K-Means Clustering Over a Large, Dynamic Network*

Souptik Datta   Chris Giannella   Hillol Kargupta [†]

Abstract

This paper presents an algorithm for K-means clustering of data distributed over a large, dynamic network. The network is not assumed to contain any special server nodes (a peer-to-peer network) and is not assumed to be stable either with respect to the topology or the data held by nodes. The algorithm requires only local communication and synchronization at each iteration: nodes communicate and synchronize only with their topologically neighboring nodes. Due to the growing prevalence of peer-to-peer and mobile/wireless sensor networks, data analysis in large, dynamic networks is likely to garner increasing importance in the near future. To our knowledge, our algorithm represents the first K-means algorithm (a common data analysis/mining technique) to be developed for a large dynamic network.

We tested our algorithm in a simulated environment of up to 1000 nodes on synthetic data. We examine its behavior in a static environment (no data or network change) and a dynamic environment. Empirical results show the algorithm demonstrates good accuracy (in both the static and dynamic environment) in that the cluster labels produced are very similar to those produced by K-means run on centralized data.

Keywords: Peer-to-peer (P2P), K-means clustering, dynamic networks.

1 Introduction

K-means clustering [11] is a well-known and well-studied exploratory data analysis technique. The standard version assumes that all data is available at a single location. However, important applications exist for which data sources are distributed over a large, dynamic network containing no special server or router nodes. Henceforth, we use the term "P2P network" with the meaning described in the previous sentence (note that, we group mobile/wireless sensor networks under the same heading). P2P networks are, by definition, highly decentralized and dynamic (and in practise very large). Therefore, collecting the data at a central location before clustering is not an attractive option. Ideally, a distributed algorithm in this setting should (1) not require global synchronization, (2) be communication-efficient, (3) be resilient to network or data change. Next we give two brief examples motivating the use of K-means clustering in large dynamic networks.

Example: Consider a P2P file-sharing network [2]. These networks allow users to freely join, provided they allow files to be stored on their computers. In return they are allowed to search and down-load files. Due to the ad-hoc nature of the network and lack of centralized control, searching for files efficiently is a major challenge. A global clustering of the files allows individual peers to produce a compact summary of their files held and communicate these to other peers. These summaries are used by other peers to more effectively route search queries [8], [17].

Example: Consider a sensor network consisting of a large number of light-weight, wireless, battery-powered sensors for environment monitoring [1]. Assume each sensor is measuring the same variables. Clustering all the data in a given region of the network over a fixed time window can offer valuable information concerning environmental phenomena *e.g.* can identify outliers. Since the power required for wireless communication goes up with the square of distance, nodes only should communicate with others in a small radius (immediate neighbors). The inherent instability of the network makes synchronization of all nodes in the desired region undesirable. Thus, a K-means clustering algorithm in this example ought to work in a locally synchronous manner *i.e.* nodes only synchronize with their immediate neighbors.

This paper addresses the problem of K-means clustering of data distributed over a large, dynamic P2P network. We present an algorithm, *P2P K-Means*, for which nodes only require local synchronization. While we are not able to provide formal convergence and accuracy[1] guarantees, our experiments show that convergence is always reached quickly and the accuracy is quite

*Primary affiliation of all authors: Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County USA, {souptik1,cgiannel,hillol}@cs.umbc.edu

†Also affiliated with AGNIK LLC, Columbia, MD USA.

[1]Accuracy is defined in terms of the similarity between the cluster labels produced by P2P K-means and those produced by K-means on centralized data. Accuracy is *not* defined with respect to any intrinsic cluster labeling of the dataset; we do not assume any such labeling exists.

good and resilient to network change. To our knowledge, P2P K-means represents the first K-means clustering algorithm to operate in a locally synchronous fashion and handle dynamic networks and data.

Section 2 describes further motivation (P2P file-sharing) and related work. Section 3 describes the P2P K-means algorithm. Section 4 describes the experimental setup (simulator and data generator). Section 5 describes the experiments conducted and their results. Finally, Section 6 concludes the paper.

2 Further Motivation and Related Work

2.1 Further Motivation
To further motivate the problem of K-means clustering in large scale distributed networks, we describe in greater detail the example of P2P file sharing given in the previous example [17]. In this system, each peer holds a collection of images. A user of the system will initiate a search query through her peer. A query is an image and the goal is to return the ℓ most similar images to the user's peer.

The approach in [17] provides each peer with a compact summary of the contents of each other peer. Using this the user's peer ranks the other peers in terms of how likely they are to contain images close to the query. Finally, the peer sends the user's query to other peers from highest rank to lowest. While [17] did not offer any mechanism to cut the search short of sending the query to all peers, they report experiments showing that most of the ℓ nearest neighbors tend to be reported among the high ranked peers.

To construct the summary structure at each peer, a distributed K-means clustering is performed over all images in the network. At the end each peer receives the correct[2] global centroids. The summary structure for each peer is a histogram of its local images with respect to the centroids. The distributed K-means algorithm is globally synchronous and works as follows [8]. At each iteration, an initiator peer starts a *probe and echo* process involving all other peers. This process creates a spanning tree with probes going along edges away from the initiator and echos returning to the initiator. The probes carry the starting centroids for the iteration to all peers. The echos perform a global sum reduction of each peer's local centroids and cluster counts around the starting centroids. Eventually, the initiator peer will produce the sum of all peers' local centroids and cluster counts and compute the new centroids for that iteration. After the final iteration, the initiator broadcasts the final centroids to all peers.

---

[2] Correct in the sense that the peers receive the same centroids that would have resulted from applying K-means on the data after centralizing it.

2.2 Related Work
The algorithm of Eisenhardt *et al.* [8] described earlier is similar to ours in that we both only transmit centroids rather than data to reduce the communication load. However, their algorithm requires a complete global synchronization at each iteration, while ours does not require global synchronization at any time. As a result our algorithm is better suited to a dynamic environment. On the other hand, their algorithm is guaranteed to produce exact results (*i.e.* same as K-means on centralized data), while ours produces an approximation. Bandyopadhyay *et al.* [3] develop a non-locally synchronized K-means algorithm in a P2P network but use random sampling of the network to reduce message load. Their algorithm is similar to ours in that an approximate result is produced. On the other hand, we only require local synchronization and only require nodes to communicate with their immediate neighbors.

Distributed clustering has been addressed recently in the Distributed Data Mining (DDM) community. For a survey of clustering techniques in DDM see [12] and [3]. In particular, some work has been done on parallel implementations of K-means. Dhillon and Modha [7] divide the dataset into p same-sized blocks, then, on each iteration, each of the p processors updates its current centroids based on its block. The processors broadcast their centroids and cluster counts. Once a processor has received all the centroids from other processors it forms the global centroids by weighted averaging. Each processor proceeds on to the next iteration. Forman and Zhang [9] take a similar approach, but extend it to K-harmonic means. Note that the methods of [7] and [9] both start by partitioning then distributing a centralized dataset over many sites. This is different than the setting we consider: the data is never located in one central repository, it is inherently distributed. However, we directly employ their idea of sending around centroids and updating based on weighted averaging.

Very recently, researchers have started to consider data analysis in large-scale dynamic networks. Here the goal is to carry-out data analysis using techniques which are highly asynchronous, scalable, and robust to network changes. Due to the fact that efficient data analysis algorithms can often be developed based on efficient primitives, some researchers have developed several different approaches for computing basic operations (*e.g.* average, sum, max, random sampling) on dynamic networks. Kempe *et al.* [13] and Boyd *et al* [5] investigate gossip based randomized algorithms. Jelasity and Eiben [14] develop the "newscast model" as part of the DREAM project. Both of the above approaches use an epidemic model of computation. Bawa, Gionis, Garcia-Molina, and Motwani, have developed [4] an approach

in which similar primitives are evaluated to within an error margin. Wolff *et al.* [18] develop a local algorithm for majority voting. Finally, some work has gone into more complex data mining tasks: association rule mining [18], facility location [15] (both based on local majority voting), genetic algorithms [6].

Related to the previous paragraph, vigorous ongoing research is being conducted on data analysis in wireless sensor networks. These algorithms must minimize communication (to reduce power consumption) and ought to be robust to network failure. An example of a paper in this area is [10] by Greenwald and Khanna. Here the authors develop communication efficient algorithms for approximating the rth ranked number in an order list. See the references therein for further works on data analysis in sensor networks.

3 The Algorithm

Below we describe our P2P K-means algorithm. The goal of the algorithm is for each node to converge on a set of centroids that are as close as possible to the centroids that would have been produced if the data from all nodes was first centralizing, then K-means was run. In Subsections 3.1 and 3.2 (the next two), we discuss the algorithm in the case of a static environment – data and network do not change. In Subsection 3.3, we describe modifications needed to handle a dynamic environment – data and network change.

Definitions and Assumptions: Let $N_1, N_2, \ldots, N_n$ denote the nodes in the system each with data set X^i. The global dataset is denoted as X which equals $\bigcup_{i=1}^{n} X^i$ (in the distributed data mining literature, this is referred to as *horizontally partitioned* data). Let $Neigh^{(i)}$ denote the set of nodes to which N_i is directly connected at a given time *i.e.* the immediate neighbors of N_i. We assume that each node can reliably compute $Neigh^{(i)}$ at any given time. As a consequence, for example, each node can determine if the link to any of its immediate neighbors from a previous time has gone down.

3.1 The Algorithm: Static Environment
The algorithm is initiated at a single node, say N_1. This node generates an initial set of centroids $\{v_{j,1}^{(1)} : 1 \leq j \leq K\}$ randomly along with a termination threshold $\gamma > 0$ (user-defined), sends these to all of its immediate neighbors $Neigh^{(1)}$, and begins iteration one. When a node receives the initial centroids and γ for the first time, it sends them to the remainder of its immediate neighbors and begins iteration one (further receipts of the initial centroids are ignored). Eventually all nodes will enter iteration one with the same initial centroids and termination threshold γ.

The P2P K-means algorithm, in summary, carries out repeated iterations of a modified K-means at each node N_i. At each iteration ℓ, N_i collects from its immediate neighbors all the centroids and their cluster counts for iteration ℓ. These, along with the local data of N_i, are used to produce the centroids for the next iteration. If these new centroids differ substantially from the old ones, then N_i goes on to the next iteration. Otherwise, it enters a "terminated" state (the meaning of this state is described later).

A formal description of the P2P K-means algorithm in a static environment is described in Algorithm 1. Informally, the algorithm proceeds as follows.

Let $V_\ell^{(i)} = \{v_{j,\ell}^{(i)} : 1 \leq j \leq K\}$ be the centroids held by node N_i at the beginning of iteration ℓ. Let $\{w_{j,\ell}^{(i)} : 1 \leq j \leq K\}$ be the centroids which result from one iteration of K-means carried out on X_i over $V_\ell^{(i)}$. Let $\sharp(w_{j,\ell}^{(i)})$ denote the cluster count of $w_{j,\ell}^{(i)}$ *i.e.* the number of tuples in X_i for which $w_{j,\ell}^{(i)}$ is closer[3] than any other $w_{h,\ell}^{(i)}$, $h \neq j$. These are called the *local centroids* and *cluster counts* of N_i at iteration ℓ. The local centroids and counts, $\{(w_{j,\ell}^{(i)}, \sharp(w_{j,\ell}^{(i)})) : 1 \leq j \leq K\}$, are stored in the *history table* of N_i whose function will be explained shortly.

Node N_i sends a message, $\langle i, \ell \rangle$, (called a *poll message*) to each of its immediate neighbors. This message serves as a request for all the neighbors to respond with their local centroids and cluster counts for iteration ℓ. Once all neighbors have responded (see Subsection 3.3 for details on the case where the neighbors change), N_i updates its jth centroid, $v_{j,\ell}^{(i)}$, to produce $v_{j,\ell+1}^{(i)}$, the jth centroid at the start of iteration $\ell + 1$. The update is a weighted average of the local centroids and counts received from all immediate neighbors (for their iteration ℓ)[4]. The precise update formula is found in step 5 of the first part of Algorithm 1. If the maximum distance between the centroids of N_i at the start and end of iteration ℓ is larger than a user-defined parameter, γ, then N_i goes on to iteration $\ell + 1$. Otherwise it enters the terminated state. From here, N_i no longer updates its centroids or sends polling messages, but continues to respond to polling messages. Next we describe how N_i responds to polling messages while not in the terminated state. Following that, we describe its response while in the terminated state.

Suppose N_i receives a poll message $\langle k, \hat{\ell} \rangle$ during iteration ℓ while not in the terminated state. The

---

[3]In our experiments, we used the Euclidean distance.

[4]If the underlying network topology is a complete graph, then the centroids produced by N_i at the end of iteration ℓ will be exactly the same as those produced by K-means on centralized data at the end of iteration ℓ.

message came from node N_k, during its iteration $\hat{\ell}$. N_i must determine how to respond. If $\hat{\ell} < \ell$, then the history table of N_i contains its local centroids and their cluster counts for iteration $\hat{\ell}$. Hence, N_i, sends these immediately in a response message to N_k. If $\hat{\ell} > \ell$, then the history table of N_i does not contain local centroids for iteration $\hat{\ell}$. So, poll message $\langle k, \hat{\ell} \rangle$ is placed in the *poll table* of N_i. If $\hat{\ell} = \ell$, then N_i checks if its history table contains local centroids and their cluster counts for iteration $\hat{\ell}$. If so, these are sent to N_k. If not, $\langle k, \hat{\ell} \rangle$ is placed in the poll table. Finally, N_i must also check its poll table during iteration ℓ. This is done immediately after producing the local centroids and their cluster counts. For all poll messages $\langle k_1, \ell \rangle$, $\cdots$, $\langle k_m, \ell, \rangle$ in the table, N_i sends its local centroids and their cluster counts for iteration ℓ in a response message to each N_{k_j} and removes these poll messages from its poll table.

Suppose N_i receives a polling message $\langle k, \hat{\ell} \rangle$ during iteration ℓ while in the terminated state. If $\hat{\ell} < \ell$, then N_i responds exactly as it would while not in the terminated state. If $\hat{\ell} \geq \ell$, then N_i sends its local centroids and their counts for iteration ℓ in a response message to N_k.

Algorithm 1: Static P2P K-means

Assumptions: Each node N_i begins the algorithm with X^i (local data), $\gamma > 0$ (termination threshold), and $V_1^{(i)} = \{v_{j,1}^{(i)} : 1 \leq j \leq K\}$ (the initial centroids, the same for each node).

Definitions: A *poll message*, $\langle k, \hat{\ell} \rangle$, received by node N_i means that N_k requests the local centroids and counts of N_i at iteration $\hat{\ell}$. A *response message*, $\langle k, \{(a_j, b_j) : 1 \leq j \leq K\} \rangle$ received by N_i means that $\{(a_j, b_j) : 1 \leq j \leq K\}$ are the local centroids and counts for N_k (sent in response to a previous poll message, so the iteration number is implicit).

Upon the start of iteration ℓ, N_i does:
/* Carry out one round of K-means on local data X^i using $V_\ell^{(i)}$, steps 1 - 2.*/.
1. For each tuple $x \in X^i$, find $1 \leq j(x) \leq K$, such that centroid $v_{j(x),\ell}^{(i)} \in V_\ell^{(i)}$ is closer to x than any other $v_{h,\ell}^{(i)} \in (V_\ell^{(i)} \setminus \{v_{j(x),\ell}^{(i)}\})$. *I.e.* $j(x)$ is the *cluster label* of x at iteration ℓ.
2. For each $1 \leq j \leq K$, let $Z(j)$ denote the set of tuples in X^i with cluster label j, let $w_{j,\ell}^{(i)}$ denote the average of $Z(j)$, and let $\sharp(w_{j,\ell}^{(i)})$ denote $|Z(j)|$. The collection $\{(w_{j,\ell}^{(i)}, \sharp(w_{j,\ell}^{(i)})) : 1 \leq j \leq K\}$ is stored in the *history table* of N_i.

/* Process the poll table, step 3 – it contains all unprocessed poll messages.*/
3. For all messages $\langle k_1, \ell \rangle$, $\cdots$, $\langle k_m, \ell, \rangle$ in the poll table of N_i, send response message, $\langle i, \{(w_{j,\ell}^{(i)}, \sharp(w_{j,\ell}^{(i)})) : 1 \leq j \leq K\} \rangle$, to each N_{k_j}. Remove these messages from the poll table.
4. Send a poll message, $\langle i, \ell \rangle$, to each node $N_k \in Neigh^{(i)}$. Let $Wait^{(i)}$ denote the set of all nodes from which N_i is waiting for a response.[5]
5. N_i waits until, for each $N_k \in Wait^{(i)}$, a response message $\langle k, \{(w_{j,\ell}^{(k)}, \sharp(w_{j,\ell}^{(k)})) : 1 \leq j \leq K\} \rangle$ is received. Node N_i updates its jth centroid as a weighted average:

$$v_{j,\ell+1}^{(i)} = \frac{\sum_{N_k \in (Wait^{(i)} \cup \{N_i\})} w_{j,\ell}^{(k)} \sharp(w_{j,\ell}^{(k)})}{\sum_{N_k \in (Wait^{(i)} \cup \{N_i\})} \sharp(w_{j,\ell}^{(k)})}.$$

6. If $max\{\|v_{j,\ell}^{(i)} - v_{j,\ell+1}^{(i)}\| : 1 \leq j \leq K\} > \gamma$, then node N_i goes on to iteration $\ell + 1$. Otherwise it enters the terminated state.

Upon receipt of a poll message $\langle k, \hat{\ell} \rangle$, N_i does:
1. If $\hat{\ell} < \ell$, then recover the local centroids and cluster counts for iteration $\hat{\ell}$ from the history table of N_i and send them in a response message to N_k, $\langle i, \{(w_{j,\hat{\ell}}^{(i)}, \sharp(w_{j,\hat{\ell}}^{(i)})) : 1 \leq j \leq K\} \rangle$.
2. If $\hat{\ell} = \ell$ and N_i is not in the terminated state, do
    2a. If the local centroids for iteration ℓ are in the history table of N_i, then recover and send them in a response message to N_k.
    2b. Else, add $\langle k, \ell \rangle$ to the poll table of N_i.
3. If $\hat{\ell} > \ell$ and N_i is not in the terminated state, then add $\langle k, \hat{\ell} \rangle$ to the poll table of N_i.
4. If $\hat{\ell} \geq \ell$ and N_i is in the terminated state, then recover the local centroids and counts for iteration ℓ from the history table of N_i and send them in a response message to N_k.

3.2 Space and Communication Cost Analysis: Static Environment
In this subsection, we calculate the space and communication cost of P2P K-means assuming that all nodes have reached the terminated state and I denotes the maximum number of iterations carried out by any node. Let L denote $max\{|Neigh^{(i)}| : 1 \leq i \leq n\}$.

---

[5]In a purely static environment, there is no need to introduce the additional $Wait^{(i)}$ notation since $Wait^{(i)}$ equals $Neigh^{(i)}$. In a dynamic environment, $Wait^{(i)}$ does not necessarily equal $Neigh^{(i)}$ as, for example, some nodes in $Niegh^{(i)}$ may go down. The additional notation is included here to ease subsequent discussion of the modifications needed for the dynamic environment.

At any given node N_i, the space required is proportional to the size of N_i's history and poll tables. Clearly the history table is of size $O(IK)$ since the local centroids and their cluster counts are added for each iteration. The poll table is of size $O(IL)$, since each of N_i's neighbors sends one poll messages per iteration, thus, a maximum of I per neighbor. Therefore, the total space is $O(I(K + L))$. The number of messages (4 byte numbers) transmitted by N_i is $O(ILK)$. This is because N_i sends a poll message (size $O(1)$) to each neighbor at each iteration ($O(IL)$ in total). On top of this, N_i sends a response of size $O(K)$ for each entry of the $O(IL)$ entries in its poll table ($O(ILK)$ in total). Therefore, total number of messages is $O(IL + ILK)$.

Hence the total amount of space and communication over all nodes is $O(nI(K + L))$ and $O(nILK)$, respectively.

3.3 The Algorithm: Dynamic Environment

Now we describe how the static environment algorithm can be modified to address a dynamic environment.

Node failure or topology change: If a node leaves the network, its immediately neighboring nodes will discover this (by monitoring $Neigh$). Likewise, if an edge goes down, the two nodes involved will detect the change to their immediate neighborhood. Each node, N_i, which lost an immediate neighbor, N_k, will remove N_k from $Wait^{(i)}$ and remove any poll messages from N_k from its poll table. Thus, N_i henceforth refrains from waiting on poll requests from N_k in step 5 of the first part of Algorithm 1.

If a new edges come up, all associated nodes will discover this and, therefore, detect new immediate neighbors. A simple way to augment P2P K-means is to have all associated nodes wait until their next iteration before considering the new immediate neighbors.

Node addition: If a new node, N_i, joins the network, a more complicated set of modifications is needed. N_i will need to synchronize itself with its immediate neighbors. A simple way to do this is as follows.

1. N_i polls its immediate neighbors $N_k \in Neigh^{(i)}$ and each returns its current iteration ℓ_k.[6]

2. N_i sets its iteration number to $\ell = min\{\ell_k : N_k \in Neigh^{(i)}\}$, and its centroids as follows. Choose $N_k \in Neigh^{(i)}$ whose iteration number is ℓ (break ties arbitrarily). Send a poll request, $\langle i, \ell \rangle$, to N_k which, in turn, responds with its centroids at the start of iteration ℓ, $V_\ell^{(k)}$. For all $1 \le j \le K$, N_i sets $v_{j,\ell}^{(i)}$ to $v_{j,\ell}^{(k)}$.

3. N_i begins iteration ℓ as described in Algorithm 1.

All immediate neighbors of N_i which are not in the terminated state would simply wait until their next iteration before polling N_i. However, immediate neighbors N_k that were in the terminated state ought to consider becoming active again. To do this, N_k polls N_i for its local centroids and cluster counts at its latest iteration. N_k computes an update of its centroids and if the resulting centroids have changed significantly, N_k becomes active again and sends a message to all its immediate neighbors. If any of these neighbors are in the terminated state, they follow the same procedure to determine if they should become active.

Data change: If the data at a node N_i changes and N_i is not in the terminated state, its behavior need not change. If N_i is in the terminated state, it must determine whether or not to become active again. To do this, N_i recomputes its local centroids and cluster counts and polls its immediate neighbors for their local centroids and cluster counts (at their latest iteration). Then N_i updates its centroids and if the resulting centroids have changed significantly, N_i becomes active again.

In either case, though, N_i sends a message to all its immediate neighbors indicating a data change. Neighbors not in the terminated state, can ignore the message. Neighbors in the terminated state will determine whether to become active using the same technique as described above in the "Node addition" case.

3.4 Comments

1. No node has an explicit condition under which all activity stops. However, if all nodes have entered the terminated state, all communication ceases *i.e.* the algorithm has terminated.

2. The algorithm does not require global synchronization in the sense that all nodes in the network must be on the same iteration. However, the algorithm is not completely asynchronous in the sense that any node can be on any iteration with respect to any other node. Local synchronization is required in the following sense. Since a node must wait for responses from its immediate neighbors (unless they go down), it cannot move more that one iteration beyond them. Hence the difference in iteration number between two nodes is always upper bounded by the number of network links between the two nodes.

3. Proving convergence of P2P K-means or bounding its accuracy appears to be a quite hard problem.

---

[6]If N_k goes down before sending a response, N_i detects this (by monitoring $Neigh^{(i)}$) and refrains from waiting on N_k.

However, in experiments the algorithm always converges quickly with quite high accuracy.

4. We point out a connection between Algorithm 1 (static P2P K-means) and Lamport's work on the ordering of events in a distributed system [16]. He argues that an event a at a node N_1 can be said to have happened before (in a global sense) event b at node N_2, if N_1 sent a message after a that was received by N_2 before b. This is analogous to the centroid update phase in Algorithm 1 (step 5). A node wants its centroid update at iteration ℓ to happen after all neighbors produce their local centroids.

4 Experimental Setup

We studied the behavior of P2P K-means in a simulated environment on a single machine. Before reporting the results, we first describe the simulator (Subsection 4.1) and the data used (Subsection 4.2).

4.1 Simulator Our simulation consists of two parts: the network topology with edge delays; the message passing and local computation behavior. For the network topology and delays, we have used an Internet topology generator developed elsewhere, BRITE[7]. It produces a weighted graph with edge weights representing communication delays (in mSeconds). We have used flat level "Autonomous System" (AS) with Waxman model to simulate the network in BRITE, where two nodes u and v are connected with a probability $P(u,v) = \alpha e^{-d(u,v)/\beta L}$, where α =0.15 and β = 0.2. An "incremental growth" version of the Waxman model with random node placement is used during topology construction. A new node surveys the existing nodes in the graph in each step and connects to m (= 2) of them with the said probability. Other parameters[8] used are HS = 1000, LS = 100 (size of the plane), constant bandwidth distribution with Max BW = 1024, and Min BW = 10.

For the message passing and local computation behavior, we built our own simulator from scratch that simulates a distributed dynamic network environment with large number of nodes. For monitoring the whole network, we designed the simulator in a way that it operates with respect to a common global clock. Note that, the existence of the global clock does not call for any sort of synchronization of computations of the nodes. The whole network just updates its state at each global clock tick.

Since the simulator operates on the above global clock ticks rather than wall clock time, the BRITE map edge delays must be converted from mSeconds to number of clock ticks. To do this, we assume a clock tick in 500 mSeconds and round edge delays to the nearest multiple of 500 mSeconds. Transmission of a message is simulated by decrementing its number of ticks to arrival at each global clock tick. If the message has arrived, it is copied into a buffer associated with the arrival node. We assume that network communication dominates local computation (a reasonable assumption if the local datasets are small). Thus, local computation time is not considered in the simulation. In one clock tick, each node is able to carry out one round of K-means producing local centroids and cluster counts and send polling requests to all neighbors. The remainder of the iteration will require more clock ticks until all responses have arrived. Once all responses have arrived, the node updates its old centroids without any additional ticks and, moreover, processing poll requests from other nodes does not incur any additional clock ticks.

4.2 Data Generation All of the data used for our experiments was 10-dimensional data sampled independently from a mixture of 10 Gaussian distributions and a uniform distribution. Each point in the dataset has a fixed probability of being generated from the uniform random distribution, or any of the 10 predefined Gaussian distributions. Throughout most experiments, the means and covariances of the Gaussian distributions are fixed (unless otherwise indicated). Figure 1 depicts a 78200 point, 2D dataset generated *without* the uniform distribution. However, in all our experiments, the generator included a uniform distribution (noise) as described above. Note that this is just a representative 2-dimensional dataset used here for visualization, the original data used in all the experiment is a 10-dimensional data.

5 Results

We varied the number of nodes up to 1000. In all experiments, γ, the termination parameter, is set to 0.01. We also start the simulations with each node having the same initial centroids to start with, thereby ignoring the initial propagation delay. Average immediate neighborhood size is 4 per node. The number of clusters (K) in all simulations is set to 8.

5.1 Static Environment We first tested our algorithm in a static environment – no data or network changes. We conducted experiments with the number of nodes increasing up to 1000 making sure that the total number of points divided by the number of nodes

[7]www.cs.bu.edu/brite
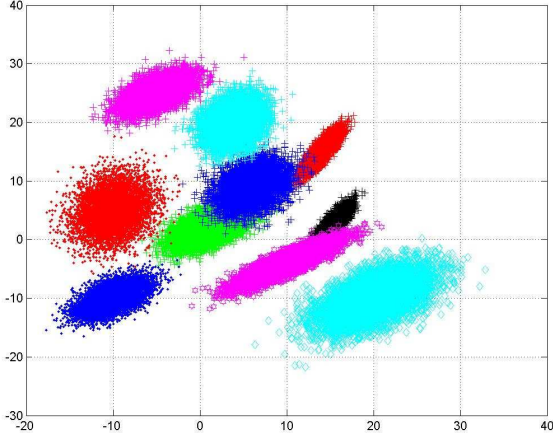[8]Refer to BRITE documentation for details.

Figure 1: A 78200 point dataset generated from a mixture of 10 Gaussian distributions.

remained constant for experiments with both the uniformly and non-uniformly distributed data. In this way, we can observe the effect of increasing the size of the network without the number of points per node becoming a confounding factor. For ν nodes, first a dataset containing 649ν points was generated, X. Then these points were assigned to nodes in one of two ways: *uniform assignment* all nodes get the same number of points, *non-uniform assignment* the number of points per node follows a Zipfian distribution with parameter 0.8 (with the constraint that each node will contain at least $K = 10$ points).

Experiments are conducted by first running a centralized K-means on X. Then P2P K-means is run on X when assigned uniformly over nodes and when assigned non-uniformly. The initial centroids in all cases are the same.

We seek to answer the following questions.

- Does our algorithm incur a large error in terms of assigning points to different clusters than a centralized K-means?

- Does our algorithm scale well with the size of the network in terms of communication cost and error?

5.1.1 Accuracy We measure accuracy of P2P K-means by comparing the final cluster assignments at each node to the final cluster assignments from the centralized K-means. Note, we *do not* measure accuracy in terms of any intrinsic cluster labeling of the global dataset X. We do not assume any such labeling exists. We measure accuracy as the difference between the cluster assignments produced by P2P K-means and

those produced by K-means run on X after centralization. When we refer to the "cluster membership" of a given point, we mean the cluster to which the point was assigned by a specified algorithm (K-means run on centralized data or P2P K-means).

The initial centroids are labeled $1, \ldots, K$ (the same centroids for each node and for the centralized K-means). For each $x \in X$, let $L_{cent}(x)$ denote the label (cluster membership) of the cluster to which x is assigned at the end of the centralized K-means. Assume x appears at node N_i i.e. $x \in X^{(i)}$. Let $L_{p2p}^i(x)$ denote the label of the cluster at N_i to which x is assigned once the node reaches the termination state after execution of P2P K-means. We define *percentage membership mismatch (PMM)* over a node i as

$$PMM^i = \frac{100|\{x \in X^i : L_{cent}(x) \neq L_{p2p}^i(x)\}|}{|X^i|}.$$

Let $Avg(PMM)$ denote the average percentage membership mismatch over all nodes i.e. $\frac{\sum_{i=1}^{n} PMM^i}{n}$. Similarly, let $STD(PMM)$ denote the standard deviation in the average percentage membership mismatch.

Once all peers have reached a termination state we measure $Avg(PPM)$ and $STD(PPM)$. Figure 2 depicts accuracy as the number of nodes increases from 100 to 1000. For uniform node assignment, the average PMM ("Avg(PMM) U") does not exceed 3% and, including three standard deviations, does not exceed 5.4%. For non-uniform node assignment, the average PMM ("Avg(PMM) NU")does not exceed 3.1% and, including three standard deviations, does not exceed 6%. Hence, the error incurred by our algorithm is quite low.

Moreover, the accuracy appears more or less flat with respect to the number of sites implying our algorithm enjoys excellent accuracy scalability with respect to network size.

Finally, the assignment of data to nodes (uniform vs. non-uniform) has only a small affect on accuracy (a slight decrease for non-uniform, as expected). As we see in the figure, the PMM values for data uniformly distributed and distributed following a Zipfian distribution with parameter 0.8 are very close to each other. Experiments with a smaller Zipfian parameter (we experimented with Zipfian parameter = 0.5 as well) shows even closer PMM values.

5.1.2 Communication Complexity We measure the communication complexity of P2P K-means by counting the number of bytes passed during the simulation. Let $Comm^i$ denote the total number of bytes received by node i during the entire run of P2P K-means.
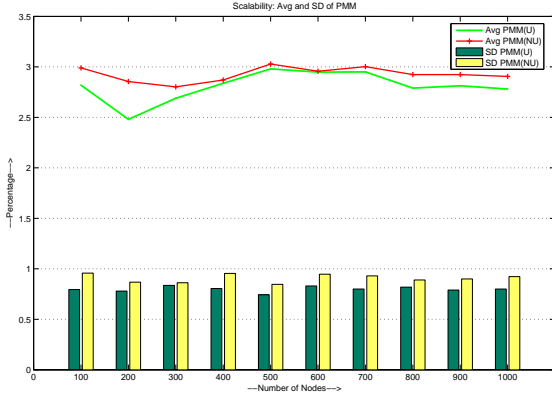
Figure 2: Variation of the average and standard deviation of PMM with the number of nodes in the network.
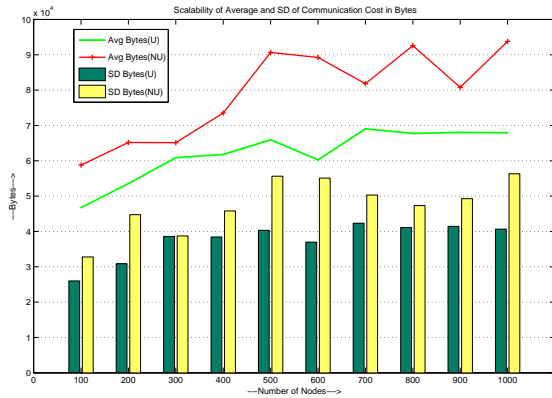

Figure 3: Variation in the average and standard deviation of communication cost with number of nodes in the network.

Let $Avg(Comm)$ denote the average number of bytes received *i.e.* $\frac{\sum_{i=1}^{n} Comm^i}{n}$ and let $STD(Comm)$ denote the standard deviation.

Figure 3 depicts the communication cost incurred by the algorithm. The number of bytes received per node increases slightly with increase in number of sites for uniform assignment of points to sites. For non-uniform assignment, a sharper increase is observed. The trend appears linear in both cases with small slope (good communication cost scaling).

5.2 Dynamic Data, Dynamic Network We also tested our algorithm in a dynamic environment – both data and network changes. We assume that changes occur every τ global clock ticks. We consider both a stationary and non-stationary distribution case. In the former, old data is replaced by new data sampled from the same distribution. In the later, each time the new data is sampled from a different distribution.

Next we describe our experimental procedure. Initially, a network is generated followed by a dataset which is assigned to nodes just as in the static case with non-uniform point assignments (Zipfian 0.8). Next, 10 percent are chosen randomly and labeled "inactive", the rest are labeled "active". The centralized K-means is run on the aggregate data over all active nodes (using a randomly chosen set of initial centroids). Next, P2P K-means is run over only the active nodes (each node uses the same set of initial centroids as the centralized K-means).

After τ clock ticks, the communication cost and accuracy of P2P K-means is tallied. Next, 10 percent of the inactive nodes are activated and the same number of previously active nodes are deactivated randomly from the network. For each of the newly activated nodes, all of their dataset is replaced by new data. In the stationary case, the new data comes from the same distribution as of the original data. In the non-stationary case, each time a new data generator is used with updated Gaussian means that vary randomly within ± 1 of the previously used distribution means. At the $(\tau + 1)$-th clock tick, 10 percent of the active nodes are selected randomly and each of these nodes have 20 percent of their dataset replaced by the new data (in the non-stationary case, updated Gaussian means are used as described before). The centralized K-means is run on the aggregate new data over all active nodes with initial centroids the same as the final centroids from the previous run of centralized K-means. Finally, P2P K-means is run over only active nodes where each node uses initial centroids the same as the final centroids at that node from the previous run (provided the node was active during the previous run, otherwise, the node sets its initial centroids as described in Section 3.3).

We seek to answer the following questions.

- Does our algorithm incur a large error at the end of any period?

- How does the accuracy and communication cost change over time?

- What impact does increasing the network size have on the previous two?

- What impact does stationary vs. non-stationary have on the previous two?

5.2.1 Accuracy Figure 4 shows the accuracy at the end of each period (three different period lengths, 50, 100, and 200 global clock ticks) with 500 total nodes
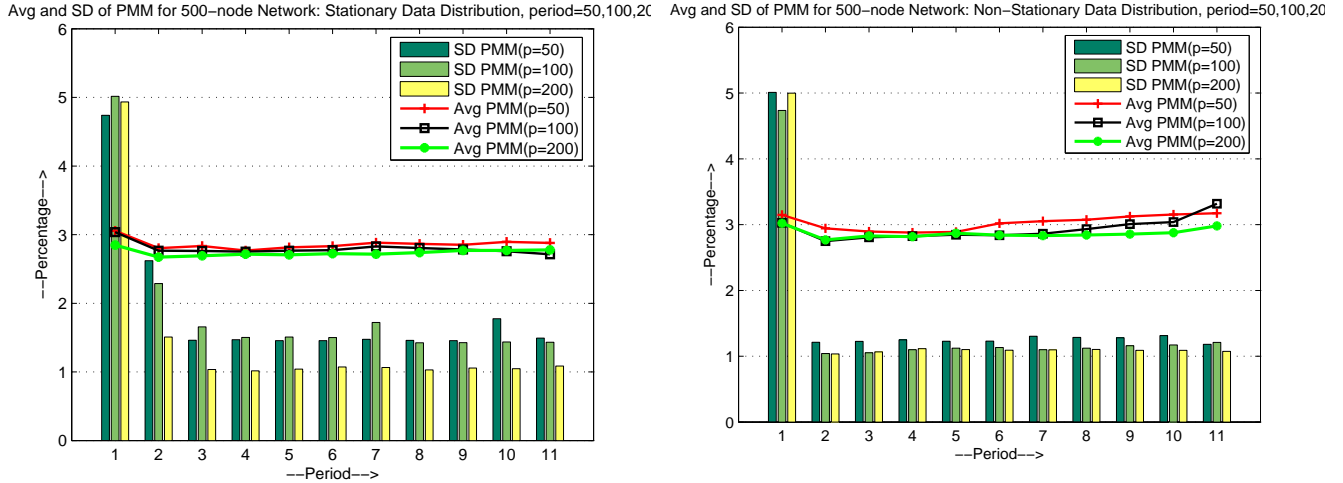
Figure 4: Variation of the average and standard deviation of PMM over time ($\tau = 50, 100,$ and 200), 500 nodes, and stationary data distribution.

Figure 5: Variation of the average and standard deviation of PMM over time ($\tau = 50, 100$ and 200), 500 nodes, and non-stationary data distribution.

and a stationary data distribution. "Avg PMM(p=50)" denotes the average PMM with $\tau = 50$ and "SD PMM(p=50)" denotes the standard deviation. As evident from the figure, our algorithm achieves very good accuracy (less than 3% average PMM) at the end of each period. Also, the average PMM appears relatively flat – accuracy does not change much over time. Moreover, the curves for different time periods are very similar showing that the frequency of data change does not seem to affect accuracy. The standard deviation curves show that a significant amount of instability exists during the first period, but dampens down quickly.

Figure 5 shows the accuracy under the same conditions as Figure 4 except a non-stationary distribution is used. The results are very similar in nature with PMM values slightly higher. Interestingly, the standard deviation is slightly lower than in the stationary data distribution case.

To check the scalability of our algorithm, we ran experiments with uniform and non-uniform node assignments over a network with size varying from 100 nodes to 1000 nodes. The period size is fixed at 100 ticks and both a stationary and non-stationary data distributions are used. Figure 6 shows the accuracy at the end of eleven periods of length 100 with a stationary data distribution. "Avg PMM(U)" denotes the average PMM over a uniform node assignment and "SD PMM(U)" denotes the standard deviation. It is evident that our algorithm shows excellent scalability and achieves high accuracy with respect to centralized K-means even for

dynamically changing data and network irrespective of network size. Figure 7 shows the corresponding result when new data is coming from a non-stationary distribution. The average PMM value is slightly higher compared to the stationary distribution case, but scalability of the algorithm is, once again, very good. These results provide evidence that our algorithm's performance is not affected by the network size.

5.2.2 Communication Complexity We measure, for each time period, the average number of bytes transmitted over all nodes ("Avg Comm") and the standard deviation ("SD Comm"). Figure 8 shows the communication complexity over each time period (period lengths 50, 100, 200 global clock ticks) with 500 nodes and a stationary data distribution. The communication cost is initially very high during the first period, since sites start from randomly assigned centroids at the beginning and execute most of their total iterations during this time, thus needing high volume of message exchange. After that it decreases sharply and becomes roughly constant after third period. Except for a start-up spike, the average amount of communication per node remains stable thereafter. The relatively high standard deviations are likely due to the fact that some nodes have many more neighbors than others.

Figure 9 shows the communication cost for a non-stationary distribution. As was the case with the accuracy figures, the result is very similar to the previous stationary distribution figure.
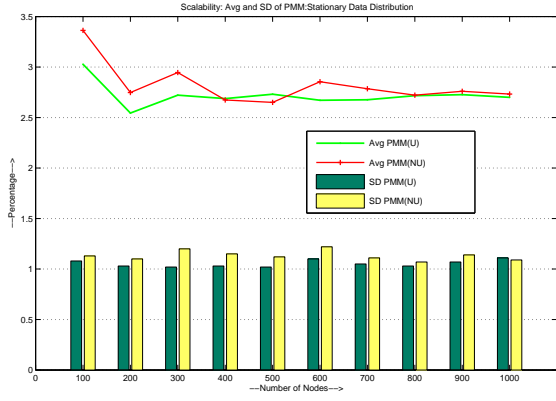
Scalability: Avg and SD of PMM:Stationary Data Distribution

Avg and SD of Communication Cost for 500−node Network: Stationary Data Distribution, period=50,100,200

Bytes for period 1

Avg Bytes(p=50)=18048
Avg Bytes(p=100)=31955
Avg Bytes(p=200)=63444

SD Comm(p=50)
SD Comm(p=100)
SD Comm(p=200)
Avg Comm(p=50)
Avg_Comm(p=100)
Avg_Comm(p=200)

Avg PMM(U)
Avg PMM(NU)
SD PMM(U)
SD PMM(NU)

—Number of Nodes—>

—Period—>

—Percentage—>

—Bytes—>

Figure 6: Variation of the average and standard deviation of PMM with number of nodes in the dynamic network: stationary data distribution, $\tau = 100$.
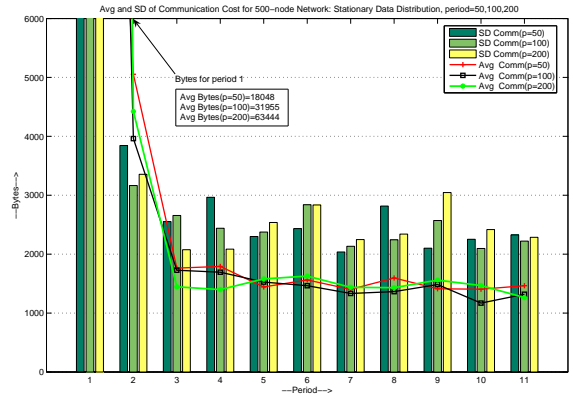
Figure 8: Variation of the average and standard deviation of communication cost over time (periods of length 50, 100, 200); 500 sites and stationary data distribution.
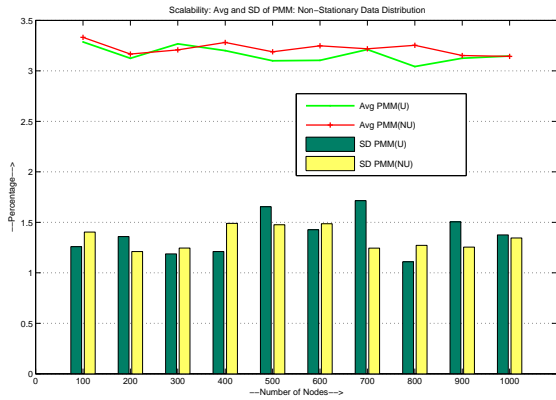
Scalability: Avg and SD of PMM: Non-Stationary Data Distribution

Avg and SD of Communication Cost for 500−node Network: Non−Stationary Data Distribution, period=50,100,200

Bytes for period 1

Avg Bytes(p=50)=17359
Avg Bytes(p=100)=33951
Avg Bytes(p=200)=63876

SD Comm(p=50)
SD Comm(p=100)
SD Comm(p=200)
Avg Comm(p=50)
Avg_Comm(p=100)
Avg_Comm(p=200)

Avg PMM(U)
Avg PMM(NU)
SD PMM(U)
SD PMM(NU)

—Number of Nodes—>

—Period—>

—Percentage—>

—Bytes—>

Figure 7: Variation of the average and standard deviation of PMM with number of nodes in the dynamic network: Non-stationary data distribution, $\tau = 100$.
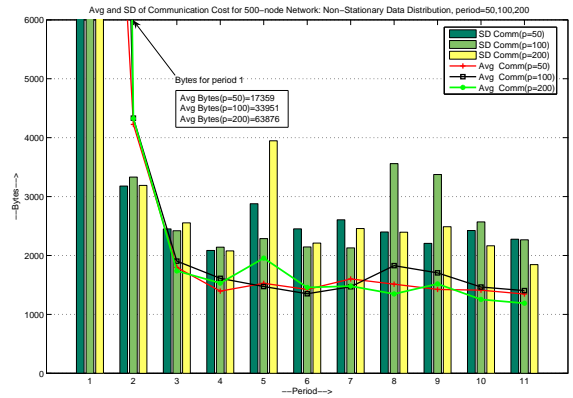
Figure 9: Variation of the average and standard deviation of communication cost over time (periods of length 50, 100, 200); 500 sites and non-stationary data distribution.

6 Conclusions and Future Work

We have considered the problem of K-means clustering on data distributed over a large, dynamic network, the data or the network itself may change. We assume the network to be peer-to-peer (does not have any special servers). Centralizing all the data to a single machine to run a centralized K-means is not an attractive option. Ideally, the algorithm ought to (1) not require global synchronization, (2) be communication-efficient, and (3) be robust to network or data change.

We have described a locally synchronous algorithm for this environment, P2P K-means. Nodes only communicate and synchronize with their topologically immediate neighbors. We conducted experiments using synthetic, 10D data generated from a mixture of 10 Gaussian distributions with 10% added uniform noise. We ran P2P K-means over a simulated environment of up to 1000 nodes.

We first considered a static environment (data and network do not change) and measured the accuracy (with respect to K-means run on the data once centralized) and communication cost of P2P K-means. We observed high accuracy (less than 3% of points per node misclassified on average) and very good scalability. We did not observe the method of assigning data points to nodes (uniformly vs. non-uniformly) to have significant impact on accuracy. However, the assignment method did have a significant impact on communication cost. The number of bytes received per node increases slowly with network size for uniform assignment. And, the cost increases more sharply for non-uniform assignment (although still appears linear).

We also conducted experiments in a dynamic environment. We observed very good accuracy (less than 3.5% misclassified on average) which remained stable as the network evolved. Moreover, changing the time period or the data distribution (stationary vs. non-stationary) did not impact accuracy significantly. Increasing the number of peers (from 100 to 1000) did not appear to affect accuracy significantly providing evidence that the algorithm exhibits good scalability.

In conclusion, we feel that P2P K-means exhibits very good accuracy and is robust in the presence of network and data change.

Acknowledgments

References

[1] Akyildiz I., Su W., Sankarasubramanium Y., and Cayirci E. Wireless Sensor Networks: A Survey. *Computer Networks*, 38:393–422, 2002.

[2] Androutsellis-Theotokis S. and Spinellis D. A Survey of Peer-to-Peer File Sharing Technologies. White paper, Electornic Trading Research Unit (ELTRUN), Athens University fo Economics and Business. citeseer.ist.psu.edu/androutsellis-theoto02survey.html, 2002.

[3] Bandyopadhyay S., Giannella C., Maulik U., Kargupta H., Liu K., and Datta S. Clustering Distributed Data Streams in Peer-to-Peer Environments. *Information Sciences (in press)*, 2006.

[4] Bawa M., Gionis A., Garcia-Molina H., and Motwani R. The Price of Validity in Dynamic Networks. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 515–526, 2004.

[5] Boyd S., Ghosh A., Prabhakar B., and Shah D. Gossip Algorithms: Design, Analysis, and Applications. In *Proceedings of IEEE Infocom'05*, volume 3, pages 1653–1664, 2005.

[6] Clemente J., Defago X., and Satou K. Asynchronous Peer-to-Peer Communication for Failure Resilient Distributed Genetic Algorithms. In *Proceedings of the 15th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'03)*, pages 769–773, 2003.

[7] Dhillon I. and Modha D. A Data-clustering Algorithm on Distributed Memory Multiprocessors. In *Proceedings of the KDD'99 Workshop on High Performance Knowledge Discovery*, pages 245–260, 1999.

[8] Eisenhardt M., Muller W., and Henrich A. Classifying Documents by Distributed P2P Clustering. In *Proceedings of Informatik 2003, GI Lecture Notes in Informatics, Frankfort, Germany*, pages 286–291, 2003.

[9] Forman G. and Zhang B. Distributed Data Clustering Can Be Efficient and Exact. *SIGKDD Explorations*, 2(2):34–38, 2000.

[10] Greenwald M. and Khanna S. Power-Converving Computation of Order-Statistics Over Sensor Networks. In *Proceedings of PODS'04*, pages 275–285, 2004.

[11] Han J. and Kamber M. *Data Mining: Concepts and Techniques*. Morgan Kaufman Publishers, San Francisco, CA, 2001.

[12] Kargupta H. and Sivakumar K. Existential Pleasures of Distributed Data Mining. In *Data Mining: Next Generation Challenges and Future Directions, edited by H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha, MIT/AAAI Press*, pages 3–26, 2004.

[13] Kempe D., Dobra A., and Gehrke J. Computing Aggregate Information using Gossip. In *Proceedings of the 44th IEEE Symposium on Foundations of Computer Science (FoCS)*, pages 482–491, 2003.

[14] Kowalczyk W., Jelasity M., and Eiben A. Towards Data Mining in Large and Fully Distributed Peer-To-

Peer Overlay Networks. In *Proceedings of BNAIC'03*, pages 203–210, 2003.

[15] Krivitski D., Schuster A., and Wolff R. A Local Facility Location Algorithm for Sensor Networks. In *Proceedings of the International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 368–375, 2005.

[16] Lamport L. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, 1978.

[17] Muller W., Eisenhardt M., and Henrich A. Efficient Content-Based P2P Image Retrieval Using Peer Content Descriptions. In *Proceedings of Internet Imaging V*, pages 57–68, 2004.

[18] Wolff R. and Schuster A. Association Rule Mining in Peer-to-Peer Systems. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 34(6):2426–2438, December 2004.