

PENS: An Algorithm for Density-Based Clustering in Peer-to-Peer Systems

Mei Li[†], Guanling Lee[‡], Wang-Chien Lee[†], and Anand Sivasubramaniam[†]
[†] Department of Computer Sciences and Engineering
Pennsylvania State University, University Park, Pennsylvania, 16801, USA
[‡] Department of Computer Science and Information Engineering
National Dong Hwa University, Hualien, Taiwan, 973, R.O.C
[†] {meli, wlee, anand}@cse.psu.edu, [‡] guanling@mail.ndhu.edu.tw

Abstract

Huge amounts of data are available in large-scale networks of autonomous data sources dispersed over a wide area. Data mining is an essential technology for obtaining hidden and valuable knowledge from these networked data sources. In this paper, we investigate clustering, one of the most important data mining tasks, in one of such networked computing environments, i.e., peer-to-peer (P2P) systems. The lack of a central control and the sheer large size of P2P systems make the existing clustering techniques not applicable here. We propose a fully distributed clustering algorithm, called Peer dENsity-based cluStering (PENS), which overcomes the challenge raised in performing clustering in peer-to-peer environments, i.e., cluster assembly. The main idea of PENS is hierarchical cluster assembly, which enables peers to collaborate in forming a global clustering model without requiring a central control or message flooding. The complexity analysis of the algorithm demonstrates that PENS can discover clusters and noise efficiently in P2P systems.

1 Introduction

Huge amounts of data are available in large-scale networks of autonomous data sources dispersed over a wide area. One such environment is peer-to-peer (P2P) systems, where a large number of nodes acting as information providers as well as information consumers self-form into a dynamic information sharing system. The information hidden in the data are valuable for many applications, such as market analysis, scientific exploration and smart query answering. Therefore, it is crucial to discover hidden and valuable knowledge from these data by exploring these sources. Data mining techniques have been shown to be effective for knowledge discovery. However, most of the recent research

efforts in P2P systems focus on data search and relevant issues (e.g., [4, 13, 14, 16, 19]). To the best of our knowledge, data mining in P2P systems has not been investigated in depth yet.

In this paper, we take an initial step to investigate clustering, one of the fundamental data mining tasks, in P2P systems. Clustering, which groups a set of data objects into clusters of similar data objects, can be applied in many different problem domains, such as spatial data analysis, scientific pattern discovery, document categorization, taxonomy generation, customer/market analysis, etc. Various clustering techniques have been proposed for either centralized systems or distributed systems, e.g., [3, 7, 8, 11, 15]. However, these existing clustering techniques are not applicable to P2P systems. Well known clustering techniques designed for centralized systems, e.g., K-mean [15], hierarchical clustering [7], and DBSCAN [8], require all data to reside in a single site. However transferring all data from widely spread data sources to a centralized server for clustering is not desirable in P2P systems due to the lack of a centralized server, excessive communication overhead, privacy concern, and etc. In addition, these clustering techniques were designed to minimize the computation cost and/or disk access cost. However, minimization of the communication cost is the primary goal for the design of clustering techniques in P2P systems. In works that perform data clustering in distributed systems, e.g., [3, 11], each site performs clustering on data objects held locally, and then transfers the local results to a central site [11] or floods the local results to the whole network [3] in order to obtain a global clustering model. Different from these distributed systems, P2P systems have no central site, and the sheer large size of P2P systems makes it impractical for network-wide message flooding. Hence, these distributed clustering techniques are not applicable to P2P systems either. This study re-examines the problem of clustering in P2P systems, identifies critical challenges involved, and provides solutions to overcome these challenges.

One challenging issue that need to be overcome in order to efficiently support clustering in P2P systems is *cluster assembly*. A general idea for clustering in P2P systems is that peers perform clustering on the data objects held locally and then collaboratively assemble the locally obtained clusters to form a global clustering model. The lack of a central site and infeasibility of network-wide flooding makes it extremely challenging for peers to conduct cluster assembly.

We propose a fully distributed clustering algorithm, called *Peer dENsity-based cluStering (PENS)* algorithm, that overcomes above-mentioned challenge. PENS follows the design principle of DBSCAN ([8]), a well-known density-based clustering algorithm¹. We illustrate our proposal in CAN [16], which is a well-accepted P2P overlay network supporting multi-dimensional data². The main idea of PENS is *hierarchical cluster assembly (HCA)*. HCA forms a global clustering model through peer communications only without requiring a central site or message flooding. Peers collaboratively assemble clusters progressively along a hierarchy leveraged by the CAN overlay. We also present optimization techniques applicable to PENS. We analyze the message complexity of PENS. The analytic result indicates that PENS performs density-based clustering in P2P systems efficiently.

The primary contributions of this paper are three-fold:

1. We identify the critical research challenge involved in clustering on P2P systems. To the best of our knowledge, this is the first attempt to investigate the problem of *P2P clustering*.
2. We propose an efficient distributed clustering algorithm, PENS, to facilitate density-based clustering in P2P systems.
3. We analyze the message complexity of PENS, and our analytic result demonstrates the efficiency of PENS.

The rest of this paper is structured as follows. Related works and background are provided in Section 2. We present the details of our proposal, PENS, in Section 3. The complexity analysis of the proposal is shown in Section 4. Finally, we conclude this paper and outline directions for future research in Section 5.

2 Preliminaries

In this section, we review some works relevant to our study and provide backgrounds on DBSCAN clustering algorithm and CAN overlay.

¹We select DBSCAN for investigation is because DBSCAN is efficient in very large databases. It requires minimum domain knowledge to determine input parameters and discovers clusters with arbitrary shapes.

²While our proposed algorithm can be applied to other P2P overlays supporting multi-dimensional data, e.g., SSW [13], we choose the simple overlay structure CAN for the clarity of presentation.

2.1 Related Works

We first review some representative clustering algorithms and then look at some studies on parallel and distributed clusters.

2.1.1 Clustering

Many different clustering algorithms for centralized systems have been proposed. These algorithms can be classified into five classes: partition-based clustering, hierarchical clustering, grid-based clustering, density-based clustering, and model-based clustering. Partition-based clustering algorithms partition n data objects into k partitions which optimize some objective function, e.g., K-mean [15]. Hierarchical clustering algorithms create a hierarchical decomposition of the data set, which can be represented by a tree structure called *dendrogram*, e.g., [7, 10, 22]. Grid-based clustering algorithms divide the data space into rectangular grid cells and then conduct some statistic analysis on these grid cells, e.g., [1, 18, 20]. Density-based clustering algorithms cluster data objects in densely populated regions by expanding a cluster towards neighborhood with high density recursively, e.g., [1, 2, 8]. Model-based clustering algorithms fit the data objects to some mathematical models, e.g., [5]. All of the above clustering algorithms are designed for centralized systems. Thus, they can not be imported to P2P systems directly.

2.1.2 Parallel and Distributed Clustering

Some studies have focused on parallel clustering and distributed clustering. References [6, 9] propose parallel K-mean clustering algorithms, which first distribute the data to multiple processors and then let these processors execute iterative K-mean algorithm in parallel. A processor broadcasts its currently obtained k centroids to all other processors. Once a processor receives centroids from all other processors, it forms global centroids before starting the next iteration of K-mean algorithm. Reference [3] proposes a distributed version of K-mean algorithm in sensor networks. The idea is very similar to the above works. The difference is that the algorithm is unsynchronized, i.e., different sites are not required to execute the same iteration of the algorithm at the same time. References [12, 17] propose distributed hierarchical clustering algorithms. The basic idea is that each site first performs clustering locally, then transfers some local statistics to a central site, which performs global clustering based on the local statistics. Reference [21] proposes a parallel density-based clustering algorithm. Similar to the parallel K-mean algorithms [6, 9], the data are first grouped into partitions and distributed to different processors. Each processor first conducts clustering on its local partition, and then sends certain data objects (called

merging candidates) to a central site for global clustering. Reference [11] proposes a distributed density-based clustering algorithm. The basic idea of [11] is similar to [21]. The difference is that the data are inherently distributed among different sites, and various representations are proposed to summarize local statistics to be sent to the central site.

In summary, the above algorithms either rely on a central site or multiple rounds of message flooding to form a global clustering model. Our study is different from these works fundamentally since we focus on large scaled and fully distributed environments where a central site is not available and flooding is not desirable.

2.2 Background

In this section, we review DBSCAN and CAN that is necessary for understanding of our proposal.

2.2.1 DBSCAN

DBSCAN is a representative algorithm for density-based clustering, which treats the regions in the data space that are densely populated by data as clusters. In the following, we give a short introduction of the DBSCAN algorithm. For details please see [8].

The key idea of basic DBSCAN algorithm is that if the neighborhood of a given radius (ϵ) for a data object has a cardinality exceeding a preset threshold value (T), this data object belongs to a cluster. In the following, we list the definitions of terminologies regarding to density-based clustering from [8] for convenience of presentation.

Definition 1: (directly density-reachable) [8] An object p is *directly density-reachable* from an object q wrt. ϵ and T in the set of objects D if 1) $p \in N_\epsilon(q)$ ($N_\epsilon(q)$ is the subset of D contained in the ϵ -neighborhood of q) 2) $|N_\epsilon(q)| \geq T$.

Objects satisfying Property 2 in above definition are called *core objects* and the property is called *core object property*.

Definition 2: (density-reachable) [8] An object p is *density-reachable* from an object q wrt. ϵ and T in the set of objects D , denoted as $p >_D q$, if there is a chain of objects $p_1, \dots, p_n, p_1 = q, p_n = p$ such that $p_i \in D$ and p_{i+1} is directly density-reachable from p_i wrt. ϵ and T .

It is possible that two objects of a cluster residing along the cluster boundary might not be density-reachable from each other. However, there must exist a third object, from which these two objects are density-reachable. Therefore, the notion of density-connectivity is introduced.

Definition 3: (density-connected) [8] An object p is *density-connected* to an object q wrt. ϵ and T in the set of objects D if there is an object $o \in D$ such that both p and q are density-reachable from o wrt. ϵ and T in D .

While density-reachability is a transitive but not symmetric relation, density-connectivity is both a transitive and symmetric relation. Figure 1 depicts the relations on some sample points where ϵ is the radius of the circles and T is 5.

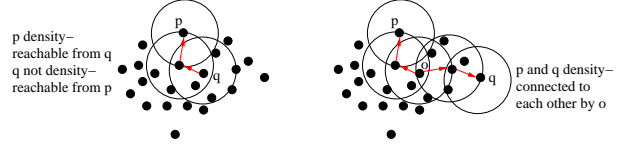


Figure 1. Density-reachability and density-connectivity.

A cluster is defined as a set of density-connected objects which is maximal with respect to density-reachability and the noise is the set of objects not contained in any cluster.

Definition 4: (cluster) [8] Let D be a set of objects. A *cluster* C wrt. ϵ and T in D is a non-empty subset of D satisfying the following conditions: 1) Maximal: $\forall p, q \in D$: if $q \in C$ and $p >_D q$ wrt. ϵ and T , then also $p \in C$. 2) Connectivity: $\forall p, q \in C$: p is density-connected to q wrt. ϵ and T in D .

Definition 5: (noise) [8] Let C_1, \dots, C_k be the clusters wrt. ϵ and T in D . Then, the noise is defined as the set of objects in the database D not belonging to any cluster C_i , i.e., $\text{noise} = \{p \in D | \forall i: p \notin C_i\}$.

Note that not all objects in a cluster are core objects. The objects in a cluster that are not core objects are called *border objects*. Different from *noise objects* that are not density-reachable from any other objects, border objects are density-reachable from some core object in the cluster. In the following discussion, we omit "wrt. ϵ and T " when no confusion is caused.

The DBSCAN algorithm efficiently discovers clusters and noise in a dataset according to above definitions. According to [8], a cluster is uniquely determined by any of its core objects (Lemma 2 in [8]). Based on this fact, the DBSCAN algorithm starts from any arbitrary object p in the database D and obtains all data objects in D that are density-reachable from p through successive region queries, which return all data objects intersecting with the specified query regions. If p is a core object, the obtained set is a cluster in D containing p . On the other hand, if p is either a border object or noise, the obtained set is empty and p is assigned to the noise. The above procedure is then invoked with next object in D that has not been examined before. This process continues till all data objects are examined.

2.2.2 Content Addressable Network (CAN)

CAN organizes the logical data space as a k -dimensional Cartesian space and partitions the space into *zones*, each of which is taken charge of by one or more peers, called as

zone owners. When a new node joins, it obtains a random point in the space and joins the zone covering the random point. The owner of the zone to be joined splits its zone into two equal-sized sub-zones along the dimension chosen in round robin fashion. A data object is mapped to a point in the space and the index for the data object is stored at the peer whose zone covers the corresponding point. In addition to indexing data objects, peers maintain routing tables which consist of pointers to neighboring subspaces along each dimension. Routing from a source zone to a destination zone is performed greedily by always forwarding the message to the peer in the routing table that is closest to the destination zone.

CAN provides two fundamental operations: *PUT* and *GET*. When a peer has a new sharable data object a , it invokes $PUT(a)$ to publish the index information for a to the owner of the zone covering a . When a peer wants to retrieve the index information for a data object a , it invokes $GET(a)$ to obtain this information from the owner of the zone covering a .

2.3 System Model

Assume that there are N peer nodes in the system, where each peer node i has a dataset U_i ($i = 1, 2, \dots, N$) consisting of n_i ($n_i \gg 1$) data objects represented by their k -dimensional feature vectors. Note that the number of data objects ($n = \sum_1^N n_i$) in the system is much greater than the number of nodes in the system, i.e., $n \gg N$. The union of the dataset U_i is U and the domain of U is $D \in R^k$, where R is the set of real numbers in the range of $\{0, 1\}$ ³. Each data object $x \in D$ is represented by a feature vector $x = \{x_1, x_2, \dots, x_k\}$. Without loss of generality, we assume the distance (dissimilarity) between two data objects, $d(x, y)$ where $x, y \in D$, is their Euclidean distance, i.e., $\sqrt{\sum_{j=1}^k (x_j - y_j)^2}$.

A data object is mapped to a point in a k -dimensional Cartesian space. The N peer nodes form into a k -dimensional CAN overlay over this Cartesian space. We modify the CAN overlay slightly as follows. When a peer joins the system, the owner of the zone to be joined splits its zone into two only if the total number of data objects mapped to this zone exceeds some threshold value M . With this minor modification, a zone is not splitted unless it is overloaded with data objects. In following discussions, only the feature vectors of data objects are required for clustering. Therefore, for brevity, we refer the feature vector of a data object as the data object itself. In addition, we refer the data objects mapped to the zone of a peer as this peer’s data objects whenever the context is clear.

³We can always normalize the attributes with domain not in the range of $\{0, 1\}$.

3 Peer Density-based Clustering

We design a fully distributed density-based clustering algorithm in P2P systems, called *Peer dENsity-based cluStering (PENS)*. PENS addresses the challenging issue raised by clustering in P2P systems, i.e., cluster assembly, through HCA (hierarchical cluster assembly). To perform clustering over the whole set of data objects stored in P2P systems, each peer first conducts clustering over the data objects mapped to its zone to obtain local clusters or noise within its zone. Then peers invoke HCA to assemble local clusters progressively to form a global clustering model.

In the following, we further introduce some definitions and terms used in the remaining discussions. We then present the details of HCA in Section 3.1. Lastly, we discuss some optimization technique applicable to PENS.

In the following, "region" refers to a portion of the data space. It may consist of one single zone or multiple neighboring zones. The precise definition of region is deferred to Section 3.1.1.

Definition 7: (cluster within Region i) Let D_i be the set of data objects mapped to Region i . A *cluster within Region i* , C , wrt. ε and T is a nonempty subset of D_i satisfying the following conditions: 1) *Maximality*: $\forall p, q \in D_i$: if $q \in C$ and $p >_{D_i} q$ wrt. ε and T , then also $p \in C$. 2) *Connectivity*: $\forall p, q \in C$: p is density-connected to q wrt. ε and T in D_i .

Definition 8: (noise within Region i) Let C_1, \dots, C_k be the clusters within Region i wrt. ε and T . Then, the *noise within Region i* is defined as the set of objects in the database D_i not belonging to any cluster C_i within Region i , i.e., $\text{noise} = \{p \in D_i | \forall j: p \notin C_j\}$.

Definition 9: (cluster covering Zone i) Let D_i be the set of data objects mapped to Zone i and D be the union of all D_i ($1 \leq i \leq N$). Let C be a cluster in D . C is a *cluster covering Zone i* if $\exists p \in C: p \in D_i$.

Recall that to perform clustering in PENS, before invoking HCA, peers first perform clustering locally based on the DBSCAN algorithm as introduced in Section 2.2.1⁴. When peers finish clustering locally, the data objects in each zone are classified as either local clusters or noise within a zone. Each local cluster is given a unique LocalClusterID, denoted by $\langle \text{ZoneID}, \text{ClusterID} \rangle$, where ZoneID is the unique identifier of a zone (how to obtain the ZoneID is described later) and ClusterID is the unique identifier of a local cluster within the zone. We also given a unique LocalClusterID to data objects that are considered noise locally.

⁴In this paper, we adopt the heuristics developed in [8] to obtain the setting for ε and T from the "thinnest" region of a cluster. For details on ε and T settings, please refer to [8].

3.1 Hierarchical Cluster Assembly

While a central site is not available and network-wide message flooding is impractical in P2P systems, it is extremely challenging to assemble local clusters and obtain a global clustering model. Hierarchical clustering assembly (HCA) is proposed to address this issue. HCA enables peers to collaboratively form a global clustering model through peer communication only. The basic idea of HCA is that the clusters within smaller regions (or zones) are assembled to form clusters within larger regions. These clusters are then assembled to form clusters within even larger regions. This hierarchical process continues until the clusters in the whole data space are formed. HCA consists of three tasks:

- **Hierarchy Formation:** To form a hierarchy in P2P systems which is used for cluster assembly.
- **Cluster Expansion Check:** To determine whether locally obtained clusters can be expanded to other zones and obtain the corresponding cluster expansion information.
- **Cluster Merging:** To merge clusters according to cluster expansion information along the hierarchy to obtain a global clustering model.

In the following, we address these three tasks in details.

3.1.1 Hierarchy Formation

We observe that the CAN overlay can be leveraged to form a hierarchical structure, which can be used for cluster assembly. We call this hierarchical structure as *virtual peer tree (VPtree)*. In the following, we introduce the concept of VPtree and some relevant terms.

The history of space partition during CAN overlay construction can be represented by a virtual binary tree. The root of the tree represents the initial data space and the two subtrees of each node represent the two subspaces generated by a partition event. We encode the binary tree by assigning bit 0 and 1 to the left and right edges of each tree node, respectively. A tree node obtains a tree label by concatenating the edge labels on the path from the root to itself. The tree label uniquely identifies a node in the tree. This encoded binary tree is the VPtree. Note that the VPtree is virtual only and it is not stored anywhere in the system.

The two equal-sized subspaces generated from a partition event are called *buddy regions*. We use *region* as a general term to represent the subspace consisting of one single zone or two buddy regions. The regions that undergo x partitions correspond to the nodes at depth x in the VPtree, and they are called *level- x regions* (thus, the leaf nodes of VPtree are also zones). For instance, the whole data space is level-0 region and the two buddy regions generated from the

first partition (corresponding to the two subtrees of the root node) are level-1 regions. The tree label for a node in the VPtree serves as the *RegionID* for the corresponding region of an internal tree node, or the *ZoneID* for the corresponding zone of a leaf node.

Figure 2 depicts the VPtree structure and the corresponding regions for a CAN in 2-dimension Cartesian space. The Cartesian space is partitioned along Dimension 1 and 2 (marked as $d1$ and $d2$ in the figure) in round robin fashion. In this example, level-5 Zones M (with ZoneID 11110) and N (with ZoneID 11111) are buddy zones. Similarly, level-4 Zone J (with ZoneID 1110) and the region (with RegionID 1111) composed of Zone M and N are buddy regions.

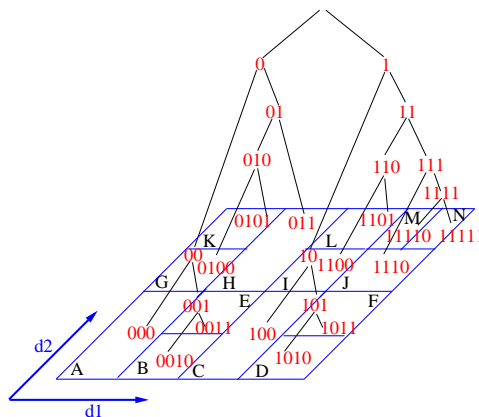


Figure 2. Illustrative example of VPtree.

3.1.2 Cluster Expansion Check

The task of cluster expansion check is to determine whether and how local clusters within a zone can be expanded to other zones and obtain the corresponding cluster expansion information. In the following, we first discuss some observations, which facilitate cluster expansion check. We then explain how cluster expansion check is performed in details.

After careful examination, we have following two observations:

- If all data objects of a cluster within Zone i are at least ε away from the boundary of Zone i , this cluster is also a cluster in the whole data space, i.e., this cluster is *non-expandable*.
- If a cluster within Zone i can be expanded to include other data objects outside of Zone i , i.e., this cluster is *expandable*, there exists at least one data object in this cluster whose ε -neighborhood contains some data objects outside of Zone i .

We call the region inside Zone i that is within ε to the boundary of Zone i as ε -*inner-boundary* of Zone i , and the

region outside of Zone i that is within ε to the boundary of Zone i as ε -outer-boundary of Zone i . Observation 1 implies that if no data object of a cluster within a zone resides in the ε -inner-boundary of the zone, this cluster is not expandable. Observation 2 implies that if a cluster within a zone can be expanded to other zones, there must exist some data objects of this cluster in the ε -inner-boundary of this zone whose ε -neighborhood contains some data objects in the ε -outer-boundary of this zone. In the following discussions, we refer the ε -inner-boundary of a zone as the ε -inner-boundary and ε -outer-boundary of a zone as the ε -outer-boundary whenever the context is clear. Figure 3 illustrates examples of an expandable cluster (A2) and a non-expandable cluster (A1) within Zone A (depicted by the rectangle). The shaded areas indicate the ε -inner-boundary and ε -outer-boundary of Zone A, respectively.

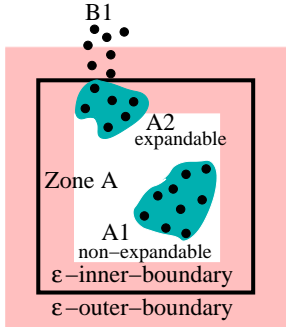


Figure 3. Expandable or non-expandable clusters.

The above two observations can be generalized to following two formal theorems:

Theorem 1: Let D , D_i and D_{i_in} be the set of data objects in the whole data space, the set of data objects mapped to Zone i , the set of data objects mapped to the ε -inner-boundary, respectively. Let C be a cluster within Zone i . If $\forall p \in C: p \in D_i - D_{i_in}$, C is a cluster in D .

Proof: Omitted due to space constraint.

Theorem 2: Let D_{i_in} and D_{i_out} be the set of data objects mapped to the ε -inner boundary and ε -outer-boundary of Zone i , respectively. Let C be a cluster within Zone i . If C can be expanded to other zones, there exists a pair of data objects p and q where $q \in C$, $q \in D_{i_in}$ and $p \in D_{i_out}$ such that p is directly density-reachable from q .

Proof: Omitted due to space constraint.

According to Theorem 2, in order to determine whether a local cluster within a zone can be expanded, we only need to examine whether this cluster is affected by the data objects in the ε -outer-boundary. In addition, only the data objects in the ε -inner-boundary might be affected, i.e., change their core object property (mentioned in Section 2.2.1), by these data objects in the ε -outer-boundary. Therefore, cluster expansion check proceeds as follows. A peer first obtains the data objects in the ε -outer-boundary by a region

query. Then through local computation, the peer examines the ε -neighborhood for each of the data objects in the ε -inner-boundary. There are four possible cases for each of such data object p :

1. Noise: $|N_\varepsilon(p)| < T$.
2. New cluster: $|N_\varepsilon(p)| \geq T$, and all data objects in $N_\varepsilon(p)$ are previous noise objects within zones.
3. Cluster extension: $|N_\varepsilon(p)| \geq T$, and some data objects in $N_\varepsilon(p)$ belong to one cluster within a zone, while others are noise objects within zones.
4. Cluster merging: $|N_\varepsilon(p)| \geq T$, and some data objects in $N_\varepsilon(p)$ belong to two or more clusters within zones while others are noise objects within zones.

Figure 4 illustrates the four cases. Note that different cases might coexist in some scenario. For instance, in Figure 4(d) the two clusters $A2$ and $B1$ are merged. In addition, some noise object is added to the newly merged cluster as well.

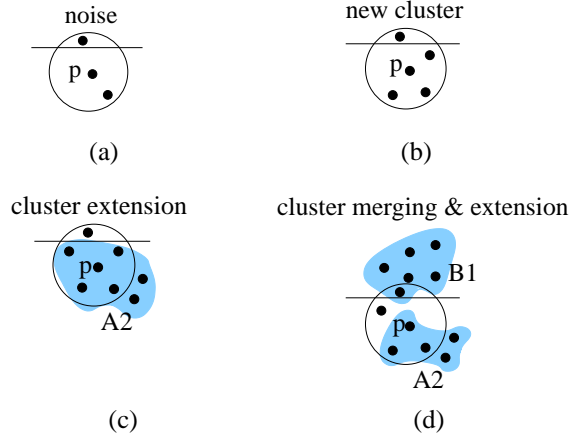


Figure 4. Illustrative examples for cluster expansion check.

For the first case, we do not need to do anything. For the latter three cases, the owner of the zone needs to record cluster expansion information, formally denoted as *cluster expansion set (CES)*, which indicates how local clusters within a zone can be expanded outside of the zone. CES consists of cluster expansion entries for each local cluster that can be expanded to other zones. Each cluster expansion entry consists of two parts, i.e., *present coverage (Pcoverage)* and *expandable coverage (Ecoverage)*, where Pcoverage includes the LocalClusterID for a local expandable cluster within the zone, and Ecoverage includes the LocalClusterIDs for the clusters or noise objects in the ε -outer-boundary that this local cluster can be expanded to.

3.1.3 Cluster Merging

Cluster merging assembles local clusters according to CESs obtained previously along the VPtree to form a global clustering model. To preform cluster merging, the CESs of two buddy zones are first delivered to their parent in the VPtree where the clusters within the two buddy zones are assembled to form larger clusters and the two CESs are merged to form a new CES for the region consisting of the two buddy zones. The CESs of two buddy regions are then delivered to their parent where the cluster assembly and CES merge take place accordingly. This process continues till the root of VPtree (corresponding to the whole data space) is reached. In order to perform above process, we need to address following four questions:

- Who should be the *arbiter* acting as the parent for merging clusters in two buddy regions?
- Who should be the *representative* to deliver the relevant information to the parent along the VPtree?
- How should the CESs be merged at an arbiter?
- What information should be forwarded along the VP-tree?

In the following, we address these four questions and discuss the algorithmic details of cluster merging.

To minimize communication overheads, the arbiter for a region is chosen as the peer in charge of the zone within the specified region that is closest to the center of data space. The representative for a region is chosen among the arbiter for this region and the two representatives representing the two corresponding buddy regions. The selection is made according to the current processing load, bandwidth, etc., at these three peers (when one of the representatives is chosen as the arbiter, one message instead of two is incurred to forward CES to the arbiter). Once the arbiter receives CESs from the two representatives of the corresponding buddy regions, it merges them to form a new set of CES (to be detailed shortly).

Figure 5 illustrates one example for arbiter selection in the top right region from Figure 2. The owner of Zone M acts as the arbiter for the Region 1111 (consisting of the two buddy zones M and N) since Zone M is closer to the center of data space. Thus, the owner of Zone N forwards its CES to the owner of Zone M . Between the owners of Zone M and N , one peer is selected as the representative for Region 1111 to forward the merged CES to the owner of Zone J , who is the arbiter for Region 111 (consisting of Zone M , N and J). Similarly, the owner of Zone L forwards its CES to the owner of Zone I , who is the arbiter for Region 110. Then, the CESs of Region 111 and 110 are merged at the owner of Zone I , who is the arbiter for Region 11.

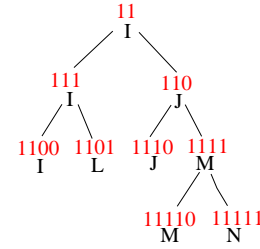


Figure 5. Illustrative example for arbiter selection during cluster merging.

We now proceed to the details of how CESs are merged and corresponding local clusters are assembled at an arbiter. We assume that the two buddy regions are Region A and B and the corresponding CESs are Set A and B , respectively. The merged result is stored back to Set B . Recall that each entry in Set A and B indicates how a local cluster in Region A and B can be expanded. Without loss of generality, we start to examine Set A first to determine whether a cluster in Region A can be expanded to Region B . For each entry i in A (A_i), the arbiter checks whether there is some overlap between Ecoverage of A_i and Pcoverage of an entry j from B (B_j). A non-empty overlapping set between these two indicates that the local cluster (or noise objects) in Region A corresponding to A_i can be combined with the local cluster (or noise objects) in Region B corresponding to B_j . In this case, these two clusters are merged as follows. Pcoverage of A_i is added to Pcoverage of B_j . The overlapping set is then removed from Ecoverage of A_i . In addition, the updated Ecoverage of A_i is added to Ecoverage of B_j . If Ecoverage of A_i becomes empty, the corresponding cluster can not be expanded any more. Thus, the examination for A_i terminates and A_i is removed from A . After all cluster expansion entries in A are examined, if A is not empty at this moment, the remaining entries in A are added to B .

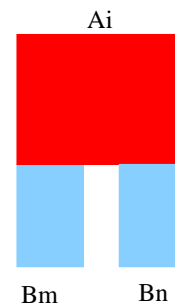


Figure 6. Illustrative example for one cluster to be merged with two clusters.

Up to now, we have merged a cluster in Region A with a cluster in Region B . In some cases, it is possible that a cluster in Region A , e.g., A_i , can be merged with two or more different clusters in Region B , B_m, B_n as shown in Figure 6. Using the procedure described above, A_i and B_m are merged into one larger cluster, and A_i and B_n are merged into another larger cluster. However, in this case, B_m, B_n and A_i should be merged (note that the symmetric case where a cluster in Region B can be merged with two or more clusters in Region A is already handled, since whenever a cluster in A is merged with B , the corresponding cluster expansion entry in B is expanded with the corresponding entry from A). To achieve this, we examine whether two clusters in set B have some overlap in their Pcoverage. If there are, merging is performed by combining their Pcoverage and Ecoverage. The algorithm for merging CESs is illustrated in Algorithm 1.

Algorithm 1 Algorithm for merging CESs at an arbiter.

Merging CESs (A and B are the two CESs to be merged. The merged results are stored in B .)

```

1: for  $i = 1$  to  $|A|$  do
2:   for  $j = 1$  to  $|B|$  do
3:      $U = A_i.Ecoverage \cap B_j.Pcoverage$ 
4:     if  $U \neq \phi$  then
5:        $B_j.Pcoverage = B_j.Pcoverage \cup A_i.Pcoverage$ 
6:        $A_i.Ecoverage = A_i.Ecoverage - U$ 
7:        $B_j.Ecoverage = B_j.Ecoverage \cup A_i.Ecoverage$ 
8:       if  $A_i.Ecoverage = \phi$  then
9:          $A = A - \{A_i\}$ 
10:      end if
11:    end if
12:  end for
13: end for
14: if  $A \neq \phi$  then
15:    $B = B \cup A$ 
16: end if
17:  $i = 1$ 
18: while  $i < |B|$  do
19:   for  $j = i + 1$  to  $|B|$  do
20:      $U = B_i.Pcoverage \cap B_j.Pcoverage$ 
21:     if  $U \neq \phi$  then
22:        $B_i.Pcoverage = B_i.Pcoverage \cup B_j.Pcoverage$ 
23:        $B_i.Ecoverage = B_i.Ecoverage \cup B_j.Ecoverage$ 
24:        $B = B - \{B_j\}$ 
25:     else
26:        $i = i + 1$ 
27:     end if
28:   end for
29: end while

```

In order to determine which information to be forwarded along VPtree, the arbiter checks the merged CES to see whether there are some clusters that can not be expanded out of current region any more (i.e., empty Ecoverage). For those clusters, there is no need to forward the corresponding cluster expansion entry any further (this information is retained at this peer for cluster membership propagation to be discussed shortly). However, in order to obtain certain

global clustering information, e.g., the total number of clusters, we need to indicate the existence of such clusters in the messages forwarded up to the root of VPtree. Therefore, we represent such a cluster by $CRegionID$, the RegionID of the smallest region enclosing the cluster, to indicate the existence and the coverage of this cluster (the coverage information is for cluster membership propagation to be discussed shortly). $CRegionID$ replaces the cluster expansion entry for this cluster and is propagated up along the VPtree. Once the arbiter for the whole data space (the root of VPtree) receives messages from the corresponding representatives, in addition to merging the two corresponding CESs, it also obtains the total number of clusters and assigns each cluster a unique numeric GlobalClusterID.

Note that even though the clusters are assembled along a hierarchy (VPtree), each node in the hierarchy only forwards one message and receives two messages. In addition, the message size is not necessarily monotonically increasing since the clusters that can not be expanded further are simply represented by their $CRegionIDs$. Thus, HCA does not impose high processing load at the root or nodes at the high level of the hierarchy. Potential single point of failure at the root or higher level of the hierarchy can be addressed by extending our current HCA algorithm to have multiple arbiters and representatives for each region, which is left for future exploration.

3.2 Optimization Techniques

We observe that region queries are issued to obtain CES (cluster expansion information) in HCA. The query region in HCA is the ε -outer-boundary of a zone (Section 3.1.2). One optimization is to let a peer store the index information for data objects mapped to the 2ε -outer-boundary of its zone (in addition to storing the index information for data objects mapped to its zone). In this case, a peer can obtain CES through local computation only. We analyze in Section 4 that when 2ε is small and the data dimensionality is high, this optimization could result in large saving in message overheads with marginal extra maintenance and storage overheads.

4 Analysis of PENS

We analyze the message complexity of PENS. Table 1 lists the symbols used in this section. For presentation clarity, we assume that the average side length of zones⁵, r , is not less than 2ε , and the clusters are randomly distributed in the data space.

⁵This is a practical assumption and we can guarantee that this condition is satisfied in most cases by requiring the threshold value M (introduced in Section 2.3), the minimum number of data objects contained in a zone to invoke zone splitting, is not less than $e \cdot (2\varepsilon)^k$.

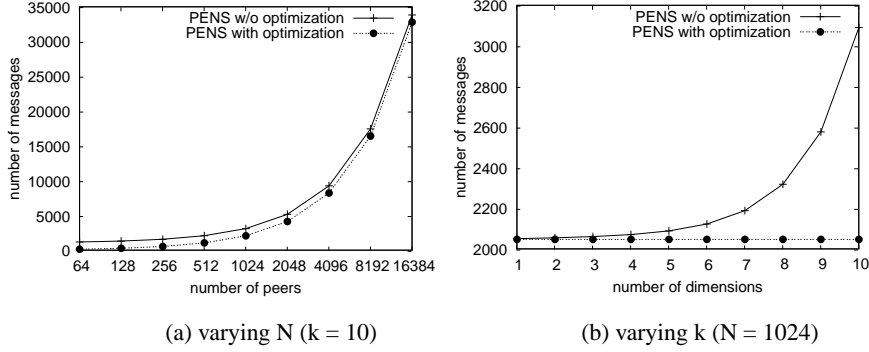


Figure 7. Message complexity incurred by clustering through PENS.

Table 1. Symbols used in the analysis.

Symbols	Descriptions
N	Number of nodes in the network
k	Dimensionality of data objects
r	Average side length of a zone
e	Average data density
α	Average percentage of clusters affected by an insertion/deletion

Recall that PENS discovers clusters and noise through local clustering and HCA (hierarchical cluster assembling). While local clustering doesn't incur any message communication, cluster expansion check and cluster merging in HCA incur message communication. To conduct cluster expansion check, a peer issues a region query to obtain the data objects in the ε -outer-boundary of its zone. For a k -dimensional Cartesian space, the ε -outer-boundary of a zone intersects with the two neighboring zones (abutting all but one dimension) along each dimension, i.e., $2k$ zones in total, and other neighboring zones sharing each of the vertex, i.e., 2^k zones in total. Therefore, cluster expansion check incurs $(2k + 2^k)$ messages. During cluster merging, at most one message is forwarded along each edge in the VP-tree. For a tree with N leaf nodes, there are $(N-1)$ internal nodes, thus, there are $(2N-2)$ edges in the tree connecting these $(2N-1)$ nodes (N leaf nodes and $(N-1)$ internal nodes). Therefore, at most $(2N-2)$ messages are incurred during cluster merging. In summary, the message complexity incurred by PENS is $(2k + 2^k + 2) \cdot N - 2 = O(2^k \cdot N)$.

Using the optimization as mentioned in Section 3.2, no messages are required for cluster expansion check in PENS. Thus, the message complexity is reduced to $2N-2$ ($O(N)$). Compared to the case without optimization, the relative maintenance and storage overhead is proportional to $\frac{e \cdot (2\varepsilon+r)^k}{e \cdot r^k}$, where r^k and $(2\varepsilon+r)^k$ denote the volume of the zone and the volume of the region consisting of this zone

and its 2ε -outer-boundary, respectively. When 2ε is much smaller than r and k is large, the extra maintenance and storage overheads of this optimization technique are marginal.

Figure 7 displays the plots of message complexity incurred by clustering through PENS with and without optimization under different network size and data dimensionality. In Figure 7(a), the data dimensionality is set to 10 and the network size is varied from 64 to 16384 peers. From this figure, we can see that the number of messages grows linearly with the network size. The number of messages incurred by PENS with optimization is smaller than the one without optimization. In Figure 7(b), the network size is set to 1024 and the data dimensionality is varied from 1 to 10. This figure indicates that when the dimensionality increases, the message complexity of PENS with optimization is much smaller compared to the message complexity of PENS without optimization.

In summary, the number of messages incurred by PENS is proportional to the network size (N) and is not related to the dataset size (n), where N is expected to be much smaller than n . This indicates that PENS is efficient. In fact, for any clustering algorithm in P2P systems, in order to form a global clustering model, any peer has to communicate with other peers at least once to exchange clustering information. Thus $O(N)$ message complexity incurred by PENS with optimization is also the lower bound on messages complexity for any clustering algorithms in a P2P system with size N .

5 Conclusion

Peer-to-peer systems have become a popular media for sharing large amount of information among millions of users. While previous research efforts are focusing on supporting search in P2P systems, obtaining hidden and valuable knowledge from these data through data mining techniques is essential for scientific findings and many other applications. In this paper, we investigate clustering, one of

the most important data mining tasks, in P2P systems.

We identify the challenge involved in performing density-based clustering on P2P systems and propose Peer dENsity-based cluSTering (PENS) algorithm, which overcomes the challenge to facilitate clustering in P2P systems. The main idea of PENS is hierarchical cluster assembly (HCA), which enables peers to collaboratively form a global clustering model through peer communication only without requiring either a central site or message flooding. We also present optimization techniques applicable to PENS. We provide complexity analysis on the messages incurred by PENS. The analytic result indicates that PENS can efficiently cluster data objects in peer-to-peer systems.

As for the next steps, we plan to implement the proposed algorithm and verify its efficiency through experiments. In addition, we plan to explore other clustering algorithms and data mining tasks in P2P systems.

6 Acknowledgement

Wang-Chien Lee and Anand Sivasubramaniam were supported in part by National Science Foundation grant IIS-0534343.

References

- [1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of SIGMOD*, pages 94–105, June 1998.
- [2] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: Ordering points to identify the clustering structure. In *Proceedings of SIGMOD*, pages 49–60, June 1999.
- [3] S. Bandyopadhyay, C. Gianella, U. Maulik, H. Kargupta, K. Liu, and S. Datta. Clustering Distributed Data Streams in Peer-to-Peer Environments. *Information Science Journal (In Press)*, 2005.
- [4] Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarca, S. Shenker, and J. Hellerstein. A case study in building layered DHT applications. In *Proceedings of SIGCOMM*, August 2005.
- [5] P. Cheeseman and J. Stutz. Bayesian classification (auto-class): Theory and results. In *Advances in Knowledge Discovery and Data Mining*, pages 153–180. AAAI/MIT Press, 1996.
- [6] I. S. Dhillon and D. S. Modha. A data-clustering algorithm on distributed memory multiprocessors. In *Proceedings of Workshop on Large-Scale Parallel KDD Systems (in conjunction with SIGKDD)*, pages 245–260, August 1999.
- [7] R. Duda and P. Hart. *Pattern classification and scene analysis*. John Wiley & Sons, New York, 1973.
- [8] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of Knowledge Discovery in Database (KDD)*, pages 226–231, 1996.
- [9] G. Forman and B. Zhang. Distributed data clustering can be efficient and exact. *SIGKDD Explorations*, 2(2):34–38, 2000.
- [10] S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In *Proceedings of SIGMOD*, pages 73–84, June 1998.
- [11] E. Januzaj, H.-P. Kriegel, and M. Pfeifle. DBDC: Density based distributed clustering. In *Proceedings of International Conference on Extending Database Technology (EDBT)*, pages 88–105, March 2004.
- [12] E. L. Johnson and H. Kargupta. Collective, hierarchical clustering from distributed, heterogeneous data. In *Proceedings of Workshop on Large-Scale Parallel KDD Systems (in conjunction with SIGKDD)*, pages 221–244, August 1999.
- [13] M. Li, W.-C. Lee, and A. Sivasubramaniam. Semantic small world: An overlay network for peer-to-peer search. In *Proceedings of International Conference on Network Protocols (ICNP)*, pages 228–238, October 2004.
- [14] P. Linga, A. Crainiceanu, J. Gehrke, and J. Shanmugasundaram. Guaranteeing correctness and availability in P2P range indices. In *Proceedings of SIGMOD*, pages 323–334, 2005.
- [15] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.
- [16] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, pages 161–172, August 2001.
- [17] N. F. Samatova, G. Ostrouchov, A. Geist, and A. V. Melechko. RACHET: An efficient cover-based merging of clustering hierarchies from distributed datasets. *Distributed and Parallel Databases*, 11(2):157–180, 2002.
- [18] G. Sheikholeslami, S. Chatterjee, and A. Zhang. WaveCluster: A multi-resolution clustering approach for very large spatial databases. In *Proceedings of VLDB*, pages 428–439, August 1998.
- [19] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of ACM SIGCOMM*, pages 149–160, August 2001.
- [20] W. Wang, J. Yang, and R. R. Muntz. STING: A statistical information grid approach to spatial data mining. In *Proceedings of VLDB*, pages 186–195, August 1997.
- [21] X. Xu, J. Jäger, and H.-P. Kriegel. A fast parallel clustering algorithm for large spatial databases. *Data Mining and Knowledge Discovery*, 3(3):263–290, 1999.
- [22] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of SIGMOD*, pages 103–114, June 1996.