributed computing

1. Introduction

Extracting the knowledge from a dataset by extracting useful pattern from the data is the task of data mining [1]. Clustering is one of the major research fields in the wide area of data mining and analysis. Clustering partitions the data objects of a dataset into a number of groups or subsets such that objects in a particular subset are similar to each other and comparatively dissimilar from objects from other subsets. There are many applications of the clustering problems in different fields such as image analyses, social science, web technology, pattern recognition, telecommunications etc. [2]. Document clustering produces clusters or groups of documents such that documents in a cluster contain similar property in comparison to documents in other clusters [3]. The grouping of documents is done by the occurrence of each word in the document files in the dataset. Thus, the clustering job is to determine the groups having many of the same words. This is achieved by using a similarity measure which lies is the core of clustering algorithm. Document clustering is used in various requirements such as document organization, document browsing, automatic hierarchical representation of documents, information filtering, search engine result generation, keyword extraction, information retrieval [4,5] etc.

The most popular clustering algorithm is k-means because of its simplicity and efficiency [6]. ICDM Conference ranked it second of top 10 clustering algorithms [7]. K-means algorithm groups N objects into K clusters maintaining high intra group similarity and low inter group similarity of the objects. Initially provided with k cluster centers (centroids), the k-means algorithm places a data object into a group by finding closest cluster center using a distances measure. This measure calculates similarity distance between each data objects and centroids.

* Corresponding author.
E-mail addresses: tanvir.cs@pace.edu.in (T.H. Sardar), zahid_cs@pace.edu.in (Z. Ansari).
Peer review under responsibility of Faculty of Computers and Information Technology, Future University in Egypt.

https://doi.org/10.1016/j.fcij.2018.03.003

Among many available distance measures, Euclidean distance is the widely used measure for distance calculation between objects and centroids [8]. K-means is an iterative algorithm. It converges after a finite number of iterations or if a prior condition for converge is met. Each iteration provides a set of objects to a particular centroid whose distance is lower. A new centroid is then calculated based on the mean of the data objects of each group. This new centroid is fed to the next iteration and so on. The time complexity of k-means is O (nkt), where t is the number of iterations [6].

The enormous development and fully digitalization of many sectors becomes inevitable and a must require for their survival. It is expected that in near future the trend of digitization will cover each area of technology. The dataset sizes are increasing at an explosive rate due to digitalization of field of work such as scientific laboratories, industrial manufacturing process, enterprise planning and management, chemical and pharmaceuticals, finance and insurance, mining, health care, construction, communication, agriculture, education and entertainment etc. to name a few. This explosive growth in data size makes the existing clustering algorithms become inadequate in performance [9]. The classical clustering algorithms lack in performance which causes enterprises to lose time and also the marketing advantage [10]. This situation requires an efficient computing platform that can exploit this potential to the maximum, keeping in mind the current challenges associated with it. Obviously we need to transform classical clustering algorithms for the efficient computing platform designed for large data processing. Enterprises have experienced that existing centralized architectures requires to be replaced with the distributed architecture in order to efficiently handle huge volume of data [11]. Apache Hadoop is arguably the most influential, established and efficient distributed computing framework for large data processing [12]. Hadoop has two key components: HDFS (Hadoop distributed file system) and MapReduce (distributed programming model). While distributing the computing process, HDFS takes responsibility to divide dataset and send them to multiple computing nodes and keep track of them whereas MapReduce process algorithmic steps in each computing node.

In this study, we have proposed a parallelized k-means algorithm using MapReduce programming model and executed on top of Hadoop. A comparison based on execution time is made with classical and modified k-means with different data size.

The rest of the paper is organized as follows. Section 2 provides a quick insight on several modifications of k-means proposed in literature. Section 3 provides the details of our document clustering study in MapReduce, specially focusing on proposed algorithm in detail. Section 4 provides the observation of experiments and its analysis. The last section concludes our work.

2. Literature survey

The simplicity and efficiency makes k-means one of the most popular clustering algorithms. It takes number of cluster k as an input prior clustering. It selects k objects, known as centroids, from the dataset then calculates distance from each object to the centroids using Euclidean distance. The closest objects belong to the centroids create a cluster. Iteratively this process executes till it reaches a finite number of iterations or a clustering criterion [13]. Traditional K-means is used intensively in several works for document clustering and found good result [33−35].

Data clustering algorithms become expensive and slow while dealing with large data repositories. This large volume of data repositories makes analytical operations, retrieval operations and process operations time consuming and difficult [14]. It is thus inevitable to develop efficient, faster and effective scalable and parallel clustering algorithms. It is also required a parallel and distributed computing platform for dealing with large volume of data and for executing parallel and distributed clustering algorithms efficiently and effortlessly.

Recently many works modified k-means in order to implement it on different platforms for clustering large datasets efficiently. Few works have modified k-means algorithm for sequential execution and other works have modified k-means in order to execute on different parallel and distributed platforms. Some of the modifications are reviewed and briefed below:

K-means clustering is sensitive to the random selection of initial cluster centers. Generally, the clustering result depends on early centroid values but there are no formal rules to select a good set of initial centroids. Instead of running k-means with random centroids, initial centroids are chosen based on the preliminary run of k-means in work of Bradley and Fayyad [15]. It provides a better clustering result. In [16], authors proposed a study which determines the right value of k by integration and splitting processes of proposed ISODATA technique. The work is effective but it again requires another user supplied threshold for specifying the number of processes.

Similarly, Arthur and Vassilvitskii [17] proposed k-means++ which provides a seeding method which intelligently chooses cluster centers from a dataset. It overcomes the possibility of poor clustering outcome due to initial centroid values. Kanungo et al. [18] modified k-means using k-dimensional tree (kd-tree) data structure which makes execution of each step of k-means faster. A k-d tree data structure used for forming certain quantity of points in a space with k dimensions. K-d tree firstly stores all data objects maintaining a subset of candidate centers. The centers are filtered to pass to its children. This tree saves time at it does not require updating at each iteration. The corset points represent a weighted version of original points. In order to get speedup in k-means, Frahling et al. [19] uses coresets to quickly find groups of similar set of points for more than one values of K. It is good and useful when user is unaware of correct number of K prior the execution of k-means. K-means is modified in [20] by removing noise sensitivity from k-means using Minkowski metric. It assigns feature weight for each cluster using Minkowski distance measurement. In this modified k-means, Feature weights in Minkowski appeal like a feature rescaling factors in a classical k-means. Likas et al. [21] proposed a work where various k-means processes is maintained for numerous clusters. Authors suggested for a modification of

k-means so that it can execute parallel in order to obtain efficiency.

Zhang and Forman [22] observe that clustering large datasets requires clustering algorithms to be parallelized. They proposed a parallel k-means in order to obtain efficiency in clustering large datasets. A performance function is designed which input data objects N and K cluster centers location so as to estimate a value M. The input data objects are disseminated among a set of computers (C). Estimated by performance function value M, K cluster centers are selected and consistent copies are kept in each C. Now iteratively each computes its part of global statistics S for N which provides information to performance function. Then S value is broadcasted to the all computers after summation across processors. Based on the value of S, each computer settles on a new centroid. A downside of this method is the computation and storage requirement at the central site for data duplication from/to remote site slows down the entire process.

K-means can also be parallelized using other parallel computing architectures such as OpenMP and MPI (Message Passing Interface) in order to get efficiency. MPI provides a message passing programming model for parallel architectures. MPI model is very helpful in creating efficient and scalable parallel applications. In [6], k-means is modified on MPI model by adding a merging algorithm in k-means. The data parallelization is achieved by evenly distributing data objects to all processes and replicating centroids. After each iteration the global operation on centroids is done through a merging algorithm which combines generated centroid sets from each process into a final centroid set with a greedy style. Then it output with K centroids, I/O execution time and time of clustering. The work is stable and efficient for large datasets. This work did not provide any theoretical analysis on performance for varying number of processes. Similarly in [23], MPI programming model is used to parallelize k-means by dividing algorithmic jobs among processors and accessing through distributed memory in order to get speed up in execution time. It is found that the communication costs become insignificant to the overall clustering time as the number of data objects increases.

K-means is experimented with different parallelization platforms in order to compare the efficiency among them. In [24], authors implemented k-means in different parallel framework such as OpenMP, MPI and Cuda-C and compared their performance. It is observed that for small datasets, OpenMP performs best whereas cuda works well with large datasets. Similarly in [25], parallel version of k-means is executed on OpenMP, MPI and Cuda and the result shows that the performance of parallel k-means is way better than sequential access and performance varies with different platform and hardware combination. In [26], author experimented and compared three standard frameworks for parallelization: MPI, OpenMP and MapReduce. It is observed that for small datasets and sufficient processor cores and memory OpenMP is a best choice for parallelization whereas for moderate data size MPI is a good choice. It is also stated that for real world large datasets, MapReduce is more efficient than MPI and OpenMP. The most time consuming part of k-means is the iterative distance computation. Authors in [27] Observes that optimizing this iterative part of the algorithm is the key area for gaining efficiency through parallel implementation of the algorithm. The requirements of using MapReduce on top of Hadoop distributed architecture in modifying k-means are provided below:

- OpenMP and MPI are found efficient in clustering only when the size of the dataset is small or moderate. MapReduce model can achieve major performance gain in processing large datasets [28].
- The distance calculation from objects to centroids is the iterative part in k-means. Hence if distance calculation part is designed on MapReduce and executed in parallel on Hadoop, then the algorithm will gain efficiency in clustering [28].
- The data distribution and result accumulation among nodes in distributed architecture requires significant complication in programming and have an adverse effect in code efficiency. A distributed architecture like Hadoop is thus required that implicitly manages these operations [29].
- The distributed architecture design should put up nodes with low cost commodity hardware so that the algorithm can be executed anywhere anytime. Hadoop cluster can be a good choice [30].
- Scalability is an important requirement in distributed processing. Hadoop is scalable as a cluster can accommodate any number of nodes at any time [31].
- Fault tolerance is another important factor in a distributed framework. Fault tolerance guarantees that if a node fails while an algorithm executing then the data lost in the node's memory should be recovered. Hadoop provides fault tolerance in node(s) failure [32].

3. Methodology

In this work, we have modified traditional K-means algorithm into parallel K-means using MapReduce paradigm and executed on the top of Hadoop platform to reduce execution time clustering in order to cluster document dataset. The major objective of this paper is to discover the clustering efficiency in terms of execution time of proposed k-means over classical k-means on different size of document dataset. The contribution of this work is in the design of traditional K-means into MapReduce based K-means which works on vector space model of text datasets for document clustering. We believe that the design of K-means for document clustering could become a framework to be used for parallelizing other clustering algorithms.

A) Hadoop Framework: Hadoop provides a simple but robust distributed programming framework for processing large dataset in parallel. It provides distributed storage and computation through a cluster of commodity computers. Each node in a cluster provides their own capacity of

computation and total capacity of a Hadoop cluster is determined by cumulative computing power of nodes connected in a cluster. Hadoop works on master/slave architecture (see Fig. 1). One node in a cluster is determined as master by the user whereas other nodes become slaves.

The user interacts only through master by a command interface and master node coordinates with slave nodes for storage and computation distribution. For instance, the user store dataset in the master node and execute programs through master node. It becomes the responsibility of master node to split the dataset across slaves for computation and then to accumulate results of computation from slaves. The capacity of storage and processing is provided by two basic Hadoop modules:

- *A Distributed File System:* The data storage and management part of the Hadoop cluster is provided by Hadoop Distributed File System (HDFS). Each node in a cluster contains HDFS. Before executing a data intensive application HDFS of master node divide the dataset and send datasets to each slave in the cluster. HDFS replicate a data splits to multiple nodes so that in case of failure of a node, data can be recovered from other node. Slave nodes can communicate among themselves to rebalance data, to transfer copies around and to keep data replication. This way HDFS provides fault tolerance hence reliability to data.
- *MapReduce Programming Model*: MapReduce programing paradigm is used by Hadoop architecture to process the large datasets across a number of connected nodes. Hadoop data processing part is provided by MapReduce. The user submits MapReduce jobs to the master. The master transfers the job to available slave nodes in the cluster. MapReduce programming model works as follows:

MapReduce model takes input as a set of <key, value> pairs and outputs <key, value> pair set. Each data processing algorithm is run on MapReduce model using two functions: Map and Reduce. Prior to submit data to the mapper, data should be transformed to <key, value> pairs as mapper can only deal with it. When a data is submitted to HDFS, it is assigned a key and a value to it. Content of a line in the dataset, apart from line terminator considered as the value and the offset of the beginning of the line of the dataset s considered as key.

- *Map function:* It takes an input <key, value> pair and outputs a set of intermediate <key, value> pairs. For each intermediate key i, all intermediate values are collected and then provided to reduce function.
- *Reduce function:* It receives a key and a set of values for that key from mapper then gathers these values to arrange a possibly reduced set of values. Generally, each reduce invocation provides zero or one output value.

A Basic MapReduce Data Processing Scheme is depicted in Fig. 2.

B) *Proposed k-means algorithm execution stages:* In this section we provide the necessary details of parallel k-means for MapReduce programming model. Fig. 3 provides the steps used for clustering voluminous document dataset using parallel K-Means algorithm based on MapReduce paradigm.

Step 1: Preprocess the voluminous document dataset: The text data of a document need to be preprocessed and transformed to a form so that k-means can use it as an

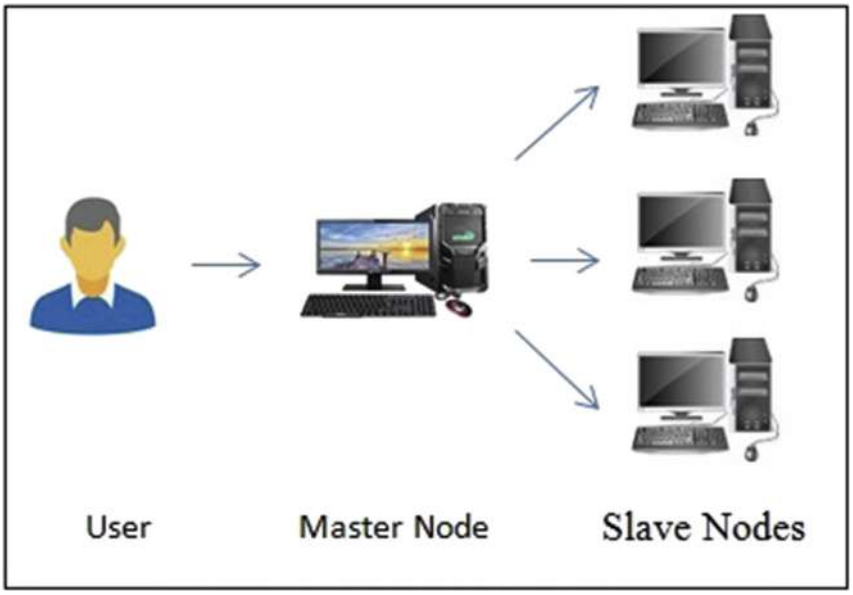

Fig. 1. Hadoop master/slave architecture.

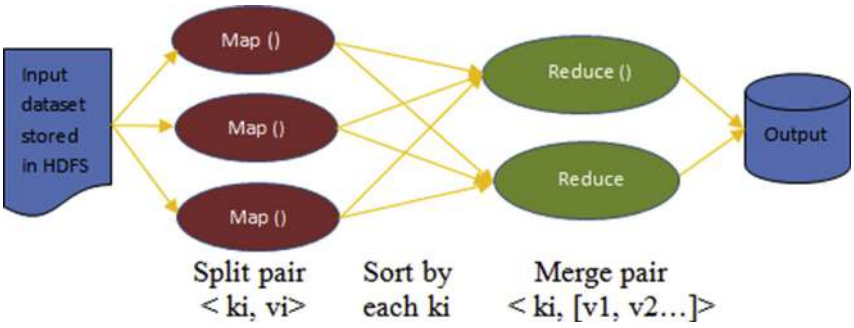

Fig. 2. A basic MapReduce data processing scheme.

appropriate input. In literature documents are preprocessed using various techniques such as vector model, graphical model, stemming etc. K-means can take clustering input of numeric values. Hence, the dataset is processed using vector-space model technique which represents each word in the dataset into its normalized numeric quantity based on its occurrences in the dataset. The normalized numeric quantity is then stored in a file and then provided as an input to the k-means through HDFS.

Transformation of documents dataset is suitable to input in k-means as it represents a dataset to a set of multidimensional numeric vector values. This multidimensional vector is constituted by many single dimensions. A dimension is the weight of each unique word (term) in the dataset. The "weight" reflects the relevancy of the corresponding terms in the given dataset. If a corpus comprises of n terms, let t_i, where $i = 1…n$, then document d from that dataset would be characterized with the vector: $d = \{w_1, w_2, …, w_n\}$, where w_n are weights linked with terms t_i. The relationship degree among documents can be characterized by distance among vectors of the corresponding documents in vector space. The dataset is converted to vector space model after implementing widely used pre-processing techniques such as normalizing the text, removing terms with very small/high frequency by using Zipf's rule (logarithmic scaling), removing the so-called stop-words, reducing words to their root form through stemming.

Step 2: Selection of K cluster centroids: K-means requires the number of output cluster number to be supplied before execution of the algorithm. Hence, we provided the number

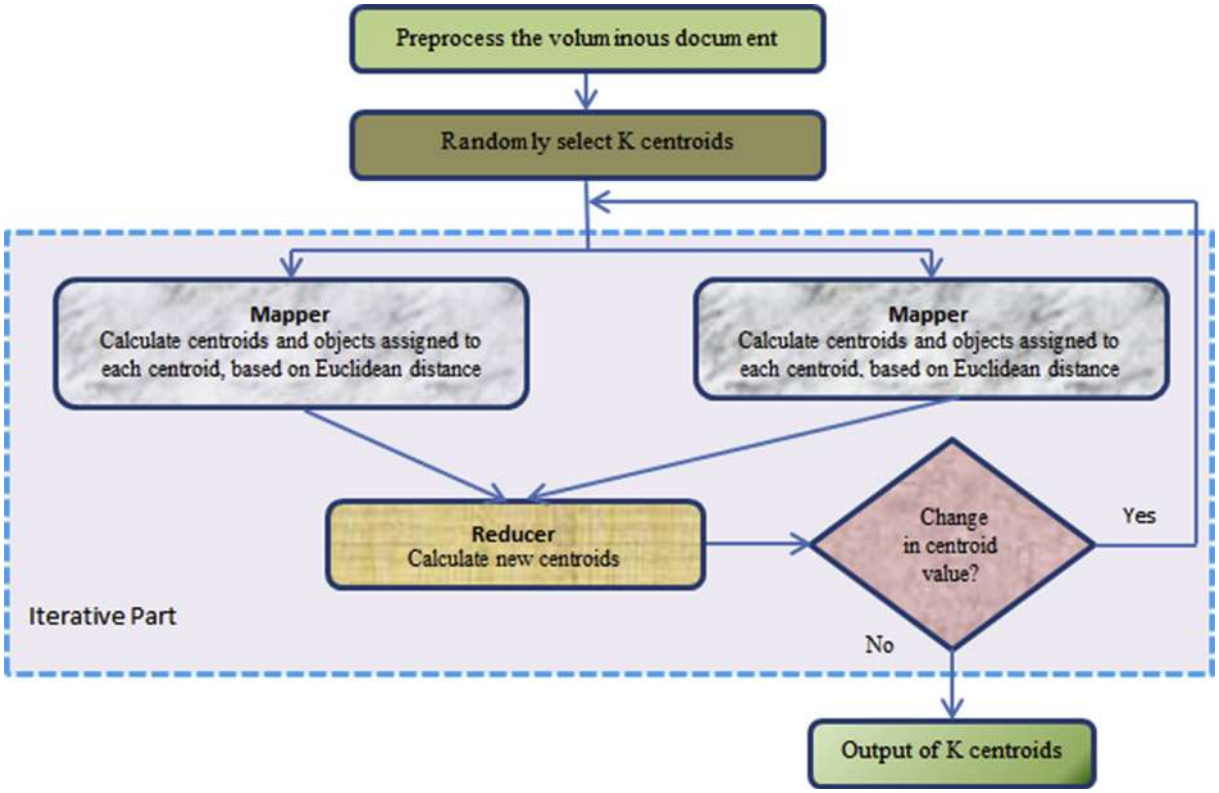

Fig. 3. The Stages of our document clustering using parallel K-means.

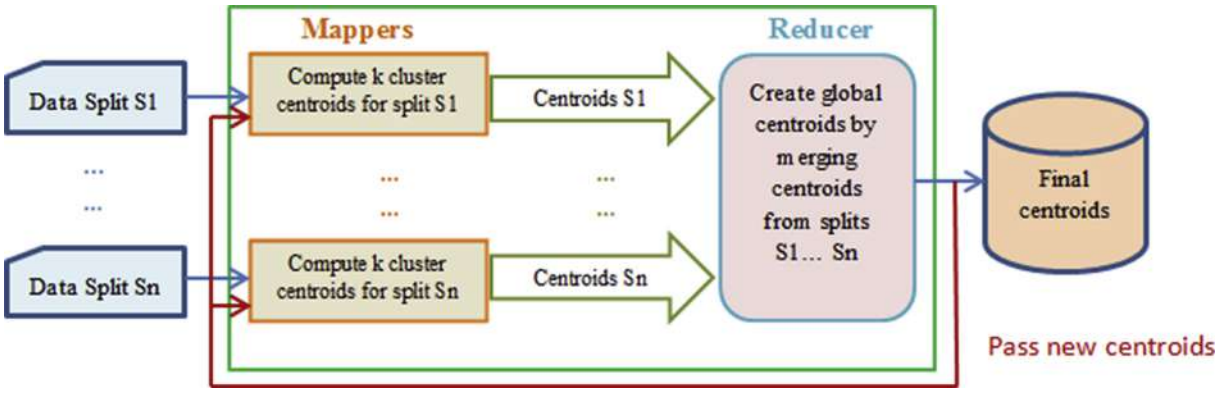

Fig. 4. Parallel K-means iterations in a Hadoop cluster.

of cluster centroid K before execution of the algorithm. The number of centroids, K, are randomly generated by the program and stored in a file named cluster_centroids. These values of centroids are used in first iteration of parallel k-means in map function. Subsequent map operations will iteratively recalculate cluster centroids and will update the centroid values, keeping the value of K same.

Step 3: *MapReduce parallel execution of k-means:* The dataset after preprocessing generates numeric vector valued representation of the input document dataset and the vectors are stored in a file named input_data. The number of clusters, k, is then taken as input from the user. The k number of vector values, the cluster centers, is taken randomly from the dataset and stored in a file named centroid_data. Input_data is split across slave nodes and centroid_data is copied to each slave by the HDFS. The parallel execution of our k-means in MapReduce is conducted on a Hadoop cluster of 10 nodes.

The choice of implementing an algorithms by dividing it into map and reduce parts is problematic. We observed that the execution of k-means can be divided into two parts: firstly, the parallel and iterative part of calculating distance between centroids and dataset objects, which as a result assign each object to the nearest centroid; secondly the sequential part of updating new centroids after objects are assigned to it after each iteration. Taking the above observation, we designed our parallel k-means such that the map function accomplishes the job of assigning each object to the nearest center while the reduce job achieves the procedure of updating the new cluster centers, until it remains unchanged. A pictorial overview is presented in Fig. 4. When the centroid values remain unchanged it indicates that the clustering job is accomplished successfully and the result is displayed.

3.1. Algorithm for mapper

Input: A set of document objects transformed in numeric vector form O = {o1, o2, …, on}, randomly chosen set of initial cluster centers C = {c1, c2, ck}

1. F1← {o1, o2,……,on}
2. existing_centroids←C
3. Distance (x, y) = $\sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$, where x, y are objects and in dimension i
4. FOR all oi ϵ F1 such that 1≤i≤f do
   bestCentroid=NULL
   dist_min=∞
5. FOR all values of C do
   similarity=distance (oi , c)
6. IF (bestCentroid = null || similarity < dist_min) then
   dist_min < similarity
   bestCentroid=c
7. END IF
8. END FOR
   release (bestCentroid, oi)
   i=i+1
9. END FOR
10. Return intermediate <key, value> pair

Output: An intermediate <key, value> pair of (C_i, O_j) where $1 \leq i \leq n$ and $1 \leq j \leq k$.

Function:

3.2. Algorithm for reducer

Input: (Key, Value) pair outputted by mapper, where key = bestCentroid and value = Objects assigned to it.

```
1. oppair =oppair (<Key, Value> pairs from mappers)
2. [centroid] = { }
3. newCentroid =NULL
4. Let α= <key, value> pair outputted from mappers
5. FOR all α ϵ oppair do
       oldCentroid ← α.key
       object ← α.value
       [centroid] ← object
6. END FOR
7. FOR all centroid ϵ [centroid] do
       sumofAllObjects= NULL
       FOR all object ϵ [centroid] do
              sumofAllObjects = sumofAllObjects + object
              numofObjects = numofObjects+ 1
8.     END FOR
              newCentroid = (sumofObjects / numofObjects)
              release (oldCentroid, newCentroid)
9. END FOR
```

Output: <Key, Value> pair where key and value are oldCentroid and newCentroid respectively; bestCentroid of mapper is used to calculate value for newCentroid.

Function:

4. Results and analysis

4.1. Experimental setup

The experiment was carried out on a Hadoop cluster of ten nodes. The nodes in the Hadoop cluster are configured with Intel Core 2 Duo CPU@ 2.53 GHZ processor, 8 GB of DDR3 RAM for each node and 80 GB of hard disk with a measured bandwidth for end-to-end TCP sockets of 100 MB/s. Operating system used is Ubuntu 14.04 LTS and Hadoop version 2.7.2. The dataset we used in this experiment are newsgroup document. It contains news from different categories such as sport, politics, religion etc. It has 20 directories, each consist of news of a particular category. The dataset is a collection oflarge amounts of unstructured text data. The dataset consist of different sizes of 100, 250, 500, 750 and 1024 megabytes.

4.2. Experimental result and its analysis

The variation in dataset sizes helps evaluate the performance gain of our proposed algorithm effectively. The sequential k-means algorithm is also experimented with the same datasets and system configuration and performance differences between sequential and proposed k-means is presented and analyzed.

In Table 1, the observation of different execution time with respect to different dataset is provided for sequential and parallel execution of k-means. To evaluate the performance of proposed k-means, data scale up method of evaluation is used. In our data scale up experiments, proposed k-means and sequential k-means are executed with respect to different dataset sizes for a fixed size of 10 node Hadoop cluster and the execution time is recorded for each experiments. The resulting execution time of experiments provides a framework to analyze and evaluate the performance differences between proposed and sequential k-means.

Table 1
Execution time in seconds.

Algorithm	Observation	Dataset size				
		100 MB	250 MB	500 MB	750 MB	1024 MB
Sequential k-means	Execution time	151	195	520	637	649
	Ratio	1	1.3	3.4	4.2	4.3
Proposed k-means in 10 node cluster	Execution time	52	87	110	133	140
	Ratio	1	1.7	2.1	2.5	2.7
Proposed: sequential execution time		1:2.9	1:2.5	1:4.7	1:4.8	1:4.6

The execution time taken by sequential k-means is 151, 195, 520, 637 and 649 s for 100 MB, 250 MB, 500 MB, 750 MB and 1024 MB of datasets whereas proposed k-means took 52, 87, 110, 133 and 140 s respectively. We have analyzed the execution time obtained from all experiments in order to retrieve the performance of proposed k-means and compared with sequential k-means. Ratio is an effective tool for comparison. Ratio represents a relationship between two quantities that the number of times one value contains or is contained within the other. To critically analyze and evaluate performance of proposed k-means, execution time of 100 MB of dataset is taken as a unit and execution time of other dataset is compared using ratio calculation and representation. For example, observed execution time of proposed k-means for 100 MB dataset which is 52 s is considered as a unit and the execution time of other dataset is compared and represented in the table. The ratio of execution time between sequential and proposed k-means with respect to different dataset size is also shown in Table 1. A ratio is also calculated for comparing the efficiency between sequential and proposed k-means. The performance gain of proposed k-means against sequential k-means can be directly obtained from this ratio. For example, the ratio between sequential and proposed k-means is 1: 2.5 for 250 MB of dataset size. Hence the performance gain of proposed k-means over sequential k-means is two and half time more than sequential k-means. The sequential and proposed k-means execution time for different dataset size is also depicted using bar charts in Fig. 5 and Fig. 6 respectively and using line charts using Fig. 7 and Fig. 8 respectively. The execution time of our proposed algorithm is significantly lesser than the sequential implementation of k-means. It is 2.9, 2.5, 4.7, 4.8 and 4.6 times faster than sequential k-means for 100 MB, 250 MB, 500 MB, 750 MB and 1024 MB datasets. It is also observed that the clustering using proposed algorithm becomes more efficient as the input dataset becomes larger. For example, it takes 52 s to cluster 100 MB dataset whereas for 1024 MB dataset, which is 10 times large, it takes only 140 s to achieve clustering, which is just 2.7 times more time consuming. This is due to the fact that MapReduce performs more efficiently for the large datasets. The execution time of proposed algorithm is depicted using bar graphs and line graphs in Figs. 5 and 6 respectively. Similarly, the execution time of sequential k-means is depicted using bar graphs and line graphs in Figs. 7 and 8 respectively.

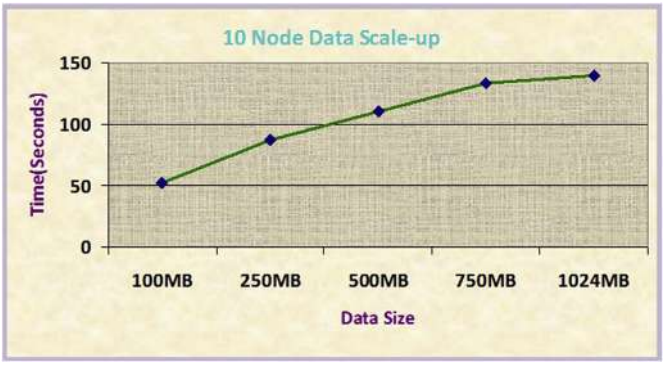

Fig. 6. Line graph of proposed K-means execution.

It is clearly observed from the bar and line charts that the increase of execution time with respect to increase of data size is not very flat i.e. it cannot be portrayed using a line graph with a straight line. This uneven increase of execution time for dataset size happens in almost all Hadoop clusters as nodes in the cluster have to carry many other overhead other than MapReduce execution. Some of the overheads a cluster has to bear is processing overhead of systems processes and tools (like background antivirus tools) execution with different priority, networking overhead between nodes, replication factor of Hadoop, HDFS checksum etc.

5. Conclusion and future work

MapReduce programming model for Hadoop cluster is a recent and popular trend in analyzing large datasets in short span of time. It is important to parallelize clustering algorithms using MapReduce for efficiency in clustering result in terms of execution time. This work proposed a parallel k-means algorithm using MapReduce for document clustering and the execution time of clustering job is compared with sequential k-means algorithm with datasets of different size. The proposed algorithm is able to cluster datasets in short span of time by utilizing the Hadoop cluster of 10 nodes. The experimental results give us the following insight:

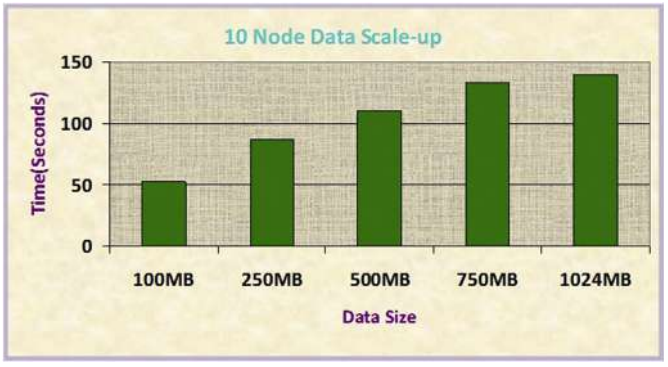

Fig. 5. Bar graph of proposed K-means execution.

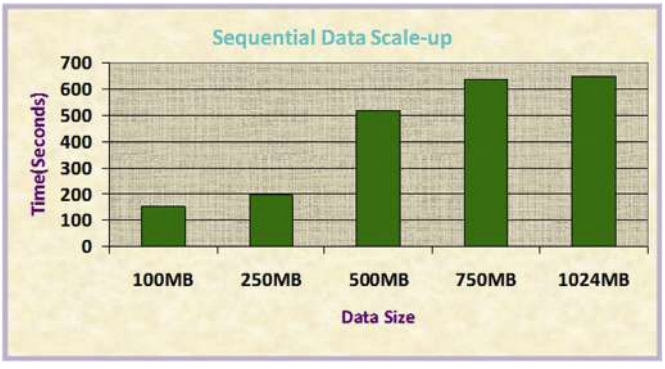

Fig. 7. Bar graph of sequential K-means execution.

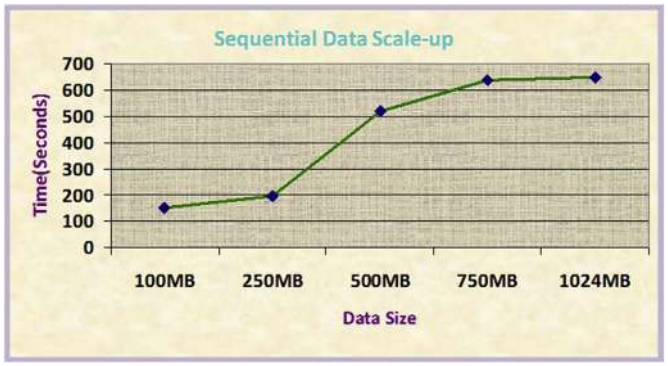

Fig. 8. Line graph of sequential K-means execution.

- Document clustering can be effectively implemented in MapReduce by writing suitable mapper and reducer parts in k-means.
- The efficiency of the proposed k-means outperforms sequential k-means in terms of execution time.
- The proposed algorithm is more efficient while clustering larger datasets than smaller.
- It is observed from line charts and bar graphs that the processing time of Hadoop cluster for different sized dataset is not uniform, although same configuration of nodes and same algorithm is used.

In this study, the proposed k-means is just modified in order to execute on top of Hadoop. The inherent issues of k-means are left overlooked and unsolved. Like sequential k-means, proposed k-means is also required to be supplied with k cluster numbers prior the execution. Likewise, the clustering result of proposed k-means is dependent on initial centroid selection. There are always many possibilities available for all research work to be improved. As a future work we can incorporate proposed algorithm and framework with the concepts provided below:

- The proposed k-means can be modified such that it can automatically settle on the number of cluster and effectively select initial centroids depending on the datasets.
- Proposed k-means can be combined with techniques like hierarchical clustering algorithms, swarm intelligence, fuzzy logic, gravitational search algorithms, neural networks etc. in order to obtain better quality clustering with efficiency.
- Optimization of the proposed algorithm can also be done by tuning the number of mapper and reducer in the code effectively and/or adding a combiner between mapper and reducer.
- Hadoop provides choices to optimize on disk, memory, network and CPU. Hadoop cluster can be optimized for each particular job for more efficiency.

Conflicts of interest

The authors declare that they have no conflict of interest.

Ethical approval

This article does not contain any studies with human participants or animals performed by any of the authors.

Acknowledgement

This study was facilitated by Government of Karnataka (GoK) Vision Group on Science and Technology (VGST) CISEE (2015−16) scheme grant GRD No-461.

References

[1] Shettar Rajashree, Bhimasen, Purohit V. A review on clustering algorithms applicable for map reduce. In: Proceedings of the international conference computational systems for health & sustainability; April, 2015. p. 17−8. Bangalore, Karnataka, India.
[2] Olman Victor, Mao Fenglou, Wu Hongwei, Xu Ying. Parallel clustering algorithm for large data sets with applications in bioinformatics. IEEE ACM Trans Comput Biol Bioinf 2009;6(2):344.
[3] Neepa Shah, Mahajan Sunita. Document clustering: a detailed review. Int J Appl Inf Syst 2012;4(5):30.
[4] Sunita Bisht, Amit Paul. Document clustering: a review. Comput Appl 2013;73(11):0975.
[5] Michael Steinbach, Karypis George, Kumar Vipin. A comparison of document clustering techniques. KDD workshop on text mining. 2000. p. 400−1.
[6] Jing Zhang, Wu Gongqing, Hu Xuegang, Li Shiying, Hao Shuilong. A parallel clustering algorithm with mpi-mkmeans. J Comput 2013;8(1):10.
[7] Wu Xindong, Kumar Vipin, Quinlan J Ross, Ghosh Joydeep, Yang Qiang, Motoda Hiroshi, et al. Top 10 algorithms in data mining. Knowl Inf Syst 2008;14(1):1. https://link.springer.com/journal/10115.
[8] Bawane Vinod S, Kale Sandesha M. Clustering algorithms in MapReduce:a review. Int J Comput Appl 2015:0975. Special Issue of National Conference on Recent Trends in Computer Science & Engineering (MEDHA 2015).
[9] Tanvir Habib Sardar, Ahmed Rimaz Faizabadi, Zahid Ansari. An analysis of data processing using MapReduce paradigm on the Hadoop framework, international conference on emerging trends in science & engineering (ICETSE−2017) conference held by IEAE India, at coorg institute of technology, Ponnampet, Karnataka. India Int J Emerg Res Manag Technol 2017;6(5):922−7.
[10] Sulun Erhan. Improvements in K-means algorithm to execute on large amounts of data (Master of Science Dissertation). Turkey: Izmir Institute of Technology Izmir; 2004.
[11] Kumar Praveen, Bariker Nirmala. Implementation of Hadoop based framework for parallel processing of biological data. Int J Sci Res (IJSR) 2015;4(4):1087.
[12] Willson Joseph C, Pushpalatha B. A survey on big data and Hadoop. Int J Innovat Res Comput Commun Eng 2017;5(3):5525.
[13] Habib Sardar Tanvir, Rimaz Faizabadi Ahmed, Ansari Zahid. An evaluation of MapReduce framework in cluster analysis. In: 2017 IEEE international conference on intelligent computing. Kannur, India: Instrumentation and Control Technologies (ICICICT); 2017.
[14] Fahad Adil, Alshatri Najlaa, Tari Zahir, Alamri Abdullah, Khalil Ibrahim, Zomaya Albert Y, et al. A survey of clustering algorithms for big data: taxonomy and empirical analysis. IEEE Trans Emerg Top Comput 2014;2(3):267.
[15] Bradley PS, Fayyad UM. Refining initial points for K-Means clustering. In: Proceedings of the fifteenth international conference on machine learning; 1998. p. 91−9.
[16] Ball G, Hall D. A clustering technique for summarizing multivariate data. Behav Sci 1967;12:153.
[17] Arthur D, Vassilvitskii S. k-means++: the advantages of careful seeding. In: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms; 2007. p. 1027−35.

[18] Kanungo T, Mount D, Netanyahu N, Piatko C, Silverman R, Wu A. An efficient K-means clustering algorithm: analysis and implementation. IEEE Trans Pattern Anal Mach Intell 2002;24(27):881.
[19] Frahling G, Sohler C. A fast K-means implementation using coresets. In: Proceedings of the twenty-second annual symposium on computational geometry; 2006. p. 135−43.
[20] Amorim R, Mirkin B. Minkowski metric, feature weighting and anomalous cluster initializing in K-means clustering. Pattern Recogn 2012; 45(3):1061.
[21] Likas Aristidis, Vlassis Nikos, Verbeek Jakob J. The global k-means clustering algorithm. Pattern Recogn 2003;36(2):451.
[22] Forman George, Zhang Bin. Distributed data clustering can be efficient and exact. ACM SIGKDD Explor Newslett 2000;2(2):34.
[23] Dhillon Inderjit S, Modha Dharmendra S. A data-clustering algorithm on distributed memory multiprocessors. Springer. Berlin, Heidelberg: Large-Scale Parallel Data Mining; 2002. p. 245−60.
[24] Bhimani Janki, Leeser Miriam, Mi Ningfang. Accelerating K-Means clustering with parallel implementations and GPU computing. In: High performance extreme computing conference (HPEC). IEEE; 2015.
[25] Yang Luobin, Chiu Steve C, Liao Wei-Keng, Thomas Michael A, et al. High performance data clustering: a comparative analysis of performance for GPU, RASC, MPI, and OpenMP implementations. J Supercomput 2014;70(71):284.
[26] Kang, Ji Sol, Yeon Lee Sang, Lee Keon Myung. Performance comparison of OpenMP, MPI, and MapReduce in practical problems. Adv Multimed 2015;7.
[27] Jinlan Tian, Lin Zhu, Suqin Zhang, Lu LIU, et al. Improvement and parallelism of k-means clustering algorithm. Tsinghua Sci Technol 2005; 10(13):277.
[28] Ping ZHOU, Jingsheng LEI, Wenjun YE. Large-scale data sets clustering based on MapReduce and Hadoop. J Comput Inf Syst 2011; 7(16):5956.
[29] Marisiddanagouda M, Mr Raghu MT. Survey on performance of Hadoop MapReduce optimization methods. Int J Rec Res Math Comput Sci Inf Technol 2015;2(1):114.
[30] Nagarjuna, Yogesh. A survey on Hadoop architecture & its ecosystem to process big data -real world Hadoop use cases. Int J Sci Res Eng Technol (IJSRET) 2015;4(2):90.
[31] Verma, Jain. Amazon hadoop framework used in business for big data analysis. Global J Eng Sci Res Manag 2017;4(5):131.
[32] Patil Vishal S, Soni Pravein D. Hadoop skeleton & fault tolerance in Hadoop clusters. IJAIEM 2013;2(2):247.
[33] Singh, Kumar Vivek, Tiwari Nisha, Garg Shekhar. Document clustering using k-means, heuristic k-means and fuzzy c-means. In: Computational intelligence and communication networks (CICN), 2011 international conference on. IEEE; 2011.
[34] Singh, Hardeep Mr. Clustering of text documents by implementation of K-means algorithms. Streamed Info Ocean January-June 2016;1(1).
[35] Balabantaray, Chandra Rakesh, Sarma Chandrali, Jha Monica. Document clustering using K-means and K-medoids. 2015. arXiv preprint arXiv: 1502.07938.