Scalable Clustering by Iterative Partitioning and Point Attractor Representation

JUNMING SHAO and QINLI YANG, University of Electronic Science and Technology of China
HOANG-VU DANG, BERTIL SCHMIDT, and STEFAN KRAMER,
Johannes Gutenberg Universität Mainz

Clustering very large datasets while preserving cluster quality remains a challenging data-mining task to date. In this paper, we propose an effective scalable clustering algorithm for large datasets that builds upon the concept of synchronization. Inherited from the powerful concept of synchronization, the proposed algorithm, CIPA (Clustering by Iterative Partitioning and Point Attractor Representations), is capable of handling very large datasets by iteratively partitioning them into thousands of subsets and clustering each subset separately. Using dynamic clustering by synchronization, each subset is then represented by a set of point attractors and outliers. Finally, CIPA identifies the cluster structure of the original dataset by clustering the newly generated dataset consisting of points attractors and outliers from all subsets. We demonstrate that our new scalable clustering approach has several attractive benefits: (a) CIPA faithfully captures the cluster structure of the original data by performing clustering on each separate data iteratively instead of using any sampling or statistical summarization technique. (b) It allows clustering very large datasets efficiently with high cluster quality. (c) CIPA is parallelizable and also suitable for distributed data. Extensive experiments demonstrate the effectiveness and efficiency of our approach.

Categories and Subject Descriptors: H.3.3 [**Information Search and Retrieval**]: Clustering; H.2.8 [**Database application**]: Data mining

General Terms: Design, Algorithms, Performance

Additional Key Words and Phrases: High-performance algorithm, scalable clustering, synchronization

ACM Reference Format:
Junming Shao, Qinli Yang, Hoang-Vu Dang, Bertil Schmidt, and Stefan Kramer. 2016. Scalable clustering by iterative partitioning and point attractor representation. ACM Trans. Knowl. Discov. Data 11, 1, Article 5 (July 2016), 23 pages.
DOI: http://dx.doi.org/10.1145/2934688

1. INTRODUCTION

Real-world datasets in today's data generating and collecting world often encompass millions or billions of objects. Clustering such large data becomes difficult for most of

This work is supported by the National Natural Science Foundation of China (61403062, 61433014), China Postdoctoral Science Foundation (2014M552344, 2015M580786), Science-Technology Foundation for Young Scientist of SiChuan Province (2016JQ0007), and Fundamental Research Funds for the Central Universities (ZYGX2014J053).
Authors' addresses: J. Shao and Q. Yang, Big Data Research Center, University of Electronic Science and Technology of China, No. 2006, Xiyuan Ave, West Hi-Tech Zone, 611731, Chengdu, China; emails: junmshao@uestc.edu.cn, qinli.yang@uestc.edu.cn; B. Schmidt and S. Kramer, Institute of Computer Science, Johannes Gutenberg Universität Mainz, Staudingerweg 9, 55128, Mainz, Germany; emails: bertil.schmidt@uni-mainz.de, kramerst@uni-mainz.de; H. Dang, Department of Computer Science, University of Illinois, Urbana, IL 61801, USA; email: hdang8@illinois.edu.

DOI: http://dx.doi.org/10.1145/2934688

the established clustering algorithms due to the problem of unacceptable computation time. To handle this problem, many scalable clustering algorithms have been proposed. Currently, there are two main strategies for clustering very large data: data sampling [Kaufman and Rousseeuw 2009; Guha et al. 1998; Milenova and Campos 2002; Havens et al. 2012] and data summarization [Zhang et al. 1996; Breunig et al. 2001; Andritsos et al. 2004]. Sampling-based approaches try to reduce the original large dataset by drawing a sample using different sampling techniques. Data summarization related methods try to condense the original data into a more compact summarized data representation by using different compression techniques, such as Cluster Features (CF) [Zhang et al. 1996], and Data Bubbles [Breunig et al. 2001]. However, both types of scalable clustering algorithms determine the cluster structure only on sampled or summarized data, and thus the quality of the clustering is difficult to be preserved.

Recently, inspired by natural synchronization phenomena, many synchronization-based clustering algorithms [Kim et al. 2008; Böhm et al. 2010; Shao et al. 2011] have been introduced and have demonstrated attractive properties compared to many existing clustering algorithms. However, like most existing clustering algorithms, synchronization-based clustering algorithms are computationally intensive and do not scale well with the number of observations.

In this paper, we propose a new scalable clustering algorithm, CIPA (Clustering by Iterative Partitioning and Point Attractor Representations), generalizing the idea of synchronization-based clustering algorithms (like Sync [Böhm et al. 2010]) to scale up to large volumes of data. The contributions of the paper are as follows:

(1) *Simple, yet intuitive and effective approach*. While adapting synchronization-based clustering in this way may appear as only a small step, it gives the overall approach a new quality and in fact overcomes its scalability limitations. Unlike traditional algorithms using the *divide-and-conquer* strategy, the crucial point is that the concept of synchronization with its focus on local neighborhoods of data points lends itself to a *random partitioning* of larger datasets into smaller datasets. To the best of our knowledge, it is the first clustering algorithm that allows handling large dataset naturally by random data partitioning. The type of partitioning required by other, related algorithms, is based on geometry rather than random sampling. It is evident that a suitable geometric partitioning of an input space is (a) not straightforward to define and (b) most likely computationally costly.
(2) *Scalability*. CIPA allows clustering on a specific representation of the original dataset, a so-called *point attractor representation*, rather than any sampled or summarized data. Thanks to the synchronization-based point attractor (PA) representation, CIPA allows partitioning data into as many subsets as necessary or possible in a given application in a natural fashion. CIPA lends itself to parallelization and also can work on distributed data. In the results section, we show that CIPA is able to cluster one hundred million data points in only a few minutes.
(3) *High-quality clustering.* Unlike other sampling or summarization-based scalable clustering algorithms, the PA representation allows maintaining the cluster structure of the original dataset. Therefore, CIPA not only allows clustering large datasets efficiently in a *divide-and-conquer* fashion, but more importantly, the properties of synchronization-based clustering are inherited and cluster quality is thus preserved. As it turns out in the results section, CIPA is not only more accurate than the well-known scalable clustering algorithms CURE, BIRCH, and reckFCM on smaller, labeled data, it also outperforms these algorithms on larger real-world data.

The remainder of this paper is organized as follows: In the following section, we briefly survey related work. Section 3 presents our algorithm in detail and we give its parallel implementation in Section 4. Section 6 contains an extensive experimental evaluation. Finally, we give a short discussion and conclude in Section 7.

2. RELATED WORK

During the past several decades, many algorithms [Kanungo et al. 2002; Bahmani et al. 2012; Orlandic et al. 2005; Guha et al. 1998; Zhang et al. 1996; Breunig et al. 2001; Kwon et al. 2010; Banerjee and Ghosh 2006; Ying et al. 2014; Havens et al. 2012] have been proposed for clustering large datasets, which can largely be divided into two categories. The first solution is to speed-up the clustering process based on some space-partitioning data structures such as the KD-tree [Kanungo et al. 2002] and space reduction [Orlandic et al. 2005] or by reducing the number of iterations [Bahmani et al. 2012]. Another main stream is to propose new scalable clustering algorithms. Here, we only provide a brief survey of scalable clustering algorithms related to our work.

Data sampling. Data sampling is probably the most widely used method to reduce a large dataset to speed-up clustering algorithms to handle large datasets. The basic idea of data sampling-related scalable clustering methods is to randomly draw a smaller sample using different sampling strategies and then apply the clustering algorithm to the small subset instead of the whole dataset. If the sample size is large enough, this type of algorithm is usually expected to detect the same cluster structure as in the original data. For instance, CLARA [Kaufman and Rousseeuw 2009] draws multiple samples from the dataset and then applies PAM (Partitioning Around Medoids) on samples and finally selects the optimal clustering results. CURE [Guha et al. 1998] employs a hierarchical clustering approach that uses a number of representative points to define a cluster instead of a single centroid and thus allows discovering clusters of complex shapes and different sizes. To handle large datasets, CURE combines the techniques of random sampling and partitioning. For partitioning, it is successful only if there is a certain amount of objects from each cluster for every partition. Banerjee and Ghosh [2006] proposed several scalable clustering algorithms by considering balancing constraints. Like CURE, these algorithms first sample a small representative subset, then cluster on the sampled data, and finally populate and refine the clusters. Recently, Havens et al. [2012] compare the efficacy of three different implementations of techniques aimed to extend fuzzy c-means (FCM) clustering to very large data. The random sample and extend kernel FCM (rsekFCM) follows an idea similar to the one of CURE and CLARA, the initial sample is first chosen and then clustered using weighted kernel FCM (wkFCM). The cluster prototypes yielded by wkFCM are then used to extend the partition to the entire data. As the kernel function is applied, the algorithm allows finding non-linear clusters in principle. However, this also leads to a high space complexity. For all these sampling-based clustering algorithms, the quality of clustering heavily depends on the sample size and the effectiveness of the sampling techniques.

Data summarization. This type of scalable clustering first compresses the original dataset by some summarization, and then performs clustering on the summarized data. BIRCH [Zhang et al. 1996] is a typical data summarization-based scalable clustering algorithm. The basic idea of this algorithm is to use a balanced data structure called CF-Tree (Cluster Feature tree) for storing statistical summary information of subclusters of objects, which attempts to preserve the cluster structure of the dataset. The hierarchical clustering is then performed on leaf nodes of the tree. Inheriting from k-centroid style clustering, BIRCH usually fails to detect non-spherical clusters. In Bradley et al. [1998], another compression technique for scaling up clustering

algorithms is proposed. The approach produces basically the same type of compressed data items as BIRCH, but separates data points into three types of sets. Based on the CF-Tree, Breunig et al. [2001] introduce the *Data Bubble*, a more specialized kind of compressed data representation, suitable for hierarchical clustering. In general, the performance of data summarization methods critically depends on whether the form of summarization (e.g., statistical information like the CF-tree or prototypes) captures the genuine cluster structure of the original data.

Hardware speed-up or parallel platforms. Besides abstractions of the original dataset, strategies to handle large datasets make use of Map-Reduce frameworks [Zhao et al. 2009] or modern parallel data processing systems [Kwon et al. 2010]. Zhao et al. [2009] have implemented the parallel K-Means clustering in the Map-Reduce framework. The core idea behind Map-Reduce is mapping your dataset into a collection of $< key, value >$ pairs, and then reducing overall pairs with the same key. Similar work can be found in the literature [Ludwig 2015; Kim et al. 2014; Fu et al. 2014]. Another main direction is to use GPUs to handle datasets [Cao et al. 2006; Böhm et al. 2009; Wasif and Narayanan 2011; Adinetz et al. 2013] or other parallel data processing system, such as DryadLINQ [Kwon et al. 2010]. In these approaches, established clustering algorithms are extended to scale up to large datasets by means of various parallel platforms and hardware.

Generally, current scalable clustering algorithms often work on sampled or summarized data. Therefore, the quality of the clustering may be compromised, if the form and bias of the summarization does not match the characteristics of a dataset. In this paper, we propose a new synchronization-based scalable clustering algorithm that clusters the original data building upon the so-called PA data representation and iterative data partitioning. As the partitions are randomly sampled, they do not have to be constructed specifically and therefore do not assume any specific properties of the data. Experimental results (see Section 6) show that synchronization-based clustering works particularly well together with random partitioning, giving accurate clusters at low running times.

3. SYNCHRONIZATION-BASED SCALABLE CLUSTERING

In this section, we present the CIPA algorithm for parallel data clustering.

3.1. Synchronization and Point Attractor Representations

Before we can give an overview of the approach, we need to explain the main underlying concepts of synchronization and PA representations.

Synchronization is a prevalent phenomenon in nature. It is known that synchronization is rooted in human life from the metabolic processes in our cells to the highest cognitive tasks we perform as a group of individuals [Arenas et al. 2008]. A paradigmatic example of a synchronization phenomenon in nature is the synchronous flashing of fireflies observed in South Asian forests [Acebron et al. 2005]. Recently, many synchronization-based models [Dirk Aeyels 2008; Böhm et al. 2010] and data-mining algorithms [Kim et al. 2008; Böhm et al. 2010; Shao et al. 2010, 2011; Hong et al. 2012; Huang et al. 2013; Shao et al. 2014] have been proposed and showed many desirable properties. The key idea of clustering approaches by synchronization (e.g., in *Sync* and *ORSC* [Shao et al. 2011]) is to view each data object as a phase oscillator, the feature vector of an object as its phase, and simulate the dynamical behaviors of the objects over time. By the interaction with similar objects, the phase of an object gradually aligns with its neighborhood, resulting in a non-linear object movement driven by the local cluster structure. Finally, the objects in a cluster are synchronized together and

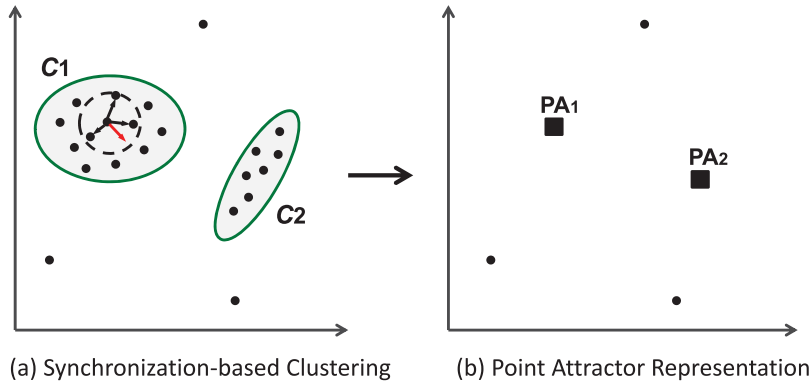


Fig. 1. Illustration of synchronization-based clustering and the point attractor representation.

have the same phase (feature vector). Figure 1 displays two snapshots of the simulated dynamical object movement for data synchronization. Specifically, there are two clusters (C1 and C2) and three outlier objects. With interaction among objects (Figure 1(a)), the objects in two clusters C1 and C2 are finally synchronized together, respectively (Figure 1(b)). In this paper, we define the synchronized phases as PAs. Formally, a PA is defined as follows.

Definition 1 (*A point attractor*). A PA is a point in the phase space, where a set of objects $S \in \mathcal{D}$ is finally synchronized together by dynamic clustering. It is defined to be triple: $PA = (p, w, ids)$, where p is the final synchronized phase (i.e., the feature vector resulting from a clustering) of the set of objects S, w is the number of objects in S, and ids is the unique IDs of these objects.

A PA characterizes the common phase for a set of objects after dynamic clustering and thereby not only enables to represent all objects in this set, but also envelops the local cluster structure of these objects (i.e., a small cluster). With this definition, finally, the whole dataset in Figure 1 is represented by two PAs and three outlier objects (Figure 1(b)). For the purpose of this paper, an outlier object is also a triple $O = (p, w, ids)$, where p is its original phase, $w = 1$ and ids is its unique ID. Unlike other data summarization techniques such as CF in BIRCH, which store the statistical information of all objects in a subcluster, the PAs really store the cluster structure (clustering results) of a dataset. As will be shown below, the combination of synchronization-based scalable clustering and random partitioning (cf. Section 3.3) exhibits many attractive properties in practice.

3.2. Intuition and Overview

We consider scalable clustering building on the PA-based data representation and a simple, yet effective strategy: *divide-and-conquer*. Inheriting from the powerful concept of synchronization, our method allows partitioning large data into thousands of subsets (*divide*) and clustering each subset separately (*conquer*). The clustering results of each subset are then represented by a set of PAs and some outlier objects, if existing. A new dataset is generated by collecting PAs and outliers from all subsets. If the size of this new dataset is still large, a new *divide-and-conquer* procedure is performed, until its size is small enough according to the user's needs. Finally, the cluster structure of the original data can be explored by clustering the final new dataset as it contains all clustering results from each level and corresponding data structure information. Figure 2 gives an overview of our scalable clustering by synchronization. The algorithm

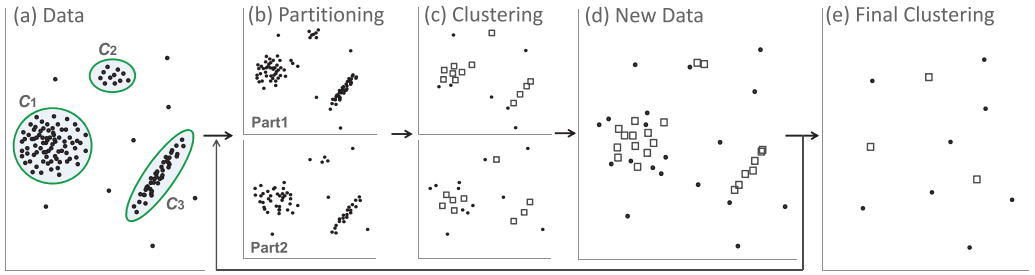


Fig. 2. Intuition of scalable clustering by synchronization.

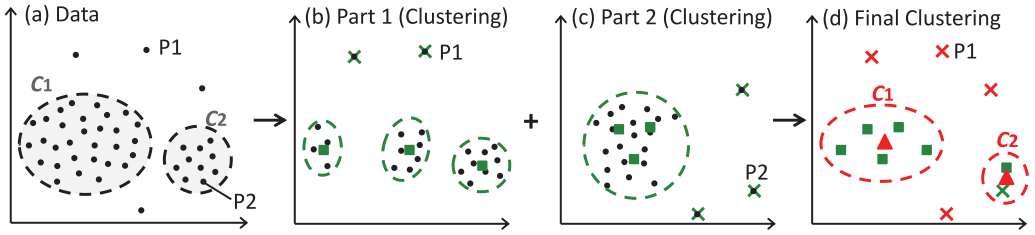


Fig. 3. Data partitioning for synchronization-based clustering.

starts with the data partitioning (Figure 2(b)). Here, only two partitions are used for better illustration. For each partition, we cluster separately using the synchronization-based dynamic clustering introduced in Section 3.4 (Figure 2(c)). After clustering, the clustering results in each partition are represented by PAs (square markers) and outlier objects. Subsequently, all PAs (square markers) and outlier objects from both partitions are collected together to form a new dataset (Figure 2(d)). As the new dataset is small enough, the final clustering is directly performed on it (Figure 2(e)) without further partitioning and clustering. In the following, we will elaborate on how to partition a dataset and why it works in conjunction with our synchronization-based scalable clustering algorithm.

3.3. Data Partitioning

Unlike other scalable clustering algorithms starting with drawing a sample or summarizing a dataset, our method handles large data by randomly partitioning it into thousands of subsets of fixed small size. However, the crucial question arises: Why does the partitioning technique work for our scalable clustering algorithm and what is the reason behind this?

For synchronization-based clustering algorithms, one salient feature is that cluster formation is driven by the *local data structure*, and all objects in a cluster dynamically *move together* (a PA). Therefore, with the PA representation, the local structure within each partition can be maintained. This means that although the size of the new dataset is smaller than the original dataset, the cluster structure is still maintained and represented by the PAs and outliers in the new dataset. In addition, synchronization-based clustering algorithms are robust to noise or outlier objects. Thus, global noise or outlier objects in the original data can be easily identified in each separate partition. To illustrate this partitioning for synchronization-based clustering, Figure 3 provides a simple example where the original data contains two clusters and four outlier objects. For instance, with random partitioning (Figure 3(b) and (c)), cluster C1 is split into three sub-clusters (two clusters in partition 1 and one cluster in partition 2), and

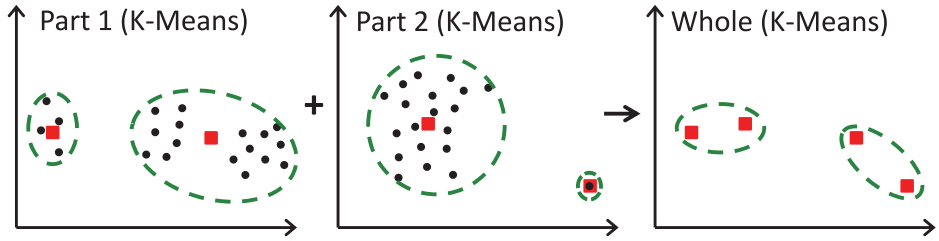


Fig. 4. K-Means clustering effect with the *divide-and-conquer* strategy.

cluster C2 is segmented as one cluster in partition 1 plus one object in partition 2. The four global outlier objects are scattered across both partitions. By dynamic clustering, which we will introduce in the following Section 3.4, partition 1 is represented by three PAs (green square markers) and two outlier objects (green cross markers). Similarly, partition 2 is replaced by three PAs and three outlier objects, where cluster object P2 in the original data is regarded as an outlier object temporarily. Finally, the new dataset has six PAs and five outlier objects (Figure 3(d)). It is intuitive to see that the cluster structure of the original dataset is maintained in the new dataset. Therefore, with the final clustering on the new dataset, we can obtain two clusters and four outlier objects, where object P2 in partition 2 becomes the cluster object in C2 finally, and the outlier objects (e.g., P1) are easily discovered.

Moreover, it is important to note that the *divide-and-conquer* strategy fits well to our synchronization-based clustering, in contrast to many other existing cluster algorithms, such as K-Means, DBSCAN, and the like. Let us take the typical K-Means algorithm for example. As K-Means is sensitive to outlier objects, to exclude the outlier effect, we remove four outlier objects before data partitioning. With the same partitioning on the two clusters in the original data (Figure 4), we perform the K-Means algorithm on each partition. However, it does not work for partition 1, as the objects in C1 and the objects in C2 in the original dataset are merged into one cluster. The reason is that K-Means splits the whole data space into distinct *Voronoi cells*, and thus the local cluster structure of the original data cannot be maintained in a partition. Finally, we cannot find the intrinsic cluster structure by re-clustering all centroids from the K-Means clustering results from the two partitions (Figure 4).

3.4. Dynamic Clustering by Synchronization

In this section, we generalize the synchronization-based notion [Böhm et al. 2010; Shao et al. 2011] to scalable clustering on large datasets. Typically, a synchronization-based clustering algorithm needs three definitions to simulate a dynamic clustering process: first, a parameter ϵ specifying the interaction range among objects, and second, the interaction model for clustering, and finally, a stopping criterion to terminate dynamic clustering. Our approach follows the definitions of synchronization-based clustering underlying the algorithm Sync [Böhm et al. 2010]. In the following, we give a short summary of all necessary definitions.

Definition 2 (*Range neighborhood*). Given a $\epsilon \in \mathcal{R}$ and $x \in \mathcal{D}$. The ϵ-range neighborhood of an object x, denoted by $N_\epsilon(x)$, is defined as follows:

$$N_\epsilon(x) = \{y \in \mathcal{D} | dist(y, x) \leq \epsilon\}, \tag{1}$$

where $dist(y, x)$ is a metric distance function, and the Euclidean distance is used in this study.

Definition 3 (*Interaction model*). Let $x \in \mathcal{R}^d$ be an object in the dataset $\mathcal{D}$ and x_i be the ith dimension of the data object x, respectively. With an ϵ-range neighborhood interaction, the dynamics of each dimension x_i of the object x is defined by the following:

$$x_i(t+1) = x_i(t) + \frac{1}{|N_\epsilon(x(t))|} \cdot \sum_{y \in N_\epsilon(x(t))} \sin(y_i(t) - x_i(t)), \tag{2}$$

where $sin(\cdot)$ is the coupling function. $x_i(t+1)$ describes the renewal phase value of the ith dimension of object x at $t = (0, \ldots, T)$ during the dynamic clustering.

To characterize the level of synchronization between oscillators during the synchronization process, we define an order parameter. Instead of considering a global order parameter, a cluster order parameter R_c is defined to measure the coherence of the local oscillator population.

Definition 4 (*Cluster order parameter*). It is used to terminate the dynamic clustering by investigating the degree of local synchronization, and is defined as follows:

$$R_c = \frac{1}{|D|} \sum_{x \in D} \left(\frac{1}{|N_\epsilon(x)|} \sum_{y \in N_\epsilon(x)} \mathbf{e}^{-||y-x||} \right), \tag{3}$$

where D is the dataset. $N_\epsilon(x)$ is the ϵ-range neighborhood.

The dynamic clustering terminates when $R_c(t)$ converges, which indicates local phase coherence. At this moment, all cluster objects have the same phase (identical location in the feature space).

For dynamic clustering, each data object is viewed as a phase oscillator and has its own phase (feature vector) at the beginning. As time evolves, each object interacts with its ϵ-range neighborhood according to the interaction model (Equation (2)). With local interactions, the phase of an object gradually aligns with its neighborhood driven by the local cluster structure for each time step. Finally, the cluster order parameter is used to terminate dynamic clustering when R_c converges.

Like in Figure 1, each object interacts with its similar objects, and finally all objects in C1 and C2 are synchronized together and form clusters. Meanwhile, since the outlier objects do not interact with other objects, during the dynamic clustering, they maintain their original values and thus can be read off immediately.

3.5. CIPA Algorithm

In this section, we describe the algorithm CIPA, which involves the following steps:

(1) *Data partitioning*. First, the original large dataset is randomly split into many partitions of fixed size $FixSize$, where $FixSize$ is usually very small compared to the original dataset size. As the final clustering results are not particularly sensitive to this parameter, we specify it as $FixSize = 200$ in all further experiments.
(2) *Dynamic clustering*. For each partition, with suitable interaction range ϵ, we perform the synchronization-based dynamic clustering, and finally each partition is represented by a set of PA and some outlier objects, if existing.
(3) *New data generation*. As the PAs and outliers represent the clustering results in each partition, a new dataset is generated by putting them together. It is important to note that the cluster structure of the new generated data is well preserved due to the desirable property of the synchronization-based clustering.
(4) *Final clustering*. If the size of the new dataset is small enough (e.g., lower than $MaxNumObj$), then the cluster structure of the original data is obtained by

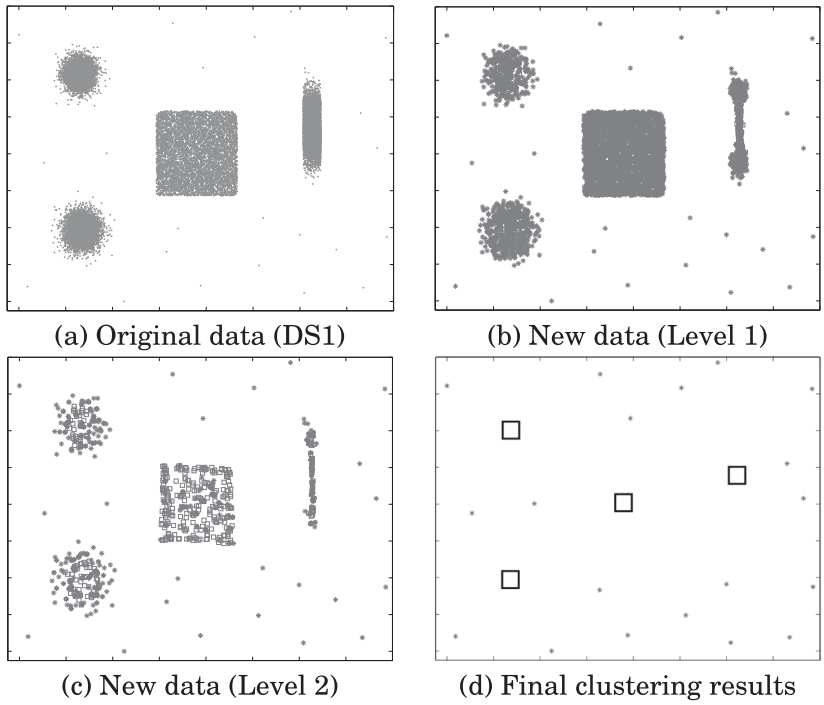


Fig. 5. Illustration of synchronization-based scalable clustering.

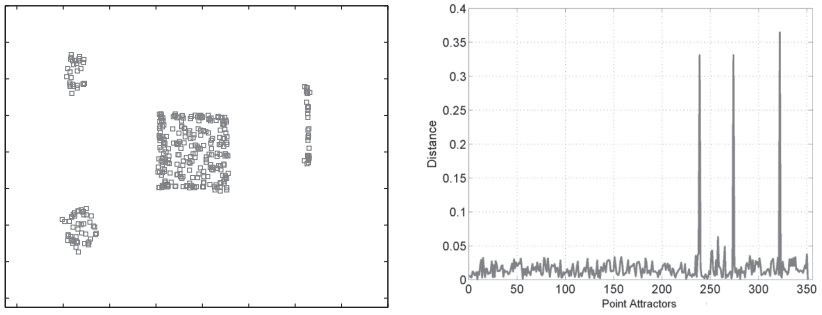
clustering on the new dataset. Otherwise, we repeat steps 1–3 until the size of the newly generated dataset is smaller than $MaxNumObj$.

Based on the data partitioning and the PA representation, our scalable clustering allows clustering very large datasets iteratively. After each *divide-and-conquer* step, the newly generated data does not change the original cluster structure, but the size of the dataset decreases significantly.

To the best of our knowledge, this is the first scalable clustering algorithm that works on exactly the original large dataset. Based on this property, CIPA inherits all benefits from synchronization-based clustering. This means that our method not only enables to handle very large data, but also yields good clustering results, which is difficult for existing scalable clustering algorithms based on sampling or summarization techniques (see Figure 5(b)–(d)). Figure 5(b)–(d) illustrates the newly generated dataset represented by PAs and outliers on different levels during scalable clustering, where □ indicates a PA and $*$ indicates an outlier object. It is interesting to note that the cluster structure of the original dataset is maintained in the newly generated data at different levels.

3.6. Interaction Range Estimation

CIPA needs to specify the interaction range ϵ at two stages: first, when clustering the partitions, and second, when performing the final clustering. For the first stage, to guarantee a stable interaction of each object, we specify ϵ with the average value of the k-nearest neighbor distance determined from a random partitioning, where k is small ranging from 3 to 10. In this paper, we set $k = 5$. A small interaction range ensures that the original cluster structure for each partitioning data is maintained. It is not important that the cluster objects in each partitioning data are regarded as noise due



(a) Point attractors (b) $NearDists$ plot

Fig. 6. Interaction range ϵ estimation.

to the small interaction range since finally these objects will re-cluster again on the next level (*divide* and *conquer*). Therefore, for the interaction range at this stage, a small interaction range (average value of the 5-nearest neighbor distance) is a good choice for most cases.

The second stage of interaction range specification is the final clustering. Here, we present a new simple and effective heuristic rather than a complicated model selection-based method (e.g., based on the MDL principle in Sync [Böhm et al. 2010]) to estimate a suitable interaction range for the final clustering by investigating the distances among clusters. If there are some clusters in the datasets, intuitively, the distances among two clusters should be large (see Figure 6). Namely, we can use the cluster distribution to infer the suitable interaction range. Here, the PAs in the final data are considered only as they can represent the main cluster structure of the dataset (see Figure 6(a)). Specifically, supposing there are two lists of objects in the beginning: *uncheck* consisting of all PAs, and $check = \emptyset$. We first randomly select one PA (named P) in *uncheck* and move it into *check*. Then, we search for the object in *uncheck* (e.g., Q) nearest to P and move Q into *check*. Meanwhile, the distance between P and Q is recorded in a vector *NearDists*. We iteratively search one object in *uncheck* with the nearest distance to any object in *check*, move it to *check* and record the distance into *NearDists*, until *uncheck* is empty. In this way, the objects in a cluster will be searched first, as the distances among them are relatively small. However, the distance in *NearDists* will suddenly peak when an object in another cluster is checked. Therefore, the distances between clusters can be inferred from the peak distances in *NearDist*. Here, the peak distances are defined as distances in *NearDists* that are five times the mean distance in *NearDists*. Figure 6(b) plots the distances in *NearDists*, and it is evident that there are three peak values which correspond to the distances between four clusters. From this plot, general information about the cluster structure can be inferred, such as the number of clusters and the distances among the clusters. Finally, based on *NearDists*, the interaction range is computed as follows:

$$\epsilon = \min\{minPeak, avgPeak/2\}, \tag{4}$$

where *minPeak* is the minimal peak distance in *NearDists* and *avgPeak* is the mean of all peak distances in *NearDists*. Finally, putting all together, the pseudocode of the CIPA algorithm is summarized in Table I.

3.7. Complexity Analysis

For traditional synchronization-based dynamical clustering, the runtime complexity with respect to the number of data objects is $O(T \cdot N^2)$, where N is the number of objects

Table I. Pseudocode of the CIPA Algorithm

```
algorithm [C, O] = CIPA(D, FixSize, MaxNumObj)
   D* = D //Initialization
   while(D*.Size > MaxNumObj)
      P = Partitioning(D*, FixSize);
      D* = null; ε = NN(k); //k = 5
      for(i = 0; i < P.Size; i++)
         R= DynamicClustering(P.get(i), ε);
         D*.add(R);
      end for
   end while
   ε = ParaEstimation(D*) // Equation (4)
   [R] = DynamicClustering(D*, ε);
   C = R.Get(C); O = R.Get(O)
Return C, O;

Function P = Partitioning(D, FixSize); //Data Partitioning.
   np = Math.Ceil(D.Size/FixSize); //Number of partitions
   for(i = 0; i < np; i++)
      Randomly select FixSize different objects, named Di, in D;
      D.remove(Di);
      P.add(Di);
   end for
return P;

Function [R] = DynamicClustering(D, ε); //Dynamic Clustering
   while(loopFlag = true)
      for(each object x ∈ D)
         Search Nε(x) of object x
         Update its new phases using Equation (2);
      end for
      Set D* to be the new phases of all objects
      Compute cluster order parameter Rc;
      if (Rc converges)
         loopFlag = false;
      end if
   end while

   // New data set generation.
   Set all objects in D* unchecked;
   for(each object x ∈ D* and x is unchecked)
      Search all objects in D* that synchronize with x, namely Set;
         If (Set.Size > 1);
            Set all objects in C checked;
            C.p = x.phase; C.w = Set.Size;
            C.ids = All ids of objects in Set;
            R.add(C); //point attractor objects
         Else
            O.p = x.phase; O.w = 1; O.ids = x.ids;
            R.add(C); //outlier object
         End If
   End For
return R;
```

and T is the iterations. In most cases, T is small with $5 \leq T \leq 20$. For the algorithm CIPA, the running time is significantly reduced due to the PA-based data representation and the *divide-and-conquer* strategy. The dynamic clustering for each partitioning data is $O(T \cdot N_p^2)$, where N_p is the fixed size number of objects for each partition data (e.g.,

200) and $N_p \ll N$. The time complexity for all partitions is thus $O(\sum_{l=1}^{L} p_l \cdot T \cdot N_p^2)$, where L is the number of levels and p_l is the number of partitions in each level. For example, there is a dataset with one hundred million objects. For *Sync*, the computation time is $T \times 10^{16}$. For CIPA, supposing $MaxNumObj = 1{,}000$, partitioning size $N_p = 200$, and new data size reduced to one-tenth of the data size at previous scale, finally, its running time is: $T \times ((500,000 + 50,000 + 5,000 + 500 + 50) \times 200^2 + 1,000^2) = T \times 2.2223 \times 10^{10}$. Comparing with *Sync*, CIPA is about 4.5×10^5 times faster. For traditional sampling methods, such as CURE (2.5% of data), for the time complexity $O(N_r^2 \cdot \log N_r)$, with N_r being the number of objects after sampling, computation time is about: 9×10^{13}, which is about 4×10^3 times slower compared to CIPA. With the parallel implementation, the algorithm can be executed independently on each part of the data. Therefore, the running time is reduced to $O(L \cdot T \cdot N_p^2)$ in theory.

4. PARALLELIZATION WITH CUDA

The design of CIPA allows parallelization since each partition can be executed independently. In this section, we use the CUDA programming model to demonstrate how an efficient parallel implementation can be designed for modern many-core accelerators (GPUs) which are attached to a host CPU via PCIe. Our CUDA parallelization uses a single-thread block for each partition. Initially, data objects are transferred to the GPU global memory and only the CIPA clustering results are transferred back to the CPU. Clustering operations are implemented as a number of CUDA kernels in order to avoid costly data movements between GPU and CPU memory.

—*Partitioning*. For data partitioning, an integer array L is first generated in a global memory to store permutation indices of all data objects at each *divide-and conquer* level. Afterwards, every partition (e.g., P_i) containing $FixSize$ data objects with the indices L_i (i.e., $i \times FixSize \leq L_i < (i+1) \times FixSize$) is generated and assigned to an independent thread block with $FixSize$ threads.

—*Dynamic clustering*. For this step, each thread block works independently and the distance computations between objects are required using Equations (2) and (3), the data objects from global memory are thus transferred into the per-block shared memory. The renewed phases of data objects during synchronization are also kept in shared memory and only updated to CUDA global memory at the end of the kernel in a coalesced manner. Finally, a new dataset is generated by collecting clustering results (PAs and outliers) from all partitions.

To represent the clustering results after each synchronization level, the integer array L of size N is kept in CUDA global memory. Initially, L_i is set to i which means object with index i has not synchronized to any object. The final result will look like a forest in which each tree has a root that is associated with a data object that has not synchronized to any other objects. Using this data structure, we can quickly record all the objects that are synchronized to an object x. Consequently, we can generate a new array L for the next *divide-and-conquer* level which is a permutation of all the root nodes. Our current CUDA implementation is able to work on input datasets with dimensionality up to 48 due to the limited size of the shared memory available in a CUDA-enabled device.

5. RELATIONSHIP TO OTHER MAJOR CLUSTERING PARADIGMS

5.1. Synchronization-Based Clustering Algorithms

Inspired by natural synchronization phenomena, many synchronization-based clustering algorithms [Kim et al. 2008; Shao et al. 2013b, 2011] have been introduced recently and have demonstrated attractive properties compared to many existing clustering

algorithms. For instance, (a) the clusters revealed by dynamic clustering truly reflect the intrinsic structure of the dataset; (b) they allows detecting clusters of arbitrary number, shape, and data distribution, even in difficult settings with noise points and outliers. Unlike traditional clustering algorithms, synchronization-based clustering is a dynamic process and driven by local data structure, and also allows for a natural hierarchical analysis (i.e., a set of synchronized objects can be viewed as a new data object (PA), which provides a natural data abstraction). However, like most existing clustering algorithms, synchronization-based clustering algorithms are computationally intensive and do not scale well with the number of observations. The proposed method, CIPA, builds upon *PA representations* and *divide-and-conquer*, a simple yet effective way to handle large-scale dataset. CIPA mainly advances the current synchronization-based clustering algorithms (e.g., Sync) by addressing its efficiency problem. It is able to cluster one hundred million data points in only a few minutes. More importantly, CIPA does not degrade the cluster quality, and the properties of synchronization-based clustering are inherited. CIPA also lends itself to parallelization (like the CUDA implementation of CIPA, see Section 4) and distributed data. Therefore, equipped with the benefits of traditional synchronization-based clustering and its scalability, CIPA shows its superiority over state-of-the-art algorithms in both effectiveness and efficiency aspects.

5.2. Other Scalable Clustering Algorithms

Following the *divide-and-conquer* strategy, the original dataset is partitioned by random sampling in the CIPA algorithm, before each partition is clustered separately. *Divide-and-conquer* is an old and simple strategy, yet is widely used in many complex situations. In the context of clustering, traditional approaches often resort to a suitable (geometric) division of the feature space to split large datasets, which may be hard to define and most likely computationally hard to find, like some space-partitioning data structures such as the KD-tree [Kanungo et al. 2002] and space reduction [Orlandic et al. 2005]. However, scaling up synchronization-based clustering with a *divide-and-conquer* strategy is a natural fit, as PA preserve local cluster structure well, even in the case of sparser densities as created by *random sampling*. The core difference between CIPA and other algorithms with the *divide-and-conquer* strategy is that the clustering results of CIPA do not rely on a specific data partitioning strategy, which gives a big advantage in its computational efficiency and clustering quality. Another main line of research on scalable clustering employs specific data structures or sampling to represent the data like BIRCH [Zhang et al. 1996], CURE [Guha et al. 1998], Bubble [Breunig et al. 2001], and FCM [Havens et al. 2012]. Those algorithms share the benefit of reduced data, which, however come with the drawback that the clustering quality tends to degrade if the data abstraction does not keep the original data distribution. The *PA representations* in CIPA, is still a data summarization technique in a strict sense. However, as the PA envelops its intrinsic cluster structure, the original cluster structure is well preserved. In addition to data-driven approaches, many classical algorithms have been extended to handle large datasets by making use of modern parallel platforms and hardware, such as Map-Reduce-style frameworks [Zhao et al. 2009; Ludwig 2015; Kim et al. 2014; Fu et al. 2014], GPU implementations [Cao et al. 2006] and others [Böhm et al. 2009; Wasif and Narayanan 2011; Adinetz et al. 2013]. Along those lines, CIPA lends itself to parallelization, which is shown in the last part of Section 6.4 (speed-up w.r.t. GPU implementation).

6. EXPERIMENTAL EVALUATION

Before we move on to present scalability experiments with CIPA, which we consider the main results of the paper, we investigate the effectiveness of the approach.

Selection of comparison methods. To study the performance of CIPA, we compare it to the synchronization-based clustering algorithm Sync[1] and two representatives of scalable clustering paradigms on synthetic and real-world data: the summarization-based hierarchical clustering algorithm BIRCH [Zhang et al. 1996], the sampling- and partitioning-based algorithm CURE [Guha et al. 1998], and the prototype-based kernel algorithm rsekFCM [Havens et al. 2012]. In the experiments, we set the parameters of the algorithms to the default values suggested in the corresponding original papers. For instance, in BIRCH, about 1,000 objects are considered for the final hierarchical clustering after re-building the CF-tree, and the sample size is chosen to be 2.5% of the original data, and 50 partitions are chosen for speeding up clustering large datasets in CURE. For rsekFCM, like CURE, the sample size is chosen to be 2.5% of the original data. Except for the CUDA version of CIPA, CIPA and all compared algorithms have been implemented in Java and all experiments have been performed on a workstation with 3.0GHz CPU and 32GB RAM. The source codes of different clustering algorithms and the synthetic datasets are available at (http://staff.uestc.edu.cn/shaojunming/files/2015/08/SourceCode.rar).

Evaluation measures. Comparing the results of different clustering algorithms with respect to effectiveness is a non-trivial problem, especially if different algorithms produce results with different numbers of clusters. To provide an objective comparison of effectiveness, we report three widely used evaluation measures: *Cluster Purity* [Zhao and Karypis 2002], *Normalized Mutual Information (NMI)* [Strehl and Ghosh 2003], and *Adjusted Rand Index (ARI)* [Rand 1971]. For all these measures, higher values indicate better clustering.

6.1. Synthetic DataSets

We start the evaluation with two-dimensional (2D) synthetic data to facilitate the presentation and demonstrate the benefits of CIPA.

Performance on small dataset. To evaluate the performance of CIPA, we first compare it to the synchronization-based clustering *Sync* and check whether the partitioning and PA representations work even on small datasets. Figure 7(a) gives the clustering results on the same dataset as in the original *Sync* publication [Böhm et al. 2010], showing the advantages over many state-of-the-art clustering algorithms. The dataset (DS2) contains 1,226 objects with eight arbitrarily shaped clusters and outlier objects. With CIPA, we partition it into 12 subsets with 100 objects each, and one with only 26 objects. Finally, CIPA successfully detects all the clusters and outlier objects. Even on this small dataset, the PA and outliers can summarize the cluster structure effectively. The clustering results of other algorithms are further plotted in Figure 7(b)–(e).

Comparison to other scalable clustering algorithms. For comparison, we created a 2D dataset DS3 consisting of six arbitrarily shaped clusters plus noise points, cf. Figure 8. For BIRCH, by re-building the CF-tree for many times, the best results have been achieved for the final hierarchical clustering with six clusters (cf. Figure 8(b)). BIRCH cannot successfully detect the clusters in the dataset, because they are not limited to uniform sizes and shapes. For CURE, clusters with few objects are difficult to detect (cf. Figure 8(c)). Similarly, due to the data sampling, rsekFCM has difficulties finding low-density clusters. The reason is that clusters of small size or low density tend to

[1]Synchronization-based clustering algorithms have already demonstrated their advantages over most traditional clustering algorithms such like DBSCAN, EM, KMeans, Meanshift, and Affinity Propagation (see, e.g., previous experiments with Sync [Böhm et al. 2010; Shao et al. 2013a]).

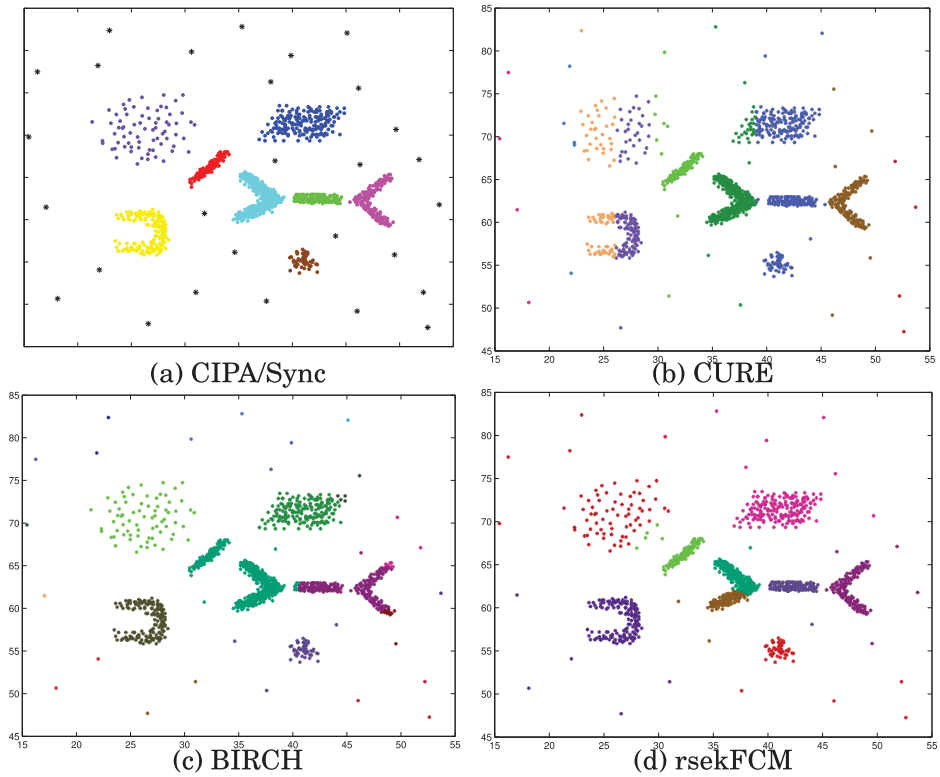


Fig. 7. The performance of CIPA and Sync on small dataset DS2.

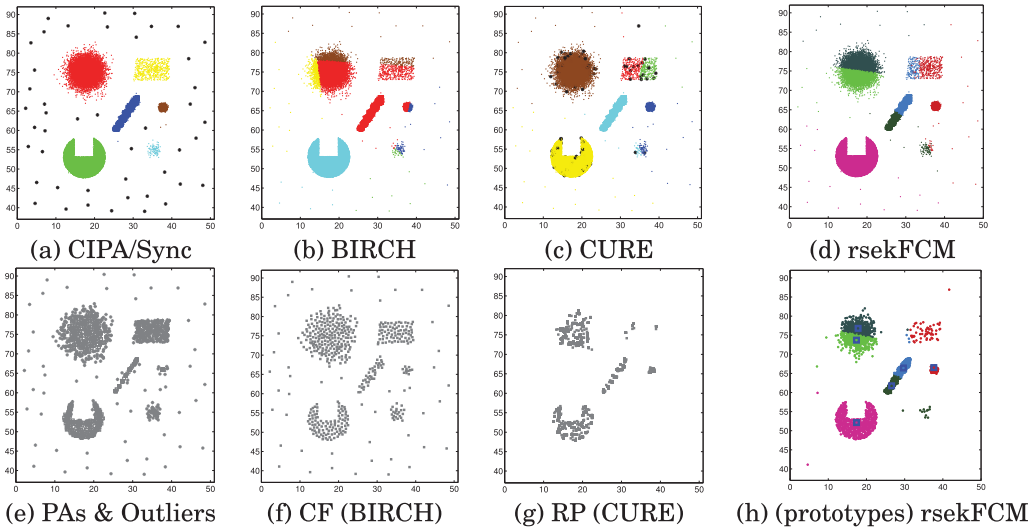


Fig. 8. The performance of different scalable clustering algorithms on DS3.

be ignored, since the sampled data might not maintain the cluster structure anymore. In addition, many cluster objects tend to be considered as outlier objects due to the effect of sampling and partitioning (e.g., a singleton cluster is regarded as noise for outlier handling in CURE). In comparison to these established scalable clustering algorithms, CIPA and *Sync* correctly detect all clusters and noise points driven by the synchronization principle(cf. Figure 8(a)). In the experimental results shown in Figure 8(d)–(f), we present the input for the final clustering with PA representations, CF for BIRCH and data representative points for CURE. As evident from Figure 8(d), PAs and outlier objects maintain the cluster structure of the original data. For BIRCH, with CF-based statistical summarization, all outlier objects and cluster objects are regarded in the same way, which results in difficulties of the final hierarchical clustering to remove outliers. For CURE, the representative points of clusters of small size or low density are missing due to its sampling and partitioning. For rsekFCM, prototypes (blue squares in Figure 8(h)) have been obtained by clustering the sampled data. These resulting prototypes are further transformed into a large dataset for final clustering. However, due to the sampling effect and limitation of K-Means, it tends to fail if the dataset consists of clusters with different densities and shapes.

Properties inherited from synchronization-based clustering. To investigate whether CIPA maintains the desirable properties of synchronization-based clustering, we compare the consistency between CIPA and *Sync*. First, for all mentioned datasets (DS1, DS2, and DS3), CIPA achieves the same clustering results as *Sync* with the same interaction ranges, which allows detecting clusters of arbitrary number, shape, size, even in the presence of noise (Figures 5, 7, 8). We further check whether the PAs really represent the cluster structure at each level, and thus the purity for each PA is examined. It is interesting to note that for all three datasets, the purity of all PAs is exactly 100%, which means that each PA really envelops the objects in a cluster for the three datasets. Therefore, with the PA representation, the desirable properties of synchronization-based clustering are preserved.

6.2. Real-World Datasets

In this section, we compare the performance of all scalable clustering algorithms on real-world data publicly available at the UCI machine learning repository (http://archive.ics.uci. edu/ml).

Wine data. The wine dataset is the result of a chemical analysis of wine grown in the same region in Italy, but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wine. For this small dataset, with 50 instances as the partition size, CIPA detects three clusters and 19 outliers. The cluster purity is 95.5%, and only four instances from type 2 are wrongly clustered together with instances from type 3. As the dataset is very small, BIRCH, CURE, and rsekFCM work on the original data with $k = 3$ and other default parameters. Many instances for BIRCH, CURE, and rsekFCM are wrongly clustered, resulting in a cluster purity of 69.1%, 70.8%, and 67.4%, respectively.

Shuttle data. The shuttle dataset contains 58,000 instances, which are labeled by seven classes. Each instance is described by nine numerical attributes. CIPA detects 9 clusters, and 77 instances are viewed as noise, resulting in good values of $NMI = 0.421$ and $Purity = 91.0\%$. For BIRCH, with $k = 7$, most instances are wrongly clustered with $NMI = 0.001$ and $Purity = 78.6\%$. Similarly, CURE hardly finds reasonable clusters, with slightly better values of $NMI = 0.008$ and $Purity = 78.6\%$. rsekFCM also difficult to find reasonable clusters on this dataset with low cluster quality ($NMI = 0.233$

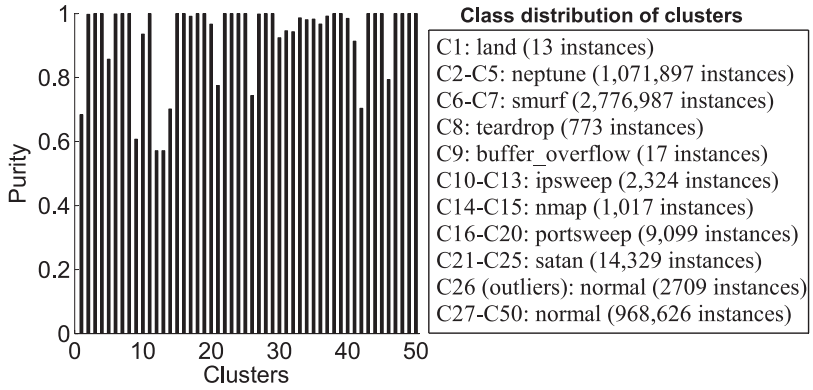


Fig. 9. The clustering results of CIPA on intrusion data. Left: the cluster purity of each cluster; Right: the major type of attack in each cluster and corresponding number of instances.

Table II. Performance of Different Scalable Clustering Algorithms on Real-World Datasets

			CIPA			BIRCH			CURE			rsekFCM		
Data	#Obj	#dim	NMI	ARI	Pur	NMI	ARI	Pur	NMI	ARI	Pur	NMI	ARI	Pur
Wine	178	13	0.703	0.768	0.955	0.399	0.352	0.691	0.374	0.401	0.708	0.352	0.320	0.674
Shuttle	58,000	9	0.421	0.389	0.910	0.001	0.002	0.786	0.008	0.006	0.786	0.233	0.313	0.827
Covtpe	581,012	54	0.136	0.052	0.651	0.057	−0.029	0.498	0.068	0.013	0.502	0.064	−0.015	0.500
Kddcup	4,898,431	34	0.756	0.918	0.989	$2\times e^{-4}$	$1\times e^{-4}$	0.573	$5\times e^{-4}$	$4\times e^{-4}$	0.574	$3\times e^{-4}$	$2\times e^{-4}$	0.573

and $Purity = 82.7\%$). As BIRCH, CURE, and rsekFCM have to work on sampled or summarized data, the quality of the clustering cannot easily be ensured.

Covtype data. The Covtype dataset containing 581,012 instances describes seven forest cover types on a 30 × 30 meter grid with 54 different geographic measurements. CIPA discovers 75 clusters, and several classes are split into multiple clusters ($NMI = 0.136$ and $Purity = 65.1\%$). For BIRCH, with $k = 7$, most instances are wrongly clustered with $NMI = 0.057$, $Purity = 49.8\%$, while CURE achieves a cluster quality of $NMI = 0.068$, $Purity = 50.2\%$. Like BIRCH and CURE, rsekFCM yields similar results with $NMI = 0.064$, $Purity = 50.0\%$.

Network intrusion data. The KDD-CUP'99 intrusion detection dataset consists of two weeks of raw TCP dump data (4,898,431 instances) for a local area network simulating a true Air Force environment with 22 occasional attacks. There are three major classes of the dataset: *neptune* (1,072,017 instances), *smurf* (2,807,886 instances), and *normal* (972,781). 34 numerical attributes out of the total 42 attributes have been selected for clustering. CIPA detects 49 clusters and 3,644 outliers, which achieves a good cluster quality of $NMI = 0.756$, $Purity = 98.9\%$ on the large dataset. The purity of each cluster and the major types of clusters are further illustrated in Figure 9. BIRCH, CURE, and rsekFCM ($k = 23$) hardly find meaningful clusters, with quite low values of NMI, ARI, and $Purity$.

The performance on these four datasets is summarized in Table II. From this table, we notice that the performance of BIRCH, CURE, and rsekFCM is still not comparable to CIPA. Although a larger number of resulting clusters usually leads to relatively lower NMI and ARI values and higher cluster purities, CIPA (yielding more clusters than the true number of clusters) still largely outperforms BIRCH and CURE with respect

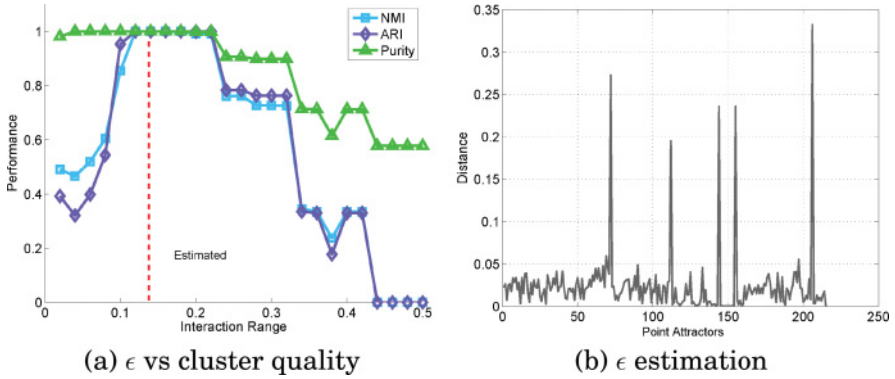


Fig. 10. The effect of the interaction range.

to NMI and ARI on all datasets.[2] For larger datasets such as Covtype or Network Intrusion, the benefits of CIPA are more pronounced.

6.3. Sensitivity to Parameters

In this section, we perform sensitivity analysis for CIPA with respect to the interaction range ϵ, random partitioning, and partition size, on the dataset DS3 used in Figure 8.

Interaction range. As stated in Section 3.6, there are two stages in the algorithm at which the interaction range ϵ needs to be specified. For the first stage, we examine the different values for ϵ with $k = 3$ to 10. CIPA obtains the same clustering results. However, when we use ϵ with $k = 20$, the quality of the clustering suffers due to many outlier objects or objects in different close clusters that are synchronized together by the large interaction range. To evaluate the estimated interaction range for the final clustering, Figure 10(a) plots the interaction range ϵ with respect to the quality of the clustering. It is evident that there is a wide suitable interaction range for our synchronization-based clustering on this dataset. Figure 10(b) plots the distances (*NearDists*) from PAs on the final level. By computing the interaction range based on Equation (4), CIPA allows finding a good ϵ (the red dashed line) with high-quality cluster results. In addition, Figure 11 shows the clustering results on the dataset consisting of multiple clusters with different densities. Unlike DBSCAN, CIPA allows detecting clusters with multiple densities in a dataset with one interaction range. The reason lies in its dynamic clustering process: Objects in clusters with multiple densities will synchronize together within the same interaction range.

Partition size. We assess the effect of the partition size on clustering with different settings ranging from 2 to 30,000. Figure 13 shows the clustering results based on different evaluation measures. Even on the two extreme cases: two objects in each partition and the dataset is only split into two subsets with a partition size up to 20, CIPA still achieves success. However, as expected, the performance becomes more stable with a partition size up to 20.

Random partitioning. For evaluating the impact of random partitioning, we cluster the dataset for 100 times and check the differences of clustering results based on the evaluation measures. Interestingly, the clustering results turn out to be not very

[2]Note that NMI and ARI do not necessarily get better with increasing cluster numbers; they usually peak around the correct number of classes. Discovering more clusters than necessary should lead to worse performance, which is not observed here.

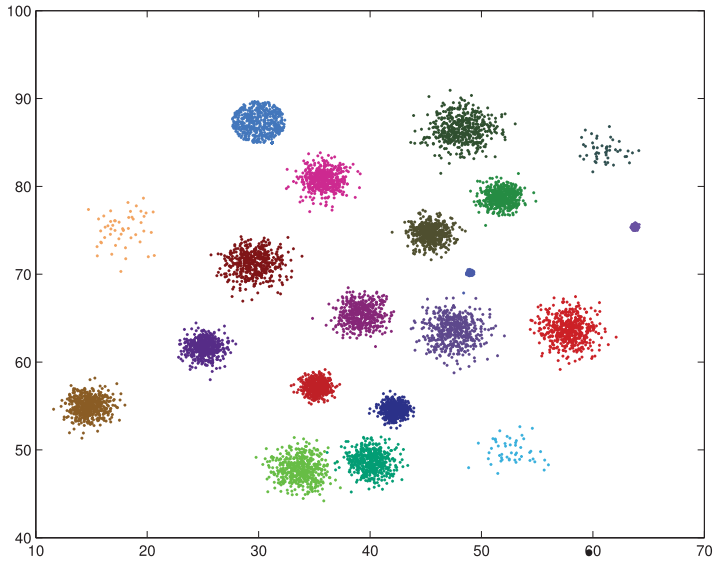


Fig. 11. The effect of the interaction range on dataset consisting of multiple clusters with different densities.

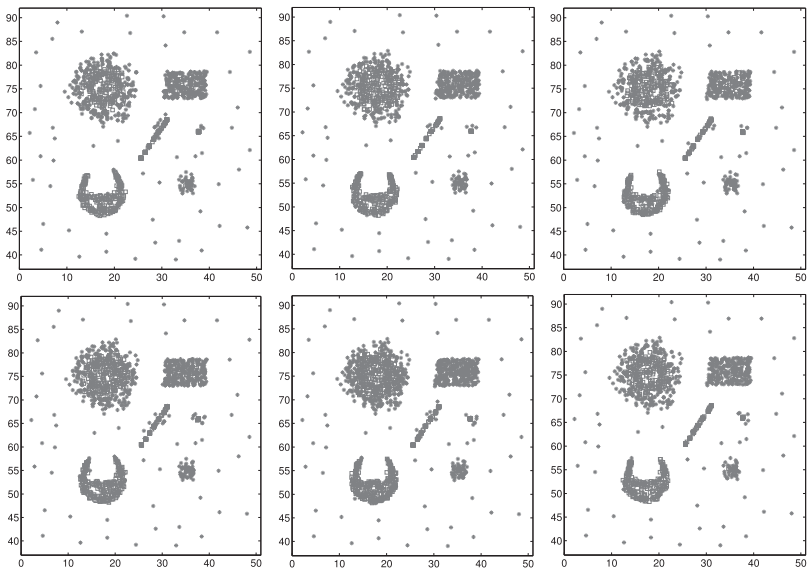


Fig. 12. The effects of randomly partitioning.

sensitive against the random partitioning ($NMI = 0.999 \pm 2.64e^{-4}$, $ARI = 0.999 \pm 2.55e^{-5}$, $Purity = 0.999 \pm 4.27e^{-5}$). In addition, to further check the random partitioning for clustering, we plot the obtained PAs and outlier objects before final clustering (see Figure 12). Due to space limitations, only six plots are given. From these plots, it is clear that the cluster structure for all plots is maintained and almost the same. The reason behind this is the local and dynamical synchronization-based clustering.

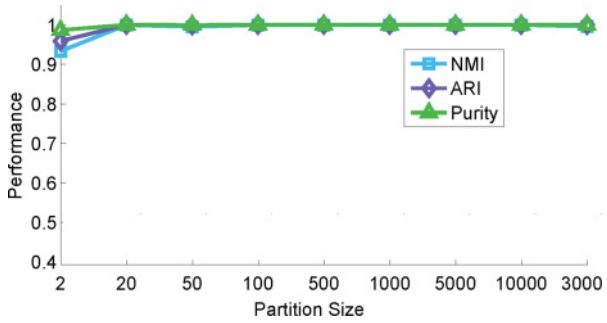


Fig. 13. The effect of the partition size on clustering.

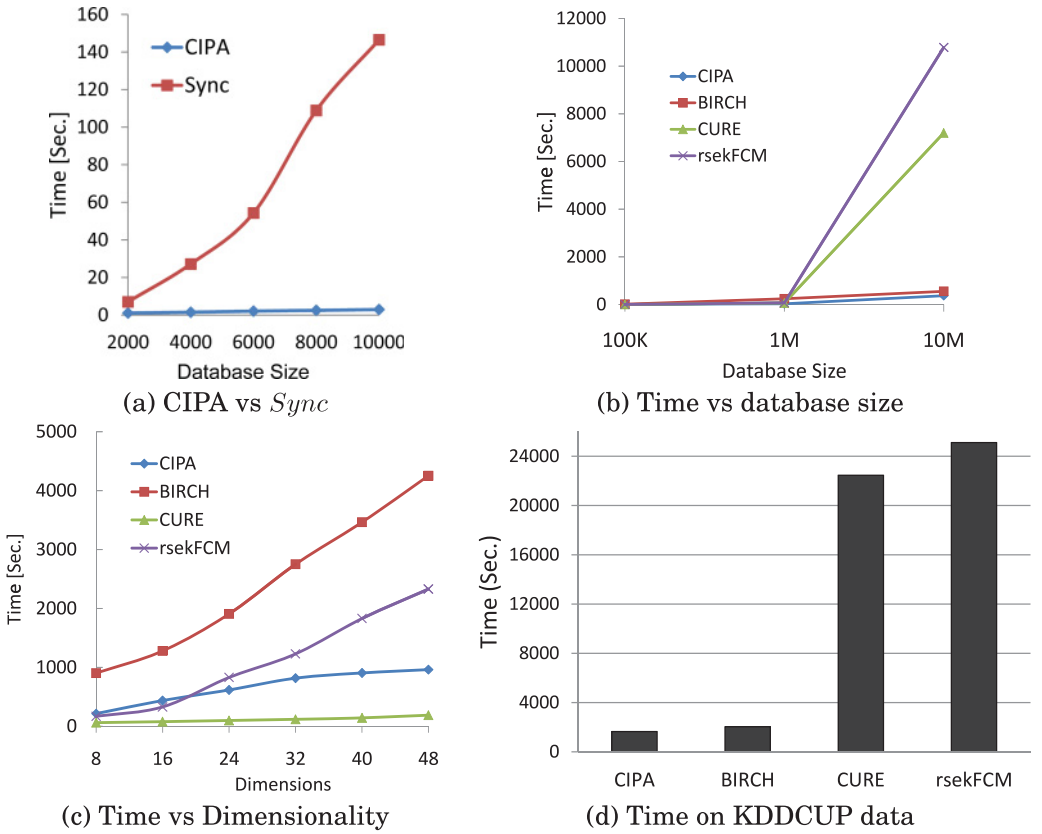


Fig. 14. Running time depending on the database size and dimensionality.

6.4. Scalability Experiments

In this section, we compare the scalability for different scalable clustering algorithms. In addition, the speed-up of the GPU parallel implementation of CIPA is further assessed.

CIPA vs Sync. In Figure 14(a), we first investigate the runtime of CIPA compared to *Sync* with respect to database size ranging from 1,000 objects to 10,000 objects. It

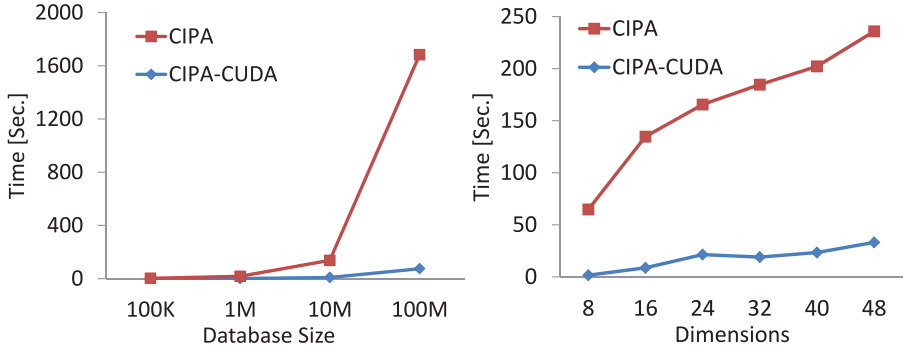


Fig. 15. Speed-up w.r.t. GPU implementation.

indicates that the larger the database size, the higher the speed-up of the clustering (e.g., approximately 100 times faster for the 10,000 objects).

Runtime w.r.t. database size. To assess the scalability of CIPA with respect to database size, we generate three 2D synthetic data with different database sizes (one hundred thousand objects, one million objects, and ten million objects). Figure 14(b) shows the running time for different scalable clustering algorithms. The running time of CIPA is comparable to BIRCH and CURE when the database size is smaller than one million objects and is much faster when the size of the database increases, especially for CURE and rsekFCM. The reason is when the data size is very large, BIRCH needs to re-build the CF-tree again and again, while the sample size (2.5% of original dataset) is still large for CURE and rsekFCM due to its time complexity of $O(N^2 \cdot \log(N))$. Figure 14(d) further plots the running times of different algorithms on the real-world kddcup dataset.

Runtime w.r.t. dimension. To assess the scalability of CIPA with respect to dimensionality, five synthetic datasets containing one million objects with different dimensions are generated (ranging from 8 to 48). All three algorithms show a linear running time against the number of dimensions (Figure 14(c)). CIPA is faster than BIRCH and comparable with rsekFCM, while CURE is faster than BIRCH and CIPA due to the advantage of the sampling technique (only 25,000 objects in the sampled dataset).

Speed-up w.r.t. GPU implementation. To further assess the speed-up of CIPA with GPU, we implement CIPA-CUDA on the following platforms:

—CPU: Intel Xeon X5650 Dual Hex-Core, 96GB RAM, java 1:6:0 27 in OpenJDK 64-Bit Server VM (build 20.0-b12) with multi-threading enabled (i.e., our CPU implementation uses 12 threads).
—GPU: Nvidia Tesla K20c with 5GB RAM, CUDA version 5.0, attached to the same workstation as mentioned for the CPU.

Figure 15(a) shows the performance comparison for clustering four 2D synthetic datasets with different data sizes ranging from $100K$ to $100M$ ($1K = 10^3$, $1M = 10^6$) on a logarithmic scale. Overall, CIPA-CUDA achieves an average speed-up of around 15 compared to multi-threaded CPU implementation CIPA w.r.t. database size, and the benefits of CIPA-CUDA are more pronounced with increasing database size. Figure 15(b) plots the running times for synthetic datasets containing one million objects against different dimensions ranging from 8 to 48 (with a step size of 8). CIPA-CUDA achieves an average speed-up of 13 compared to the CPU implementation of CIPA.

7. DISCUSSION AND CONCLUSIONS

The proposed method, CIPA, builds upon *PA representations* and *divide-and-conquer*. Scaling up synchronization-based clustering with the divide-and-conquer strategy is a natural fit, as PAs preserve local cluster structure well, even in the case of sparser densities as created by random sampling. Many other clustering algorithms would require a suitable geometric division of the feature space, which is hard to define and perhaps computationally hard to find. Although a PA can be viewed as a special form of summarization in a strict sense, it largely differs from the statistical summarization of objects as, e.g., by CF in BIRCH. One main difference is the PA represents the local cluster structure instead of summary statistics. Although one might expect differences between CIPA and *SynC* (with and without PA representations), the consistency of the results turns out to be surprisingly high. One other attractive property of CIPA is its suitability for parallelization on various technical platforms. One current limitation of CIPA is its focus on continuous data. Since data points are considered as phase oscillators in a feature vector space, the generalization to discrete or mixed-type data seems not straightforward. In comprehensive experiments, we have shown that CIPA outperforms state-of-the-art scalable clustering methods in computing accurate clusters (labeled data) and that it scales up favorably in the large. In future work, we are planning to focus on synchronization-based mining of complex network data and data visualization techniques based on simulated object movements.

REFERENCES

Juan A. Acebron, L. L. Bonilla, Conrad J. Perez Vicente, Felix Ritort, and Renato Spigler. 2005. The Kuramoto model: A simple paradigm for synchronization phenomena. *Rev. Mod. Phys.* 77, 2 (Jan. 2005), 137–185.

Andrew Adinetz, Jiri Kraus, Jan Meinke, and Dirk Pleiter. 2013. GPUMAFIA: Efficient subspace clustering with MAFIA on GPUs. In *Euro-Par 2013 Parallel Processing*. Springer, 838–849.

Periklis Andritsos, Panayiotis Tsaparas, Renée J. Miller, and Kenneth C. Sevcik. 2004. Limbo: Scalable clustering of categorical data. In *Advances in Database Technology-EDBT 2004*. Springer, 123–146.

Alex Arenas, Albert Diaz-Guilera, Jurgen Kurths, Yamir Moreno, and Changsong Zhou. 2008. Synchronization in complex networks. *Phys. Rep.* 469 (2008), 93–153.

Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. 2012. Scalable k-means++. *Proc. VLDB Endowment* 5, 7 (2012), 622–633.

Arindam Banerjee and Joydeep Ghosh. 2006. Scalable clustering algorithms with balancing constraints. *Data Min. Knowl. Discovery* 13, 3 (2006), 365–395.

Christian Böhm, Robert Noll, Claudia Plant, and Bianca Wackersreuther. 2009. Density-based clustering using graphics processors. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*. ACM, 661–670.

Christian Böhm, Claudia Plant, Junming Shao, and Qinli Yang. 2010. Clustering by synchronization. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 583–592.

Paul S. Bradley, Usama M. Fayyad, Cory Reina, and others. 1998. Scaling clustering algorithms to large databases. In *KDD*. ACM, 9–15.

Markus M. Breunig, Hans-Peter Kriegel, Peer Kröger, and Jörg Sander. 2001. Data bubbles: Quality preserving performance boosting for hierarchical clustering. In *ACM SIGMOD Record*, Vol. 30. ACM, 79–90.

Feng Cao, Anthony K. H. Tung, and Aoying Zhou. 2006. Scalable clustering using graphics processors. In *Advances in Web-Age Information Management*. Springer, 372–384.

Filip De Smet Dirk Aeyels. 2008. A mathematical model for the dynamics of clustering. *Phys. D, Nonlinear Phenom.* 273, 19 (2008), 2517C2530.

Xiufen Fu, Yaguang Wang, Yanna Ge, Peiwen Chen, and Shaohua Teng. 2014. Research and application of DBSCAN algorithm based on Hadoop platform. In *Pervasive Computing and the Networked World*. Springer, 73–87.

Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. 1998. CURE: An efficient clustering algorithm for large databases. In *SIGMOD Conference*. ACM, 73–84.

Timothy C. Havens, James C. Bezdek, Christopher Leckie, Lawrence O. Hall, and Marimuthu Palaniswami. 2012. Fuzzy c-means algorithms for very large data. *IEEE T. Fuzzy Syst.* 20, 6 (2012), 1130–1146. DOI:http://dx.doi.org/10.1109/TFUZZ.2012.2201485

Lei Hong, Shi-Min Cai, Jie Zhang, Zhao Zhuo, Zhong-Qian Fu, and Pei-Ling Zhou. 2012. Synchronization-based approach for detecting functional activation of brain. *Chaos, Interdiscip. J. Nonlinear Sci.* 22, 3 (2012), 033128.

Jianbin Huang, Heli Sun, Jianmei Kang, Junjie Qi, Hongbo Deng, and Qinbao Song. 2013. ESC: An efficient synchronization-based clustering algorithm. *Knowl.-Based Syst.* 40 (2013), 111–122.

Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. 2002. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Tran. Pattern Anal. Mach. Intell.,* 24, 7 (2002), IEEE, 881–892.

Leonard Kaufman and Peter J. Rousseeuw. 2009. *Finding Groups in Data: An Introduction to Cluster Analysis*. Vol. 344. John Wiley & Sons.

Chang Sik Kim, Cheol Soo Bae, and Hong Joon Tcha. 2008. A phase synchronization clustering algorithm for identifying interesting groups of genes from cell cycle expression data. *BMC Bioinformat.* 9, 56 (2008).

Younghoon Kim, Kyuseok Shim, Min-Soeng Kim, and June Sup Lee. 2014. DBCURE-MR: An efficient density-based clustering algorithm for large data using MapReduce. *Inf. Syst.* 42 (2014), 15–35.

YongChul Kwon, Dylan Nunley, Jeffrey P. Gardner, Magdalena Balazinska, Bill Howe, and Sarah Loebman. 2010. Scalable clustering algorithm for n-body simulations in a shared-nothing cluster. In *Proceedings of the 22nd International Conference on Scientific and Statistical Database Management, SSDBM 2010, Germany, June 30–July 2, 2010.* 132–150.

Simone A. Ludwig. 2015. MapReduce-based fuzzy c-means clustering algorithm: Implementation and scalability. *Int. J. Mach. Learn. Cybern.* (2015), 1–12.

Boriana L. Milenova and Marcos M. Campos. 2002. O-cluster. 2002. Scalable clustering of large high dimensional data sets. In *IEEE International Conference on Data Mining*. IEEE, 290–297.

Ratko Orlandic, Ying Lai, and Wai Gen Yee. 2005. Clustering high-dimensional data using an efficient and effective data space reduction. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*. ACM, 201–208.

William M. Rand. 1971. Objective criteria for the evaluation of clustering methods. *J. Am. Stat. assoc.* 66, 336 (1971), 846–850.

Junming Shao, Zahra Ahmadi, and Stefan Kramer. 2014. Prototype-based learning on concept-drifting data streams. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 412–421.

Junming Shao, Christian Böhm, Qinli Yang, and Claudia Plant. 2010. Synchronization based outlier detection. In *Machine Learning and Knowledge Discovery in Databases*. Springer, 245–260.

Junming Shao, Xiao He, Christian Bohm, Qinli Yang, and Claudia Plant. 2013a. Synchronization-inspired partitioning and hierarchical clustering. *IEEE Trans. Knowl. Data Eng.* 25, 4 (2013), 893–905.

Junming Shao, Xiao He, Qinli Yang, Claudia Plant, and Christian Böhm. 2013b. Robust synchronization-based graph clustering. In *Advances in Knowledge Discovery and Data Mining*. Springer, 249–260.

Junming Shao, Claudia Plant, Qinli Yang, and Christian Bohm. 2011. Detection of arbitrarily oriented synchronized clusters in high-dimensional data. In *IEEE 11th International Conference on Data Mining (ICDM)*. IEEE, 607–616.

Alexander Strehl and Joydeep Ghosh. 2003. Cluster ensembles—A knowledge reuse framework for combining multiple partitions. *J. Mach. Learn. Res.* 3 (2003), 583–617.

Mohiuddin K. Wasif and P. J. Narayanan. 2011. Scalable clustering using multiple GPUs. In *18th International Conference on High Performance Computing (HiPC)*. IEEE, 1–10.

Wenhao Ying, Fu-Lai Chung, and Shitong Wang. 2014. Scaling up synchronization-inspired partitioning clustering. *IEEE Trans. Knowl. Discovery Data Eng.* 26, 8 (2014), 2045–2057. DOI:http://dx.doi.org/10.1109/TKDE.2013.178

Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. An efficient data clustering method for very large databases. In *SIGMOD Conference*. ACM, 103–114.

Weizhong Zhao, Huifang Ma, and Qing He. 2009. Parallel k-means clustering based on mapreduce. In *Cloud Computing*. Springer, 674–679.

Ying Zhao and George Karypis. 2002. Evaluation of hierarchical clustering algorithms for document datasets. In *Proceedings of the 11th International Conference on Information and Knowledge Management*. ACM, 515–524.

Received January 2015; revised September 2015; accepted April 2016