This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TKDE.2015.2391123

GDCluster: A General Decentralized Clustering Algorithm

Hoda Mashayekhi, Jafar Habibi, Tania Khalafbeigi, Spyros Voulgaris, Maarten van Steen, *Senior Member, IEEE*

Abstract—In many popular applications like peer-to-peer systems, large amounts of data are distributed among multiple sources. Analysis of this data and identifying clusters is challenging due to processing, storage, and transmission costs. In this paper, we propose GDCluster, a general fully decentralized clustering method, which is capable of clustering dynamic and distributed data sets. Nodes continuously cooperate through decentralized gossip-based communication to maintain summarized views of the data set. We customize GDCluster for execution of the partition-based and density-based clustering methods on the summarized views, and also offer enhancements to the basic algorithm. Coping with dynamic data is made possible by gradually adapting the clustering model. Our experimental evaluations show that GDCluster can discover the clusters efficiently with scalable transmission cost, and also expose its supremacy in comparison to the popular method LSP2P.

Index Terms—Distributed Systems, Clustering, Partition-based Clustering, Density-based Clustering, Dynamic System

1 Introduction

Clustering, or unsupervised learning, is important for analyzing large data sets. Clustering partitions data into groups (clusters) of similar objects, with high intra-cluster similarity and low inter-cluster similarity. With the progress of large-scale distributed systems, huge amounts of data are increasingly originating from dispersed sources. Analyzing this data, using centralized processing, is often infeasible due to communication, storage and computation overheads. Distributed Data Mining (DDM) focuses on the adaptation of data-mining algorithms for distributed computing environments, and intends to derive a global model which presents the characteristics of a data set distributed across many nodes.

In fully distributed clustering algorithms, the data set as a whole remains dispersed, and the participating distributed processes will gradually discover various clusters [1]. Communication complexity and overhead, accuracy of the derived model, and data privacy are among the concerns of DDM. Typical applications requiring distributed clustering include: clustering different media metadata (documents, music tracks, etc.) from different machines; clustering nodes' activity history data (devoted resources, issued queries; download and upload amount, etc.); clustering books in a distributed network of libraries; clustering scientific achievements from different institutions and publishers.

A common approach in distributed clustering is to combine and merge local representations in a central node, or aggregate local models in a hierarchical structure [2], [3]. Some recent proposals, although being completely decentralized, include synchronization at the end of each round, and/or require nodes to maintain history of the clustering [4], [5], [6], [7].

In this paper, a General Distributed Clustering algorithm (GDCluster) is proposed and instantiated with two popular partition-based and density-based clustering methods. We first introduce a basic method in which nodes gradually build a summarized view of the data set by continuously exchanging information on data items and data representatives using gossip-based communication. Gossiping [8] is used as a simple, robust and efficient dissemination technique, which assumes no predefined structure in the network. The summarized view is a basis for executing weighted versions of the clustering algorithms to produce approximations of the final clustering results.

GDCluster can cluster a data set which is dispersed among a large number of nodes in a distributed environment. It can handle two classes of clustering, namely partition-based and density-based, while being fully decentralized, asynchronous, and also adaptable to churn. The general design principles employed in the proposed algorithm also allow customization for other classes of clustering, which are left out of the current paper. We also discuss enhancements to the algorithm particularly aimed at improving communication costs.

The simulation results presented using real and synthetic data sets, show that GDCluster is able to achieve a high-quality global clustering solution, which approximates centralized clustering. We also explain effects of various parameters on the accuracy and overhead of the algorithm. We compare our proposal with central clustering and with the LSP2P algorithm [4], and also show its supremacy in achieving higher quality clusters. The main contributions of this paper are as follows:

- Proposing a new fully distributed clustering algorithm, which can be instantiated to at least two categories of clustering algorithms.
- Dealing with dynamic data and evolving the clustering model.
- Empowering nodes to construct a summarized view of

- *H. Mashayekhi, J. Habibi and T. Khalafbeigi are with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran, -E-mail: mashayekhi@ce.sharif.edu, tkhalafb@ucalgary.ca, jhabibi@sharif.ir*
- *S. Voulgaris and M. van Steen are with the Department of Computer Science, Vrije Universiteit Amsterdam, De Boelelaan 1081A, 1081 HV Amsterdam, The Netherlands. E-mail: {spyros, steen}@few.vu.nl.*

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TKDE.2015.2391123

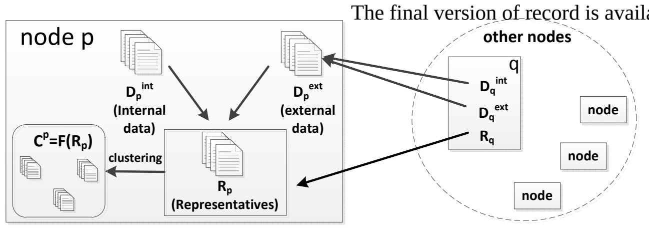


Fig. 1. A graphical view of the system model.

the data, to be able to execute a customized clustering algorithm independently.

This paper is organized as follows. The system model is described in Section 2. In Section 3, the basic decentralized algorithm is introduced. In the succeeding section we propose adjustments to deal with churn. Section 5 discusses enhancements. Simulation results are discussed in Section 6, followed by related work and conclusion.

2 System Model

We consider a set $P = \{p_1, p_2, \dots, p_n\}$ of n networked **nodes**. Each node p stores and shares a set of data items D_p^{int}, denoted as its **internal data**, which may change over time. $\mathbf{D} = \bigcup_{p \in P} D_p^{int}$ is the set of all data items available in the network. Each data item d is presented using an attribute (meta data) vector denoted as $\mathbf{d_{attr}}$. Whenever transmission of data items is mentioned in the text, transmission of the respective attribute vector is intended.

While discovering clusters, p may also store attribute vectors of data items from other nodes. These items are referred to as the **external data** of p, and denoted as D_p^{ext}. The union of internal and external data items of p is referred to as $D_p = D_p^{int} \cup D_p^{ext}$.

During algorithm execution, each node p gradually builds a summarized view of $\mathbf{D}$, by maintaining representatives, denoted as $R_p = \{r_1^p, r_2^p, \dots, r_{k_p}^p\}$. Each representative $r \in R_p$ is an artificial data item, summarizing a subset D_r of $\mathbf{D}$. The attribute vector of r, $\mathbf{r_{attr}}$, is ideally the average of attribute vectors[1] of data items in D_r. The intersection of these subsets need not be empty, i.e., $\forall r, r' \in R_p . |D_r \cap D_{r'}| \geq 0$. The actual set D_r is not maintained by the algorithm, and is discarded once r is produced.

Each data item or representative x in p, has an associated weight $w_p(x)$. The weight of x is equal to the number of data items which, p believes, x is composed of. Depending on whether x is a representative or a data item, $w_p(x)$ should ideally be equal to $|D_x|$ or one, respectively.

The goal of this work is to make sure that the complete data set is clustered in a fully decentralized fashion, such that each node p obtains an accurate clustering model, without collecting the whole data set. The representation of the clustering model depends on the particular clustering method. For partition-based and density-based clustering, a centroid and a set of core points can serve as cluster indicators, respectively. Whenever the actual type of clustering is not important, we refer to the clustering method simply as **F**. Fig. 1 provides a summarized view of the system model.

3 Decentralized Clustering

Each node gradually builds a summarized view of $\mathbf{D}$, on which it can execute the clustering algorithm **F**. In the next subsections,

1. In the context of this work, all operations on data items or representatives, are vector operations on the corresponding attribute vectors.

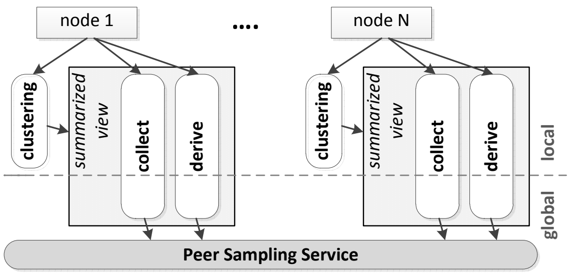


Fig. 2. The overall view of the algorithm tasks.

we first discuss how the summarized view is built. Afterwards, the method of weight calculation is described, followed by the execution procedure of the clustering algorithm.

3.1 Building the summarized view

As described in Section 2, we assume that the entire data set can be summarized in each node p, by means of representatives. Each node p is responsible for deriving accurate representatives for part of the data set located near D_p^{int}. For other parts, it solely collects representatives. Accordingly, it gradually builds a global view of $\mathbf{D}$. Each node continuously performs two tasks in parallel: i) Representative derivation, which we name DERIVE and ii) representative collection, which we name COLLECT. The two tasks can execute repeatedly and continuously in parallel. An outline of the tasks performed by each node is demonstrated in Fig. 2. We use two gossip-based, decentralized cyclic algorithms to accomplish the two tasks, as described in the next subsections.

3.1.1 *DERIVE*

To derive representatives for part of the data set located near D_p^{int}, p should have an accurate and up-to-date view of the data located around each data $d \in D_p^{int}$. In each round of the DERIVE task, each node p selects another node q for a three-way information exchange, as shown in Fig. 3. It should first send D_p^{int} to node q. If size of D_p^{int} is large, it can summarize the internal data by an arbitrary method such as grouping the data using clustering, and sending one data from each group. Node p then receives from q, data items located in radius ρ of each $d \in D_p^{int}$, based on a distance function δ. ρ is a user-defined threshold, which can be adjusted as p continues to discover data (to which we return to in Section 6.1). In the same manner, it will also send to q the data in D_p that lie within the ρ radius of data in D_q^{int}. The operation updateLocalData() is used to add the received data to D_p^{ext}.

Knowing some data located within radius ρ of some internal data item d, node p can summarize all this data into one representative. This is performed periodically every τ gossip rounds using the algorithm of Fig. 4. The mergeWeights function, updates the representative weight, and is later described in Section 3.3

3.1.2 *COLLECT*

To fulfill the COLLECT task, each node p selects a random node every T time units, to exchange their set of representatives with each other (Fig. 5). Both nodes store the full set of representatives. The summarize function used in the algorithm, simply returns all the representatives given to it as input. A special implementation of this function is described in Section 5.1, which reduces the number of representatives.

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TKDE.2015.2391123

```
Process ActiveThread($p$):                    Process PassiveThread($q$):
  loop                                          loop
    wait $T$ time units                           .
    preprocessTask1()                             .
    $q \leftarrow$ selectNode()                   receiveFromAny($p$, $D_p^{int}$)
    sendTo($q$, summarize($D_p^{int}$))           preprocessTask1()
    .                                             $D_q^* \leftarrow \{d \in D_q | \exists d' \in D_p^{int} : \delta(\mathbf{d}_{\mathbf{attr}}, \mathbf{d'}_{\mathbf{attr}}) \le \rho\}$
    .                                             sendTo($p$, $D_q^*$, summarize($D_q^{int}$))
    receiveFrom($q$, $D_q^*$,$D_q^{int}$)         .
    $D_p^* \leftarrow \{d \in D_p | \exists d' \in D_q^{int} : \delta(\mathbf{d}_{\mathbf{attr}}, \mathbf{d'}_{\mathbf{attr}}) \le \rho\}$    .
    sendTo($q$, $D_p^*$)                          receiveFrom($p$, $D_p^*$)
    updateLocalData($D_q^*$)                      updateLocalData($D_p^*$)
  end loop                                      end loop

          (a)                                           (b)
```

Fig. 3. Task DERIVE: (a) active thread for p and (b) passive thread for selected node q.

```
Process extractRepresentative($p$):
  for $d \in D_p^{int}$ do
    $\Delta_d = \{d\} \cup \{d' | d' \in D_p^{ext} \wedge \forall d'' \in D_p^{int} : \delta(\mathbf{d}_{\mathbf{attr}}, \mathbf{d'}_{\mathbf{attr}}) < \delta(\mathbf{d''}_{\mathbf{attr}}, \mathbf{d'}_{\mathbf{attr}})\}$

    $r = \frac{\sum_{d' \in \Delta_d} (w(d') \times \mathbf{d'}_{\mathbf{attr}})}{\sum_{d' \in \Delta_d} w(d')}$
    for $d' \in \Delta_d$ do
      mergeWeights( $r$ , $d'$)
    end for
    $R_p = R_p \cup \{r\}$
  end for
  removeRepetitives($R_p$)
  $D_p^{ext} = \emptyset$

          (a)

Process removeRepetitives($R$)
  for $r, r' \in R | \mathbf{r}_{\mathbf{attr}} = \mathbf{r'}_{\mathbf{attr}}$ do
    $R = R - \{r'\}$
    mergeWeights($r$, $r'$)
  end for

          (b)
```

Fig. 4. (a) Extracting representatives from the collected data, (b) removing repetitive representatives

Initially, each node has only a set of internal data items, D_p^{int}. Thus, the set of representatives at each node is initialized with all of its data items, i.e., $R_p = D_p^{int}$.

The two algorithms of tasks DERIVE and COLLECT, start with a *preprocessing* operation. In this basic algorithm, these operations have no special function, thus we defer their discussion to Section 4. The graphical representation of the communication performed in DERIVE and COLLECT is depicted in Fig. 6.

The operation selectNode() used in Figures 3 and 5, employs a peer-sampling service to return a node selected uniformly at random from all live nodes in the system (see, e.g., [9]).

```
Process ActiveThread($p$):                    Process PassiveThread($q$):
  loop                                          loop
    wait $T$ time units                           .
    preprocessTask2()                             .
    $q \leftarrow$ selectNode()                   .
    sendTo($q$, summarize($R_p$, $|R_p|$))        .
    .                                             receiveFromAny($p$, $R_p^*$)
    .                                             preprocessTask2()
    receiveFrom($q$, $R_q^*$)                     sendTo($p$, summarize($R_q$,$|R_q|$))
    $R_p = R_p \cup R_q^*$                        $R_q = R_q \cup R_p^*$
    removeRepetitives($R_p$)                      removeRepetitives($R_q$)
  end loop                                      end loop

          (a)                                           (b)
```

Fig. 5. Task COLLECT: (a) active thread for p and (b) passive thread for selected node q.

3.2 Diffusion Speed

In tasks DERIVE and COLLECT we use gossiping as a propagation media. This is in particular different from aggregation protocols [8] which employ gossiping to reach consensus on aggregations. Using vocabulary of [8] and ignoring the details, the general approach of GDCluster can be simplified as follows. At all times t, a node p maintains an ordered set (not a sum) $s_{t,p}$, initialized to $s_{0,p} = D_p^{int}$, and an ordered set of corresponding weights $w_{t,p}$. At each time step t, p chooses a target node $f_t(p)$ uniformly at random and sends both collections to that node and itself. It calculates union of the received pairs $(\hat{s}_r, \hat{w}_r)$ from other nodes with its own s and w sets. In step t of the algorithm, $s_{t,p}$ is the view p has on the entire dataset, while $w_{t,p}$ contains the corresponding weight of each view element. As the set s quickly becomes large, the notion of representatives are introduced. Node p can summarize the elements of $s_{t,p}$ by removing a subset, computing the average of its elements (locally), and replacing the average value in $s_{t,p}$. The corresponding weights should also be removed and replaced by the aggregate weight. This summarized view is labeled R_p in this paper.

According to [8] and [10], a message that originates with p at time t_0 and is forwarded by all nodes that have received it, will reach all nodes in time at most $4 \log N + \log \frac{2}{\delta}$ with probability at least $1 - \frac{\delta}{2}$. Therefore, after the same time order, the summarized view of p, will have elements from all other nodes, either in their raw form or embedded in a representative.

3.3 Weight calculation

When representatives are merged, for example in the function removeRepetitives, a special method should be devised for weight calculation. The algorithm does not record the set D_r for each representative r, due to resource constraints. Also, there is a possibility of intersection between summarized data of different representatives. To address the weight calculation issue, representative points are accompanied by a (small size) “estimation field”, that allows us to approximate the number of actual items it represents.

We adopt the method of distributed computing of a sum of numbers, introduced in [11]. The algorithm is based on properties of exponential random variables, and reduces the problem of computing the sum to determining the minimum of a collection of numbers. After briefly introducing the method, we describe the algorithm of weight calculation.

3.3.1 General counting

We aim to compute the number of items in a set X. We consider s independent hash functions mapping an item $x \in X$ to s real numbers exponentially distributed with rate 1. These values are called weight estimators and are denoted as $\hat{w}^1(x), \ldots, \hat{w}^s(x)$. Next, the minimum value per each of the s numbers should be computed. Let $\hat{W}^l = min\{\hat{w}^l(x) | x \in X\}$. Upon establishing the minimum values, an estimate of the total number is given by the formula:

$$\hat{c} = \frac{s}{\sum_{l=1}^{s} \hat{W}^l} \quad (1)$$

The basic intuition behind the estimation, is that the minimum of n independent random variables, each with exponential distribution of rate λ_j, is an exponential random variable of rate $\lambda = \sum_{j=1}^{n} \lambda_j$.

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TKDE.2015.2391123

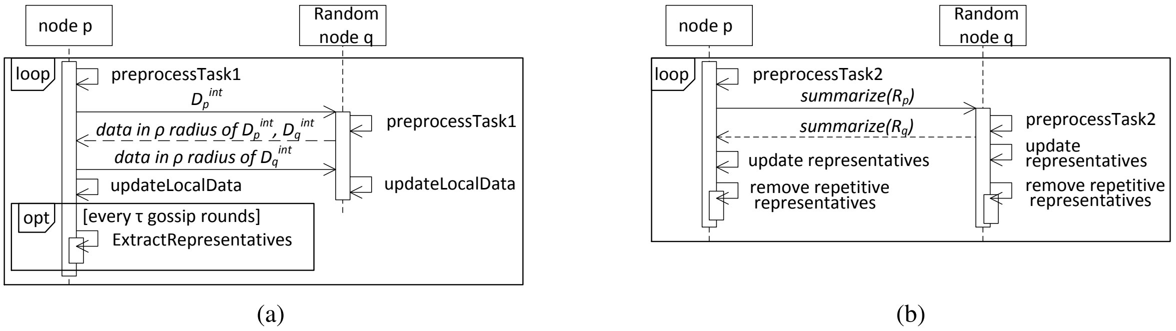


Fig. 6. Communication sequence of tasks (a) DERIVE and (b) COLLECT, for a typical node p

```
Process mergeWeights($r, x$):
    for $l \in 1 \ldots s$ do
        $\hat{w}^l_p(r) = \begin{cases} min\{\hat{w}^l_p(r), \hat{w}^l_p(x)\} & \hat{w}^l_p(r) \neq null \\ \hat{w}^l_p(x) & \text{otherwise} \end{cases}$
    end for
```

Fig. 7. The update procedure for weight estimators. x is a new representative or data item which is embedded in representative r

3.3.2 Weight calculation

To incorporate the described counting procedure into the basic algorithm, node p should store s values, per each data item $d \in D_p$ and each representative $r \in R_p$. Whether x is a data item or a representative, let $\hat{w}^1_p(x), \ldots, \hat{w}^s_p(x)$ denote the weight estimators stored for x in node p. The s random values associated with each real data item, should be deterministically generated by any node based on the attribute vector. Therefore, we have to provide s independent hash functions that deterministically map any given point to s hash values, exponentially distributed with rate 1.

For a representative r, when r is first created, s weight estimators are assigned to it with initial null values. The weight estimators are then updated in the mergeWeights function of Fig. 7,which updates the estimators incrementally. The s values per each representative r, accompany r when it is transferred to another node in task COLLECT.

The estimated number of data items summarized by a representative $r \in R_p$, i.e., $w_p(r)$, is given by the following formula:

$$w_p(r) = \frac{s}{\sum_{l=1}^{s} \hat{w}^l_p(r)} \tag{2}$$

An important observation is that by using the minimum operator, re-assignment of a data item to a representative does not increase its weight.

3.4 Final clustering

The final clustering algorithm $\mathbf{F}$ is executed on the set of representatives in a node. Node p can execute a weighted version of the clustering algorithm on R_p, any time it desires, to achieve the final clustering result. In a static setting, continuous execution of DERIVE and COLLECT will improve the quality of representatives causing the clustering accuracy to converge. In the following, we discuss partition-based and density-based clustering algorithms as examples.

3.4.1 Partition-based clustering

K-means [12] considers data items to be placed in an m-dimensional metric space, with an associated distance measure δ. It partitions the data set into k clusters, $C_1, C_2, \ldots, C_k$. Each cluster C_j has a centroid μ_j, which is defined as the average of all data assigned to that cluster. This algorithm tries to minimize the following objective function:

$$\sum_{j=1}^{k} \sum_{d_l \in C_j} \| d_l - \mu_j \|^2 \tag{3}$$

Weighted K-means assumes a positive weight for each data item and uses weighted averaging. The centroids themselves will be assigned weight values, indicating number of data assigned to the clusters. The formal definition of the weighted K-means is given in Fig. 8. The algorithm proceeds heuristically. A set of random centroids are picked initially, to be optimized in later iterations.

The available approaches of distributed partition-based clustering typically assume identical initial K-means centroids in all nodes [4], [5], [6]. This is, however, not required in our algorithm as each node can use an arbitrary parameter k with an arbitrary set of initial centroids.

Convergence

The weighted K-means algorithm is executed on a set of representatives, each extracted from data within ρ distance of a data item, and its ultimate goal at node p is to compute the mean of data in each cluster. Let D_{C_i} denote the the data items of a typical cluster C_i, and R_{C_i} denote representatives computed from data in D_{C_i}. If D_{C_i} is uniform, the expected value of the representatives will be equal to μ_i. The COLLECT step actually performs continuous random sampling of the set R_{C_i} and the convergence bound to the expected value is given by the Hoeffding inequality.

If D_{C_i} is not uniform, the expected value of representatives in R_{C_i} will deviate from μ_i. In such cases, we can consider subsets of D_{C_i}, each being approximately uniform. The finest region considered to be uniform is a ρ-neighbourhood. Representatives in such a neighbourhood share high ratio of data items with each other. Inspired from this property, R_{C_i} can be decomposed as follows. To extract subset $R^j_{C_i} \subseteq R_{C_i}$ an arbitrary representative $r \in R_{C_i}$ is added to $R^j_{C_i}$, followed by all representatives $r' \in R_{C_i}$ such that $\exists r \in R^j_{C_i} . |r - r'| \leq \rho$. This is similar to the DBSCAN algorithm [13] with representatives considered as core points and ε being equal to ρ. Number of regions produced by the mentioned algorithm depends on intrinsic features of data and number of samples. Computing the average and count of each subset (section 3.3), the average of D_{C_i} can be correctly computed using weighted averaging.

To be consistent, we can use the averaging as described above in all iterations of the weighted K-means algorithm. The set of representatives in each cluster, are identified with the usual nearest centeroid method of K-means. It is obvious that if clusters

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TKDE.2015.2391123

```
Process weightedK-means($R$,$k$):
  Output Partition of $R$ into $k$ clusters
  Select $k$ data items[a] from $R$ as the initial centroids: $\mu_1,\ldots,\mu_k$
  Define $k$ assignment sets $A=\{A_1,\ldots,A_k\}$ for holding members of the clusters

  repeat
    $A_j=\{r|r\in R\wedge\delta(r,\mu_j)\le\delta(r,\mu_i),i\neq j\}$
    $A_j \leftarrow$reduce($A_j$)
    $\mu_j=\frac{\sum_{r\in A_j}(w(r)\times r)}{\sum_{r\in A_j}w(r)}$
  until $|\mu_j^{new}-\mu_j^{old}|\le\varepsilon$
  for $j\in 1\ldots k$ do
    for $r\in A_j$ do
      mergeWeights($\mu_j$, $r$)
    end for
  end for
```

```
Process reduce($A_j$):
  while $\exists r,r'\in A_j.\delta(r,r')\le\rho$ do
    $\Delta_r=\{r\}\cup\{r'|\delta(r,r')\le\rho\}$
    $r=\frac{\sum_{r'\in\Delta_r}(w(r')\times \mathbf{r'_{attr}})}{\sum_{r'\in\Delta_r}w(r')}$
    for $r'\in\Delta_r$ do
      mergeWeights( $r$ , $r'$)
    end for
    $A_j=\{A_j-\Delta_r\}\cup\{r\}$
  end while
```

a. The first initial centroid is selected randomly, and each next centroid is selected such that its minimum distance to each of the previous centroids is maximum.

Fig. 8. The weighted K-means algorithm

are intertwined, the ρ parameter should be decreased and more sampling should be performed which introduces an accuracy/cost trade-off.

3.4.2 *Density-based clustering*

In density-based clustering, a node p can execute, for example, a weighted version of DBSCAN [13] on $R_p$with parameters *minPts* and ε. In DBSCAN, a data item is marked as a core point if it has at least *minPts* data items within its ε radius. Also, two core points are within one cluster, if they are in ε range of each other, or are connected by a chain of core points, where each two consecutive core points have a maximum distance of ε. A non-core data item located within ε distance from a core point, is in the same cluster as that core point, otherwise it is an outlier.

In our algorithm, each representative may cover a region with radius[2] greater than ε. Also a representative does not necessarily have the same attribute vector as any regular data item. Therefore, representatives do not directly mimic core points. Nevertheless, core points in DBSCAN are a means of describing data density. Adhering to this concept, representatives can also indicate dense areas.

The ρ parameter of the DERIVE task can be set to ε. This ensures that if some data item is a core point, the corresponding derived representative will have a minimum weight of *minPts*. This customization also suggests that per each internal data item, at most *minPts* data items should be transferred in COLLECT.

One of the benefits of DBSCAN is its ability to detect outliers. To achieve this in our algorithm, task COLLECT should be customized to transfer only representatives with weight larger than *minPts*. This causes representatives located outside the actual clusters not to be disseminated in the network, and improves the overall clustering accuracy.

The density-based clustering method just described can be considered a slightly modified version of the distributed density-based clustering algorithm GoScan [14]. In GoScan nodes detect core points and disseminate them through methods very similar to COLLECT and DERIVE. GoScan is an exact method, whereas here we are providing an approximate method.The approximation imposes less communication overhead, and faster convergence of the algorithm.

2. The maximum distance of the representative to points embedded in it.

4 Dynamic data set

Real-world distributed systems change continuously, because of nodes joining and leaving the system, or because their set of internal data is modified.

To model staleness of data, each data item will have an associated **age**. $age_p(d)$ denotes the time that node p believes has passed since d was obtained from its originating, owning node. Time is measured in terms of gossiped rounds. The age of data items accompany them in the DERIVE task. The age of an external data item at node p is increased (by p) before each communication; the age of an internal data always remains zero to reflect that it is stored (and up-to-date) at its owner. If a node p receives a copy d' of a data item d it already stores, $age_p(d)$ is set to $\min\{age_p(d),age_p(d')\}$ (and d' is further ignored).

When a data item d is removed from the original peer, the minimal recorded age among all its copies will only increase. Node p can remove data item d if $age_p(d)>MaxAge$, where *MaxAge* is some threshold value, presuming that the original data item has been removed. An **age** argument is also associated with each representative; $age_p(r)$ is set to zero when r is first produced by p, and increased by one before each communication.

The weight of a data item or a representative is a function of its age. For a data item d, the weight function is ideally one for all age values not greater than *MaxAge*. The data items summarized by a representative have different lifetimes according to their age. Therefore, the weight of the representative should capture the number of data items summarized by the representative at each age value. When the weight value falls to zero, the representative can be safely removed. We will see below that instead of the actual weight, the weight estimators are stored per each age value to enable further merging and updating of representatives.

The weight function of a representative will always be in the form of a descending step function for values greater than $age_p(r)$, and will reach zero at most at $age_p(r)+MaxAge$. All of the data currently embedded in the representative will be gradually removed, and no data can last longer that *MaxAge* units from the current time.

With the weight function being dependent on age, the weight estimators are in turn bound to the age values. $\hat{w}_p^l(x,t)$ presents the l'th weight estimator of item x in age t, from the view of peer p. For a data item d, while $age_p(d)\le MaxAge$, each weight estimator preserves its initial value, and is null otherwise. For a

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TKDE.2015.2391123

representative r, s weight estimators are recorded at each age value greater than $age_p(d)$, up to the point where all data embedded in the representative are removed. At that point all weight estimators will become null. The new method of updating estimates is shown in Fig. 9. In this figure, a set of data/representatives x is to be merged with the representative r. For each age value between 0 and $MaxAge$, first the data/representatives which have positive weight are put in the set X^*. Then, the minimum of weight estimators is calculated for members of this set. The weight function, with the age argument can be computed on demand from the estimators:

$$w_p(x,t) = \begin{cases} \frac{s}{\sum_{l=1}^{s} \hat{w}_p^l(x,t)} & t \geq age_p(r) \wedge (\hat{w}_p^i(x,t) \neq \ null, \\ & \qquad i = 1 \dots s) \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

When r is sent to another node in COLLECT the weight estimate values for ages greater than $age_p(r)$ should accompany it.

To incorporate these new concepts in the basic algorithm, the two preprocessing operations of DERIVE and COLLECT should be modified to increase age values of data and representatives, and remove them if necessary. Moreover, before storing the received data in DERIVE, the age values for repetitive data items should be corrected. These operations are shown in Fig. 10.

5 Enhancements

In this section we discuss a number of improvements to the basic algorithm, to enhance the consumed resources.

5.1 Summarization

Nodes may have limited storage, processing and communication resources. The number of representatives maintained at a node increases as the DERIVE and COLLECT tasks proceed. When the number of representatives and external data items stored at p exceeds its local capacity LC_p, first the representative extraction algorithm of Figure 4 is executed to process and then discard external data. Afterwards, the summarization task of Figure 11 is executed with parameters R_p and αLC_p, and the result is stored as the new R_p set. $0 < \alpha < 1$ is a locally determined parameter, controlling consumption of local resources. Dealing with limitations of processing resources is similar.

If the number of representatives and external data items to be sent by p in the DERIVE and COLLECT tasks, exceeds its communication capacity CC_p, the same summarization task of Figure 11 is executed with parameters R_p and βCC_p. Thus, a reduced set of representatives is obtained. $0 < \beta < 1$ is a parameter controlling the number of transmitted representatives. If the external data items to be sent in the DERIVE task exceed the communication limits, sampling is used to reduce the amount of data.

The summarization task actually makes use of weighted K-means (described in Section 3.4.1), which effectively "summarizes" a collection of data items by means of a single representative with an associated weight.

5.2 Weight estimators

According to the adopted approach of weight estimation and dynamicity handling, the amount of data transmitted in COLLECT can get large. Actually, the algorithm has to transmit up to $s\times$ $MaxAge$ values for each representative. Large values of $MaxAge$ can hence increase the communication costs. To diminish this cost, we use regression analysis to model each of the s values using an exponential function.

Each of the s weight estimators of a representative r, is a function of age. This function can be identified by at most $MaxAge$ tuples of $(age, value)$ pairs. Based on these tuples, an exponential regression in the form of ab^{age} can be derived for each estimator, after which the tuples can be discarded. Consequently, per each representative, at most $2s$ values should be transmitted.

6 Performance evaluation

We evaluate the GDCluster algorithm in static and dynamic settings. We will also compare GDCluster with a central approach and with LSP2P, a recently proposed algorithm being able to execute in similar distributed settings.

6.1 Evaluation model

We consider a system of N nodes, each node initially holding a number of data items, and carrying out the DERIVE and COLLECT tasks iteratively. For simplicity and better understanding of the algorithm, we consider only data churn in the dynamic setting. In each round, a fraction of randomly selected data items is replaced with new data items. By using the peer sampling service, the network structure is not a concern in the evaluations [9].

Each cluster in the synthetic data sets consists of a skewed set of data composed from two Gaussian distributions with different values of mean and standard deviation. The real data sets used for the partition-based clustering are the well-known Shuttle, MAGIC Gamma Telescope, and Pendigits data sets[3]. These data sets contain 9, 10, and 16 attributes, and are clustered into 7, 2, and 10 clusters, respectively. From each data set, a random sample of 10240 instances are used in the experiments.To assign the data set **D** to nodes, two data-assignment strategies are employed, which aid at revealing special behaviours of the algorithm:

- Random data assignment (RA): Each node is assigned data randomly chosen from **D**.

- Cluster-aware data assignment (CA): Each node is assigned data from a limited number of clusters.

The second assignment strategy abates the average number of nodes that have data close to each other. Such a condition reduces the number of other nodes which have target data for the COLLECT task. When applying churn, in the first assignment strategy, data items are replaced with random unassigned data items. The second data assignment strategy allows *concept drift* when applying churn, by reserving some of the clusters and selecting new points from these clusters. Concept drift refers to change in statistical properties of the target data set which should be clustered.

Nodes can adjust the ρ parameter during execution based on the incurred communication complexity. In the evaluations, for simplicity, the ρ parameter is selected such that the average number of data located within the ρ radius of each data item is equal to 5.

Different parameters used in conducting the experiments, along with their value ranges and defaults, are presented in Table 6.1. The parameter values are selected such that special behaviours of the algorithm are revealed. LC and CC are measured as multiples of the required resource for one representative.

3. http://archive.ics.uci.edu/ml/

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TKDE.2015.2391123

```
Process mergeWeights($r$, $x$):
1: for $t \in 0 \dots MaxAge$ do
2:     $X^* = \{x' | x' \in \{r,x\} \wedge w_p(x', age_p(x')+t) \neq 0\}$
3:     $\hat{w}^l_p(r, age_p(r)+t') = \begin{cases} \min \bigcup_{x' \in X^*} \{\hat{w}^l_p(x', age_p(x')+t)\} & X^* \neq \emptyset \\ \text{null} & \text{otherwise} \end{cases}$
       $(l = 1 \dots s)$
4: end for
```

Fig. 9. The updated mergeWeights procedure, with an extra age argument.

```
Operation preprocessTask1():
    for each $d \in D^{ext}_p$ do $age_p(d) \leftarrow age_p(d)+1$
    for each $d \in D^{ext}_p$ with $age_p(d) > MaxAge$ do $D^{ext}_p \leftarrow D^{ext}_p - \{d\}$
Operation updateLocalData($D^*_q$):
    for each $d \in D_p \cap D^*_q$ do $age_p(d) \leftarrow \min\{age_p(d), age_q(d)\}$
    for each $d \in D^*_q$ with $d \notin D_p$ do $D^{ext}_p \leftarrow D^{ext}_p \cup \{d\}$

Operation preprocessTask2():
    for each $r \in R_p$ do $age_p(r) \leftarrow age_p(r)+1$
    for each $r \in R_p$ with $w_p(r) = 0$ do $R_p \leftarrow R_p - \{r\}$
```

Fig. 10. Operations used in the DERIVE and COLLECT tasks in a dynamic setting.

```
Process Summarize($R$,$k$):
  Output: $R^*$: A set of reduced representatives.
  if $|R| \geq k$ then
      $R^* = \text{weightedK-means}(R,k)$
  end if
```

Fig. 11. Summarization of representatives.

TABLE 1
Simulation parameters

Symbol	Description	Range (default)
N	Number of nodes	128-16384 (128)
$\|C\|$	Number of real clusters in the data set	8-50
N_{int}	Number of internal data items per node	2-1000 (10)
s	Number of weight estimators	20
τ	The period between representative extraction in DERIVE	<0.4
churn ratio	Fraction of data replaced in each gossip round	10-50% (10%)
$MaxAge$	Threshold for the age parameter	2-38 (10)
LC	Node storage capacity	20-1280 (100)
α	The parameter used in summarizing local representatives	0.5
CC	Node communication capacity	$< 3\|C\|$
β	The parameter used in summarizing communicated representatives	<0.5

The majority of the evaluations is performed with partition-based clustering. Partial evaluation on density-based clustering is discussed at the end of the section.

6.2 Evaluation metrics

In order to assess the efficiency of our algorithm in detecting clusters, we mainly compare its outcome to that of (centralized) K-means using the same initial centroids in the central and distributed settings. Executing K-means centrally on a given data set results in a set of clusters $C_1, C_2, \dots, C_k$, which will be referred to as **real clusters**. Likewise, at any time while executing the algorithm, each node p can derive a set of clusters $C^p_1, C^p_2, \dots, C^p_k$, which we will call **computed clusters** of node p. $map(c)$ is the mapping function that maps a computed cluster c to some equivalent real cluster. Here we use the Kuhn-Munkres algorithm [15] for mapping clusters, which is also used in [16]. Without loss of generality, we assume that computed clusters in each node are ordered according to the mapping, such that $map(C^p_j) = C_j$. Each data item $d \in D$, belongs to a specific global cluster $C(d)$, and a specific computed cluster in each node p, denoted as $C^p(d)$.

The performance metrics are introduced below with respect to a given node. To show aggregate results, we average across all nodes in the system.

- Accuracy (AC). This metric measures the ratio of data items which are located in correct clusters, and is defined as follows [16]:

$$AC = \frac{\sum_{d \in D} eq(C(d), map(C^p(d)))}{|D|} \tag{5}$$

Where $eq(x,y)$ equals one if $x = y$ and zero otherwise.

- Rand index (RandI). RandI is a measure of similarity between two clusters, defined as follows:

$$RandI = \frac{a+b}{\binom{n}{2}} \tag{6}$$

Where a is the number of pairs of elements that are in the same real cluster, and also in the same computed cluster, while b is the number of pairs of elements that are in different real clusters and in different computed clusters. We also use the corrected RandI measure [17].

-Communication/storage overhead The communication/storage cost is measured in terms of average amount of data (in KB) transmitted/stored by each node, per gossip round. Note that the dimensions of data and the weight estimators are considered to be 8 byte doubles.

The error interval in all simulations was lower than 1%, so it is omitted in the graphs.

6.3 Simulation results

We start by presenting the simulation results for the static network, and then proceed to dynamic configurations. Unless explicitly stated, all evaluations involve the algorithm improvements discussed in Section 5. Evaluation of different parameters is mainly performed with the synthetic data set, as we can efficiently control the number of clusters, data density and the churn ratio.

Static settings

When network data is persistent, each node gradually learns the data through its representatives, and the clustering accuracy converges. The algorithm behaviour in a static setting is shown in Fig. 12, where the number of internal data items of each node,

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TKDE.2015.2391123

N_{int}, varies from 2 to 10. The trend of clustering accuracy convergence against simulation rounds, is shown for basic and enhanced GDCluster. Convergence is identified by three rounds of minor (less than 1 percent) change in results. The accuracy converges in to 100% and more than 95% in basic and enhanced GDCluster respectively. The enhanced GDCluster offers less converged accuracy values due to limited transmission of representatives and data, which reduces the quality of the constructed view of data in each node. As observed, in this setting, when nodes have few data (e.g., $N_{int} = 2$), detecting accurate clusters is harder, due to sparseness of clusters.

The same figure compares the basic GDCluster with three improved versions, when N_{int} varies. Communication and storage overheads show average per round values until convergence for each node. The values are considerable for the basic GDCluster due to the storage and transmission of a large number of external data items and representatives. The first improved version involves regression to reduce the weight estimators. As expected, this improvement preserves the clustering accuracy, while reducing the resource consumption up to 80%. In the next improvement, the communication capacity is restricted. In this setting, the AC values decrease by approximately 2 percent, while the communication overhead experiences a major reduction. Further limitation of storage capacity in the last improvement, still keeps AC above 95%, but deallocates local resources.

The RandI values remain near 100% regardless of the applied improvements. The figure also shows that the algorithm is not sensitive to the number of internal data when evaluated in terms of accuracy.

Figure 13 shows performance of GDCluster when number of internal data of nodes varies from 200 to 1000, and compares its performance with a devised central approach with ideas from [2], [18], [19]. Initially, each node computes k representatives from local data and sends them to the central node. Then, iteratively, the central node aggregates the collected representatives by executing K-means. Each node executes one round of K-means with its local data and the central centroids, and sends back the new centroids to the central node. The algorithm terminates when the centroids remain approximately constant.

As observed, GDCluster can cope well with large data without loosing its clustering accuracy. The interesting point is that, with summarizing internal data before transmission in task DERIVE (as discussed in section 3.1.1), the communication overhead can be kept low and independent of size of internal data (with same number of clusters). The central approach, has very low communication overhead because of only transmitting cluster centroids. However, its accuracy can not surpass 80% and its extension to dynamic data requires re-execution of the algorithm. The larger communication overhead of GDCluster is mainly due to the weight estimators. Nevertheless, these estimators empower the algorithm to preserve its performance even when nodes have repetitive data. In algorithms such as K-means which involve several rounds of mean computation, repetitive data can bias the results and negatively affect the clustering results. This is the fact observed in figure 13 for the central approach. GDCluster however, can resist such situations by accurately computing the weight of centroids after each calculation of the mean value.

Alternatively, we can consider a central approach which collects all data from the network and centrally executes the K-means algorithm. Despite the processing overhead of executing K-means on large data sets, here we only concentrate on communication costs. Assuming links with 1KB per second capacity and 1000 data points per each node, the execution time of GDCluster before converging to more than 95% accuracy, is approximately 60 seconds, while for the central approach it is about 2000 seconds.

TABLE 2
Performance differences when N varies with respect to $N = 1024$

	Network size	1024 (baseline)	2048-16384
RA	AC (%)	93.85	<0.11
	corrected RandI (%)	100	<0.0011
	communication (KB)	25.44	<0.67
	storage (KB)	13.4	<0.028
CA	AC (%)	93.77	<0.019
	corrected RandI (%)	100	0
	communication (KB))	25.45	<0.31
	storage (KB)	13.33	<0.12

TABLE 3
Average values of evaluation metrics for 50 runs of the algorithm with real data sets

	AC		RandI		Comm. overhead
Data set	Central Kmeans	GDCluster	Central Kmeans	GDCluster	GDCluster
Shuttle	0.84	0.86	0.75	0.80	16.8
MAGIC	0.68	0.67	0.56	0.56	17.17
Pendigits	0.63	0.53	0.80	0.88	20.39

Fig. 14 shows the behaviour of GDCluster when the network size varies from 1024 to 16384 nodes. The AC values have converged to more than 90%. This shows the efficiency and scalability of the algorithm. In the random data-assignment strategy, AC values are initially higher. This is due to each node initially having internal data items from different clusters, enabling it to identify more clusters. As the performance of the algorithm for different network sizes is very similar, we used the average values of different metrics for $N = 1024$ as a baseline in table 2, and showed the difference of values for other network sizes. The RandI values converge to 100%. The communication and storage overheads of the algorithm remain constant due to restricting resource consumption. As observed, the differences of values for different network sizes are small, showing scalability of the algorithm.

In the evaluation of the algorithm using real data sets, both central K-means and GDCluster are evaluated against the actual labels of data, and the results are presented in Table 3. GDCluster is executed in a network of 1024 nodes, each having 10 data items. The AC and RandI values for GDCluster are very close to those of the central K-means. Because GDCluster executes K-means on the representatives instead of data, when compared to actual data labels, its accuracy may even surpass the central results for some data sets. The results show the efficiency of the algorithm in conforming to central clustering for real-world data.

Dynamic settings

The *MaxAge* parameter puts an upper bound on the storage period of external data items, and representatives. Fig. 15 shows the evaluation of the basic GDCluster algorithm when *MaxAge* varies. Very low values of *MaxAge* prohibit complete propagation of information in the network, and also cause early removal of data and representatives. Large values, on the other hand, maintain invalid information longer than required and degrade accuracy. The

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TKDE.2015.2391123

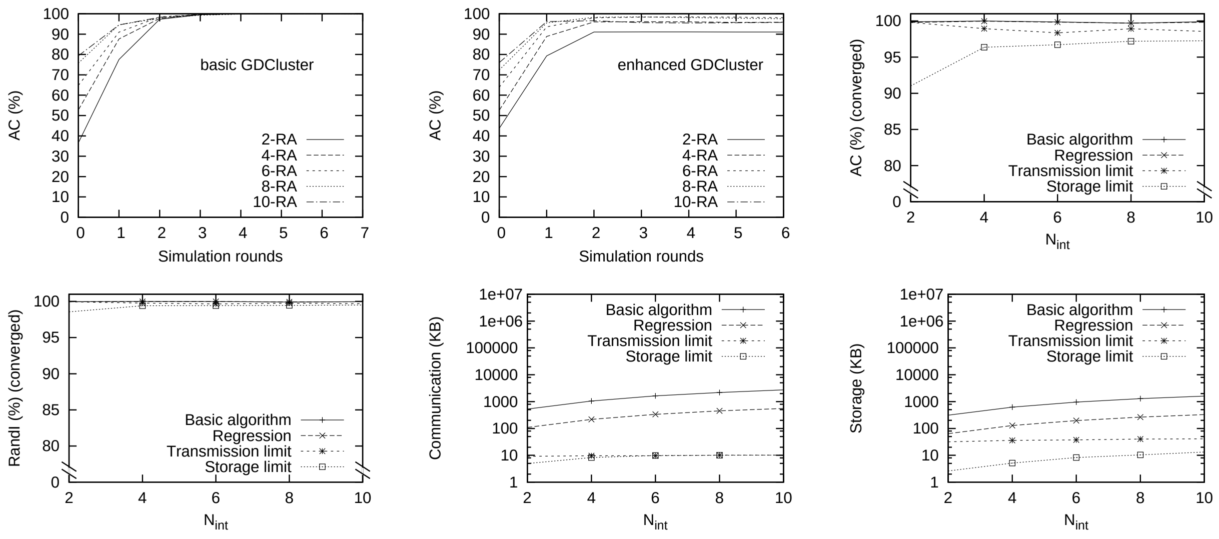


Fig. 12. Convergence and cost evaluation in static settings when N_{int} varies. Comparing incremental configurations: basic; regression; reduced communication; reduced storage (enhanced GDCluster).

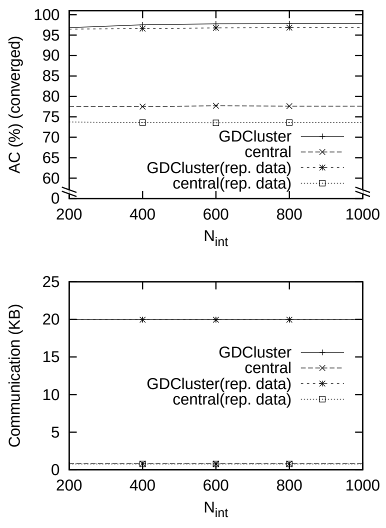


Fig. 13. Evaluation of GDCluster in static settings when N_{int} varies, and comparison to a central approach. Nodes either have unique or repetitive (rep.) data.

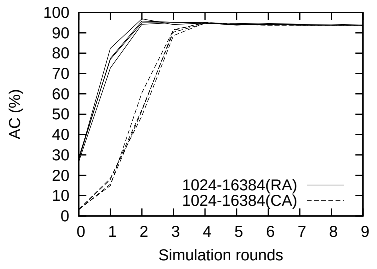


Fig. 14. Convergence in a static setting, when N varies (average values in table 2)

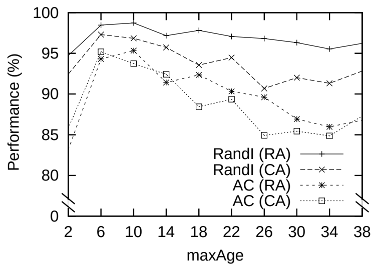


Fig. 15. Effect of changing *MaxAge*

optimum behaviour of the algorithm is observed when *MaxAge* is equal to 6. This is consistent with the earlier observation of quick convergence of the algorithm. Therefore, *MaxAge* should be chosen to be compatible with algorithm convergence rate, as to remove the data at a reasonable pace.

Fig. 16 shows the evaluation of the algorithm against different metrics in a dynamic setting, with 10% churn. With the CA strategy, concept drift is observed as some clusters are introduced later to the network.

As illustrated in Fig. 16, for all network sizes, the AC value rises to approximate average values of 94% and 93% with the RA and CA strategies, respectively. Although data changes regularly, the RA strategy ensures that previously discovered clusters remain valid through data change. This ensures higher AC values. With concept drift, nodes should move on to discover representatives in the new clusters. It also takes some time for the removed data to be discarded by the embedding representatives.

Similar trends are observed for the RandI metric, where approximate average values of 98% and 96% are achieved for RA and CA strategies, respectively. The algorithm has acceptable performance in detecting clusters, even in dynamic settings. Finally, the same figure shows that the communication overhead for different network sizes remains roughly the same. This is mainly due to removal of representatives in the dynamic setting which reduces the amount of transferred data between nodes.

High churn rates may affect distributed clustering performance, due to delay in propagation of information. With significantly high churn rates, some new data items may be removed even before all nodes get the chance to update their cluster model,

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TKDE.2015.2391123

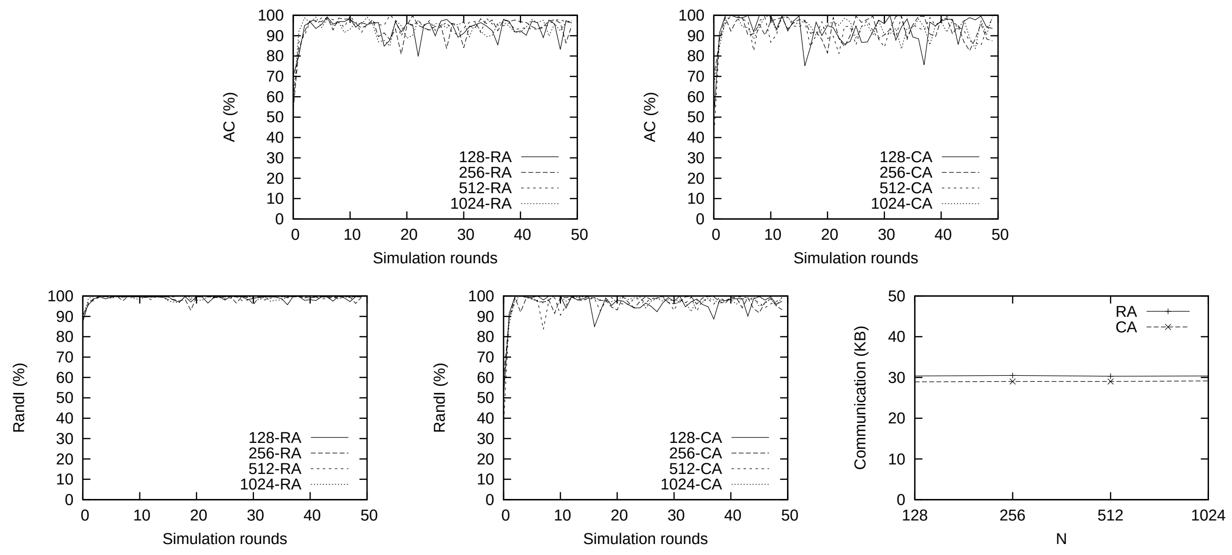


Fig. 16. Evaluation of GDCluster in dynamic setting, when N varies.

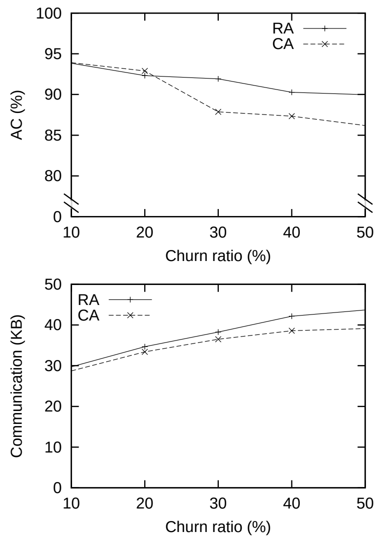


Fig. 17. Effects of varying churn ratio

or invalid clusters can persist longer. Evaluation of the algorithm when churn ratio varies, is presented in Fig. 17, where churn ratio changes from 10% to 50%. Note that along with increasing churn rates, larger data sets are used in the simulations.

Recalling that the algorithm requires a few gossip rounds to spread cluster information in the system, it is seen that the cluster accuracy does not degrade much with high churn rates. In the RA strategy, the clusters do not change significantly with data changes. With the CA strategy, on the other hand, concept drift is present and new clusters emerge as the old ones fade out. This explains the higher decreases in the AC values of the CA strategy.

The communication overhead has an increasing trend in both strategies. With higher churn rates, while the removing rate of external data and representatives from nodes is dependent on *MaxAge*, the addition of internal data speeds up. This causes more information transmission between nodes.

Comparison with LSP2P

The LSP2P algorithm [4] executes the K-means in an iterative manner, with each node synchronizing with its neighbors during each iteration. In a static setting, the algorithm is initiated at a single node p, which picks a set of random initial centroids along with a termination threshold $\gamma > 0$ (which we explain shortly). p sends these to all its immediate neighbors, and begins iteration 1. When a node receives the initial centroids and threshold for the first time, it forwards them to its remaining neighbors and initiates iteration 1. In each iteration, every node p executes one round of K-means on its local data based on the centroids computed in the previous iteration. It then prompts its immediate neighbors for their corresponding cluster centroids, and updates local centroids based on the received information. Once the computed centroids of two consecutive iterations, deviate less than γ from each other, p enters the terminated state. In a dynamic setting, the change of data may reactivate the nodes.

Regarding the above descriptions on LSP2P, it is observed that the initial centroids are identical in all nodes, which prohibits changing the number of produced clusters. Also, if K-means is to be executed with different initial centroids, a new instance of LSP2P should be started. Moreover, the history of executing the K-means algorithm is particularly important, and maintained in each node.

Our algorithm, adopting a different design and communication scheme, overcomes all above limitations. However, for a fair comparison with LSP2P, a small modification is applied to GDCluster in which the K-means initial centroids are available to all nodes. These initial centroids are used when summarizing data and also in the final clustering. The storage capacity of GDCluster is set to approximately the required memory of nodes in LSP2P. To find this value, LSP2P is executed several times, with the observation that global termination occurs at a minimum of 30 rounds. After this state, no more memory is consumed in a static setting; So, the memory threshold for our algorithm is set to the memory consumed in 30 execution rounds of LSP2P.

Fig. 18 shows a comparison of our algorithm with LSP2P in a static setting, against different evaluation metrics. Our algorithm achieves higher AC and RandI values. This is valid in all network

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TKDE.2015.2391123

sizes, and is due to establishing an accurate view on the whole data set.

The communication costs of our algorithm are higher than LSP2P in static settings (and much lower in dynamic settings as later seen in figure 19). As discussed earlier, GDCluster overcomes limitations of LSP2P and also offers a general solution, such that each node can autonomously execute the K-means algorithm (as well as other classes of clustering), to obtain the desired number of clusters. This generality demands more information exchange so that nodes have a sufficiently accurate view on the data set. If both algorithms take same number of rounds to converge (which is typically the observed behavior in the simulations), with a same data rate, GDCluster will require more time (in seconds) to transmit all required data in a static setting.

Fig. 19 compares the two algorithms in a dynamic setting. Again, GDCluster outperforms LSP2P. The design of LSP2P prohibits adaptation of the algorithm to discover different number of clusters on the fly. Our algorithm, can handle churn, while at the same time, being able of discovering arbitrary number of clusters. The communication overhead of both algorithms increases when churn is in place. In LSP2P, this is due to more message passing for handling churn. In our algorithm this is due to more data communicated in the algorithm tasks. However, in dynamic setting, LSP2P has a higher communication overhead.

Density-based clustering

For density-based clustering to be accurate, the representatives should provide sufficient coverage of all parts of a cluster, and a finer granularity to be able to distinguish clusters. This leads to a larger number of representatives stored at each node. As the algorithm is very similar to the basic version of the distributed density-based clustering method proposed in [14], here we only offer a limited evaluation with 1024 nodes.

Fig. 20 (a) shows a synthetic data set generated with the data generator tool introduced in [20]. We also use the points data set from the SEQUOIA 2000 benchmark [21]. This data set contains 62584 names of landmarks in California, extracted from the US Geological Surveys Geographic Names Information System, together with their location. Regarding the number of data items required in the experiments, a random sample of this data set is used. For both data sets, the *minpts* values in central DBSCAN and GDCluster are identical. To compensate for the approximation involved in GDCluster due to the limited storage and communication, the ε value used in final clustering of GDCluster is set to 8 times the ε value of central DBSCAN.

Fig. 20 (b) shows the result of running the algorithm with 1024 nodes, under the RA strategy. As observed, the RandI metric converges towards approximate values of 97% and 99% for the synthetic and SEQUOIA data sets in less than 15 rounds. The average communication overhead per each node for the setting of Fig. 20 is 200 KB for both synthetic and SEQUOIA data sets. Note that each node ends up having detected all clusters in the network. To reduce the communication costs, nodes can be limited to discover only interested clusters, or only detect representatives around their local data, and leave the final clustering task to some crawler which visits all nodes to discover actual clusters.

7 Related work

Distributed Data Mining (DDM) is a dynamically growing area. A discussion and comparison of several distributed centroid based partitional clustering algorithms is provided in [22]. Reference [18] propose parallel K-means clustering, by first distributing data to multiple processors. In each synchronized algorithm round, every processor broadcasts its currently obtained centroids, and updates the centroids based on the information received from all other processors.

Different from many existing distributed clustering algorithms, our algorithm does not require a central site to coordinate execution rounds, and/or merge local models. Also, it avoids global message flooding. RACHET [23] is a hierarchical clustering algorithm in which, each site executes the clustering algorithm locally, and transmits a set of statistics to a central site. A distributed partition-based clustering algorithm for clustering documents in a peer-to-peer network is proposed by Eisenhardt et al. [24]. The algorithm requires rounds of information collection from all peers in the network. A K-means monitoring algorithm is proposed in [25]. This algorithm executes K-means by iteratively combining data samples at a central cite, and monitoring the deviation of centroids in a distributed manner.

A method of combining local k-window clustering models in a central site is proposed in [26]. A partition-based clustering algorithm for clustering distributed high-dimensional feature vectors is presented in [27], which uses a central site to build the global model. SDBDC [2] is a distributed density-based clustering algorithm that summarizes local statistics, and transmits them to a central site to be merged. Aouad et al. [28] propose a lightweight distributed clustering technique based on merging of independent local sub clusters according to an increasing variance constraint. Merugu et al. [29] propose a distributed clustering algorithm, in which each node computes a probabilistic clustering model and a central node attempts to aggregate the local models in to reduce an approximate cost function.

Some distributed clustering proposals impose a special structure in the network. A hierarchical clustering method based on K-means for P2P networks is suggested in [19]. Summary representations are then transferred up the hierarchy and merged to obtain k global clusters. Lodi et al. [3] introduce a distributed density-based clustering which again uses a semantic overlay as the infrastructure. Embeddings of kernel clustering on the MapReduce framework is proposed in [30].

Some solutions which consider pure unstructured networks, require state-aware operation of nodes, work in static settings, or are aimed at computing basic functions like average and sum. Fellus et al. [7] propose a decentralized K-means algorithm which executes in iterations, and in each iteration nodes compute an approximation of the new centroids in a distributed manner. Datta et al. [4] propose a distributed K-means clustering algorithm for P2P networks in which nodes communicate with their immediate neighbours. Each node is required to store history of cluster centroids per each K-mean iteration. Elgohary et al. [5] propose a similar algorithm, with different local computation of centroids. Eyal et al. [31] provide a generic algorithm for clustering in a static network. Fatta et al. [32] propose a gossip-based distributed k-means clustering, which is initiated with similar initial centroids, and proceeds towards centroid convergence with rounds of gossiping. Shen et al. [33] propose a distributed clustering in a static network, incorporating information theory measures.

The major drawback of the majority of existing approaches, is lack of efficient solutions for adaptability in dynamic settings, which introduces significant challenges for applying the algorithms in large-scale real-world networks. Also, majority of

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TKDE.2015.2391123

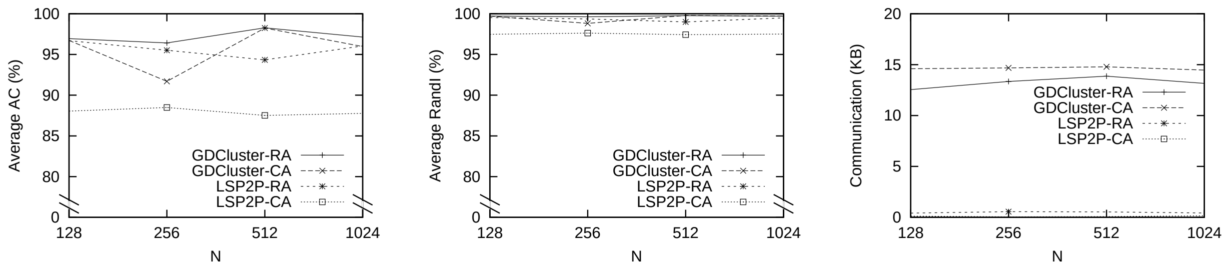


Fig. 18. Comparing algorithms in static settings with the RA strategy

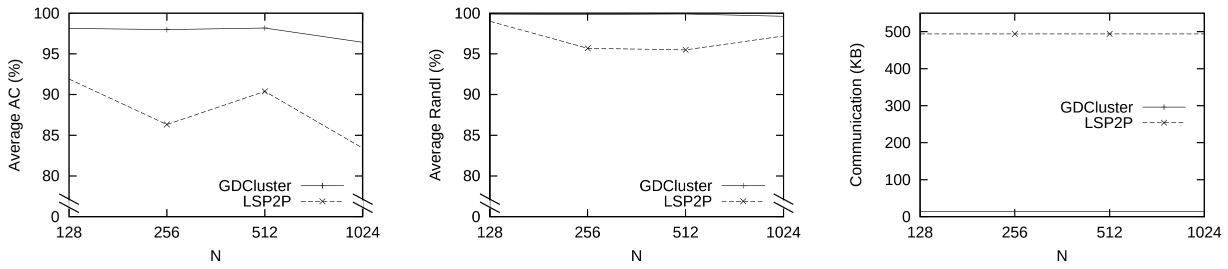
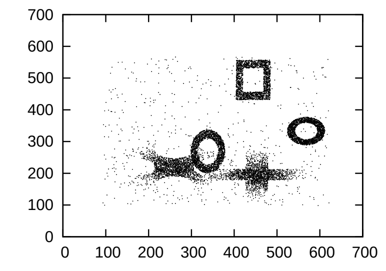
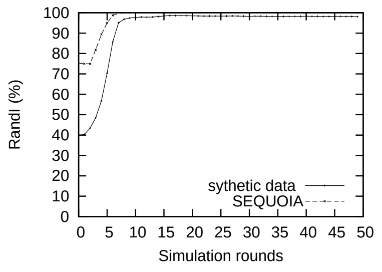


Fig. 19. Comparing algorithms in dynamic settings with RA strategy and churn ratio=10%



(a) Data set



(b) RandI for $N = 1024$

Fig. 20. Evaluation of GDCluster for density-based clustering

approaches limit nodes to finding the same number of clusters.

8 Conclusions

In this paper we first identified the necessity of an effective and efficient distributed clustering algorithm. Dynamic nature of data demands a continuously running algorithm which can update the clustering model efficiently, and at a reasonable pace. We introduced GDCluster, a general fully decentralized clustering algorithm, and instantiated it for partition-based and density-based clustering methods. The proposed algorithm enabled nodes to gradually build a summarized view on the global data set, and execute weighted clustering algorithms to build the clustering models. Adaptability to dynamics of the data set was made possible by introducing an age factor which assisted in detecting data set changes updating the clustering model. Our experimental evaluation and comparison showed that the algorithm allows effective clustering with efficient transmission costs, while being scalable and efficient.

GDCluster can be customized for other clustering types, such as hierarchical or grid-based clustering. To accomplish this, representatives can be organized into a hierarchy, or carry statistics of approximate grid cells. Further discussion of these algorithms is deferred to future work.

References

[1] K. M. Hammouda and M. S. Kamel, "Models of distributed data clustering in peer-to-peer environments," *Knowledge and Information Systems*, pp. 1–27, 2012.

[2] E. Januzaj, H.-P. Kriegel, and M. Pfeifle, "Scalable Density-Based Distributed Clustering," in *8th European Conference on Principles and Practice of Knowledge Discovery in Databases*. Berlin: Springer-Verlag, 2004, pp. 231–244.

[3] S. Lodi, G. Moro, and C. Sartori, "Distributed Data Clustering in Multi-Dimensional Peer-to-Peer Networks," in *21st Australasian Conference on Database Technologies*, vol. 104, 2010, pp. 171–178.

[4] S. Datta, C. R. Giannella, and H. Kargupta, "Approximate distributed k-means clustering over a peer-to-peer network," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 10, pp. 1372–1388, 2009.

[5] A. Elgohary and M. A. Ismail, "Efficient data clustering over peer-to-peer networks," in *11th International Conference on Intelligent Systems Design and Applications (ISDA)*. IEEE, 2011, pp. 208–212.

[6] G. Di Fatta, F. Blasa, S. Cafiero, and G. Fortino, "Epidemic k-means clustering," in *International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2011, pp. 151–158.

[7] J. Fellus, D. Picard, and P.-H. Gosselin, "Decentralized k-means using randomized gossip protocols for clustering large datasets," in *Data Mining Workshops (ICDMW)*. IEEE, 2013, pp. 599–606.

[8] D. Kempe, a. Dobra, and J. Gehrke, "Gossip-based computation of aggregate information," *44th Symposium on Foundations of Computer Science*, pp. 482–491, 2003.

[9] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based Peer Sampling," *ACM Transactions on Computer Systems*, vol. 25, no. 3, Aug. 2007.

[10] A. M. Frieze and G. R. Grimmett, "The shortest-path problem for graphs with random arc-lengths," *Discrete Applied Mathematics*, vol. 10, no. 1, pp. 57–77, 1985.

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TKDE.2015.2391123

[11] D. Mosk-Aoyama and D. Shah, “Computing separable functions via gossip,” in *25th ACM symposium on Principles of distributed computing*. ACM, 2006, pp. 113–122.

[12] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. California, USA, 1967, pp. 281–297.

[13] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” in *2nd International Conference Knowledge Discovery and Data Mining*. New York, NY: ACM Press, 1996, pp. 226–231.

[14] H. Mashayekhi, J. Habibi, S. Voulgaris, and M. van Steen, “Goscan: Decentralized scalable data clustering,” *Computing*, vol. 95, no. 9, pp. 759–784, 2013.

[15] L. Lovász and M. Plummer, “Matching theory, vol. 367,” 2009.

[16] W. Xu, X. Liu, and Y. Gong, “Document clustering based on non-negative matrix factorization,” in *26th International ACM SIGIR conference*. ACM, 2003, pp. 267–273.

[17] L. Hubert and P. Arabie, “Comparing partitions,” *Journal of classification*, vol. 2, no. 1, pp. 193–218, 1985.

[18] I. S. Dhillon and D. S. Modha, “A Data-Clustering Algorithm on Distributed Memory Multiprocessors,” in *Large-Scale Parallel Data Mining*, ser. Lecture Notes in Computer Science, vol. 1759. Berlin: Springer-Verlag, 2000, pp. 245–260.

[19] K. M. Hammouda and M. S. Kamel, “Hierarchically Distributed Peer-to-Peer Document Clustering and Cluster Summarization,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, pp. 681–698, May 2009.

[20] Y. Pei and O. Zaïane, “A synthetic data generator for clustering and outlier analysis,” *Department of Computing science, University of Alberta, edmonton, AB, Canada*, 2006.

[21] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith, “The sequoia 2000 storage benchmark,” in *ACM SIGMOD Record*, vol. 22, no. 2. ACM, 1993, pp. 2–11.

[22] N. Visalakshi and K. Thangavel, “Distributed Data Clustering: A Comparative Analysis,” in *Foundations of Computational Intelligence*, A. Abraham, A.-E. Hassanien, A. de Leon F. de Carvalho, and V. Snasel, Eds. Berlin: Springer-Verlag, 2009, vol. 206, pp. 371–397.

[23] N. F. Samatova, G. Ostrouchov, A. Geist, and A. V. Melechko, “RACHET: An Efficient Cover-Based Merging of Clustering Hierarchies from Distributed Datasets,” *Distributed Parallel Databases*, vol. 11, pp. 157–180, Mar. 2002.

[24] M. Eisenhardt, W. Muller, and A. Henrich, “Classifying Documents by Distributed P2P Clustering,” in *Informatik*, Sep. 2003, pp. 286–291.

[25] R. Wolff, K. Bhaduri, and H. Kargupta, “A generic local algorithm for mining data streams in large distributed systems,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 4, pp. 465–478, 2009.

[26] D. K. Tasoulis and M. N. Vrahatis, “Unsupervised distributed clustering,” in *Parallel and Distributed Computing and Networks*, 2004, pp. 347–351.

[27] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz, “Approximated clustering of distributed high-dimensional data,” in *Advances in Knowledge Discovery and Data Mining*. Springer, 2005, pp. 432–441.

[28] L. M. Aouad, N.-A. Le-Khac, and T. M. Kechadi, “Lightweight Clustering Technique for Distributed Data Mining Applications,” in *7th International Conference on Data Mining*. Berlin: Springer-Verlag, 2007, pp. 120–134.

[29] S. Merugu and J. Ghosh, “A privacy-sensitive approach to distributed clustering,” *Pattern Recognition Letters*, vol. 26, no. 4, pp. 399–410, 2005.

[30] A. Elgohary, “Scalable embeddings for kernel clustering on mapreduce,” *M.Sc. Thesis, University of Waterloo*, 2014.

[31] I. Eyal, I. Keidar, and R. Rom, “Distributed data clustering in sensor networks,” *Distributed Computing*, vol. 24, no. 5, pp. 207–222, 2011.

[32] G. Di Fatta, F. Blasa, S. Cafiero, and G. Fortino, “Fault tolerant decentralised k-means clustering for asynchronous large-scale networks,” *Journal of Parallel and Distributed Computing*, vol. 73, no. 3, pp. 317–329, 2013.

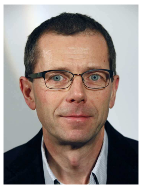
[33] P. Shen and C. Li, “Distributed information theoretic clustering,” *SIGNAL PROCESSING, IEEE Transactions on*, vol. 62, no. 13, pp. 3442–3453, 2014.

Hoda Mashayekhi is an assistant professor at the Department of Computer Engineering of the Shahrood university. She received her Ph.D. from Sharif University of Technology in 2013. Her research interests include parallel and distributed computing, data mining, decision making, peer-to-peer (P2P) networks and semantic structures.

Jafar Habibi is a faculty member at the computer engineering department at Sharif University of Technology and the managing director of Electronic Computing Machines Services. He is supervisor of Sharif’s Robo-Cup Simulation Group. His research interests are mainly in the areas of computer engineering, simulation systems, MIS, DSS and evaluation of computer systems performance.

Tania Khalafbeigi is a PhD student in deprartment of Geomatic engineering at University of Calgary. She received her BSc and MSc in computer engineering from Sharif University of Technology and her major was software systems. Her research focuses on big data analytics in Internet of Things. She also worked on distributed data mining and P2P clustering during her MSc studies.

Spyros Voulgaris is an assistant professor at the Department of Computer Science of the VU University in Amsterdam. His main research interests include Large-Scale Distributed Systems, Peer-to-Peer Algorithms, Mobile Ad-Hoc Networks, Sensor Networks, Self-Organization and Epidemic Algorithms. Prior to his current position, Spyros Voulgaris served as a senior researcher at ETH Zurich, from 2006 to 2008.



Maarten van Steen is full professor in the Computer Systems Group, Vrije Universiteit Amsterdam. He teaches modules and courses covering distributed systems, computer networks, operating systems, and complex networks to academics and professionals. He has coauthored two textbooks on networked computer systems. His research is focused on large-scale distributed systems with a strong emphasis on adaptive techniques that support automatic replication, management, and organization of wired and wireless systems.

Copyright (c) 2015 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.