Sreedhar *et al. J Big Data (2017) 4:27*
DOI 10.1186/s40537-017-0087-2

Journal of Big Data

RESEARCH Open Access

Clustering large datasets using K-means modified inter and intra clustering (KM-I2C) in Hadoop

Chowdam Sreedhar[1*], Nagulapally Kasiviswanath[2] and Pakanti Chenna Reddy[3]

*Correspondence:
csrgprec@gmail.com
[1] Faculty of Computer Science and Engineering, G Pulla Reddy Engineering College, Kurnool, India
Full list of author information is available at the end of the article

Abstract

Big data has become popular for processing, storing and managing massive volumes of data. The clustering of datasets has become a challenging issue in the field of big data analytics. The K-means algorithm is best suited for finding similarities between entities based on distance measures with small datasets. Existing clustering algorithms require scalable solutions to manage large datasets. This study presents two approaches to the clustering of large datasets using MapReduce. The first approach, K-Means Hadoop MapReduce (KM-HMR), focuses on the MapReduce implementation of standard K-means. The second approach enhances the quality of clusters to produce clusters with maximum intra-cluster and minimum inter-cluster distances for large datasets. The results of the proposed approaches show significant improvements in the efficiency of clustering in terms of execution times. Experiments conducted on standard K-means and proposed solutions show that the KM-I2C approach is both effective and efficient.

Keywords: Clustering, Big data, Hadoop, MapReduce, Scalability

Introduction

In the recent years, datasets generated by machines have been large in terms of volume and have been globally distributed [1]. Big data can be described based on various characteristics (namely volume, velocity, variety, veracity, value and volatility). Big data include datasets that are large and difficult to manage, acquire, store, analyse and visualize. A dataset can be defined as a collection of related sets of information that can have individual or multidimensional attributes. Large datasets include massive volumes of data such that traditional database management systems cannot manage them. Big data has become popular due to their ability to manage structured, unstructured and semi-structured data sources and formats [2] through the use of advanced data intensive technologies. The increased size of datasets has boosted demand for efficient clustering techniques that satisfy memory use, document processing and execution time requirements. An issue related to big data concerns the grouping of objects such that data of the same group are more similar than those of the other groups or clusters. Applications of big data are used in telecommunications, healthcare, bioinformatics, banking,

marketing, biology, insurance, city planning, earthquake studies, web document classification and transport services.

Clustering is an important tool for data mining and knowledge discovery. The objective of clustering is to find meaningful groups of entities and to differentiate clusters formed for a dataset. Traditional K-means clustering works well when applied to small datasets. Large datasets must be clustered such that every other entity or data point in the cluster is similar to any other entity in the same cluster. Clustering problems can be applied to several clustering disciplines [3]. The ability to automatically group similar items enables one to discover hidden similarities and key concepts while combining a large amount of data into a few groups. This enables users to comprehend a large amount of data. Clusters can be classified as homogeneous and heterogeneous clusters. In homogeneous clusters, all nodes have similar properties. Heterogeneous clusters are used in private data-centres in which nodes have different characteristics and in which it may be difficult to distinguish between nodes [4].

Clustering methods require the application of more precise definitions of observation and cluster similarities. When grouping is based on attributes, it is natural to employ familiar concepts of distance. A problem with this procedure is associated with the measurement of distances between clusters comprising of two or more observations. Unlike existing conventional statistical methods, most clustering algorithms do not rely on statistical distributions of data and thus can be helpful to apply when little prior knowledge exists on a certain issue [5]. McCallum et al. [6] described how the number of iterations can be reduced by partitioning a dataset into overlapping subsets and by only iterating data objects within overlapping areas.

When using flat file formats, data objects are represented as vectors in n-dimensional space that each describe an object, and this object is characterized by n attributes, each of which has a single value. Almost all existing data analysis and data mining tools such as clustering tools, inductive learning tools, and statistical analysis tools assume that datasets to be analysed are represented through a structured file format. The well-known inductive learning environment [7] and a similar decision tree based rule induction algorithm [8], as well as conceptual clustering algorithms (e.g., COBWEB [9], AutoClass [10, 11], and ITERATE [12]) and statistical packages, make this assumption. Issues of scalability are becoming a major concern when applying clustering algorithms as datasets increase in size; most are computationally expensive in terms of space and time. There is a need to manage such large volumes of data and to cluster them easily for data analytics while minimizing maximum inter-cluster distances and managing large datasets. Furthermore, such algorithms should be efficient, scalable and highly accurate.

The K-means clustering algorithm is a popular unsupervised clustering technique used to identify similarities between objects based on distance vectors suited to small datasets. Currently, datasets generated by sources such as Wikipedia, meteorological departments, telecommunications systems, and sensors are so large that traditional K-means clustering algorithms are no longer able to group related objects to develop meaningful insights. There is a need to enhance clustering algorithms to suit large datasets. Hadoop was designed to store datasets that can be scaled up to the petabyte level. Hence, the present work develops a clustering solution using Hadoop.

We apply the Hadoop MapReduce standard K-means clustering algorithm to manage large datasets and introduce a new metric for similarity measurements such that the distances between objects exhibit high levels of intra-cluster similarity and low levels of inter-cluster similarity.

The remainder of this study is organized as follows. Popular distance metrics and the need for Hadoop are discussed in "Background". Approaches recently developed to improve clustering results are discussed in "Related work". The proposed KM-HMR and KM-I2C algorithms are described in "Proposed work". An extensive set of experiments on the datasets along with a comparison of existing and proposed solutions is presented in "Experimental evaluation". Finally, conclusions and future work are discussed in "Conclusions".

Background

Applications based on similarity measures include the following: data mining [13], image processing [14], and document clustering [15]. The most popular continuous distance metric is the Euclidean distance. City-block measures partition based on a central median [16].

The following are popular distance metrics used in various clustering techniques:

a. Euclidean distance.
b. Manhattan distance.
c. Minkowski distance.
d. Jaccard distance.

Euclidean distance is defined as:

$$d(i,j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{in} - x_{jn})^2} \tag{1}$$

where, $i = (x_{i1}, x_{i2},\ldots, x_{in})$ and $j = (x_{j1}, x_{j2},\ldots, x_{jn})$ are two n-dimensional data objects.

The Manhattan (or city-block) distance, is defined as:

$$d(i,j) = \left|(x_{i1} - x_{j1})\right| + \left|(x_{i2} - x_{j2})\right| + \cdots + \left|(x_{in} - x_{jn})\right| \tag{2}$$

where, $i = (x_{i1}, x_{i2},\ldots, x_{in})$ and $j = (x_{j1}, x_{j2},\ldots, x_{jn})$ are two n-dimensional data objects.

The Minkowski distance, is defined as:

$$d\,(i,j) = \left(\left|x_{i1} - x_{j1}\right|p + \left|x_{i2} - x_{j2}\right|p + \cdots + \left|x_{in} - x_{jn}\right|p\right)1/p \tag{3}$$

where, $i = (x_{i1}, x_{i2},\ldots, x_{in})$ and $j = (x_{j1}, x_{j2},\ldots, x_{jn})$ are two n-dimensional data objects.

The Jaccard distance computes similarity between two sets of concepts as follows:

$$J(U,A) = \frac{|U \cap A|}{|U \cup A|} \tag{4}$$

Jaccard distance measure is defined as the size of intersection divided by size of union of datasets. $|U \cap A|$ is the number of concepts in the intersection of U and A; $|U \cup A|$ represents the number of concepts in the union of U and A. It is a measure of similarity

for the two datasets U, A which is used to find binary differences between two or more objects. Summary of the various distance metrics used in clustering is shown in Table 1.

Hadoop was created by Doug Cutting in 2005 and has its origins in Apache Nutch, an open source Internet search engine. Apache Hadoop is an open source iteration of MapReduce, which is a framework designed for the in-depth analysis and processing of large volumes of data. Hadoop can analyse complex data of various formats such as structured, semi-structured and unstructured data across several machines. Examples of unstructured data include books, journals, documents and metadata. Structured data include quantitative facts such as numbers and dates. Xml and json documents are considered as an examples for semi-structured data. Hadoop manages its own file system, the Hadoop distributed file system (HDFS), which replicates data to several nodes to ensure the availability of data. Large collections of unstructured data must be managed using a better clustering algorithm with high levels of intra-cluster similarity and with low levels of inter-cluster similarity to derive insights. Applications using unstructured data include, for instance, porphyry copper deposit datasets [17].

Hadoop and Spark serve the same purpose of processing large datasets. Spark is an open source platform that can execute map and reduce jobs several times faster than Hadoop. The overall performance of a particular platform cannot be based on a single criterion [18]. Hadoop offers the following advantages over Spark. Spark's performance is mainly hindered by the fact that it executes more map and reduce tasks than Hadoop and this can affect resources and node workloads when running clustering jobs. Spark uses a considerable amount of memory to shuffle files written onto disks because it does not manage its own file management system [19]. Hadoop may be used over Spark for the other following reasons: low-cost hardware, lower latency value, and the capacity to manage large datasets with the help of an HDFS.

In this work, we use Hadoop instead of Spark to enhance overall performance levels over a reasonable execution period using low-cost machines whereby data operations and reporting requirements are static and where data are processed via batch processing.

Table 2 illustrates the benefits of Hadoop technologies relative to traditional relational database management systems. Hadoop is best known for offering two main features: HDFS and MapReduce. Other sub-projects of Hadoop technologies include the following:

- Pig [20]: A high-level data flow language and execution framework for parallel computation.

Table 1 Summary of distance metrics

Distance metric	Equation	Explanation
Euclidean distance	$D(i,j) = \sqrt{\sum_{i=1}^{n} \lvert Xi - Yi \rvert^2}$	Popular distance metric used. Suitable for small datasets
Manhattan distance	$D(i,j) = \sum_{i=1}^{n} \lvert Xi - Yi \rvert$	Distance based on an absolute value. Measures each partition based on the mediancentre. Suitable for compact clusters
Minkowski distance	$D(i,j) = \sqrt[p]{\sum_{i=1}^{d} \lvert Xi - Yi \rvert^p}$	Features with large values and variances tend to dominate other features. Suitable for numeric datasets
Jaccard distance	$J(U,A) = \frac{\lvert U \cap A \rvert}{\lvert U \cup A \rvert}$	Used as a measure of similarity. Generally applied to binary values to measure distances between objects

Table 2 Need for Hadoop technology

	Traditional RDBMS	Hadoop MapReduce
Size	To gigabytes	Extends to petabytes
Processing	Realtime and batch	Batch
Updates	Reads and writes many times	Writes once and reads many times
Structure	Static schema	Dynamic schema
Integrity	High	Low
Scaling	Non-linear	Linear

- Zookeeper [21]: A high-performance coordination service for distributed applications.
- HCatalog [22]: A table and storage management service for Hadoop data.
- Hive [23]: An SQL-like interface for data stored.
- Avro [24]: A language-neutral data serialization system and language-independent scheme associated with read and write operations.
- HBASE [25]: An open-source non-relational distributed database that runs on top of an HDFS that affords Hadoop with Bigtable-line capabilities.
- Chukwa [26]: An open-source tool for monitoring the Hadoop cluster.

A MapReduce job splits a dataset into chunks of data such that map tasks are processed in a parallel manner. The framework sorts the outputs of maps, which are then input to reduce the number of tasks involved. Inputs and outputs of a job are stored in a file system. The framework manages scheduling tasks, monitors tasks and re-executes failed tasks. The HDFS is a distributed file system that uses master/slave architecture and that is capable of storing large volumes of data. MapReduce jobs are executed on a cluster of nodes while carrying out large-scale data operations that are scalable and feasible to execute in a reasonable amount of time for complex or large datasets. MapReduce works in parallel with several clusters of computers, making it easy to scale to large datasets.

Related work

MapReduce programming is designed for computer clusters. MapReduce applications can process large datasets using several low-cost machines referred to as clusters. Individual computers in a cluster are often referred to as nodes in that cluster. MapReduce involves two main broad computational stages (a map phase and reduce phase) that are applied in sequence on large volumes of data. The map function applies to each line of data and breaks data into chunks to form key-value pairs. A reducer function is then applied to all key-value pairs sharing the same key.

Robson et al. [27] focus on problems faced when clustering large datasets(e.g., obtaining efficient clusters over better execution times for large datasets) and propose a parallel clustering (ParC) method for data partitioning that leverages data localities in MapReduce to cluster large datasets. When clustering large datasets, parallel processing is best suited to Hadoop MapReduce due to its capacity to divide large datasets into chunks

and to store them in worker nodes while an algorithm is applied to the data where it is stored. Best of both Worlds (BoW) is an adaptive technique used to dynamically select the best ways to minimize costs. The top ten eigenvectors of Twitter stored as an adjacency matrix have been considered and experiments conducted using proposed solutions reveal good results. ParC focuses on two issues: minimizing I/O costs and reducing costs related to networks. In developing a good clustering algorithm, speedup and scale up processes are two major issues that must be exploited through parallelism techniques to generate efficient clustering algorithms.

Parallel K-means clustering based on MapReduce [28] clusters large datasets and applies a K-means clustering algorithm via the Hadoop MapReduce framework suited for large datasets. Applying the Fast K-means clustering algorithm through MapReduce has been proposed. The mean values of data points in clusters are used to measure cluster similarities and MapReduce computes distances between data points simultaneously. The map function stores input datasets in an HDFS which is used to assign data objects or points to the closest centre and which reduces functions in updating calculated centres based on the Euclidean distance. The results have been analysed based on various sizes of datasets, better results have been achieved with increasing dataset size. The Fast K-means algorithm is suited to large datasets but for a good clustering algorithm addressing large datasets, cluster efficiency and inter-cluster and intra-cluster measures should be considered apart from execution times and should be scaled up to meet input data size requirements.

Chen [29] proposed the use of MapReduce-based soft clustering for large datasets. A divide and conquer approach is used to split large amounts of data into chunks and the MapReduce model causes each of map and reduce phase to work in independent of other running map and reduce phases which are run in parallel. Clustering quality metrics have not been considered and evaluations based on different workloads (dataset size) should also be conducted. To overcome this drawback, a two-phase K-means (TPK) method [30] that can efficiently process large volumes of data has been introduced.

Chu et al. [31] applied a big data parallel programming technique involving K-means clustering through the MapReduce framework. Xia et al. [32] proposed a parallel K-means optimization algorithm (Par3PKM) implemented through the MapReduce framework. As a distance metric used in Par3PKM, the Euclidean distance is employed to calculate distances between data points or objects and cluster centroids. Parallelization and optimization techniques can cluster large datasets through MapReduce. The parallel CoMK-means clustering algorithm [33] uses MapReduce to distribute input data across several slave nodes by using an HDFS to overcome large dataset clustering instabilities.

The clustering of large bio-informatics datasets [34] focuses on gene sequence analyses conducted via the MapReduce framework. The sizes of bioinformatics datasets range from a few gigabytes to several petabytes depending on the genes of living creatures and plants. In addressing large volumes of data, clustering plays a vital role in analysing a particular pattern and in finding similarities between two different genes. With

such volumes of data, parallelism serves as the only means to generate efficient clustering results. The expressed sequence tag (EST) [35] is a clustering tool used to group sequences originating from the same gene to form clusters so that they are similar to the representative. As a drawback of EST clustering, when finding similarities within the same cluster and between two different clusters from genetic datasets, good inter-cluster and intra-clustering measures must be used.

Fang et al. [36] used large-scale meteorological datasets and an MK-means algorithm applied through MapReduce to manage meteorological information stored in high-cost servers over several years. Optimal K-centroids were calculated using the standard distance measure of similarity and the Euclidean distance. The performance of MK-means is investigated for different datasets of different sizes. Stability and scalability were found to be two major factors considered in MK-means. The MK-means clustering algorithm is limited in terms of the quality of clusters and distance metric efficiency.

Tsai et al. [37] propose a MapReduce black hole (MRBH) for leveraging the strength of the black hole algorithm through the MapReduce framework to accelerate clustering processes.

Proposed work

The proposed work involves two approaches: K-means Hadoop MapReduce (KM-HMR) and K-means modified inter and intra clustering (KM-I2C). The following section describes the proposed algorithms for efficient clustering.

K-means Hadoop MapReduce (KM-HMR)

Algorithm 1 describes KM-HMR, a MapReduce implementation of K-means that can find clusters faster than standard K-means clustering methods. KM-HMR focuses on the MapReduce implementation of standard K-means, which serves as a framework for parallelization problems (e.g., clustering) that takes advantage of localities of data and which involves two major phases: a map phase and a reduce phase. The MapReduce job involves splitting a dataset into parts of a fixed sized referred to as chunks, which are consumed by a single map. The map phase calculates the distances between each object and each cluster and assigns each object to its nearest cluster. One map task is created for each input split and is executed by map functions for each record of the input split. The input split size is set to 128 MB and can the split size can be set to the size of an HDFS block. The distance metric used in KM-HMR is the Euclidean distance. The objects that belong to the same cluster are sent to reduce phase. The reduce phase calculates the new cluster centroids for the next MapReduce job. Figure 1 depicts the overall flow of KM-HMR. Cluster centroids produced at the end of an initial iteration are stored in an old cluster file and are tested for the appearance of new cluster centroids with each iteration. When new cluster centroid values are obtained, new cluster centroid values

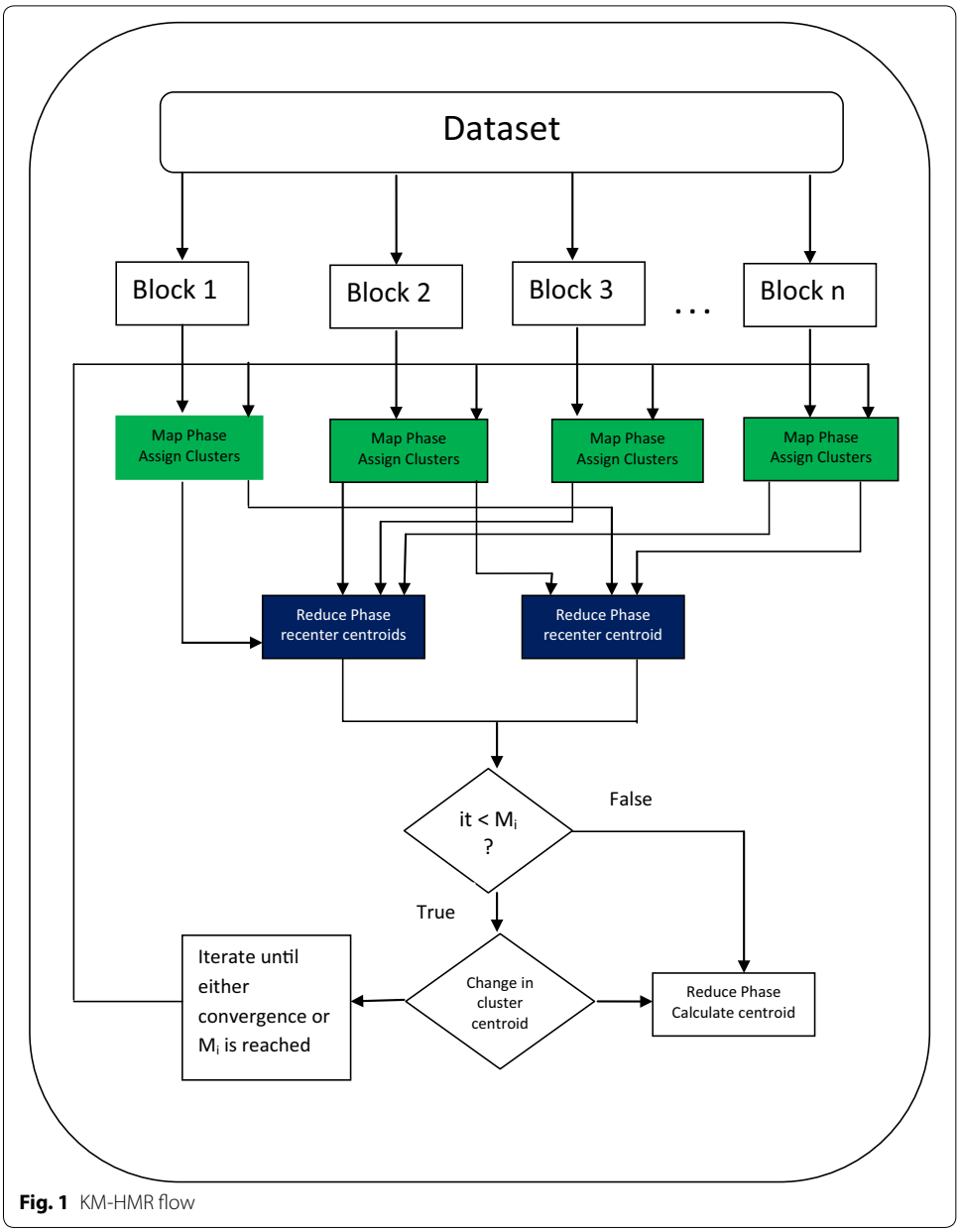

Fig. 1 KM-HMR flow

are updated in a new file and the number of iterations is increased by one. This process is repeated until no more changes in cluster centroid values are found, and this state is referred to as convergence. The final output clusters are stored in a result file. Table 3 describes notation used in Algorithm 1.

Table 3 Notations used in Algorithm 1

Notation	Description
It	Number of iterations
ic	Initial centroid
D	Dataset
k	Number of clusters
oc	Previous centroid values
nc	New cluster centroid values
Result	Final result
select()	Function for selecting data based on the k value
input()	Function for data file uploading
job.mapper()	Map function
job.reducer()	Reduce function
write()	Function for writing centroid values to a file
read()	Function for reading centroid values to a file
update()	Function for testing for updated centroid values

Algorithm 1. KM-HMR

```
Input:
      O : { o1,o2,o3,…..on}; set of objects/entities to be clustered
      K : K clusters (number of clusters)
      Mi: Maximum number of iterations
Output:
        Final output clusters
KM-HMR(data)
it ←0
for each datapointd∈ D do
   ic ← select(k, d)
   input(d)
   write(ic)
oc←ic
  while (true)
     call to job.mapper( )
    call to job.reducer( )
nc = read()
// repeat until convergence
if update((nc, oc) > 0 )
  oc = nc
Else
   update nc to result
it++
result = read( )
```

To improve the efficiency of the clustering stage, we used the KM-HMR algorithm and the large datasets described in "Datasets" section. Large datasets provided as input were divided into chunks and stored in the HDFS. We selected K data points and objects passed as initial cluster centres and updated the centroid of every cluster after a cluster completed several iterations. The map phase involved receiving a sequence file containing initial cluster centres and forms as keyvalue (k,v) pairs. In this phase, the distances between data objects and each cluster were computed using the Euclidean distance. The standard K-means clustering algorithm is suited for application to small datasets and structured data. When datasets are large in size in terms of volume and contain unstructured data, data processing and result generate take a considerable amount of time and

KM-HMR makes an attempt to process faster within a reasonable execution time. The proposed KM-HMR achieves the above goal of processing large volumes of data in parallel with the MapReduce programming model.

K-means modified inter and intra clustering (KM-I2C)

All techniques used to cluster datasets using the K-means algorithm for estimating the number of clusters suffer from deficiencies of cluster similarity measures in forming distinct clusters. A good clustering algorithm must produce high-quality clusters with high levels of intra-cluster similarity and with low levels of inter-cluster similarity. The KM-I2C algorithm divides distinct clusters in datasets into sub clusters based on inter-cluster and intra-cluster similarities. Clusters formed maximize inter-cluster distances and minimize intra-cluster distances. The quality of clustering results is dependent on similarity and implementation measures. The KM-I2C algorithm uses a set of 'O' data points in a dataset 'D' and an initial number of clusters 'k'.

Algorithms 2 and 3 describing algorithmic steps of the map and reduce phases of KM-I2C.

Algorithm 2: Map phase

```
Input:
      N dimensional data objects (n1,n2, n3,....nm) in each mapper
      K: K clusters (number of clusters)
      Initial cluster centroids c1,c2,c3,c4,....ck
Output:
      output list <k, v>
list_new: New centroid list
v = 0
list_new = 0
for all d ∈ D
  for all ci∈ M do
bi ← Ǿ          where bi represents the centroid closest to the data object
Ie← ∞
Ia ← ∞
for  all oi∈ O do
   l(oi) ← ECD(oi,oj) j ∈ {1,2,3,...k}
  it ← 0
  b ← 0
repeat
for each ei∈ E do
    minDist ← ECD (oi, cj)    j ∈ {1,2,3,...k}
if (curr_centr = 0 or l (oi) < minDist) then
     update Ie using equation (5)
  else
update Ia using equation (6)
bi ← bi+1
it ← it+1
create an output list <k, v> with each object and the centroid cluster that it belongs to
repeat until convergence
```

The KM-I2C algorithm follows similar KM-HMR steps as those of Hadoop MapReduce. In KM-I2C, distance metrics of the Ie (inter-cluster distance) and the Ia (intra-cluster distance) are used as similarity and dissimilarity measures. The new distance measures Ie and Ia are calculated as follows:

The new distance measure Inter cluster distance is calculated as follows:

$$Ie = \frac{1}{2}\left(\frac{\sum_{i=1}^{O1}\sum_{j=1}^{O2}\left(Ai - Bj\right)^2}{O1 * O2}\right) \tag{5}$$

where Ie is the Inter cluster distance, O1 and O2 are data points in Cluster 1 and Cluster 2, respectively, and Ai represents the ith data point in Cluster 1 and the jth data point in clusters A and B, respectively.

$$Ia = \frac{1}{2}\left(\frac{\sum_{i,j=1}^{O1+O2}\left(Ai - Bj\right)^2}{(O1 + O2) * (O1 + O2 - 1)}\right) \tag{6}$$

Algorithm 3: Reduce phase

```
Input:
      (k,v): key, value pair where key=l(o_i),
      value = objects assigned to centroids by mappers
      O_l: output list from mappers
Output:
list_new: New centroid list(Nc)
     list_new = 0

Nc ← Ǿ
for all x ∈ O_l
   centroid ← x.key
   data object ← x.value
Nc ←dataobject
for all c_i∈ M do
   Nc ← Ǿ
   Sum_objects ← Ǿ
   Num_objects ← Ǿ
for  all o_i∈ O do
   Sum_Objects + = Object
Num_object++
Nc ← (Sum_objects/Num_objects)
Outputlist ← Nc list U Nc
Return Nc
```

Algorithm 2 describes the map phase of the proposed KM-I2C algorithm and Algorithm 3 describes reduce phase of the proposed KM-I2C algorithm. The input dataset is stored locally in the HDFS as a ⟨k, v⟩ pair where k is the key and v is the value for a given record in dataset D. The input dataset is distributed across several mappers. In the map phase, each map task is read in a shared file containing all cluster centroids. A cluster centroid file can typically be stored in a Hadoop distributed cache. Map tasks are read from a small number of input data points. It assigned each data point to the cluster whose centroid is closest to the data. The map phase then produces intermediate ⟨key, value⟩ pairs, which are grouped automatically by the Hadoop system in preparation for the reduce phase. The final output is stored in an Nc file.

Experimental evaluation

The Hadoop cluster is a special parallel computational cluster that includes a master node and several slave nodes. For a given dataset, a file is split into numerous components equal to the block size set for the HDFS cluster (which is 64 MB by default). Several experiments were conducted on various datasets to evaluate the quality and

scalability of our proposed algorithms. To evaluate the performance of the KM-I2C algorithm, we compared it to other algorithms (e.g., standard K-means and KM-HMR).

Experiments were conducted on a cluster of one master node acting as a NameNode and on ten slave nodes acting as DataNodes as described in Table 4. VMware virtual nodes are used in the CentOS 6.3 operating system.

A master node acts as a Namenode and JobTracker via an Intel Core i3—5005U CPU 2.66 GHz with a RAM capacity of 8 GB and with four processor cores. Slave or worker nodes are defined as Datanodes and TaskTrackers with an Intel Core i3—5005U CPU 2.66 GHz configuration and with a specified RAM capacity and two processor cores. The VMWare Workstation is used to create slave nodes from virtual machines. JDK 1.8.0 with CentOS operating system is used in our experiments.

Key configurations of the Hadoop environment modify conf/master files according to the specified number of slave nodes involved. Namenode and JobTracker are configured under a conf/hadoop-site.xml file and necessary changes are made to other Hadoop configuration files such as hadoop-default.xml and hadoop-core.xml. Experiments were conducted on a varied number of Datanodes and with various block sizes to evaluate the performance of the proposed algorithms.

Datasets

Table 5 describes the datasets used in our experiment. Project Gutenberg (PG) consists of approximately 50,000 free ebooks downloaded from [38]. PG contains approximately 3000 English documents written by 142 authors and makes an effort to create and distribute ebooks of mostly public domain documents. Unstructured data refers to information that does not have a predefined data model and is not organized in a pre-defined manner. We selected this dataset due to the reason that there is no clearly defined observation and variables (rows and columns). Analysing the document manually is an impossible task. Clustering makes it easier for grouping of documents according to their similarities. We applied the implementation method to the subset of PG documents. All of the proposed algorithms were run with document sets of different sizes taken from

Table 4 Experimental setup

Node	CPU	No. of cores	RAM (GB)
MN: master node	Intel i3 5005U	4	8
SN_A: slave node 1–3	Intel i3 5005U	2	2
SN_B: slave node 4–6	Intel i3 5005U	2	2
SN_C: slave node 7–10	Intel i3 5005U	2	1

Table 5 Datasets description

Dataset	Size (GB)	Description
DS_A: million song dataset	300	Collection of 53 audio features and metadata
DS_B: US climate reference network (USCRN)	200	Collected from 143 stations to maintain high quality climate observations
DS_C: Project Gutenberg	110	Includes over 50,000 free ebooks

the above dataset to generate effective clustering results. Categories of focus include the following: audio, music, entertainment, children, education, comics, crafts, finance, health and markets. This dataset was used to investigate and explore documents to conduct a cluster analysis of unstructured data with better execution times.

The US climate reference network (USCRN) [39] is a systematic and sustained network of climate monitoring stations with sites across the United States. Approximately 114 stations are equipped with high-quality devices that measure temperatures, precipitation levels, soil conditions and wind speeds. The purpose of this dataset is to obtain high-quality climatic observations for monitoring climatic patterns. The million song dataset (MSD) [40] is a collection of audio features and metadata available through the Amazon Public Dataset that can be attached to an Amazon EC2 virtual machine to run experiments. The MSD is an unstructured dataset that contains information on approximately 1 million songs with 53 features and is approximately 300 GB in size. We have considered the subset of the entire datasets in our experiment. The split size of the complete dataset is based on the total input file size divided by the number of map tasks launched. The number of mappers is based on the number of input splits made. In the MapReduce framework, the split size of input files and of InputFormat describes input specifications for a MapReduce job. InputFormat used in MapReduce is directly proportional to the number of files and to the total number of blocks of input files. The default split size is the size of the HDFS block size. We can control the split size by applying the mapred.min.split.size parameter available through hadoop-site.xml. In this work, we used the HDFS block size of 128 MB as the split size.

Figure 2 compares the execution times of the proposed algorithms for document sets of the PG dataset. As the number of documents to be clustered increases, the time dedicated to standard K-means exceeds that used for the proposed solution. Parallel processing is essential in processing large volumes of data. In this experiment, we only considered some documents of a subset of the PG dataset. Table 6 shows the clustering results for Project Gutenberg. The proposed KM-HMR takes less time to employ than the standard K-means algorithm due to its processing of large datasets through several different machines. The proposed KM-I2C takes less time to employ than the standard K-means and KM-HMR. KM-HMR uses the Euclidean distance as a similarity measure, and in our proposed KM-I2C we use inter and intra-clustering measures to obtain

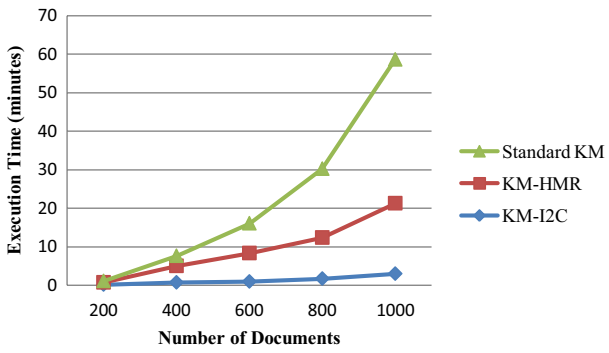

Fig. 2 Comparison of execution times with number of documents

Table 6 Clustering results

Broad categories	Subcategories formed/total		
	KM-I2C	KM-HMR	Standard K-means
Digital audio	21/22	19/22	15/22
Space	26/28	17/28	14/28
Education	61/64	51/64	42/64
Entertainment	50/52	40/52	30/52

efficient and high-quality clusters with high levels of intra-cluster similarity and low levels of inter-cluster similarity to obtain valuable insights from large datasets.

One of the most complex problems associated with clustering concerns is identifying the optimal number of clusters. Unfortunately, there is no general theoretical solution used to find the optimal number of clusters for a dataset. The silhouette coefficient (SC) [41, 42] is used to validate the optimal number of clusters and the degree of inter- and intra-clustering cohesiveness. The SC is a dimensionless quantity valued at −1 to 1. Negative values are undesirable and values closer to 1 denote an overall measure of goodness for a cluster. The SC value can be obtained by the following steps:

1. For object *d*, calculate a_d, the average distance to all other objects in the cluster, using inter- and intra-clustering equations used in the proposed work.
2. For object *d* and any cluster that does not contain the object, the object's average distance (b_d) is calculated for all objects in a cluster.
3. For the dth object, the SC is given by:

$$S_c(d) = \frac{(b_d - a_d)}{\max(a_d, b_d)} \tag{7}$$

Table 7 compares the performance of the proposed model to that of the KM-HMR and KM-I2C algorithms when applied to the USCRN dataset. Parallel K-means worked well with the datasets studied over a reasonable amount of time due to inherent levels of data parallelism. In TPK, an overhead resulted after splitting the algorithm into two phases. KM-I2C performed better in terms of execution time, as clusters formed in less time than when using the other existing clustering techniques. Average lengths of time used for each cluster indicate that with parallelization, the proposed clustering algorithm was more efficient than the single node standard K-means clustering method. Through the experiment, we found that block sizes should depend upon the input datasets employed.

Table 7 Comparisons of total execution periods (s)

	Parallel K-means	Standard K-means	KM-HMR	KM-I^2C
	4320.14	4502.18	4214.16	3876.23
	3880.23	4025.06	3705.29	3518.34
	3691.18	3907.12	3519.12	3412.49
Average	3963.85	4144.78	3812.85	3602.35

When the block size is too small, the number of collaborations involved can increase, affecting performance results and disabling opportunities for parallelization.

Figures 3 and 4 show the execution times of five clusters for the datasets (DS_A, DS_B, and DS_C) considered in this work for KM-HMR and KM-I2C respectively. As the size of datasets increases, time used by KM-I2C improved relative to the used via the standard K-means and KM-HMR methods. The x-axis shows the number of virtual machines used and the y-axis shows the amount of time taken to cluster the datasets in seconds. The results illustrate the superiority of the proposed algorithm when applied to different parameters relative to the traditional algorithm. The execution time is another measure used to compare the two algorithms. We have improved the clustering response time with respect to Hadoop's characteristics. Effects of the number of nodes used on the clustering algorithms were also studied for the same set of data objects. Scalability levels were tested by increasing the number of nodes used for execution while maintaining the same datasets and the same number of nodes but while varying the dataset sizes.

Figures 5 and 6 show execution times dedicated for varied numbers of clusters (ten clusters for KM-HMR and KM-I2C respectively) and using a specified number of virtual machines. The more parallel processing was executed, the less time was taken for algorithm execution. KM-I2C generated better execution times than the other algorithms.

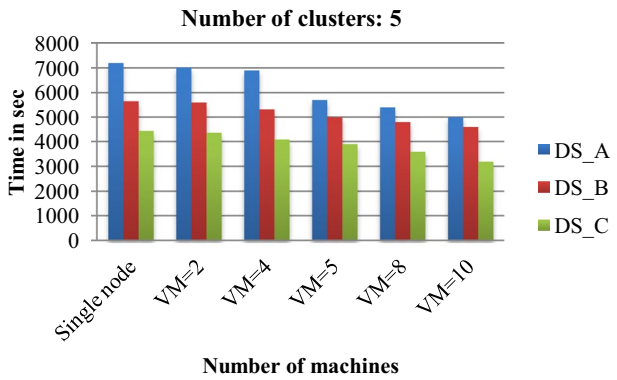

Fig. 3 Execution time comparison with five clusters for KM-HMR

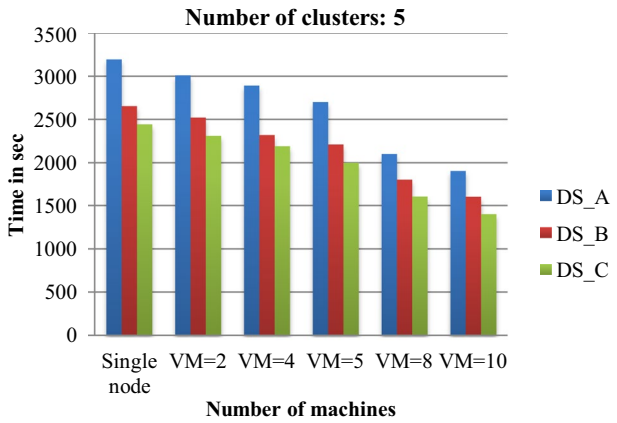

Fig. 4 Execution time comparison with five clusters for KM-I2C

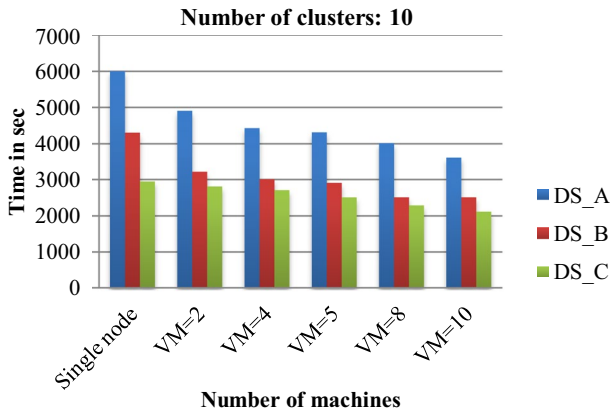

Fig. 5 Execution time comparison with ten clusters for KM-HMR

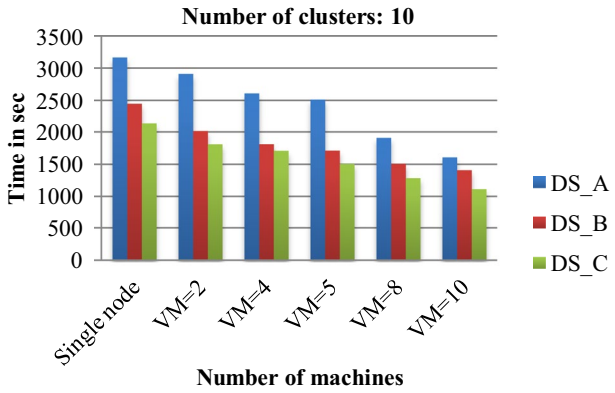

Fig. 6 Execution time comparison with ten clusters for KM-I2C

KM-HMR and KM-I2C are comparable in terms of execution time with KM-HMR performing better than the standard K-means algorithm and with KM-I2C generating better results than the KM-HMR algorithm. This was expected, as KM-I2C uses inter and intra-cluster distances rather than Euclidean distances to perform the benchmark. We also measured execution times of the proposed clustering algorithms against those of the standard K-means clustering algorithms.

Figures 7 and 8 show execution times dedicated for varied numbers of clusters (15 clusters for KM-HMR and KM-I2C respectively). The execution time is a key facet of a successful clustering technique. The proposed KM-HMR and KM-I2C algorithms are better suited for clustering large datasets via MapReduce than the standard K-means clustering algorithm. Effects of the dimensionality of data on time taken were computed. To obtain more conclusive results from our comparisons, the number of processes used for sum computations was varied for each series of runs. To generate efficient results, the K-means dependency of the number of clusters was determined for different variations of cluster numbers.

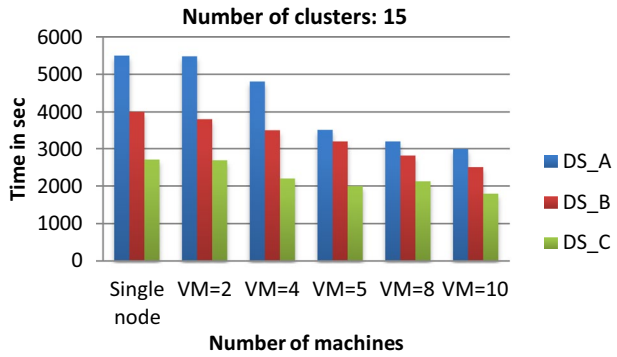

Fig. 7 Execution time comparison with 15 clusters for KM-HMR

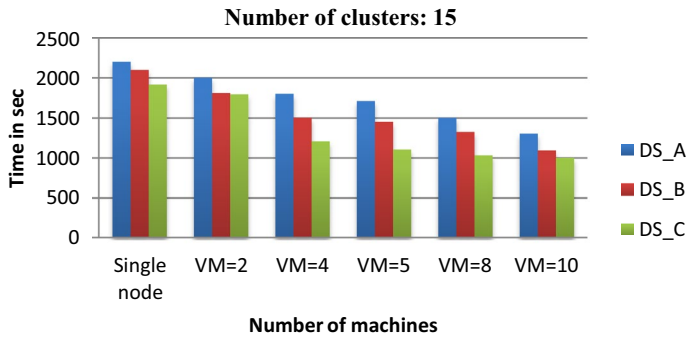

Fig. 8 Execution time comparison with 15 clusters for KM-I2C

Conclusions

Clustering is a challenging issue that is heavily shaped by data used and problems considered. The proposed algorithms show improvements in terms of execution time. Hadoop can compute map and reduce jobs in parallel to cluster large datasets effectively and efficiently. The main objective of this work was to accelerate and scaleup large datasets to obtain efficient high-quality clusters. The standard K-means method is the most popular clustering method due to its simplicity and reasonable execution efficiency when applied to small datasets. A clustering approach should however also maintain cluster efficiency levels when large datasets are involved by considering inter and intra-clustering distances between data objects in a dataset. We have introduced a KH-HMR algorithm to make use of parallelization tools through Hadoop and to obtain better execution times than those of the standard K-means approach. We have developed a novel KM-I2C algorithm by making modifications to the clustering distance metric. We, in turn, have developed an efficient and more effective method relative to other clustering techniques. Future work should enhance the performance of map and reduce jobs to suit large datasets. The performance of Hadoop can be enhanced by using multilevel queues for the efficient scheduling of jobs suitable for large datasets.

Authors' contributions

CS: prepared manuscript and contributed two major algorithms: KM-HMR and KM-I2C and had been made extensive study on the literature. Introduced new distance metric which is able to produce clusters at a faster rate when compared with the other clustering techniques. NK: conception and design of the study. PCR: analysis and interpretation of data. All authors read and approved the final manuscript.

Authors' information
Sreedhar C, is currently working as Associate Professor in Department of Computer Science and Engineering at G Pulla Reddy Engineering College, Kurnool, India. His research includes Big Data, Network Communications. He has 13 years of teaching experience and published 19 papers in various International Journals/Conferences.

Kasiviswanath is currently working as Professor and Head in the Department of Computer Science and Engineering at G Pulla Reddy Engineering College, Kurnool, India. He has over 20 years teaching experience. His research areas include Computer Networks, Security and Big data. To his credit, he published around 25 papers in various International Journals/Conferences.

Chenna Reddy is currently working as Professor in the Department of Computer Science and Engineering at Jawaharlal Nehru Technological University Anantapur, Anantapuram, India. He is high profiled, worked and working as Director and various esteemed capacities at University level.

Author details
[1] Faculty of Computer Science and Engineering, G Pulla Reddy Engineering College, Kurnool, India. [2] Department of Computer Science and Engineering, G Pulla Reddy Engineering College, Kurnool, India. [3] Department of Computer Science and Engineering, Jawaharlal Nehru Technological University Anantapur, Anantapuram, India.

Acknowledgements
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Availability of data and materials
Data will not be shared at this moment, as the datasets are for use in extension for my research work. Complete results and datasets of my research work will be shared in github.

Consent for publication
Not applicable.

Ethics approval and consent to participate
Not applicable.

Funding
Not applicable.

Publisher's Note

Received: 13 June 2017 Accepted: 29 August 2017
Published online: 05 September 2017

References

1. De Camargo RY, Goldchleger A, Kon F. InteGrade: a tool for executing parallel applications on a grid for opportunistic computing. In: Proceedings of 23th Brazilian symposium on computer networks. 2005.
2. Sreedhar C, Kasiviswanath N, Reddy PC. A survey on big data management and job scheduling. Int J Comput Appl. 2015;130(13):41–49.
3. Gonzalez TF. Clustering to minimize the maximum intercluster distance. Theor Comput Sci. 1985;38:293–306.
4. Han J. Data mining: concepts and techniques. San Francisco: Morgan Kaufmann Publishers Inc.; 2005. ISBN 1558609016.
5. Babuska R. Fuzzy clustering. http://homes.di.unimi.it/~valenti/SlideCorsi/Bioinformatica05/Fuzzy-Clustering-lecture-Babuska.pdf. Accessed 4 Jan 2016.
6. McCallum A, Nigam K, Ungar LH. Efficient clustering of high-dimensional datasets with application to reference matching. In: Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining. 2000. p. 169–78.
7. Quinlan JR. Programs for machine learning. San Francisco: Morgan Kaufmann Publishers Inc; 1993. ISBN 1-55860-238-0.
8. Domingos P. Linear-time rule induction. In: Proceedings of knowledge discovery and data mining (KDD-96), Portland, Oregon. 1996.
9. Fisher D. Knowledge acquisition via incremental conceptual clustering. Mach Learn J. 1987;2(2):139–72.
10. Cheeseman P, Stutz J. Bayesian classification (AutoClass): theory and results. In: Advances in knowledge discovery and data mining, 1996. p. 153–80. ISBN: 0-262-56097-6.
11. Fayyad UM, Piatetsky-Shapiro G, Smyth P, Uthurusamy R, editors. Cambridge: AAAI/MIT Press; 1996. p. 153–80.
12. Biswas G, Weinberg J, Li C. ITERATE: a conceptual clustering method for knowledge discovery in databases. In: Braunschweig B, Day R, editors. Innovative applications of artificial intelligence in the oil and gas industry. 1995.
13. Das G, Mannila H, Ronkainen P. Similarity of attributes by external probes. In: Proceedings of the fourth international conference on knowledge discovery and data mining KDD'98. New York: AAAI Press; 1998. p. 23–9.

14. Ortega M, Rui Y, Chakrabarti K, Mehrotra S, Huang T. Supporting ranked boolean similarity queries in mars. IEEE Trans Knowl Data Eng. 1998;10(6):905–25.
15. Dhillon IS, Modha DS. Concept decompositions for large sparse text data using clustering. Mach Learn. 2001;42(1):143–75.
16. Mao J, Jain AK. Self-organizing network for hyperellipsoidal clustering (HEC). Proc IEEE Neural Netw. 1994;5:2967–72.
17. Shanley RJ, Mahtab MA. Delineation and analysis of clusters in orientation data. J Int Assoc Math Geol. 1974;8(1):9–23.
18. Sreedhar C, Kasiviswanath N, Reddy PC. A novel multilevel queue based performance analysis of Hadoop job schedulers. Indian J Sci Technol. 2016;9(44). doi:10.17485/ijst/2016/v9i44/96414
19. haimov N, Malony A, Canon S, Iancu C, Ibrahim KZ, Srinivasan J. Scaling spark on HPC systems. In: Proceedings of 25th ACM international symposium on high-performance parallel and distributed computing. 2016. p. 97–110.
20. Olston C, Reed B, Srivastava U, Kumar R, Tomkins A. Pig latin: a not-so-foreign language for data processing. In: Proceedings of 2008 ACM SIGMOD international conference on management of data. 2008. p. 1099–110.
21. Hunt P, Konar M, Junqueira FP, Reed B. ZooKeeper: wait free coordination for internet-scale systems. In: Proceedings of USENIX conference. 2010.
22. Wadkar S, Siddalingaiah M. HCatalog and Hadoop in the enterprise. In: Proceedings of Apache Hadoop 2014; Apress, Berkeley, CA.
23. Thusoo A, Sarma JS, Jain N, Shao Z, Chakka P, Anthony S, Liu H, Wyckoff P, Murthy R. Hive: a warehousing solution over a map-reduce framework. J VLDB Endow. 2009;2(2):1626–9.
24. Vohra D. Apache Avro. Book of practical Hadoop ecosystem. 2016. p. 303–23.
25. Shastry K, Madhyastha S, Kumar S, Bresniker KM, Battas G. Transaction support for HBase. In: Proceedings of international conference on management of data. 2014. p. 117–20.
26. Wadkar S, Siddalingaiah M. Apache Ambari. Book of Pro Apache Hadoop. 2014. p. 399–401.
27. Ferreira Cordeiro RL, Traina Junior C, Machado Traina AJ, López J, Kang U, Faloutsos C. Clustering very large multi-dimensional datasets with MapReduce. In: Proceedings of KDD'11, ACM, California, August 21–24. 2011.
28. Zhao W, Ma H, He Q. Parallel K-means clustering based on MapReduce. In: CloudCom 2009, LNCS 5931. Berlin: Springer; 2009. pp. 674–9.
29. Chen M. Soft clustering for very large data sets. Comput Sci Netw Secur J. 2017;17(11):102–8.
30. Pham DT, Dimov SS, Nguyen CD. A two-phase K-means algorithm for large datasets. Mech Eng Sci J. 2004;218(10):1269–73.
31. Chu C-T, Kim SK, Lin Y-A. Map-reduce for machine learning on multicore. In: Proceedings of the 20th annual conference on neural information processing systems (NIPS'06), Vancouver, Canada. 2006. p. 281–8.
32. Xia D, Wang B, Li H, Li Y, Zhang Z. A distributed spatial–temporal weighted model on MapReduce for short-term traffic flow forecasting. J Neurocomput. 2016;179(C):246–63.
33. Chao L, Yan Y, Tonny R. A parallel Cop-K means clustering algorithm based on MapReduce framework. Adv Intell Soft Comput J. 2011;123:93–102.
34. Wang C, Guo M, Liu Y. EST clustering in large dataset with MapReduce. In: Proceedings of pervasive computing, signal processing and applications, Sept 2010. 2010.
35. Zhang LD, Yuan DJ, Zhang JW, Wang SP and Zhang QF. A new method for EST clustering. J Yi chuan xue bao = Acta genetica Sinica. 2003.
36. Fang W, Sheng VS, Wen X, Pan W. Meteorological data analysis using MapReduce. Sci World J. 2014;2014. doi:10.1155/2014/646497
37. Tsai CW, Hsieh CH, Chiang MC. Parallel black hole clustering based on MapReduce. In: Proceedings of IEEE international conference on systems, man and cybernetics. 2015.
38. https://www.gutenberg.org/. Accessed 10 Aug 2016.
39. Palecki MA, Lawrimore JH, Leeper RD, Bell JE, Embler S, Casey N. US climate reference network processed data from USCRN database version 2. 2015.
40. Bertin-Mahieux T, Ellis DP, Whitman B, Lamere P. The million song dataset. In: Proceedings of the 12th International conference on music information. Retrieval (ISMIR 2011). 2011.
41. Rousseeuw PJ. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. J Comput Appl Math. 1987;20:53–65.
42. Kaufman L, Rousseeuw PJ. Finding groups in data: an introduction to cluster analysis. New York: Wiley; 1990.