Journal of Big Data

RESEARCH Open Access

Decreasing the execution time of reducers by revising clustering based on the futuristic greedy approach

Ali Bakhthemmat[1*] and Mohammad Izadi[2]

*Correspondence:
bakhthemmat@ce.sharif.edu
[1] Kish International Campus, Sharif University of Technology, Tehran, Iran
Full list of author information is available at the end of the article

Abstract

MapReduce is used within the Hadoop framework, which handles two important tasks: mapping and reducing. Data clustering in mappers and reducers can decrease the execution time, as similar data can be assigned to the same reducer with one key. Our proposed method decreases the overall execution time by clustering and lowering the number of reducers. Our proposed algorithm is composed of five phases. In the first phase, data are stored in the Hadoop structure. In the second phase, we cluster data using the MR-DBSCAN-KD method in order to determine all of the outliers and clusters. Then, the outliers are assigned to the existing clusters using the futuristic greedy method. At the end of the second phase, similar clusters are merged together. In the third phase, clusters are assigned to the reducers. Note that fewer reducers are required for this task by applying approximated load balancing between the reducers. In the fourth phase, the reducers execute their jobs in each cluster. Eventually, in the final phase, reducers return the output. Decreasing the number of reducers and revising the clustering helped reducers to perform their jobs almost simultaneously. Our research results indicate that the proposed algorithm improves the execution time by about 3.9% less than the fastest algorithm in our experiments.

Keywords: Futuristic greedy, MapReduce, Load balancing, Decreasing the number of reducers

Introduction

The amount of data generated on the internet grows every day at a high rate. This rate of data generation requires rapid processing. The MapReduce technique is applied for distributed computing of huge data, whose main idea is job parallelization. The MapReduce algorithm deals with two important tasks, namely Map and Reduce. Initially, the Map includes a set of data, which is broken down into tuples (key/value pairs). Secondly, reduce task takes the map output as an input whereby Reducers run the tasks. Job clustering can determine an allocation of jobs to the reducers and mappers. In recent years, this method has been used frequently for job allocation in MapReduce for shortening the execution time of big data processing [1].

Previous research has shown that clustering methods can be useful for big data analysis. K-means as one of the clustering methods (partitioning based) is simple and fast, which outperforms many other methods. Another clustering method is known as hierarchical clustering method which is performed by splitting or merging data. However, the time complexity of the hierarchical clustering method is not suitable in practice. Also, in this method, the number of clusters is not constant. Grid-based method is a type of clustering method which uses on spatial data, with the EM algorithm functioning effectively in big data clustering. Finally, the density-based clustering method offers adequate precision and proper execution time [2].

Decreasing the execution time of jobs is the main motivation of clustering methods. Therefore, the purpose of this paper is to present a new method based on clustering for big data processing in Hadoop framework using the MapReduce programming model. We use the MR-DBSCAN-KD method as it is one of the fastest density-based clustering methods. However, MR-DBSCAN-KD has two main drawbacks: First, the outliers' allocation to reducers is not determined in this method. Thus, we propose an FGC algorithm in order to solve this challenge. Secondly, MR-DBSCAN-KD creates various clusters with significantly different densities. Use of this method of clustering does not lead to load balancing in the clusters [3]. Accordingly, we propose an algorithm in order to solve this problem. Our proposed method is based on MR-DBSCAN clustering, the futuristic greedy approach, and approximated load balancing.

Related work

Clustering operations of big data involve expensive computation. Hence, the execution time of sequential algorithms is very long. Parallelization of clustering algorithms is recommended for processing big data which can be fulfilled by MapReduce programming. The MapReduce programming can decrease the execution time provided that it uses proper density-based clustering techniques. In this section, we focus on new approaches to big data processing. We also try to categorize these approaches into the idea structure of prior research and discuss their strengths plus weaknesses. We have classified the new approaches into five categories: clustering based on pure parallelizing, clustering based on load balancing, clustering based on traffic-aware, clustering based on innovative methods, and clustering based on cluster optimization.

The first category of this categorization is called density-based clustering based on pure parallelizing. Zhao et al. [3] proposed a clustering algorithm based on pure parallelizing. They applied parallel k-means clustering in MapReduce for big data processing. The results of their work showed that the proposed algorithm functions in a reasonable time as the data grow. Srivastava et al. [4] proposed a parallel K-medoid clustering algorithm in Hadoop to be accurate in clustering. When the number of reducers increases in this method, the make span time diminishes as it correlates with the data growth. Dai et al. [5] stated that the parallel DBSCAN algorithms are not efficient for big data processing in MapReduce when the number of the reducers with small data increases. They illustrated the MR-DBSCAN-KD algorithm for bulky data. In this method, the execution time of small data in reducers was negligible. Most methods based on pure parallelizing in density-based clustering create heterogeneous clusters. The jobs in heterogeneous clusters are never executed simultaneously, which prolong the run of jobs.

We place the load balancing approach in the second category for solving heterogeneous clusters' problem in parallelized clustering. A number of scientists investigated the total execution time by load balancing in parallel clustering. He et al. [6] used this approach to big data processing. They proposed the MR-DBSCAN algorithm based on load balancing for heavily skewed data. Their method was implemented completely in parallel. They achieved load balancing in heavily skewed data and their results verified the efficiency and scalability of MR-DBSCAN. Also, Verma et al. [7] studied job scheduling in MapReduce in order to minimize the span and improve clustering. They presented an innovative heuristic structure for job scheduling. This method generated balanced workload, thereby reducing the completion time of jobs. Ramakrishnan et al. [8] and Fan et al. [9] load-balancing methods based on MapReduce have also been reviewed. Xia et al. [10] used a new greedy algorithm with load balancing for MapReduce programming. In this method, data were allocated to the reducers based on iterative calculation of sample data. This method used a greedy algorithm instead of a hash algorithm since its execution time was shorter than that of hash portioning algorithms. Clustering methods based on load balancing have not focused very much on issues of online job arrival and clustering accuracy. In clustering methods, jobs traffic changes irregularly when a job arrives online. Thus, load balancing in clusters disappears.

In recent years, a third category has been introduced, which is based on traffic awareness for arrival of irregular jobs. Xia et al. [11] proposed an algorithm based on traffic awareness. They applied efficient MapReduce-based parallel clustering algorithm for distributed traffic subarea division. In this research, a metric distance is innovated for the k-means-parallelized algorithm. Evaluation of the experimental results indicates the efficiency in execution time and high accuracy of clustering. Ke et al. [12] and Reddy et al. [13] also proposed a traffic-aware partition and aggregation in big data applications. They classified data based on job traffic. Also, Venkatesh et al. [14] investigated MapReduce based on traffic-aware partition and aggregation for huge data via an innovative approach. This method considerably reduces the response time for big data in the Hadoop framework and consists of three layers: The first layer performs partitioning and mapping on big data. In the second layer, data are shuffled based on traffic aware mapping. In the third layer, data are reduced; this layer reduces the network traffic in the traffic aware clustering algorithm in response to which the execution time diminishes. However, in spite of the available methods, clustering time in some data sets is very high.

Indeed, recent methods could still be implemented within a shorter time by decreasing the clustering computation. Hence, innovative methods of the fourth category have aided to reducing the clustering computations. For example, HajKacem et al. [15] presented a one-pass MapReduce-based clustering method called the AMRKP method, for mixed large-scale data. The AMRKP method reduces the computation required for calculating the distance between clusters. Also, the data are read and written only once. Consequently, the number of I/O operations on the disk would be reduced and operation iterations would improve the execution time. Sudhakar Ilango et al. [16] developed an algorithm with an artificial bee colony based on clustering approach for big data processing. It minimized the execution time but did not always provide good precision for clustering. Fan et al. [17] focused on multimedia big data. They observed that Canopy + K-means algorithm operates faster than k-means as the amount of data

increases. Canopy algorithm is composed of two steps. In the first step, the data are grouped based on new distance function calculation through greater precision in clustering. These group data are introduced as canopy. Thereafter, the groups are assigned to clusters. This structure could improve the total execution time. Jane et al. [18] proposed an algorithm by sorting based on the K-means algorithm and the Median-based algorithm for clustering. This algorithm uses the multi-machine technique for big data processing (SBKMMA). It also reduces the number of iterations in the k-means algorithm. The drawback of this algorithm is the determination of the number of clusters as the primary number of clusters affects the execution time of algorithm. Kaur et al. [19] presented SUBSCALE, a novel clustering algorithm, to find non-trivial subspace clusters for a k-dimensional data set. Their algorithm is applied for high dimensionality of the dataset. Note that parallelism in this method is independent of multiple dimensions, and thus iterations of SUBSCALE algorithm diminish. Kanimozhi et al. [20] proposed an approach for clustering based on bivariate n-gram frequent items. This approach reduced the amount of big data for processing in reducers, leading to an increase in the speed of execution in big data. Nevertheless, in innovative methods, many clusters are not clustered precisely because of the border points (outliers) in them. Accordingly, it is better to optimize the clusters.

Finally, the fifth category of algorithms is designed to optimize clusters for improving the clustering accuracy. Zerabi et al. [21] developed a new clustering method using conditional entropy index. This method involves a process with three tasks with each dealing with MapReduce operations. These tasks operate based on the conditional entropy index, whereby the clusters will be optimized. Hosseini et al. [22] proposed a scalable and robust fuzzy weighted clustering based on MapReduce through micro array gene expressions. This method merges data based on similarity index. Data are processed in a parallelized and distributed platform offering a reasonable execution time in this method. Hemant Kumar Reddy et al. [23] improved the map-reduce performance by novel-entropy-based data placement strategy (EDPS). They extracted data-groups based on dependencies among datasets. Then, data-groups are assigned to the data centre heterogeneity. Finally, data-groups are assigned to clusters based on their relative entropy, whereby clusters are optimized approximately. Beck et al. [24] applied mean shift clustering for grouping big data. They applied NNGA+algorithm for dataset pre-processing. They could improve the quality of clustering and execution time via the mean shift model for big data clustering. Gates et al. [25] showed that random models can have an impact on similar clustering pairs. These models can be applied for evaluating several methods in Map-Reduce. Heidari et al. [26] discussed clustering with variable density based on huge data. They presented MR-VDBSCAN in this method. Their idea search local density of points for avoiding of connecting clusters with various densities. In this way, clustering optimization is performed.

Researchers have tried to improve execution time by approaches such as parallelism, load balancing, jobs categorization based on traffic-aware, reducing clustering computation and cluster optimization. Parallelism creates heterogeneous clusters, which significantly affect the runtime in the reducers. In this way, the total execution time of jobs in clusters increases. Load balancing in clustering could create approximately homogenous clusters. Nevertheless, the jobs arriving online disrupt the load balance while also generating heavy computations in the clustering based on load balancing. For this reason,

clustering was performed based on job traffic. This approach did not solve the problem of high computation in clustering either. We consider parallelization and reduction of calculations as well as optimization of clusters and load balances in the proposed method, respectively. Innovative methods have reduced runtime by reducing computation and using local options. Nonetheless again because of the boundary points, the clusters are not carefully clustered. Cluster optimization can be done with the minimum number of clusters suitable for reducers. Lowering the number of reducers with proper clustering and load balancing can diminish the total runtime as reducers can function almost simultaneously. Since with fewer reducers the execution time decreases, we can consider maximizing the usage of reducers with load balancing. Hence, we tried to present a new method that would decrease the number of reducers by clustering jobs and load balancing in reducers. The main challenge is the bounded points (outliers) created in density-based clustering. We try to cluster data based on density and subsequently, apply approximated load balancing to the clusters. The proposed idea presents a distance function called Futuristic Greedy Index for appending outliers to clusters. Also, it can shorten the execution time by correcting the clusters. Cluster correction is done by discovering similar data and assigning them to clusters, provided that interdependence between clusters is minimized.

Methods

We consider two main goals for designing the proposed algorithm (diminishing and load balancing of reducers). We design the proposed algorithm by mapping, where reducing operations are performed in the Hadoop structure. In the proposed method, the jobs are stored in HDFS structure in order and without heavy computation. The jobs are stored in file systems equally. Thereafter, the file systems are assigned to mappers sequentially. Each mapper is clustered by MR-DBSCAN algorithm. Accordingly, clusters and outliers are generated. Then, the generated outliers merge together or other clusters based on FGI. Next, the generated clusters merge together based on centroids distance. Subsequently, new clusters are created by load balancing, which are assigned to reducers. Finally, the results of reducers are combined together, and the output is returned. The proposed method is composed of five phases.

In the first phase, jobs are stored in the HDFS structure (V_i). They are assigned to the file systems equally. Each file system (fs) can store a limited number of jobs because each file system accommodates limited capacity.

In the second phase, mapping operations are performed. This phase consists of three steps. In the first step, the data are assigned to the mappers, and then data in each mapper are clustered using the MR-DBSCAN method. The output of this operation is an uncertain number of heterogeneous clusters and outliers.

(C_{ij}, O_{ij}). In the second step, FGC algorithm is employed in order to assign outliers to existing clusters or together. In the final step, some of the generated clusters merge together based on centroid distance. The output of the second phase contains new clusters (C'_{ij}). Hence, the number of reducers diminishes.

In the third phase, clusters must be assigned to reducers. Clusters have almost similar jobs, but they are heterogeneous. Therefore, if clusters are assigned to reducers, then reducers will have a variable work load, which can increase the total execution

time. Thus, clusters are grouped based on the average cluster workload (ETA_k). Accordingly, the grouped clusters are assigned to reducers based on the approximated load balancing.

In the fourth phase, jobs are assigned to reducers, and then each reducer executes the related jobs. We expect that the execution time decreases as the clusters are being assigned to fewer reducers with load balancing. It results in diminished communication cost of data transmission.

In the fifth phase, the outputs of the reducers are combined together, and then the final outputs are displayed.

The phases of the proposed method are illustrated in Fig. 1. Table 1 presents the notations utilized in the proposed algorithm.

Phase 1: Storing the data set in HDFS

Data are stored in the Hadoop structure as a set of data nodes where each data node presents a data point. Clustering operations are performed on each data point separately. Data points are presented by V_1, V_2, V_3,...,V_n. Each of the data points is stored in a file in the distributed file system denoted by fs. Algorithm1 presents the storing operations. The time complexity of storing the data set in HDFS is:

$i.$ $O(N)$

```
While (input data exists)   // N number of the input data = loop iteration
   i=i+1
   Vi =Read( Input data)   //  Vi  is  a data point  { V1,V2,V3,...,Vn}
   Store Vi in Hadoop distributed file system (fsi)
end
```

Algorithm 1. Storing operations in Hadoop.

Phase 2: FGC-Mapping

FGC-Mapping is performed in the following steps (clustering mappers, assigning outliers to clusters, merging clusters).

Step 1. Clustering mappers

In the previous phase, big data were split equally among V_1,V_2,V_3,...,V_n data nodes which were assigned to the mappers. In this step, first the MR-DBSCAN-KD algorithm is applied to the mappers, both in parallel and separately. This step is illustrated in Fig. 2 where the data are split among three mappers. The data points assigned to each mapper are clustered, generating mappers that are comprised of clusters and outliers. The clusters are presented by C_{ij} with each of them possessing different densities. For example, mapper 1 includes clusters $\boldsymbol{c^{11}}$, $\boldsymbol{c^{12}}$, $\boldsymbol{c^{13}}$, and four outliers. Outliers are jobs that do not fall in any cluster. Algorithm 2 presents the first step of FGC-Mapping.

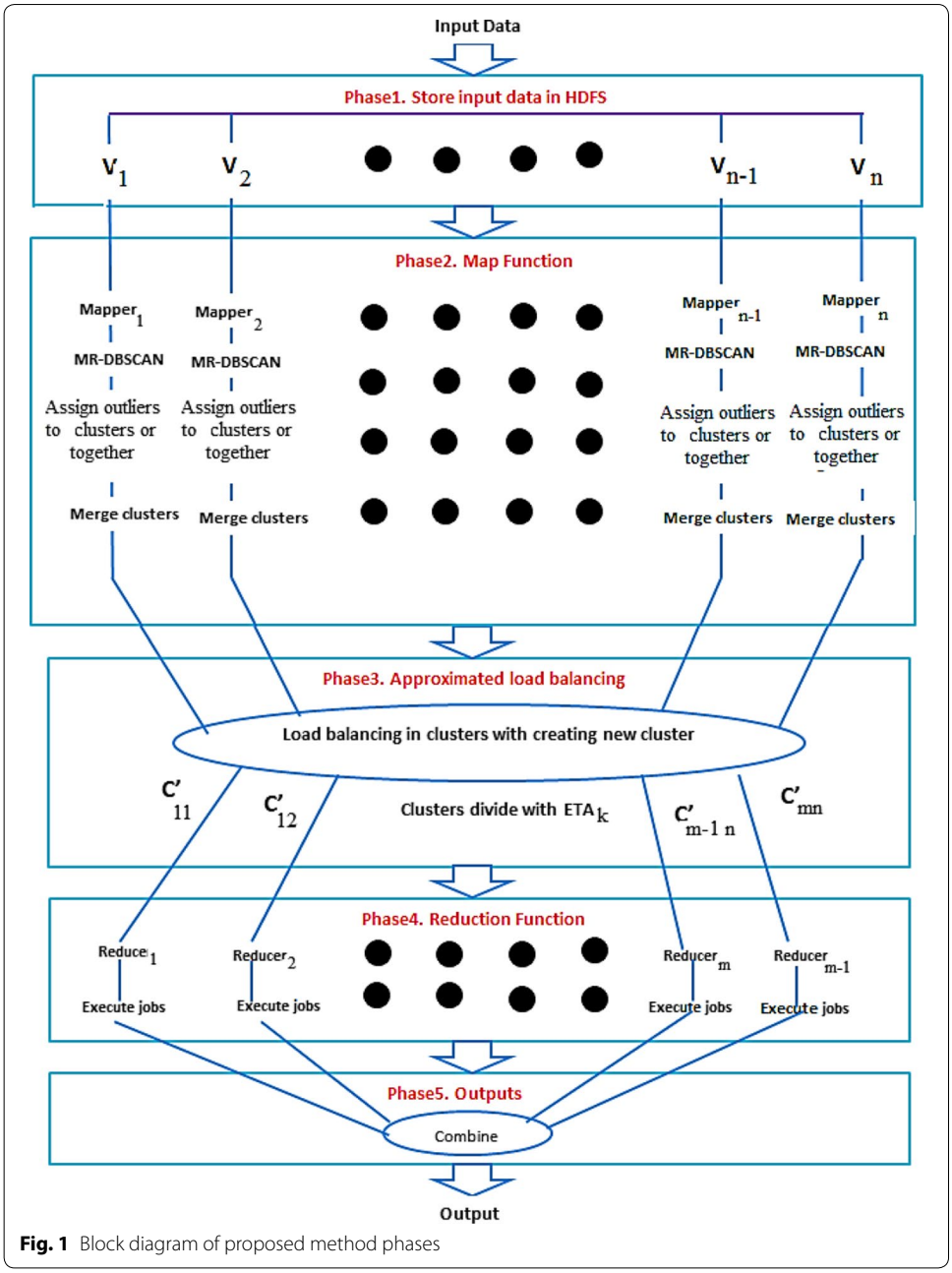

Fig. 1 Block diagram of proposed method phases

The time complexity of the MR-DBSCAN algorithm equals $\left(\frac{N}{n}\right)^2$ such that $\frac{N}{n}$ is the number of jobs in each mapper, n represents the number of mappers, and N denotes the number of all data. In the parallel structure, the time complexity of step 1 is:

Table 1 Notations

Notation	Description
V_i	i-th data point
$\boldsymbol{Mapper_i}$	i-th mapper
$\boldsymbol{C_{ij}}$	j-th cluster of i-th mapper
$\boldsymbol{O_{ij}}$	j-th outlier if i-th mapper
$\boldsymbol{\hat{C}_{ij}}$	cluster centre of C_{ij} or
n	Number of data points
N	Number of input data
$\boldsymbol{reducer_i}$	i-th reducer
j	Number of jobs in each cluster
k	Number of new clusters before *centroids clustering*
p	Number of new clusters after *centroids clustering*
$\boldsymbol{C'_{ij}}$	New cluster after *centroids clustering*
$\boldsymbol{C''_{ij}}$	Produced cluster of centroids clustering
F	number of reducers
$\boldsymbol{ETA_k}$	Density average of jobs in k-th cluster.

Notice: cluster centre for each outlier is same as $\boldsymbol{O_{ij}}$

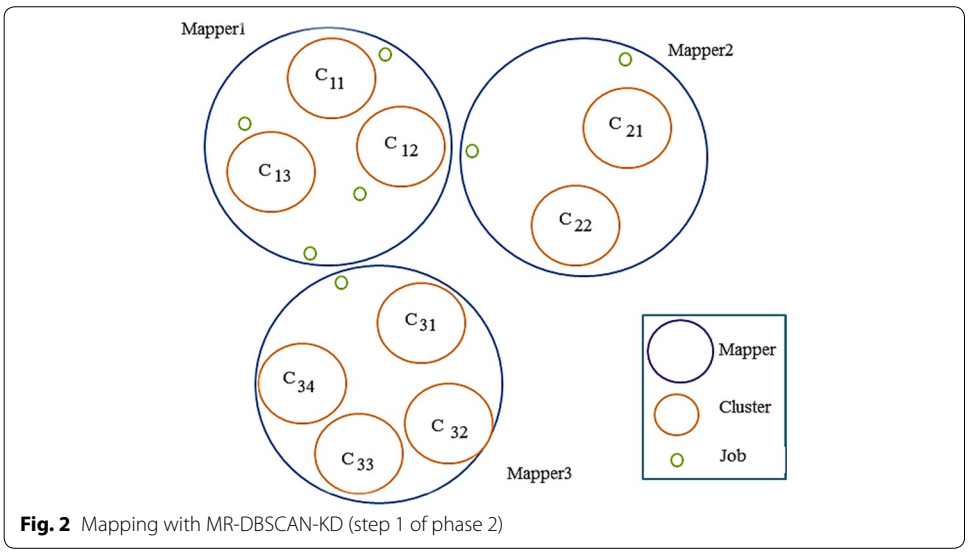

Fig. 2 Mapping with MR-DBSCAN-KD (step 1 of phase 2)

$$ii. \qquad O(n) + O\left(n * \left(\frac{N}{n}\right)^2\right) = O\left(n * \left(\frac{N}{n}\right)^2\right) = O\left(\frac{N^2}{n}\right)$$

For each datapoint(V_1 ,V_2 , ...,V_n) // *n=loop iteration*

$Mapper_i$ = Assign(V_i)

For each mappers in parallel // *n= parallel loop iteration*

MRDBSCAN-KD ($Mapper_1$, $Mapper_2$, $Mapper_{3,....}$) // $O\left(\frac{N}{n}\right)^2$

Algorithm 2. Assigning data points to mappers.

Step 2. Assigning outliers to clusters.

Each of the clusters includes a cluster centre called centroid. Further, each cluster may include outliers as illustrated in Fig. 2. The accuracy of the MR-DBSCAN-KD algorithm is high; however, creating outliers and heterogeneous clusters are some of the drawbacks of the MR-DBSCAN-KD algorithm [3]. The proposed algorithm appends outliers to the existing clusters or other outliers using the Futuristic Greedy Index function (FGI). FGI is a new distance function which calculates the distance between the outliers and the clusters. The outliers are assigned to the closest cluster using this function. Algorithm 3 illustrates the steps of assigning the outliers to the clusters in the second step of the FGC-Mapping.

```
For each of O_ij      // b=number of O_ij
    For each of  Ĉ_ij in Mapper_i   // n=number of Mapper_i
        calculate  FGI(O_ij , Ĉ_ij , Mapper_i) by Futuristic_Index (O_ij , Ĉ_ij)
        //cluster center for each outlier is same with O_ij
        //j =number of FGI calculates = Number of jobs in each cluster
For each of O_ij   // b=number of O_ij
    Find (C_mn or O_mn) by Minimum_distance (FGI(O_ij , Ĉ_ij , Mapper_i))
    //j =number of Find calculates = Number of jobs in each cluster
    Append O_ij to C_mn  or  Append O_mn other outlier with a new cluster
```

Algorithm 3. Assigning outliers by FGA.

The FGI function is calculated using Eqs. 1 and 2. Equation 1 calculates the Euclidean distance, while Eq. 2 computes the futuristic index. FGI function is designed based on of two parts (futuristic and greedy) in Eq. 3.

$$dist(O_{ij}, \hat{C}_{ij}) = \sqrt{\sum_{j=1}^{n} (O_{ij} - \hat{C}_{ij})^2} \quad for\ Mapper_i \tag{1}$$

$$Futuristic\_Index = \frac{1}{\sum_{j=1}^{n} dist\left(O_{ij}, \hat{C}_{ij}\right)} \tag{2}$$

$$FGI\left(O_{ij}, C_{ij}, Mapper_i\right) = \frac{dist\left(O_{ij}, \hat{C}_{ij}\right)}{\sum_{j=1}^{n} dist\left(O_{ij}, \hat{C}_{ij}\right)} \tag{3}$$

FGI assigns each outlier to a cluster. In some cases, the distance between an outlier and different clusters may not be significantly different. The futuristic in FGI function determines that the outlier point is far from the other clusters, while the greedy in FGI function specifies that the outlier point is the closest to one cluster. Closeness in Eq. 1 is determined by the greedy distance index denoted by $dist\left(O_{ij}, \hat{C}_{ij}\right)$. The greedy distance index may create improper clustering since the boundary point may not be assigned to

the proper cluster. Thus, we consider future in Eq. 2. Equation 3 presents the futuristic greedy index. Indeed, Eq. 3 is an outcome of the multiplication of Eq. 1 and Eq. 2.

Finally, FGI is calculated by Eq. 2. We append the outlier points to the clusters provided that clustering does not deteriorate in the next iteration of the greedy algorithm as greedy selections do not guarantee appropriate selections in the subsequent steps. Figure 3 illustrates the output of Algorithm 3. Value 'j' is number of jobs in mappers and value 'c' denotes the number of mappers. The time complexity of step 2 is:

$$iii.\ O\big(b*n*j+b*j\big)=O\big(n*j\big),\ j=\frac{N}{c},\ c<N\Rightarrow O\left(n*\frac{N}{c}\right)=O(N)$$

Step 3. Merging clusters.

In this step, clusters that are located close in the mappers are merged together. Algorithm 4 presents the merging method used in the third step of the FGC-Mapping. Initially, centroids are clustered by MR-DBSCAN-KD. The outputs of this clustering are presented by $(C''_{11}C''_{11}(1), C''_{12}(2), \ldots, C''_{ij}(k))$, where $C''_{ij}(t)$ denotes centroid of the t-th cluster. Clusters linked by centroids merge together according to centroids' clustering. A new set of clusters is updated in the existing mappers. Hence, new clusters will be considered $(C'_{11}(1), C'_{12}(2), \ldots, C'_{ij}(p))$, which have different densities. Figure 4 illustrates algorithm 4. Note that when the clusters are merged, a to new cluster is formed. Hence, the number of merged clusters is fewer than the number of previous clusters. Accordingly, clustering based on centroids can lower the number of clusters. The time complexity of step 3 is:

$$k \ll N\ ,k\ is\ the\ number\ of clusters\ before\ centoids'\ clustring$$

$$\frac{N}{p}\ \ is\ the\ average\ of\ the\ number\ of\ jobs\ in\ each\ cluster$$

$$P \ll N,\ \ P\ is\ the\ number\ of\ clusters\ after\ centroids'\ clustering$$

$$iv. \qquad O\left(k+k^2+\frac{N}{p}+p\right)=O(N)$$

For each of C_{ij} *// k= number of clusters*

Assign ($\hat{C}_{ij}$ *) to* C_{ij}

Centroids=Set of $\hat{C}_{ij}$ *{* $\hat{C}_{11}$, $\hat{C}_{12}$ $\hat{C}_{13}$ ….*}*

($\hat{C}_{ij}$, C''_{ij} *,p)= MR-DBSCAN-KD (centroids)*

// $O(k^2)$ *is complextity of* $MR-DBSCAN-KD$ *algorithm*

// p is number of new clusters , C''_{ij}*'s are clusters after centroids clustering*

$C'_{ij}(p)$ = *transfer& merge (Linked cluster with (* $\hat{C}_{ij}$, C''_{ij} *,p))*

// merge list (cluster) with $O(\frac{N}{p})$

For t=1 to p

// p=number of loop iteration =number of clusters after centroid clustering

Cluster(p)= $C'_{ij}(p)$

Algorithm 4. Merging clusters based on centroids.

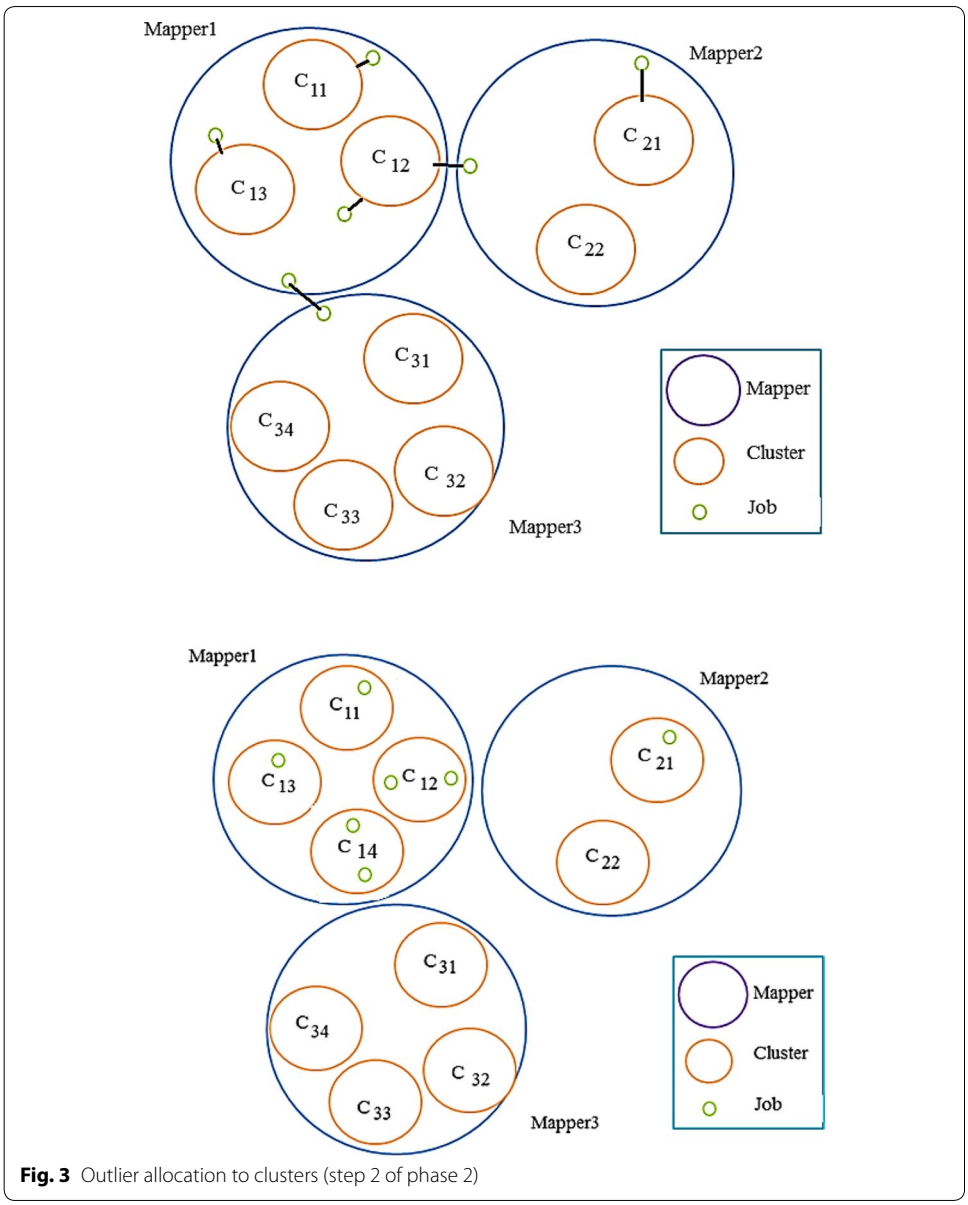

Fig. 3 Outlier allocation to clusters (step 2 of phase 2)

Phase 3: Load balancing in clusters

The result of the MR-DBSCAN-KD algorithm is a set of heterogeneous clusters. In the previous phase, the number of clusters was reduced by merging some clusters. In phase 3, the clusters are modified with load balancing. The number of clusters is represented by P and the proper number of reducers is denoted by F. ETA_k presents the average density of the k-th cluster in MR-DBSCAN. We design Algorithm 5 (the third phase of the proposed method) based on ETA_k. We assign reducers to clusters with an approximately similar density to ETA_k. Hence, load balancing will be accomplished in cluster densities.

In Algorithm 5, the clusters with densities greater than ETA_k are split into equal clusters with ETA_k density. They are then assigned to reducers. The remaining clusters are assigned to other reducers using the Best Fit algorithm [27]. Accordingly, load balancing in reducers leads to distribution of traffic balance.

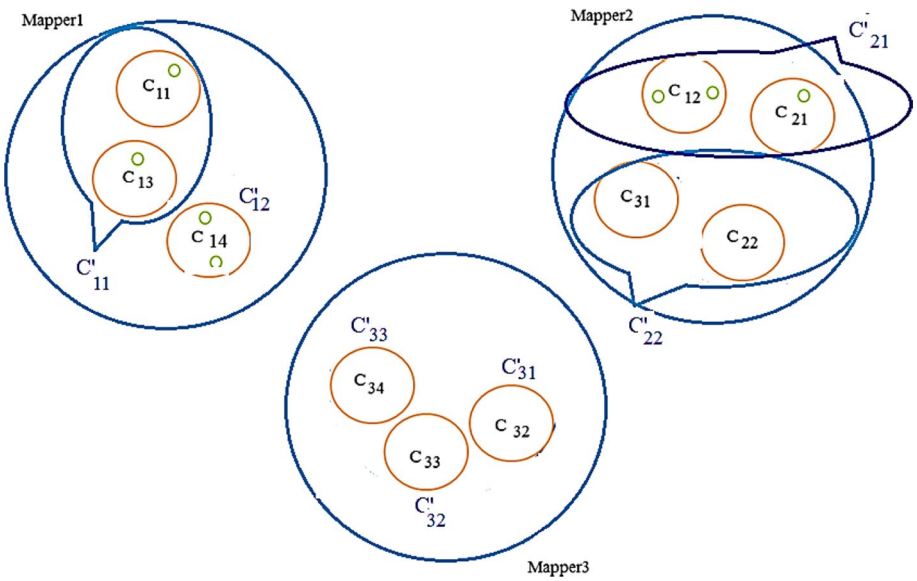

Fig. 4 Merge of clusters to new clusters in mappers (step 3 of phase 2)

The time complexity of phase 3 is:

$v.\quad O(P*logP+P+No*P+m*P*logP)\,, m<p=O(P^2*logP)$

//P is number of cluster of previous step

Sc=sort clusters based on ETA_k *Descending* $\{ETA_1, ETA_2, \ldots, ETA_k, \ldots, ETA_p\}$ *//O(P*logP)*

$Mean1=\frac{\sum_{i=1}^{p} ETA_i}{p}$ *//O(p)*

t=1

f=1

Cluster(t)=select (Sc(t))

While $(ETA_k(t) \geq Mean1)$ *//O(p)*

$No=\left\lfloor \frac{ETA_k(i)}{Mean2} \right\rfloor$

For i=1 to No //O(No)

Assign $cluster^i(t)$ *to reducer (f)*

f=f+1

End

t=t+1

Cluster(t)=select (Sc(t))

end

while (t<=p) //O(p-t) , p-t= m ,m<p

*Assign by bestfit (cluster(t) to reducers) //O(p*logp)*

Cluster(t)=select (Sc(t))

t=t+1

end

Algorithm 5. Cluster revising based on load balancing.

Phase 4: Job execution in reducers

In phase 4, clusters are assigned to reducers, and then reducers execute the jobs in parallel. Load balancing in reducers leads them to execute jobs almost simultaneously. As a result, the execution time of the jobs diminishes. It is because the fewer number of reducers results in less communication cost. Subsequently, the total execution time decreases. Algorithm 6 illustrates phase 4.

```
For each of  cluster(i)       //loop iteration =P is number of cluster of previous step
        Assign cluster(i) to Reducer(i)
For each Reducers in parallel    //loop iteration = number of reducers=p
    execute jobs in each reducer
```

Algorithm 6 Assigning clusters to reducers.

Complexity time of step 4 is:

$$vi.O(p+p) = O(p)$$

Phase 5: Determining the outputs

In the last phase, the outputs of the reducers are combined together where the final output is returned. Algorithm 7 depicts the last phase of the proposed algorithm. Complexity time of step 5 is:

$$vii.O\left(p * \frac{N}{p}\right) = O(N)$$

```
Solution={ }
i=1
For each reducer(i)    //loop iteration = number of reducers=p
   Solution= Combine(solution, solution(reducer(i))
    //combine reducers with number of N/p
    i=i+1
end
```

Algorithm 7 Combining the outputs of reducers.

Results and Discussion

The time complexity of the proposed method is calculated by the complexity from phase 1 to phase 5 in Sect. 3 of this paper, which is shown in Table 2. Thu, the complexity of the proposed algorithm is:

$$O(N) + \left(O\left(\frac{N^2}{n}\right) + O(N) + O(N)\right) + O\left(P^2 * logP\right) + O(P) + O(N) = O\left(\frac{N^2}{n}\right)$$

The time complexity of phase 3 (load balancing in clusters) is $O\left(PlogP\right)$. P represents the number of clusters which is far lower than N. Phase 3 is additional phase that we add to the main steps of MapReduce. Phase 3 creates waiting time with complexity $O\left(P^2 * logP\right)$. Note that it is negligible in contrast to $O\left(\frac{N^2}{n}\right)$ since $P \ll N$. Also, the approximated load balancing in reducers improves the execution. The experimental results confirm this claim.

The experimental platform is implemented using Hadoop and is composed of one master machine plus eight slave machines. All of the machines had the following specifications: Intel Xeon E7-2850 @ 2.00 GHz (Dual Cpu) and 8.00 GB RAM. All of the experiments were performed on Ubuntu 16.04 with Hadoop 2.9.1 and JDK 1.8. The codes were implemented by Java in the Hadoop environment.

Table 3 presents datasets employed in this research. These datasets are big data in this research for two reasons: Firstly, part of the main memory of available computers is occupied by the operating system and other information required. Thus, it is not possible to load all of data in datasets into the existing main memory of the available computer. Datasets in this research are too large and complex to be processed by traditional algorithms and computers. As the second reason, datasets are composed of several types and several attributes.

Four types of datasets have been used in the experiments called NCDC, PPG-DaLiA, HARCAS, and YMVG. The NCDC dataset [29] contains files with every station subhourly (5-min) data in terms of year from U.S. Climate Reference Network (USCRN). Sub-hourly data include air temperature, precipitation, global solar radiation, surface infrared temperature, relative humidity, soil moisture and temperature, wetness, and 1.5-m wind speed. Instances are from 2006 to 2019, where the size of this dataset is about 26G. PPG-DaLiA dataset contains data from 15 subjects wearing physiological and motion sensors, providing a PPG dataset for motion compensation and heart rate estimation in daily life activities. PG-DaLiA dataset is a publicly available dataset for PPG-based heart rate estimation. This multimodal dataset features physiological and motion data, recorded from both a wrist- and a chest-worn device, of 15 subjects while performing a wide range of activities under close to real-life conditions. The included ECG data provides the heart rate ground truth. The included PPG- and 3D-accelerometer data can be used for heart rate estimation, while compensating for motion artifacts. Human activity recognition from Continuous Ambient Sensor dataset (HARCAS) represents the ambient data collected in houses with volunteer residents. Data are collected continuously while residents perform their normal routines. Ambient PIR motion sensors, door/temperature sensors, and light switch sensors are placed throughout the house of the volunteer, which are related to specific activities of daily living we wish to capture.

Table 2 Complexity of the algorithm phases

Phase	Complexity
Complexity of phase 1	$O(N)$
Complexity of phase 2	$O\left(\frac{N^2}{n}\right) + O(N) + O(N)$
Complexity of phase 3	$O\left(P^2 * logP\right)$
Complexity of phase 4	$O(P)$
Complexity of phase 5	$O(N)$

Table 3 Datasets

Data set	Type	Attribute Characteristics	Number of attributes	Number of instances	Size	Description
YMVG [28]	Multivariate	Integer, Real	1,000,000	120,000	9.2 G	Video youtube
NCDC [29]	Multivariate	Categorical	9	14	26 G	Weather data
PPG-DaLiA [28]	Multivariate	Real	11	8,300,000	8.4 G	Physiological and motion data
HARCAS [28]	Multivariate	Integer, Real	37	13,956,534	28.9 G	Ambient data in home

The dataset should be useful particularly for research on multi-view (multimodal) learning, including multi-view clustering and/or supervised learning, co-training, early/late fusion, and ensemble techniques. YouTube Multiview Video Games (YMVG) dataset consists of feature values and class labels for about 120,000 videos (instances). Each instance is described by up to 13 feature types, from three high level feature families: textual, visual, and auditory features. There are 31 class labels, 1 through 31. The first 30 labels correspond to popular video games. Class 31 is not specific, which means none of the 30. Note that neither the identity of the videos nor the class labels (video-game titles) are released. Again, the dataset should be useful particularly for research on multi-view (multimodal) learning, including multi-view clustering and/or supervised learning, co-training, early/late fusion, and ensemble techniques.

The results are compared with K-means-parallel, GRIDDBSCAN, DB-Scan, Mean shift clustering, and EM clustering methods. Table 4 summarizes the execution time for these algorithms. The proposed method executes jobs faster than the other algorithms due to four reasons. Firstly, big data are categorized and assigned to mappers equally without heavy calculations. Also, each mapper consists of small data. Hence, the clustering operation is performed on small data of each mapper within a short execution time. Then, clusters and outliers are created and outliers are assigned to other clusters or together within a short execution time. Secondly, the generated clusters are merged based on their centroids' distance. Therefore, the distance function is not computed for each node of cluster and only is calculated based on centroids. It prevents from high computation. Accordingly, computation of distance in clustering is decreased. Thirdly, the load balancing in clusters divides the workload between the reducers almost equally. Thus, reducers execute the jobs almost simultaneously. Also, it results in diminished number of reducers. The low number of reducers shortens the time of data transmission in the Hadoop framework. Accordingly, the communication cost in

the Hadoop framework drops. Note that the communication cost consists of coordination between reducers, which is performed by a coordinator. Also, load balancing in the traffic of reducers leads to less data transmission between reducers. If load balancing is not established, then it is possible to transfer high loads to one reducer. Hence, the total execution time increases in the parallel structure of MapReduce. Furthermore, each of the clusters is composed of jobs with almost similar computation. These clusters are assigned to the reducers and since the computation of jobs is almost similar, the reducers execute jobs very fast. Accordingly, some similar operations in reducers do not need to be recalculated. Also, several similar operations can be processed fast (for example, a similar key in the key-value structure of MapReduce for counting the number of one word). Consequently, iterations of operations and execution time will be reduced. Figure 5 indicates that the proposed algorithm performs faster than other methods when applied to the four datasets. The speed of algorithms is shown based on the percentage of improvement in the total execution time. Near clusters in the mappers are merged together in order to lower the total number of clusters.

We can compare clustering methods with a similar index. The Rand Index in data clustering is a measure of the similarity between two clusters. It shows view the measure of the percentage of correct decisions made by the algorithm. The Rand Index is calculated using Eq. (4) [30]:

$$Rand\ Index = \frac{TP + FN}{TP + FP + TN + FN} \tag{4}$$

Rand Index is calculated for every two clusters. Subsequently, we consider the average Rand Index, and compare clusters with it. TP is the number of true positives, TN represents the number of true negatives, FP shows the number of false positives, and FN denotes the number of false negatives. Rand Index can calculate clustering accuracy, and it is applied even when class labels are not used [31].

Table 5 shows that the K-means-parallel has the minimum Rand Index while our proposed method offers the largest Rand Index. Figure 6 illustrates the Rand Index of the five algorithms when applied to the four. It demonstrates the percentage of improvement of Rand Index. Table 5 presents the Rand index in various algorithms. This table shows that the proposed method performs more efficiently compared to the other clustering methods in creating similar clusters. This efficiency is the result of usage of FGI for assigning outliers and merging near small clusters. In the second phase of the proposed algorithm, the data points in the mappers are clustered quickly using MR-DBSCAN-KD. The quick clustering is a result of the fact that the data are of normal size and are not considered big data as

Table 4 Comparison of execution time (seconds)

Datasets	K-means parallel	GRIDDBSCAN	DB-scan	Mean shift clustering	EM clustering	Proposed method
YMVG	745.13	763.12	633.74	814.93	876.61	619.78
NCDC	2458.37	2345.67	1880.46	2671.47	2593.21	1790.34
PPG-DaLiA	616.23	697.65	457.21	491.43	623.81	447.16
HARCAS	2766.26	2537.23	1983.71	2719.39	2643.12	1837.37

we apply distributed processing by mappers and reducers. The clusters are merged based on centroids' clustering. This clustering and calculating the distance between the centroids occur quickly as number of the centroids is k ($k \ll N$). Meanwhile, merging the clusters together leads to a reduction in the number of clusters, and the merged clusters in this step are approximately similar. In subsequent phases, the operations are performed on the clusters created in this phase. Thus, less calculation is established in each phase of the proposed algorithm. Load balancing of jobs in the third step leads to a balanced assignment of jobs to the reducers. Hence, reducers finish jobs almost simultaneously. Our proposed method improves the average execution time compared to other methods.

Before executing Algorithm 5, the proposed algorithm allocates each cluster to similar jobs, which may suggest that heterogeneous clusters are created. Some heterogeneous clusters have high density. Hence, these clusters cannot be assigned to reducers since load imbalance in cluster density causes some reducers to execute jobs within a long time. Subsequently, the total execution time increases.

Algorithm 5 creates two states. In the first state, if a cluster has a density higher than the average number of clusters, then the cluster is evenly divided into multiple partitions. Next, each partition is allocated to a reducer. In the second state, if the cluster has density less than the average density, then it is allocated to a reducer. Thus, the entire capacity of reducers is not used. Accordingly, load balancing is not fully achieved in reducers. It is important to note that the majority of the clusters have density higher than mean density of clustering before Algorithm 5 due to merging smaller clusters in the previous steps. An excessive increase in the number of clusters with a higher density than the average density causes a very small amount of imbalance to be increased.

Reducers must process similar jobs in one cluster with approximal similar density. Empirical tests show that Algorithm 5 performs properly when the number of clusters with high density is very large. Also, experimental results suggest that the number of high-density

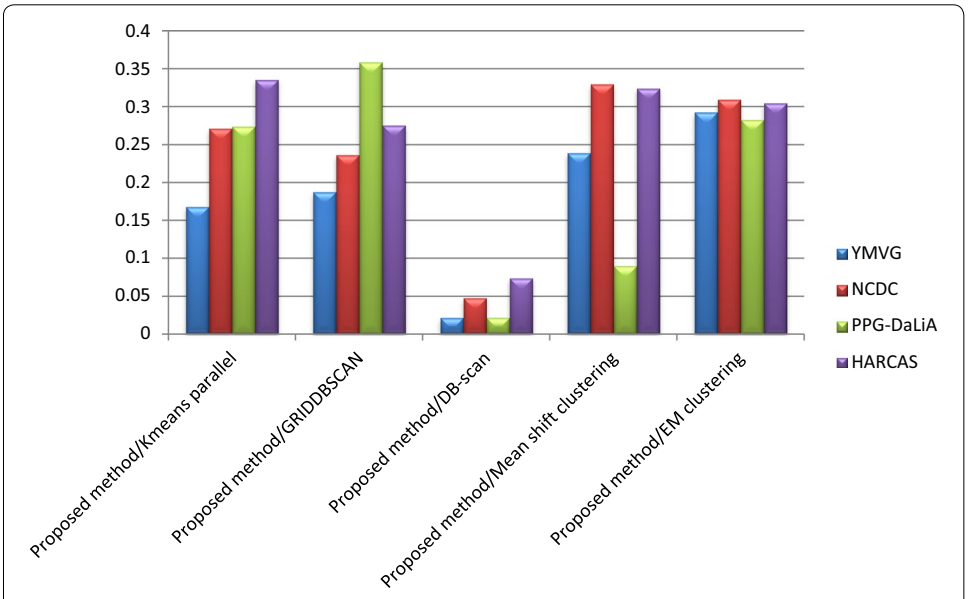

Fig. 5 Percentage of improvement for execution time

Table 5 Comparison of Rand Index

Datasets	K-means parallel	GRIDDBSCAN	DB-scan	Mean shift clustering	EM clustering	Proposed method
YMVG	70.79	84.31	81.64	90.78	89.83	92.41
NCDC	69.46	83.64	79.54	90.21	87.93	91.83
PPG-DaLiA	72.39	85.76	82.17	91.52	90.26	93.47
HARCAS	67.37	81.93	76.87	89.86	86.87	90.57

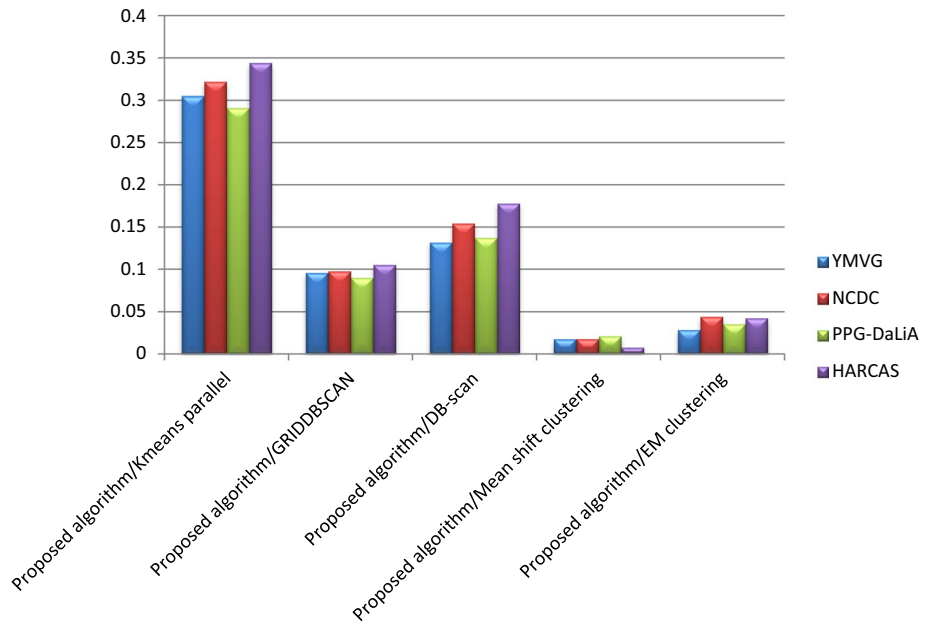

Fig. 6 Percentage of improvement for Rand Index

clusters increases before running Algorithm 5 as many small clusters merge together in previous steps. If the proposed algorithm presents proper load balancing, then the total execution time will decrease as reducers execute jobs almost simultaneously. Note that our proposed algorithm presents approximated load balancing. Table 6 reports the load balancing and total execution time in datasets by the proposed method. The load balancing of reducers is defined by Eq. 5, which indicates the average difference of execution time in the reducers. Table 6 shows that dataset size enlargement has had a proper effect on execution time and little effect on load balancing in reducers.

t_r *is the execution time of reducer rth*

$\bar{t}$ *represents the average execution time of reducers*

r *is the number of reducers*

LB denotes load balancing in the execution time

Table 6 Correlation between load balancing and total execution time

Datasets	Load balancing time (phase 3)	Total time	LB
YMVG	32.96	619.78	0.8937
NCDC	88.31	1790.34	0.793
PPG-DaLiA	22.61	447.16	0.836
HARCAS	93.26	1837.37	0.813

$$LB = 1 - \frac{\sum_{i=1}^{r} \left| t_r - \bar{t} \right|}{r} \tag{5}$$

Conclusions

In this paper, we proposed a new method based on MR-DBSCAN-KD and futuristic greedy index for processing big data. Our proposed method was composed of 5 steps. In the first step, big data were partitioned into data points and the data points were stored in the Hadoop structure. In the second step, the data points stored in Hadoop were clustered using MR-DBSCAN-KD in parallel. The outliers were then assigned to the existing clusters using the Futuristic Greedy Index. At the end of the second step, the clusters were merged together based on the distance between their centroids. As a result, the number of clusters decreased. In the third step, the clusters were classified based on the decline in the number of reducers. In the fourth step, the clusters were assigned to the reducers, and in the fifth step the outputs of the reducers were merged together.

Our experimental results indicated that use of this method reduced the execution time of jobs. A reasonable execution time was achieved since less data were processed in parallel in mappers and reducers throughout each phase. Meanwhile similar data were located in the same reducer. This led the reducers to execute the jobs faster. A decrease in the number of reducers resulted in shorthand execution time. Note that creation of outliers is a drawback of MR-DBSCAN-KD. As a solution, the proposed method used futuristic greedy method for assigning these outliers to existing clusters. Exchange of jobs between clusters is likely to improve load balancing, and is recommended to be considered in future research. Also, utilisation of novel density-based algorithms instead of MR-DBSCAN-KD might decrease the execution time.

Abbreviations

FGI: Futuristic Greedy Index; *FGC*: Futuristic Greedy Clustering; *fs*: File System; *HDFS*: Hadoop Distributed File System.

Authors' contributions

All authors read and approved the final manuscript.

Funding

The authors declare that they have no funding.

Availability of data and materials

All data used in this study are publicly available and accessible in the cited sources. Dataset: including details of Dataset that used in experiment see the web site: https://archive.ics.uci.edu/ml.

Ethics approval and consent to participate

The authors ethics approval and consent to participate.

Consent for publication
The authors consent for publication.

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

Author details
[1] Kish International Campus, Sharif University of Technology, Tehran, Iran. [2] Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.

Received: 2 August 2019 Accepted: 23 December 2019
Published online: 09 January 2020

References

1. Tsai C-W, Lai C-F, Chao H-C, Vasilakos AV. Big data analytics: a survey. J Big data. 2015;2(1):21.
2. Sanse K, Sharma M. Clustering methods for Big data analysis. Int J Adv Res Comput Eng Technol. 2015;4(3):642–8.
3. Zhao W, Ma H, He Q. Parallel k-means clustering based on mapreduce. In: IEEE international conference on cloud computing. 2009. p. 674–9.
4. Srivastava DK, Yadav R, Agrwal G. Map reduce programming model for parallel K-mediod algorithm on hadoop cluster. In: 2017 7th international conference on communication systems and network technologies (CSNT). 2017. p. 74–8.
5. Dai B-R, Lin I-C. Efficient map/reduce-based dbscan algorithm with optimized data partition. In: 2012 IEEE Fifth international conference on cloud computing. 2012. p. 59–66.
6. He Y, Tan H, Luo W, Feng S, Fan J. MR-DBSCAN: a scalable MapReduce-based DBSCAN algorithm for heavily skewed data. Front Comput Sci. 2014;8(1):83–99.
7. Verma A, Cherkasova L, Campbell RH. Two sides of a coin: Optimizing the schedule of mapreduce jobs to minimize their makespan and improve cluster performance. In: 2012 IEEE 20th international symposium on modeling, analysis and simulation of computer and telecommunication systems. 2012. p. 11–8.
8. Ramakrishnan SR, Swart G, Urmanov A. Balancing reducer skew in MapReduce workloads using progressive sampling. In: Proceedings of the Third ACM symposium on cloud computing. 2012. p. 16.
9. Fan L, Gao B, Zhang F, Liu Z. OS4M: Achieving Global Load Balance of MapReduce workload by scheduling at the operation level. arXiv Prepr arXiv14063901. 2014.
10. Xia H. Load balancing greedy algorithm for reduce on Hadoop platform. In: 2018 IEEE 3rd international conference on big data analysis (ICBDA). 2018. p. 212–6.
11. Xia D, Wang B, Li Y, Rong Z, Zhang Z. An efficient MapReduce-based parallel clustering algorithm for distributed traffic subarea division. Discret Dyn Nat Soc. 2015;2015.
12. Ke H, Li P, Guo S, Guo M. On traffic-aware partition and aggregation in mapreduce for big data applications. IEEE Trans Parallel Distrib Syst. 2015;27(3):818–28.
13. Reddy YD, Sajin AP. An efficient traffic-aware partition and aggregation for big data applications using map-reduce. Indian J Sci Technol. 2016;9(10):1–7.
14. Venkatesh G, Arunesh K. Map Reduce for big data processing based on traffic aware partition and aggregation. Cluster Comput. 2018. p. 1–7.
15. HajKacem MA, N'cir C-E, Essoussi N. One-pass MapReduce-based clustering method for mixed large scale data. J Intell Inf Syst. 2019;52(3):619–36.
16. Ilango SS, Vimal S, Kaliappan M, Subbulakshmi P. Optimization using artificial bee colony based clustering approach for big data. Cluster Comput. 2018. p. 1–9.
17. Fan T. Research and implementation of user clustering based on MapReduce in multimedia big data. Multimed Tools Appl. 2018;77(8):10017–31.
18. Jane EM, Raj E. SBKMMA: sorting based K means and median based clustering algorithm using multi machine technique for big data. Int J Comput. 2018;28(1):1–7.
19. Kaur A, Datta A. A novel algorithm for fast and scalable subspace clustering of high-dimensional data. J Big Data. 2015;2(1):17.
20. Kanimozhi K V, Venkatesan M. A novel map-reduce based augmented clustering algorithm for big text datasets. In: Data Engineering and Intelligent Computing. New York: Springer; 2018. p. 427–36.
21. Zerabi S, Meshoul S, Khantoul B. Parallel clustering validation based on MapReduce. In: International conference on computer science and its applications. 2018. p. 291–9.
22. Hosseini B, Kiani K. FWCMR: a scalable and robust fuzzy weighted clustering based on MapReduce with application to microarray gene expression. Expert Syst Appl. 2018;91:198–210.
23. Reddy KHK, Pandey V, Roy DS. A novel entropy-based dynamic data placement strategy for data intensive applications in Hadoop clusters. Int J Big Data Intell. 2019;6(1):20–37.
24. Beck G, Duong T, Lebbah M, Azzag H, Cérin C. A Distributed and approximated nearest neighbors algorithm for an efficient large scale mean shift clustering. arXiv Prepr arXiv190203833. 2019.
25. Gates AJ, Ahn Y-Y. The impact of random models on clustering similarity. J Mach Learn Res. 2017;18(1):3049–76.
26. Heidari S, Alborzi M, Radfar R, Afsharkazemi MA, Ghatari AR. Big data clustering with varied density based on MapReduce. J Big Data. 2019;6(1):77.
27. Kenyon C, others. Best-Fit Bin-Packing with Random Order. In: SODA. 1996. p. 359–64.
28. Data set. https://archive.ics.uci.edu/ml/. Accessed 9 Feb 2018.
29. Data set. ftp://ftp.ncdc.noaa.gov/pub/data/uscrn/products/subhourly01. Accessed 11 Feb 2019.

30. Sammut C, Webb GI. Encyclopedia of machine learning. New York: Springer; 2011.
31. Rand WM. Objective criteria for the evaluation of clustering methods. J Am Stat Assoc. 1971;66(336):846–50.

Publisher's Note