Distributed Data Mining in Peer-to-Peer Networks

Peer-to-peer (P2P) networks are gaining popularity in many applications such as file sharing, e-commerce, and social networking, many of which deal with rich, distributed data sources that can benefit from data mining. P2P networks are, in fact, well-suited to distributed data mining (DDM), which deals with the problem of data analysis in environments with distributed data, computing nodes, and users. This article offers an overview of DDM applications and algorithms for P2P environments, focusing particularly on local algorithms that perform data analysis by using computing primitives with limited communication overhead. The authors describe both exact and approximate local P2P data mining algorithms that work in a decentralized and communication-efficient manner.

Souptik Datta, Kanishka Bhaduri, Chris Giannella, and Hillol Kargupta
University of Maryland, Baltimore County

Ran Wolff
Technion – Israel Institute of Technology

LANs, peer-to-peer (P2P) networks, mobile ad hoc wireless networks (Manets), and other pervasive distributed computing environments often include distributed data and computation sources. Data mining in such networks naturally calls for proper utilization of these distributed resources in an efficient, decentralized manner. Data mining algorithms that require substantial communication among the nodes, synchronous computing nodes, and complete centralized control have difficulty scaling in such distributed environments. Moreover, privacy concerns and resource issues in multiparty applications often dictate that data sets collected at different sites be analyzed in a distributed fashion without collecting everything to central sites. Most off-the-shelf data mining products are designed to work as monolithic centralized applications, downloading relevant data to centralized locations to perform data mining operations, but this centralized approach doesn't work well in many emerging distributed data mining applications.

Distributed data mining (DDM) offers an alternate approach to address this problem of mining data using distributed resources. DDM pays careful attention to distributed data, computing, communication, and human resources to use them in a near-optimal fashion. Distributed P2P systems are emerging as a solution of choice for a new breed of applications such as file sharing, collaborative movie and song scoring, electronic commerce, and surveillance using sensor networks. DDM is gaining increasing attention in this domain for advanced data-driven applications.

This article presents an overview of efforts to use DDM technology in P2P networks. Our goal is to present a high-level introduction to this field with pointers for

 Published by the IEEE Computer Society 1089-7801/06/$20.00

further exploration. We illustrate the ideas using some exact and approximate DDM algorithms.

P2P Data Mining: Why Bother?

The term "data mining" generally implies analysis of large databases to detect useful patterns. In most commercial applications, data mining systems run as vertical applications on top of large centralized data warehouses. Although this model serves well for many applications, including customer-relationship management and financial fraud detection, many emerging domains such as P2P systems call for new thinking. High-speed network connectivity and cheap digital storage and data-recording devices are increasing the popularity of P2P networks such as the e-Mule and Kazaa file-sharing networks, which are based on point-to-point connections without central servers. Such networks host a substantial array of widely varying data, collected from different sources and distributed over large numbers of peers. If integrated, that data would present a valuable repository for mining, but computational resource constraints, privacy issues, and so on make it difficult to integrate distributed data into one repository.

Many popular Web servers use Web-mining applications to analyze and track users' click-stream behavior. Now imagine client-side Web mining that did the same for Web site visitors (rather than host servers) by analyzing the browsing histories of many users connected via a P2P network. Today, site visitors have no direct access to the results of Web mining algorithms running on the servers, but a client-side P2P Web-mining algorithm could empower visitors with click-stream data mining for advanced applications such as P2P search, interest-community formation, and P2P-based electronic commerce. Figure 1 illustrates such a case, in which the application categorizes visited URLs according to three subjects (movies, baseball, and hurricanes) by exchanging information with other peers. Clearly, maintaining users' privacy will be an important issue in such applications, and the field of privacy-preserving DDM might offer some solutions.[1]

Although most current P2P networks deal primarily with file-sharing applications (for music and movies, for example), in this article, we consider a P2P network to be any large, serverless network with point-to-point connections. This opens up other potential application areas for P2P data mining, including mobile ad hoc networks (Manets), sensor networks, and federated databases without central coordinator sites. These application areas differ in some respects, but all would benefit from data-analysis and mining algorithms that could operate effectively on dynamic, large-scale P2P networks.

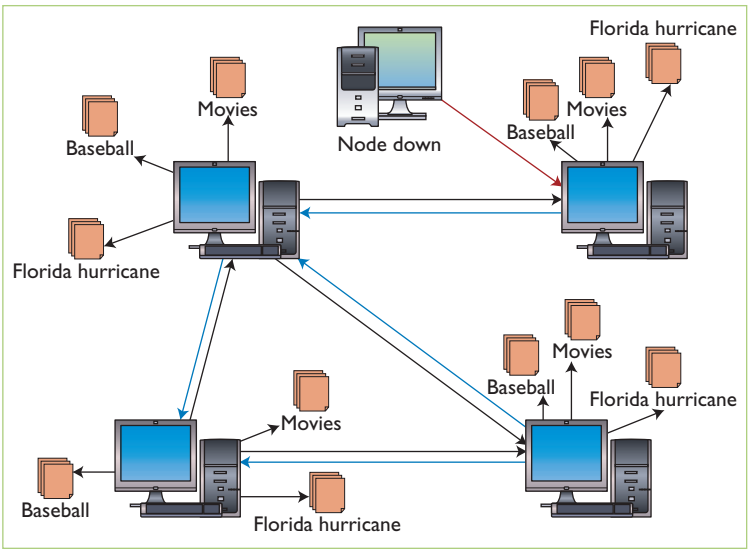

Figure 1. Example client-side P2P Web-usage-mining application. Client-side P2P Web-mining algorithms could enable an application to cluster every peer's browsing history into three categories (baseball, hurricane, movies) based on information exchanged with other peers. The application would update each peer's profile according to the number of visited Web pages in each category, and use the profile to form Internet groups of peers with common interests.

The computational environment in P2P systems differs dramatically from those for which traditional centralized data-mining algorithms were intended. Some important requirements include:

- *Scalability.* Modern P2P systems can include millions of peers, which makes scalability the foremost requirement for data-mining algorithms. Computational and communication (bandwidth) resource requirements should ideally be independent of system size, or at least bounded by a function that grows slowly with increases in system size.
- *Availability.* Because data can change at some peers during computation, the algorithms must work incrementally and should be able to report partial, ad hoc solutions at any time.
- *Asynchronism.* Algorithms developed for P2P system shouldn't depend on global synchronization; any attempt to synchronize an entire network is likely to fail due to connection latency, limited bandwidth, or node failure.
- *Decentralization.* Although some P2P systems still use central servers for various needs, next-

P2P Data Mining's Evolution

P2P data mining is a new field that has grown out of distributed data mining (DDM), which itself is a fairly new research area. DDM has evolved over the past five to 10 years as an effort to introduce distributed versions of many standard data mining algorithms, such as association-rule mining, Bayesian network learning, and clustering. However, most such efforts assume a stable network and data, and so they can't be applied directly to P2P network conditions.

Given that researchers can, in principle, develop efficient complex algorithms by applying efficient primitives, many P2P data-mining efforts have focused on developing primitive operations (average, sum, max, random sampling, and so on), laying a foundation for more sophisticated data analysis and mining algorithms Wojtek Kowalczyk and colleagues developed the newscast model and used it to calculate mean of data distributed in P2P networks.[1] They relied on empirical accuracy results rather than guaranteed correctness. In another approach using an epidemic model of computation, David Kempe and colleagues investigated gossip-based randomized algorithms for computing aggregate information and proved that aggregate estimation error probability is zero when algorithms run uninterrupted.[2] Mayank Bawa and colleagues developed an approach for evaluating similar primitives within a specified error margin.[3]

In contrast to these approaches, which all require resources that scale directly with system size *local algorithms*[4] can compute results and make definite claims regarding correctness using information from just a handful of nearby neighbors in a P2P system. The resources required by such algorithms are often independent of system size, which presents obvious benefits for scalability and fault tolerance. However, local algorithms apply to a limited class of functions.

Various researchers have focused on developing local algorithms for primitive operations. Mortada Mehyar and colleagues used a Laplacian-based approach to compute the average of data points distributed over a P2P network.[5] Ran Wolff and Assaf Schuster developed a local algorithm for computing the majority vote over a P2P network.[6] They've subsequently used the primitive to develop local algorithms for more complicated problems, including *K*-facility location,[7] which is finding the *k*-best gateways for information exchange in a sensor network, and association-rule mining,[6] (deriving associations between attributes when data is homogeneously distributed in a P2P network). Ran Wolff, Kanishka Bhaduri, and Hillol Kargupta have also proposed algorithms for monitoring *K*-means clustering in P2P networks.[8] Although Brian Babcock and Chris Olston addressed a similar problem earlier, their approach assumed a centralized coordinator site and a hierarchical topology for effective monitoring and global conflict resolution.[9] They also focused on detecting change in top *k*-ranked entities in a distributed scenario, whereas this work (described in the main text) is designed to monitor any shift in existing clusters.[8] Souptik Datta and colleagues have focused on developing approximate local algorithms for solving data-mining problems such as *K*-means clustering in P2P networks.[10]

References

1. W. Kowalczyk, M. Jelasity, and A. Eiben, "Towards Data Mining in Large and Fully Distributed Peer-to-Peer Overlay Networks," *Proc. 15th Belgian-Dutch Conf. Artificial Intelligence* (BNAIC 03), Univ. of Nijmegen Press, 2003, pp. 203–210.
2. D. Kempe, A. Dobra, and J. Gehrke, "Computing Aggregate Information using Gossip," *Proc. 44th IEEE Symp. Foundations of Computer Science* (FoCS), IEEE CS Press, 2003, pp. 482–491.
3. M. Bawa et al., "The Price of Validity in Dynamic Networks," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, ACM Press, 2004, pp. 515–526.
4. B. Awerbuch et al., "Compact Distributed Data Structures for Adaptive Network Routing," *Proc. 21st ACM Symp. Theory of Computing* (STOC), ACM Press, 1989, pp. 479–489.
5. M. Mehyar et al., "Distributed Averaging on a Peer-to-Peer Network," *Proc. IEEE Conf. Decision and Control*, IEEE CS Press, 2005.
6. R. Wolff and A. Schuster, "Association Rule Mining in Peer-to-Peer Systems," *IEEE Trans. Systems, Man, and Cybernetics, Part B*, vol. 34, no. 6, 2004, pp. 2426–2438.
7. D. Krivitski, A. Schuster, and R. Wolff, "A Local Facility Location Algorithm for Sensor Networks," *Proc. Int'l Conf. Distributed Computing in Sensor Systems* (DCOSS 05), Springer, 2005, pp. 368–375.
8. R. Wolff, K. Bhaduri, and H. Kargupta, "Local l2-Thresholding Based Data Mining in Peer-to-Peer Systems," *Proc. 2006 SIAM Conference on Data Mining* (SDM 06), SIAM Press, 2006, pp. 430-441.
9. B. Babcock and C. Olston, "Distributed Top-*K* Monitoring," *Proc. ACM SIGMOD 2003 Int'l Conf. Management of Data*, ACM Press, 2003, pp. 28–39.
10. S. Datta, C. Giannella, and H. Kargupta, "K-Means Clustering over Large, Dynamic Networks," *Proc. 2006 SIAM Conf. Data Mining* (SDM 06), SIAM Press, 2006, pp. 153–164.

generation P2P algorithms might need to run without any coordinator (server or router) and calculate results in-network rather than collect data in a single peer.

- *Fault tolerance.* Given that multiple peers can leave or join a P2P system at any given moment, algorithms must be robust enough to let systems recover from peer failures and subsequent data loss.
- *Privacy.* Privacy is an enabling factor that lets users contribute data without fearing consequences such as revealing sensitive information. This is particularly important with multiparty applications, such as P2P network surveillance for threat management, community formation, and match-making.
- *Security and trust.* As with any large distributed system, security is a crucial issue in P2P data mining because exchanging information with other peers can increase a peer's vulnera-

bility to network threats such as denial of service or selfish behavior. Trust management is also likely to be an important issue because users of P2P systems must deal with peers they might not have directly interacted with otherwise. In a mobile vehicular ad hoc network (Vanet), for example, a vehicle might need to communicate with a group of nearby vehicles that changes every few minutes.

We now turn our attention to algorithms for P2P data mining, focusing particularly on *local* algorithms, which perform computations by communicating information only with neighbor nodes. Approaching these algorithms from a computational perspective, we distinguish between exact and approximate approaches.

Algorithms for P2P Data Mining

A P2P algorithm is unlikely to scale if it requires that every node communicate with every other node in the network. Unfortunately, many data mining tasks on P2P networks demand this very situation. Consider, for example, a P2P network in which every node has a data tuple and our goal is to compute the distance matrix (in some metric space) where the (i, j)th entry represents the distance between tuples stored at the ith and jth nodes. To compute this in an exact manner, we have little choice but to exchange information between every possible pair of peers. One solution is to make sure that every node talks to every other node in the network and computes the corresponding pair-wise distance, but that approach might not scale in P2P networks with millions of nodes. On the other hand, we might be able to approximate the problem and eliminate the need for such an extensive communication load.[2] For example, we could identify only the significant entries of the distance matrix and develop an efficient P2P algorithm that doesn't necessarily require exchanging information between every pair.[2] Many other problems are inherently decomposable and don't require that every node directly share data with every other node in the network.

The notion of *locality* is very important in developing P2P algorithms because it facilitates P2P data mining in a scalable manner through a collection of local computations. Consider a P2P network represented by a graph in which nodes represent peers and edges represent the links between them. Let $G = (V; E)$ be the graph representing the network in which V denotes the set of nodes and E represents the edges between them. The α-neighborhood of a vertex $v \in V$ is the collection of vertices at distance α or less from it in G: $\Gamma_\alpha(v; V) = \{u | dist(u; v) \leq \alpha\}$, where $dist(u; v)$ denotes the length of the shortest path between u and v, and a path's length is defined as the number of edges in it. Let each node $v \in V$ store a data set X_v. An α-local query by some vertex v is a query whose response can be computed using some function $f(X_\alpha(v))$, where $X_\alpha(v) = \{X_v | v \in \Gamma_\alpha(v; V)\}$ and the response size is bounded by some constant c. An algorithm is α-local if it never requires computation of a β-local query such that $\beta > \alpha$. When we speak of local algorithms, we thus imply α-local algorithms in which α is a small constant.

In this article, we can broadly classify local algorithms under two categories:

- *Exact local algorithms* produce the same results as a centralized algorithm.
- *Approximate local algorithms* offer approximations of the results that a centralized algorithm would produce.

To compare the two, we begin by discussing an exact local algorithm for majority voting, which we can use as a primitive for monitoring a K-means clustering[3] — in which the goal is to divide objects into a fixed number of clusters K while minimizing the sum of the average distances to the cluster centroids over all clusters. We then describe the approximate local algorithm we developed to offer an alternative solution for incrementally computing a K-means clustering.

Exact Local Algorithms

In discussing exact local algorithms, we assume an overlay tree topology is maintained over the P2P network. Current P2P networks commonly use such overlay information. This tree structure guarantees that the algorithms produce correct answers, given that peers communicate directly only with their neighbors in the tree.

Majority voting. Building on work by Ran Wolff and Assaf Schuster, the majority-voting problem serves as a nice primitive from which we can develop more complicated exact local algorithms, such as frequent-item-set mining.[4] In this problem, each peer P_i holds a number b_i (0 or 1) and a threshold $\tau > 0$ (the same threshold for all peers). The peers seek to collectively determine whether $\Sigma_i b_i$ is above $n\tau$ where n is the number of peers in the network. We

can easily extend the approach described here to two other general scenarios:

- Each peer has a real number x_i and the collective goal is to decide whether $avg(x_i) > \tau$.
- Each peer has a pair of real numbers x_i, y_i and the collective goal is to decide whether $\Sigma_i x_i - (\Sigma_i y_i)\tau > 0$.

For simplicity we don't describe these here, but we'll use them later when discussing the exact local algorithm for *K*-means monitoring and frequent-item-set mining.

Peer P_i communicates only with its neighbors and uses the information it receives to maintain estimates of both the global sum S_i and the number of nodes in the network C_i. Based on these estimates, P_i believes that the majority threshold is met (that is, whether $\Sigma_i b_i$ is above $n\tau$) if $S_i - C_i\tau > 0$; otherwise, it believes the majority is unmet. We call this P_i's *threshold belief*.

Let S_{ji} denote the most recent sum estimate that P_i received from its neighbor P_j. Likewise, we define C_{ji} as P_j's most recent estimate of total nodes. The crux of the approach lies in deciding whether P_i needs to send a message (composition explained later) to P_j. It must send a message unless it can be certain that it doesn't have any information that will change P_j's threshold belief. To make this decision, P_i must estimate P_j's sum and count, based on the information it knows for certain that P_j has — namely, the information that P_i sent to and received from it: S_{ij};C_{ij} and S_{ji}, C_{ji}.

If P_i estimates that P_j believes the threshold to be met is $S_{ij} + S_{ji} - (C_{ij} + C_{ji})\tau > 0$, it doesn't need to send a message if its own estimate would only strengthen the belief (that is, $S_{ij} + S_{ji} - (C_{ij} - C_{ji})\tau \leq S_i - C_i\tau$). In this case, P_i could be certain that it had no information that could change P_j's threshold belief. Similar reasoning applies if P_i estimates that P doesn't believe the threshold to be met. If P_i decides to send a message, it sends all of its information except that received from P_j about the global sum (that is, S_{ij} is set to $b_i + \Sigma_{\ell \neq j \in N_t} S_{\ell i}$) and the global count (that is, C_{ij} is set to $1 + \Sigma_{\ell \neq j \in N_t} C_{\ell i}$).

This approach is naturally robust to data and network changes. If its data changes (b_i flips), P_i recomputes S_i and C_i and applies the conditions explained in the preceding paragraph to all its neighbors. If a neighbor P_j drops out of the network, P_i recomputes S_i and C_i without S_{ji} and C_{ji} and applies the same conditions to all remaining neighbors.

Frequent-item-set mining. The majority-voting primitive leads directly to a local algorithm for frequent-item-set mining of data distributed over P2P networks.[4] Frequent-item-set mining has gained great popularity for analyzing centralized data (for example, determining customers' buying patterns), and we believe it also has interesting applications in analyzing data distributed over P2P networks.[5] For example, imagine a P2P music-sharing network in which each peer records the artist name for all songs downloaded. For recommendation purposes, it might be useful to determine which artists X and Y tend to be downloaded together. If a user downloaded a song by X, the system could inform the user that Y's songs might be of interest.

Generally speaking, each peer is said to have an item set if it holds all its items (in the previous example, a peer that downloaded songs by artists X and Y would hold itemset {X}, itemset {Y}, and itemset {X,Y}). An item set's *support* over an entire network is the total number of peers holding it. The goal of frequent-item-set mining is to find all *frequent* item sets — those whose level of support is above $n\tau$.

Peers can undertake a majority vote to determine whether any given item set is frequent. To find all frequent item sets, peers engage in the following procedure.

For each *size-one* item set (a set containing one item), the peers engage in a majority vote to determine if they're frequent. Next, peer P_i considers the *size-two* item sets. For each size-two item set I, if the peer isn't currently running a majority vote for I, but is confident that all the size-one subsets of I are frequent, the peer initiates a majority vote for I, which could be frequent though not currently under consideration. If the peer is currently running a majority vote for I and is confident that one of its size-one subsets is infrequent, it stops running the majority vote. If one of I's subsets is infrequent, then I must also be infrequent, which means there's no need to keep it under consideration.

In this way, peers continuously initiate and terminate majority votes for item sets. Provided that the data and network remain static for long enough, however, all majority votes will eventually terminate at which point all peers will know the precise set of frequent item sets.

Monitoring a *K*-means clustering. Large-scale applications collect P2P system status data (such as network traffic or user-specific data) as part of their daily routines. System administrators or common

users are often interested in using this status data to build complex models of the collected data, which give us insight into system behavior, user profiles, network conditions, and so on. Some of these complex predicates includes *K*-means, eigenvectors, eigenvalues, and means of the data. Keeping these models up to date is very important because data can change frequently, and the model needs to follow the changes to accurately reflect current system status. For volatile scenarios in which data and topology change frequently, periodically centralizing the data to build new data models can be expensive. The periodic-update scheme has two major disadvantages. The model is certain to be inaccurate during the period when the distribution has changed but the model has yet to be rebuilt. Moreover, when the distribution changes infrequently (static periods), this computation mode unnecessarily rebuilds the same model, thereby wasting valuable resources. We could potentially monitor these predicates if we had a local algorithm that let each peer quickly identify that the current model no longer represented its data.

Although this work doesn't solve the problem of computing a *K*-means clustering in a distributed setting, we've developed an algorithm for monitoring *K*-means clusters of data distributed over P2P networks.[3] The *K*-means monitoring algorithm has two major parts:

- an exact local algorithm that monitors the data distribution to trigger new runs of the *K*-means algorithm as soon as the current centroids no longer represent the data, and
- sample data collected from the network and transferred to a central location to compute the centroids using a standard *K*-means algorithm.

Simply put, the local algorithm raises an alert if the centroids need updating. Any peer for which the alert flag is set sends a sample of its local data to its parent up the tree topology that's already been laid on top of the network. Once the root node has all the data, it executes a new run of *K*-means and ships the new centroids back to all peers. Henceforth, we focus only on the monitoring part because computing the new centroids and flooding the network with them is a relatively simple problem. For ease of discussion, we assume a 2D problem, but extending it to a multidimensional case is straightforward.

Each peer P_i begins with a 2D local data set S_{it} at time t, as well as a list of the current centroids (each of which is a 2D vector) computed by running the centralized *K*-means algorithm on the complete data set. We can think of this initial centroid computation as an offline task that isn't part of the distributed algorithm. That said, this step is necessary so that all peers start with the same reference point. P_i computes an average of all its data points as a vector with one average value for each dimension. It then computes the difference between the local average vector and the current centroids to get a vector we call $\boldsymbol{X}_i$, the *knowledge* of P_i. The monitoring problem is to figure out if the average $\boldsymbol{X}_i$ of all the peers exceeds a user-defined threshold ε. Given that $\boldsymbol{X}_i$ is a vector, we use a standard metric, popularly known as the L2 norm, to find the value of $\boldsymbol{X}_i$. By definition, the L2 norm of a vector is computed as the square root of the sum of its individual components. Thus, the problem becomes: Is the L2 norm of the average vector less than a user-defined threshold ε? If so, the algorithm doesn't need to do anything; otherwise, the current centroids no longer accurately represent the data, so a new round of *K*-means is necessary to accurately represent the data.

For the 2D case, the vector's L2 norm is a circle (sum of squares of individual components), and we can view the problem as a way to determine

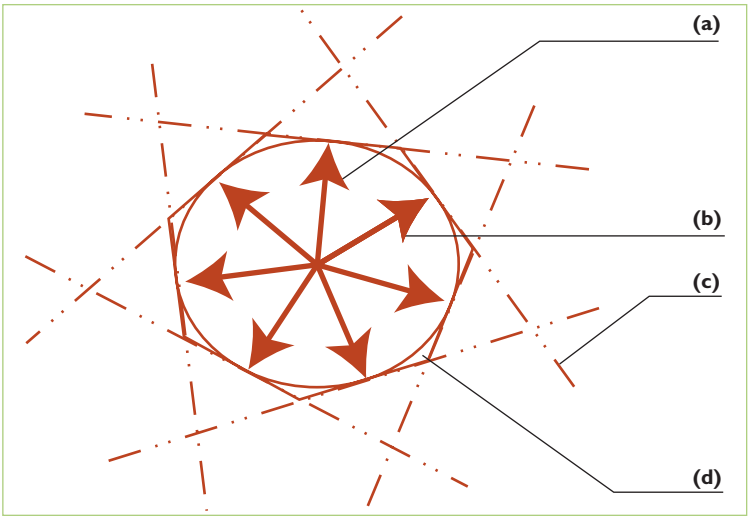

Figure 2. Peer node's global-average computation. (A) Each peer begins by trying to bound its local knowledge vector within a circle of radius ε. (B) Seven evenly spaced vectors define tangent planes that partition the domain outside the circle into half spaces. (C) The borders of the seven half spaces define polygons in which the circle is circumscribed.This polygon is our approximation of the circle. (D) The area between the circle and the union of the half spaces is our "flooding zone." If any of the vectors is in this region, the algorithm forces the peer to flood the network with its local data.

whether the global average is inside or outside a circle of radius ε. For each peer P_i to decide this locally, the algorithm uses two auxiliary vectors (other than its local knowledge X) — an agreement vector (Y) for each pair between P_i and P_j and a withheld-knowledge vector ($Z = X - Y$) between each P_i and P_j. Peers then use the following decision rule: If, for every peer P_i and all its neighbors P_j, both the agreement and the withheld knowledge are inside any convex shape, the global average is as well.

That leaves us three cases to solve:

- In case 1, all three vectors (X, Y, and Z) are inside the circle.
- In case 2, all vectors are outside.
- In case 3, some are inside and some are outside.

For case 1, we know from the rule that the global average is inside if Y and Z are. Hence, if the knowledge agrees (is inside), the peer knows that its knowledge is correct and that it can cease sending messages.

Case 2 presents a different scenario. If we assume that, for any peer P_i and any neighbor P_j, both sets of knowledge are outside the circle, we must still solve to see if the average is inside (the knowledge vectors could be in opposite directions). To do so, we use a set of tangent lines. For each peer P_i with neighbors P_j, if it can identify a tangent line such that its own knowledge is outside the circle with respect to the line and the agreement and withheld knowledge are also outside with respect to the same tangent line, the global average is also guaranteed to be outside the circle. We can still use the decision-rule here because the tangent line slices the space into sets of half spaces that are each convex regions. If for all peers and all neighbors the agreement and withheld knowledge are outside, the global average is therefore outside as well. Given that P_i's current knowledge matches the global average, it needn't communicate any further.

In case 3, P_i must communicate with its neighbors because the rule can't help it decide whether the global average is inside or outside. Figure 2 illustrates the details of the peer's computation using the circle of radius ε. Seven tangent lines exist, and for every new point that changes its local knowledge, P_i must test the three conditions stated above (by projecting its local vectors along the tangent line if necessary) until it can decide, based on the local decision rule, that it needs no more communication. If the point is in the small space between the circle and the tangent lines, the peer must communicate all its local knowledge because this case can't be solved locally. We can make this space arbitrarily small with additional tangent lines, but the computation cost increases for each peer with the number of lines.

Approximate Local Algorithms

Although many existing local algorithms can eventually achieve exact solutions, they're usually limited to problems that can be reduced to threshold predicates. In contrast, approximate local algorithms can solve more-complicated problems, such as clustering, with approximate solutions that closely estimate the actual solutions. Here, we propose an approximate P2P K-means clustering algorithm for data that's homogeneously distributed over a network (every node observes the same feature set).

Martin Eisenhardt and colleagues previously addressed K-means clustering in P2P networks through an algorithm that uses a probe-and-echo mechanism to produce an exact solution.[6] However, their approach requires synchronization of all peers at each iteration and doesn't account for network or data changes. Our P2P K-means algorithm relaxes the global-synchronization requirement and addresses the dynamic aspects of a typical P2P network. Rather than guarantee that the centroids at each peer are the same, it's designed to ensure that they're all close to the centroids produced by a centralized algorithm.

Algorithm Description

Our iterative algorithm[7] is based on message exchange between directly connected peers. It assumes that every peer knows its immediate neighbor peers, the termination threshold, and the value of K (number of clusters). The algorithm is initiated with a set of randomly chosen centroids distributed over all peers. In each iteration, every peer runs a two-step process.

The first step is identical to an iteration of the standard K-means algorithm, in which the ith peer P_i assigns each of its data points to its nearest cluster centroids. After assigning all the data points to their respective clusters, P_i calculates K local cluster centroids by taking the average of data points belonging to each cluster, and counts how many data points are assigned to each of the K clusters (cluster count). P_i stores these local centroids and cluster counts to answer queries from its neighbors.

In the second step, P_i sends a poll message, comprising its ID and current iteration number, to its immediate neighbor peers and awaits their

responses. Each response message from a neighboring peer P_a contains the peer's locally updated centroids and cluster counts for the current iteration. Once all P_i's immediate neighbors respond or cease to be neighbors, P_i updates K cluster centroids by taking a weighted average of its own centroid plus all the centroids it received (using cluster counts as weights). It then moves to the next iteration of K-means and repeats the whole process.

If the new centroid's maximum change in position after an iteration remains above termination threshold, P_i goes on to iteration $k + 1$; otherwise, it enters the *terminated* state.

In addition to executing these steps, each peer must respond to any polling message received from its neighbors at any point. If P_i receives a polling message during its kth iteration from peer P_h, which is at iteration $\hat{k}$ as long as $\hat{k} \leq k$, P_i sends its local centroids and counts corresponding to iteration $\hat{k}$. Otherwise, P_i places this poll message in a queue and checks at every iteration whether its current iteration equals $\hat{k}$. The moment P_i finishes its own $\hat{k}$th iteration, it responds to P_h's pending request. Any peer P_i can enter a terminated state at the end of iteration k if its cluster centroids change less frequently than the termination threshold. Once P_i is in the terminated state, it no longer updates its centroids or sends polling messages; instead, it sends only responses — for example, responding to polling messages from peer h for iteration $\hat{k}$ by sending its local centroids and counts corresponding to iteration $\min\{\hat{k}, k\}$. Therefore, once all peers enter the terminated state, all communication ceases and the algorithm terminates.

The algorithm includes a simple mechanism to detect and adjust to network and data changes: any peer joining the network can synchronize its clustering computation with the ongoing clustering computation in the network by starting from the minimum iteration of K-means in its neighborhood. Changes in any peer's data during clustering simply reassign the peer's cluster centroids before the peer moves on to the next iteration.

In extensive experimentation, our algorithm showed better than 90 percent clustering accuracy, in comparison to the centralized K-means clustering algorithm (the hypothetical case in which data present in all peers are integrated and standard K-means is applied on the data as a whole).[7]

Although this algorithm lacks a theoretical proof of convergence because it chooses only immediate neighboring peers rather than uniformly randomly sampled peers, its high accuracy and low communication cost (compared to centralizing the data from all peers) shows great potential for addressing the clustering problem in P2P networks.

Several algorithms have emerged to address basic DDM problems, but multiple challenges remain before they'll be mature enough to integrate with real P2P applications. For example, most existing P2P data-mining algorithms rely on asymptotic convergence properties. We need P2P data mining algorithms with performance bounds (regarding accuracy and communication costs, for example). We also need to quantify how the algorithm would behave over a given finite amount of time, and we might need a way to quantify advanced properties, such as the algorithm's stability, and develop techniques that can handle the nonstationary distributions generated by environments such as Manets.

Exact P2P algorithms are usually restricted to functions with local representations in the given network as with mean computation. Approximate techniques like those we've employed can help computing functions (such as clustering in P2P networks) for which no local algorithm have yet been developed. We believe we'll see several interesting applications for P2P network threat detection, P2P search, information retrieval, electronic commerce in Manets, and other such areas. As we mentioned earlier, privacy-preserving and trust-management techniques will likely play important roles in future P2P applications.

References

1. J. Vaidya and C. Clifton, "Privacy Preserving K-Means Clustering over Vertically Partitioned Data," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, ACM press, 2003, pp. 206–215.
2. K. Das, K. Bhaduri, and H. Kargupta, "Identifying Significant Inner Product Elements in a Peer–to–Peer Network," In communication, to be published 2006.
3. R. Wolff, K. Bhaduri, and H. Kargupta, "Local l2-Thresholding Based Data Mining in Peer-to-Peer Systems," *Proc. 2006 SIAM Conference on Data Mining* (SDM 06), SIAM press, 2006, pp. 430–441.
4. R. Wolff and A. Schuster, "Association Rule Mining in Peer-to-Peer Systems," *IEEE Trans. Systems, Man, and Cybernetics, Part B*, vol. 34, no. 6, 2004, pp. 2426–2438.
5. M. Wurst, K. Morik. "Distributed Feature Extraction in a P2P Setting — A Case Study." *Future Generation Computer Systems*, special issue on data mining; to be published.
6. W. Muller, M. Eisenhardt, and A. Henrich, "Classifying Documents by Distributed P2P Clustering," *Proc. Informatik*

2003, GI Jahrestagung, Lecture Notes in Informatics, 2003, pp. 286–291.

7. S. Datta, C. Giannella, and H. Kargupta, "*K*-Means Clustering over Large, Dynamic Networks," *Proc. 2006 SIAM Conf. Data Mining* (SDM 06), SIAM Press, 2006, pp. 153–164.

Souptik Datta is a PhD student and research assistant at the University of Maryland, Baltimore County. His research interests include peer-to-peer data mining and privacy-preserving data mining. Datta has an MS in computer science from University of Maryland, Baltimore County. He is a student member of the IEEE and winner of the best research paper award at the IEEE International Conference on Data Mining (ICDM) in 2003. Contact him at souptik1@cs.umbc.edu.

Kanishka Bhaduri is a PhD student at the University of Maryland, Baltimore County. His research interests include distributed data mining, data stream mining, and statistical data mining. Bhaduri has a BE in computer science and engineering from Jadavpur University, Kolkata, India. Contact him at kanishk1@cs.umbc.edu.

Chris Giannella is a post-doctoral research fellow at the University of Maryland, Baltimore County. His research interests include database theory and distributed and privacy-preserving data mining. Giannella has a PhD in computer science from Indiana University. Contact him at cgiannel@cs.umbc.edu.

Ran Wolff is a research associate at the Technion – Israel Institute of Technology. His research interests include data mining (and other similar computations) and privacy issues in large-scale distributed systems, such as grid systems and peer-to-peer and sensor networks. Wolff has a PhD in computer science from Technion. Contact him at ranw@cs.umbc.edu.

Hillol Kargupta is an associate professor at the University of Maryland, Baltimore County, and a cofounder of Agnik. His research interests include distributed and ubiquitous data mining. Kargupta has a PhD in computer science from the University of Illinois at Urbana Champaign. He is coauthor of *Advances in Parallel and Distributed Data Mining* (MIT Press). Contact him at hillol@cs.umbc.edu, or see www.cs.umbc.edu/~hillol.